Noisy Language Modeling Framework Using Neural Network Techniques

By

Jun Li

April 2009

DECLARATION

I hereby declare that all the work presented in the thesis in partial fulfillment of the requirements for the degree of Doctorate of Philosophy is original and my own except where otherwise specified.

Jun Li

ABSTRACT

The text entry interaction between human and computer could be noisy. For example, the typing stream is a reflection of user typing behaviours which include user particular vocabulary, typing habits and typing performance. As computer users inevitably make errors, a typing stream generated from using computer QWERTY keyboard implies all users' self-rectification actions rather that a clean text. Therefore this research develops a novel intermediate layer language modeling framework called ALMIL (i.e. Adaptive Language Modelling Intermediate Layer) which is seen as a communication language layer between human and computer to analyze noisy language stream and provide users with two fundamental functions, namely Text Prediction and Text Correction. A specific research case of ALMIL called Intelligent Keyboard (IK) aiming to develop a user oriented hybrid framework with self-adaptive function to help people using QWERTY keyboard more effectively is also conducted.

In order to explore the methodologies, influential factors and demonstrate the feasibility of the frameworks, a comprehensive neural networks and language modeling process is carried out. Several neural network models which include a Focused Time-Delay Neural Network model (FTDNN) to model non-noisy, noisy and typing stream datasets, a Time Gap Neural Network model (TGNN) to simulate and predict user typing time gap between two consecutive letters, a Prediction using

Time Gap model (PTG) to predict right symbols based on user typing speed, a Probabilistic Neural Network based model (PNN) to simulate 'Hitting Adjacent Key Errors', and a Word List real-time ranking model (WLR) on prioritizing prediction results are developed. All the models have been demonstrated, and shown high performance through a set of experiments using a range of dataset.

In essence, this research brings forth a creative concept – intermediate layer language modeling framework for noisy language processing, pioneers a comprehensive neural networks modelling process, and originally develops a hybrid solution to combine multiple correction functions based on an evolutionary ranking approach. It produces a significant contribution in the area of neural networks application and shows a direction for Human-Computer noisy language interaction research. Also a full report on disabled people typing behaviour, a development of EIM application and a universal pre-processing tool for all neural networks modelling and *n*-gram calculation will benefit both research and commerce.

A thesis submitted in partial fulfillment of the requirements of London

Metropolitan University for the degree of Doctor of Philosophy

Faculty of Computing

London Metropolitan University, London

April 2009

To my family

ACKNOWLEDGMENTS

This report is an outcome of the contributions of many people. I would like to express the deep gratitude to my director of research Dr. Karim Ouazzane for his dutiful supervision and invaluable guidance. And also I would like to thank my supervisors Dr. Yanguo Jing for his constructive suggestions and full support in improving the quality of the research, and Prof. Hassan Kazemian for his help.

This thesis would not have been possible without the support of the staff of Charity – Disability Essex. In particular, I would like to thank Richard Boyd, Marielle Brouwer and Pete Collings. It is your inspiration, generosity and the love to life encourage me not only to fulfill this research but also to do my best in all matters of life. This research is funded by Technology Strategy Board Programme, Disability Essex, and Faculty of Computing of London Metropolitan University. I am very grateful to them for the full financial support.

Thanks also to my parents, Xiuying Song, Shugong Li who have been extremely understanding and supportive in my research, and my brother Yong Li for all the long walks with deep discussions on just everything that is interesting in this strange but fantastic world.

TABLE OF CONTENT

CHAPTER ONE INTRODUCTION 10

1.1 Computer text entry system 11

1.2 Noisy language modelling 12

1.3 Motivation, aim and objectives 13

1.3.1 An intermediate layer framework development and noisy language modelling 15

1.3.2 Typing stream framework development and typing stream modelling ... 15

1.3.3 Framework with combination functions and word list ranking 16

1.4 Chapters overview 17

CHAPTER TWO LITERATURE REVIEW 19

2.1 Introduction 20

2.2 Neural networks 21

2.2.1 Focused time-delay neural network 25

2.2.2 Elman network 26

2.2.3 Probabilistic neural network 28

2.2.4 Structuring neural network 30

2.3 Statistical language modelling 31

2.3.1 Prediction by Partial Matching 33

2.3.2 Information Entropy 34

2.4 Data pre-processing 35

2.5 String distance 40

2.6 Fitt's law 43

2.7 Text prediction and correction applications 44

2.7.1 Text prediction applications 44

2.7.2 Typing Correction Applications 49

2.7.3 Dasher 52

2.7.4 ProtoType 54

2.8 Summary 55

CHAPTER THREE A NOVEL FRAMEWORK FOR NOISY LANGUAGE ANALYSIS 57

3.1 Introduction 58
3.2 A novel intermediate layer language framework 58
3.3 Disabled keyboard users investigation.....66
3.4 Intelligent Keyboard framework 69
3.4.1 Intelligent Keyboard framework and Rationale 70
3.4.2 Intelligent Keyboard framework demonstration.....75
3.5 Summary 77
CHAPTER FOUR NEURAL NETWORK AND LANGUAGE MODELS DEVELOPMENT 79
4.1 Introduction 80
4.2 Experimental datasets.....81
4.3 System environment 85
4.4 Data processing tools 86
4.5 Input coding 90
4.6 Neural network models development outline.....92
4.7 Focused time-delay neural network modelling 94
4.7.1 FTDNN N-Gram Prediction.....96
4.7.2 N-Gram Prediction with noise 115
4.7.3 N-Gram Prediction with Typing Data.....121
4.7.4 FTDNN Modelling Summary 127
4.8 Time gap modelling 128
4.9 Prediction using time gap.....133
4.10 Probabilistic neural network modelling 136
4.11 Summary 147
CHAPTER FIVE INTELLIGENT KEYBOARD FURTHER DEVELOPMENT 149
5.1 Introduction 150
5.2 Further framework development 151
5.3 Word list neural network ranking 153
5.3 A pilot application 166
5.4 Summary 170
CHAPTER SIX DISCUSSION, CONCLUSION AND RECOMMENDATION FOR FUTURE WORK.....172
6.1 Discussions and conclusions.....173

6.1.1 Adaptive language modelling intermediate layer framework.....173
6.1.2 Intelligent Keyboard and its pilot application 174
6.1.3 FTDNN language modelling 175
6.1.4 Specific typing behaviours analysis using neural networks.....176
6.1.5 Conclusions.....177
6.2 Contributions.....179
6.3 Recommendations for future work.....183
REFERENCES 188
APPENDICES.....200
Appendix A MATLAB SOURCE CODE.....xxx
Appendix B MAIN ENSTATISTICS SOURCE CODE.....xxx
Appendix C MAIN INTELLIGENT KEYBOARD – ENGLISH INPUT METHOD PROGRAMS.....xxx
Appendix D NEURAL NETWORKS MODELLING RESULTS DIAGRAMS.....xxx
Appendix E INTELLIGENT KEYBOARD – UNITS AND MODULES ILLUSTRATION.....xxx
Appendix F VIRTUAL KEY CODES.....xxx
Appendix G USUAL NEUAL NETWORK ACTIVATION FUNCTION SIN MATLAB.....xxx
Appendix H PUBLICATIONS.....xxx

LIST OF FIGURES

Figure 1.1 Chapters structure of Noisy Language Modelling Framework Using Neural Network Techniques.....17

Figure 2.1 A representation of nervous system.....21

Figure 2.2 A neural network model with a neuron.....22

Figure 2.3 A three layer Focused Time-Delay Neural Network.....25

Figure 2.4 Elman neural networks architecture (1990).....27

Figure 2.5 A representation of Probabilistic Neural Network architecture.....28

Figure 2.6 *1*-gram relative frequencies of letters in general English plain text.....32

Figure 2.7 An example of preprocessing sampling methods.....40

Figure 2.8 Fitt's law expression based on keyboard.....44

Figure 2.9 Jianhua & Graeme word prediction model.....45

Figure 2.10 Repeat and Predict key composition rules.....46

Figure 2.11 A Dasher interface example.....53

Figure 2.12 ProtoType program flow diagram.....54

Figure 3.1 Adaptive language modelling intermediate layer framework.....61

Figure 3.2 The architecture of Intelligent Keyboard.....72

Figure 4.1 A piece of KeyCapture log sample.....82

Figure 4.2 A statistic of alphabet occurrences in typing stream.....84

Figure 4.3 Schematic representation of intelligent modelling.....87

Figure 4.4 Interface representation of intelligent models data pre-processing....89

Figure 4.5 Neural network with unary coding.....91

Figure 4.6 Neural network with ASCII coding.....91

Figure 4.7 The experiments procedure with datasets one, two & three.....93

Figure 4.8 Architecture of Focused Time-Delay Neural Network.....98

Figure 4.9 Presentation of *n*-gram FTDNN language modelling process.....99

Figure 4.10 One & nine-gram Hitting Rate curves.....102

Figure 4.11 N-gram First and First Three Hitting Rate curves.....105

Figure 4.12 [1, 2, 3, 5, 7, 9, 15] hidden neurons Hitting Rate curves.....107

Figure 4.13 Symbols distribution with 2&3-Gram and 25 hidden neurons.....109

Figure 4.14 [1, 2, 3, 5, 7, 9] gram s-entropy curves.....113

Figure 4.15 N-gram prediction with Noise Rate = 0.001.....117

Figure 4.16 N-gram prediction with Noise Rate = 0.01.....119

Figure 4.17 N-gram prediction with Noise Rate = 0.1.....119

Figure 4.18 2 & 3-gram Hitting Rate curves under noise rates.....121

Figure 4.19 [1, 3, 7, 9] – gram typing stream individual Hitting Rates.....124

Figure 4.20 [1, 3, 5, 7, 9] – gram typing stream Hitting Rates.....126

Figure 4.21 [1, 3, 5, 7, 9] – gram typing stream s-entropy curves.....126

Figure 4.22 Modelling time gap using A→Z sequence.....130

Figure 4.23 Modelling time gap using QWERTY sequence.....131

Figure 4.24 Absolute Frequency of PTG model Correction Rate.....135

Figure 4.25 A QWERTY keyboard layout sample.....139

Figure 4.26 Relationship – angle between keys and its surrounding keys D,E,A.....139

Figure 4.27 Key distances coordinate for PNN classification.....141

Figure 4.28 Hitting adjacent key prediction rates based on PPN network.....145

Figure 5.1 Presentation of word list neural network ranking model (WLR).....158

Figure 5.2 Sampling Points representation of WLR modeling.....162

Figure 5.3 A sample of WLR model experimental dataset.....162

Figure 5.4 The comparison of Neural network ranking First Hitting Rate and L.M.T Rates.....165

Figure 5.5 Relationship of Windows, applications and EIM.....167

Figure 5.6 EIM interface demonstration.....169

Figure 5.7 Sampling – Evolution of Correction Rates.....169

LIST OF TABLES

Table 4.1 Unary code and ASCII samples.....90

Table 4.2 Experimental results of FTDNN model with dataset one.....114

Table 4.3 Neural Networks modelling and the related performances.....147

Abbreviations

AI	Artificial Intelligence
ALMIL	Adaptive Language Modelling Intermediate Layer
BNC	British National Corpus
BP	BackPropagation
BPC	Bits Per Character
BPTT	BackPropagation Through Time algorithm
DAT	Distance, Angle and Time Gap NN model
DATP	Distance, Angle and Time Gap PNN model
DPSD	Discrete Prediction Symbols Distribution
EDPA	Essex Disabled People Association
EIM	English Input Method
EMD	Error Margin Distance
FLM	Factored Language Model
FTDNN	Focused Time-Delay Neural Network
MHP	the Model Human Processor
MLP	Multilayer Perceptions
MRF	Mistakes Recording function
NPM	Nestor Prediction Measurement
HMM	Hidden Markov Model
HR	Hitting Rate
IMM	Input Method Manager

FT	First Three hitting rate
HCI	Human Computer Interaction
IK	Intelligent Keyboard
IME	Input Method Editor
LM	Language Modelling
ML	Machine Learning
L.M.T	Levenshtein word distance, Metaphone and Two-Gram word
NLP	Natural Language Processing
NN	Neural Network
PMI	Pointwise Mutual Information
PNN	Probabilistic Neural Networks
PPM	Prediction by Partial Matching
PRI	priority
PTG	Prediction using Time Gap
SM	SiMulation
SP	word-list Success Prediction
SRS	Simple Random Sample
SRSWOR	Simple Random Sample Without Replacement
SRSWR	Simple Random Sample With Replacement
TGNN	Time Gap Neural Network model
VKC	Virtual Key Code
WLR	Word List neural network Ranking model

CHAPTER ONE

INTRODUCTION

1.1 Computer text entry system

Computer text entry may be full of noises. For example, computer keyboard users inevitably make typing mistakes and their typing stream implies all users' self-rectification actions. These may produce a great negative influence on the accessibility and usability of applications that need text entry. Efforts have been made based on different technologies such as spell checking and natural language processing, but few tools can intelligently identify new genre of mistakes. Moreover although distinct solutions such as Metaphone [Philips, 1990] and *n*-grams have been implemented to correct user typing mistakes, an optimum solution is hardly identified among them. It is desirable to develop a hybrid solution based on these technologies to achieve an optimal result.

For input efficiency and accuracy considerations, text entry requires text prediction as well as correction. Research on predictive text input technologies have been undertaken in multiple directions such as language modelling and natural language processing, and many products such as Dasher [Ward & MacKay et al., 1997-2008] and Prototype [Sensory Software International Ltd, 2007] have been available on the market. However, those technologies have been used exclusively and the whole issue hasn't been addressed well.

Neural Network is a non-linear statistical data modelling tool, which can be used to model complex relationships between inputs and outputs or to find patterns in datasets. Its related research has been flourishing in areas like human activity recognition and category classification, although they are hardly traced to be applied to noisy text entry such as user typing stream processing.

1.2 Noisy language modelling

The goal of Statistical Language Modelling (SLM) is to build a statistical language model that can estimate the distribution of natural language as accurately as possible. Language model assigns probabilities to sequences of symbols or words, and is used in many natural language processing applications such as speech recognition and data prediction. For example, given string *S*= *{student}*, some *2*-gram prediction cases are,

'st' → *'u'*
'tu' → *'d'*
'en' → *'t'*

Compared with natural language modelling, noisy language modelling is used to estimate the probabilities of a set of symbols or words based on noisy historical data. For example given a noisy string *S*= *{sutdent}* whose corresponding right string should be 'student', where there exists a Letters Reverse error ('u' → 't'), some *2*-gram prediction with noise cases are,

'su' → *'u'*
'ut' → *'d'*
'en' → *'t'*

As shown above, the prediction cases not only include one step forward symbol forecast but also symbol correction (e.g. 'su' → 'u' rather that 'su' → 't'). In language modelling area, quite a few of research have been carried out such as *n*-gram Prediction and Prediction by Partial Matching, which have been applied to clean text efficiently, but the usage on active text with noisy data is hardly found. For example, the research on typing stream generated by computer QWERTY keyboard user, which implies all users' self-rectification actions, has been underestimated [Soukoreff & MacKenzie, 2003].

1.3 Motivation, aim and objectives

Through computer, an individual interacts with applications by producing events which are triggered by appropriate input devices and transformed into values expected by a target system. Typical input devices which can help users with text entry include keyboard, mouse, and camera so on.

As indicated in the previous section, the text entry interaction between human and computer could be noisy. For example, the typing stream is a reflection of a user's typing behaviour that includes a user's particular vocabulary, typing habits and typing performance. As computer users inevitably make errors, a typing stream implies all users' self-rectification actions rather that a clean text. Here is another example: one of the facial recognition functions is to allow computer to interpret the speaker's speech and spot the text they intend to express along with their facial expression changes, which is hardly completely accurate.

Currently all these happen during a communication process without recognizing the mistakes that may be incurred by a user when using input devices. A requirement to design an intelligent framework as a communication interface between input devices and applications to tackle the noisy input in certain language context is needed. It will also provide a platform for cooperation between text entry applications and input devices, or between each two input devices such as the audio and video capture in bimodal speech recognition.

Motivated by these requirements, this research is intended to propose a self-adaptive intermediate layer language modelling framework which can be seen as a mapping language layer as well as a filter platform between user and computer to process the noisy language stream. All input noises can be identified and

filtered through this layer, and a clean data stream can be presented to upper layers (e.g. applications). In turn, user feedback is used to further strengthen the accuracy of this intermediate layer.

As the framework will be extremely useful for text entry such as typing stream from a computer keyboard, a demonstration will be studied aiming to help people in using a QWERTY keyboard more effectively. Simultaneously user typing streams can be researched in particular and simulated systematically by developing distinct neural network models both based on generally analyzing plain text, noisy data and user typing stream, and specifically studying specific user typing behaviours such as 'hitting adjacent key errors'. Some state of the art neural networks such as Focused Time-Delay Neural Network, Elman Network, and Probabilistic Neural Network can be investigated and further applied. The neural networks structures can be extendable both at input layer and at hidden layer, and able to handle real-time interaction issues. The neural networks experimental results are expected to be helpful in practical use such as in prediction and correction.

The major purpose of this research is to demonstrate the feasibility and the rationale of the intelligent intermediate layer language framework through its development and related neural networks modelling. Factors which may have an influence on the functionality, accuracy and efficiency of the framework are expected to be identified.

The objectives of research are intended to make the following contributions to knowledge by studying areas shown below, which are also the main hypotheses of this thesis.

1.3.1 An intermediate layer framework development and noisy language modelling

To develop an intelligent framework to analyze noisy language stream data and offset the communication gap between a user and computer applications. This framework should provide two fundamental functions at least, namely, text correction and prediction. Through it, the noises of text entry can be filtered significantly; meanwhile the text entry efficiency can be improved. Along with the framework, several neural network language models will be developed. This will be a simulation to the designed intermediate layer framework on a global basis and require a time-efficient neural network with high computation and memory capacity. As prediction rates and correction rates can be varied with different neural network structures, an extendable neural network design might be needed. Moreover, the experimental datasets could be collected from difference sources such as articles and disabled user typing stream, and a noise incremental distribution scheme may be conducted along with the modelling process.

1.3.2 Typing stream framework development and typing stream modelling

As user typing stream from computer QWERTY keyboard is full of noises, it can be a good research case to demonstrate the intermediate layer framework. Because user typing behaviours are varied, particularly with disabled people, a user investigation needs to be conducted first. Subsequently a user oriented framework as an intermediate layer language model to filter typing mistakes and predict typing intention is required to be developed. Since it is difficult for a

single algorithm to deal with all the problems appeared in typing stream, distinct prediction and correction algorithms such as Metaphone and word *2*-gram may be integrated into this framework. As user typing streams imply user specific typing behaviours, it may also be possible to analyze this language stream using multiple neural network models. Distinct factors which may affect user typing efficiency and accuracy are expected to be identified. These neural network models are able to provide a comprehensive data analysis such as data prediction and correction on a noisy language basis.

1.3.3 Framework with combination functions and word list ranking

Given a vast variety of user typing behaviours, it is difficult to use a single algorithm to deal with the noisy text entry. In practice, a combination model based on multiple correction functions to present multiple answers for user selection may be a better solution, and it is also necessary to develop a ranking model to prioritize the answers. However, the ranking may be affected by many factors such as time, text content and user feedback, so an evolutionary ranking model based on data updates and neural network learning to combine distinct correction algorithms to produce an optimal prediction needs to be developed.

Intermediate layer framework concept, typing data stream neural networks modelling and word list online ranking are all unique and preliminary contributions. This thesis will lead to a new research area on noisy language modelling study.

1.4 Chapters overview

The thesis is structured in chapters as shown in Figure 1.1. Chapter 2 reviews related research work. The research work undertaken for this study is illustrated in chapter 3 & 4. Chapter 5 describes further work based on the findings of chapter 3 & 4. Following the figure, a more detailed description of each chapter is presented.

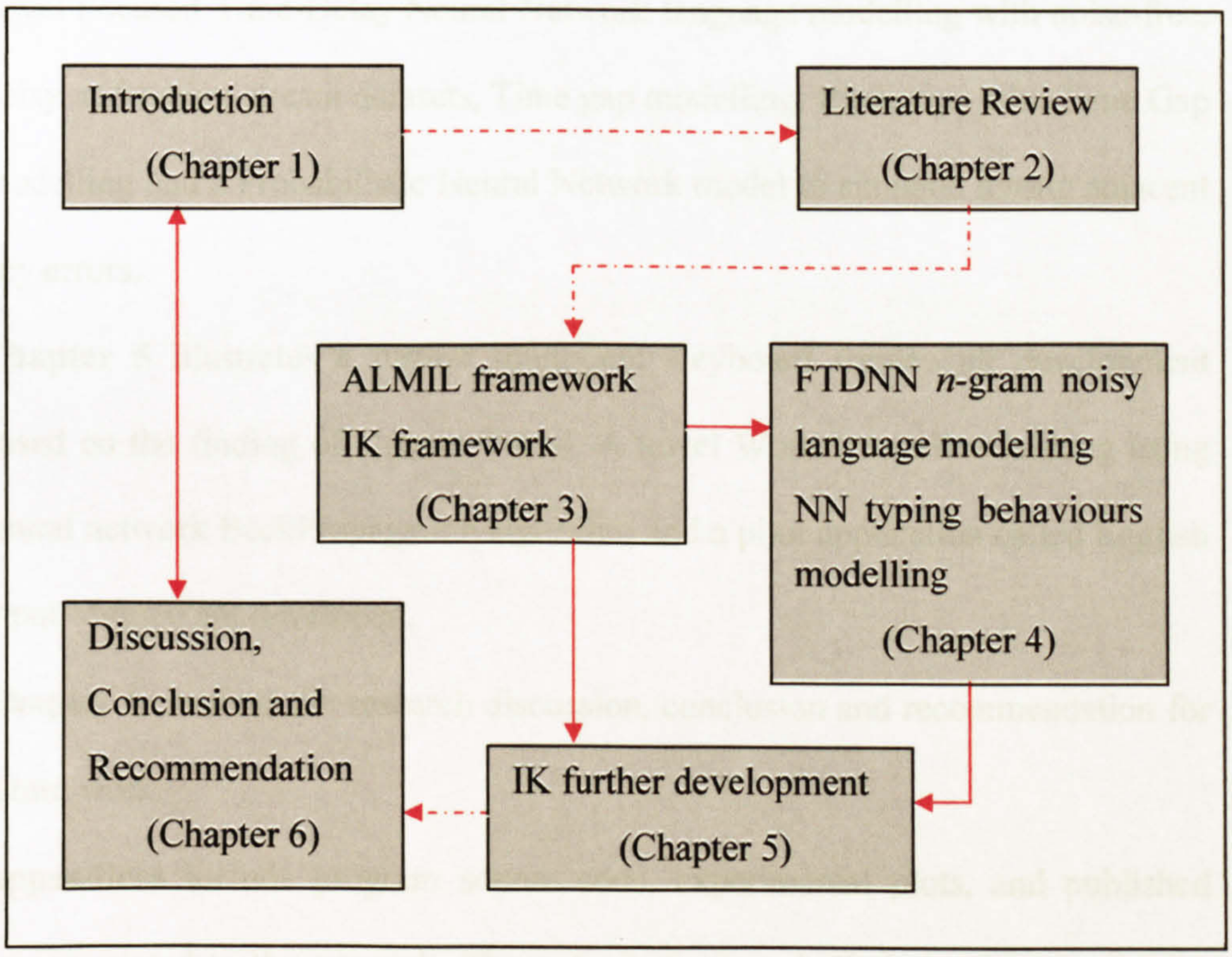


Figure 1.1 Chapters structure of Noisy Language Modelling Framework Using Neural Network Techniques

Chapter 1 gives an introduction to noisy language modelling, and presents the motivation, aim, objectives and hypotheses of the research.

Chapter 2 reviews Neural Network models, Statistical Language Modelling algorithms, Data preprocessing methods and other technologies such as String distance algorithms and Fitt's law equation used in the research.

Chapter 3 develops an intermediate layer noisy language modelling framework called ALMIL and presents a demonstration of the ALMIL framework called Intelligent Keyboard.

Chapter 4 illustrates full neural network models development, which includes novel Focused Time-Delay Neural Network language modelling with noise-free, noisy and typing stream datasets, Time gap modelling, Prediction with Time Gap modelling and a Probabilistic Neural Network model to simulate hitting adjacent key errors.

Chapter 5 illustrates a further Intelligent Keyboard framework development based on the finding of chapter 3 & 4. A novel Word list online ranking using neural network BackPropagation algorithm and a pilot application called English Input Method are developed.

Chapter 6 presents the research discussion, conclusion and recommendation for future work.

Appendices include program source code, experimental plots, and published papers related to the research. Throughout the text, italic is used to emphasize defined terms, and bold is used to highlight main ideas.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

Neural network language modelling is a method to apply particular neural network models to model natural languages. In this chapter both neural network models and statistical language modelling methods are reviewed. First, neural network rationale, architectures, learning algorithms and some state of the art networks which include Focused Time-Delay Neural Network, Elman Network, and Probabilistic Neural Network are described. Then the language modelling rationale, Prediction by Partial Matching (PPM) algorithm and entropy concept are introduced respectively.

As well known, the raw data need to be preprocessed before it can be used by neural networks. A six-step approach with distinct algorithms for each step of neural networks data pre-processing to achieve best performance for the training dataset is introduced.

One objective of the research is to develop a framework to deal with noisy data stream. Therefore, several text prediction and correction applications such as Dasher and ProtoType, and their related techniques are examined. However, the disadvantages of these technologies such as failure of meeting peculiar needs and the lack of self-learning ability are also mentioned.

Finally two other useful technologies are reviewed, namely, String Distance and Fitt's law. String Distance is a method to compare the difference between two character sequences, whereas Fitt's law uses human-computer interaction technology to study user's typing movement. In this research both of them will be applied to user's typing behaviour models and framework development.

2.2 Neural networks

The human nervous system may be viewed as a three-stage system, as depicted in the block diagram of Figure 2.1 [Simon Haykin, 1999].

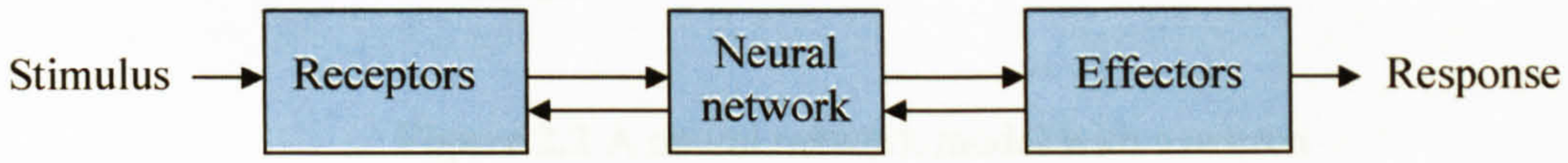


Figure 2.1 A representation of nervous system

Signals received by receptors are processed in the neural network and the responses are delivered to the outside world by effectors. Artificial neural networks are motivated by neuro-biological theory involving the behaviour of the brain as a network of units called neurons, which constitute a massively parallel-distributed processor through connections, called synapses. It is estimated that the human brain is likely to have around 10 billion neurons each connected on average to 10,000 other neurons.

The basic attributes of a neural network may be divided into architecture and functional properties. The architecture defines the network structure, that is, the number of artificial neurons in the network and their interconnectivity with familiar characteristics such as inputs, synaptic strengths, activation, outputs, and bias. Functional properties define how the neural network learns, recalls, associates and classifies. A basic model of a neuron *i* is illustrated in Figure 2.2 [Stamtios, 1996].

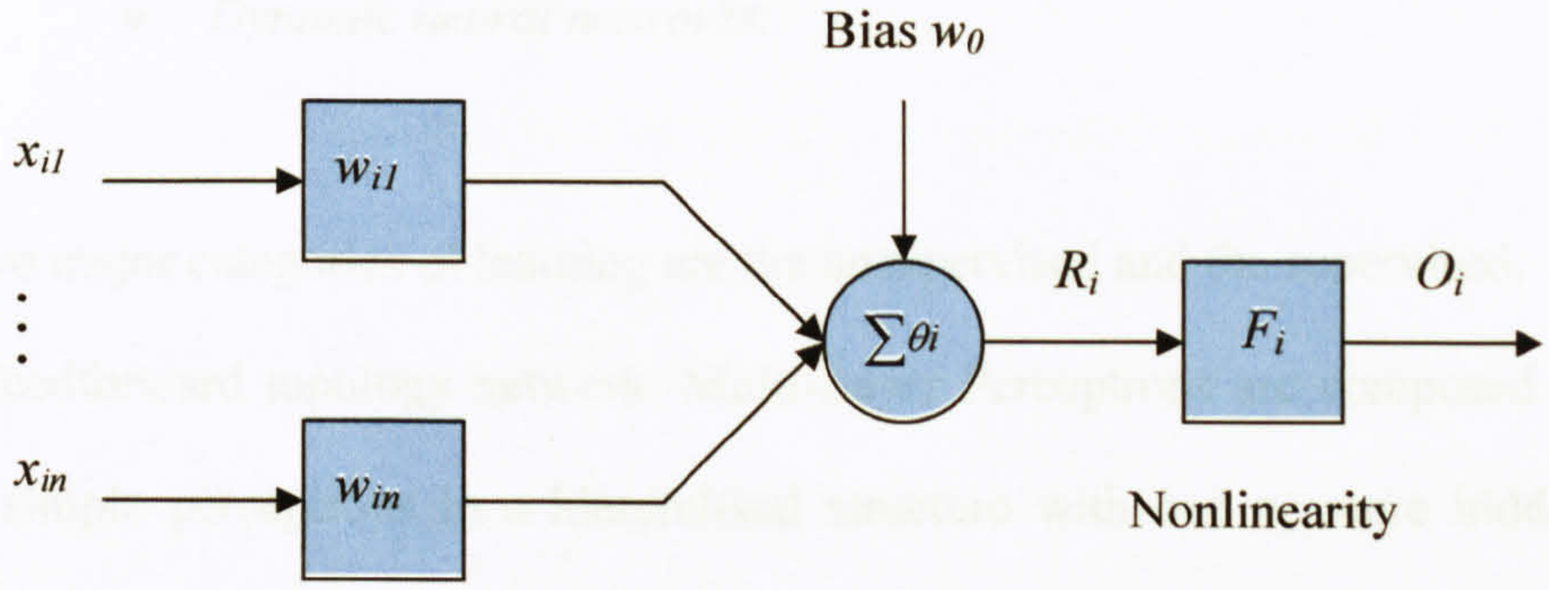


Figure 2.2 A neural network model with a neuron

The general neuron i has a set of n inputs $\{x_{i1}...x_{ij}...x_{in}\}$, where the subscript j takes values from *1* to n and indicates the source of the input signal, accordingly associated with weights w_{ij}. In addition, it has a bias term $w_{0,}$ a threshold value Θ_i that has to be reached or exceeded for the neuron to produce a signal, a nonlinearity transfer function F_i that acts on the produced activation signal R_i, and an output O_i after the nonlinearity function. The transfer function of the basic model is described as follows,

$$O_i = F_i\left(\sum_{j=1}^{n} w_{ij}x_{ij}\right), \text{ as } \quad R_i = \sum_{j=1}^{n} w_{ij}x_{ij} \tag{2.1}$$

In general three fundamentally different classes of network architectures can be identified as,

- *Single-layer feedforward neural networks*
- *Multi-layer feedforward neural networks*

- *Dynamic neural networks.*

The two major categories of learning are the unsupervised and the supervised.

As a feedforward topology network, Multi-Layer Perceptrons are composed of many simple perceptrons in a hierarchical structure with one or more hidden layers between input and output layers. The most commonly used learning algorithm is BackPropagation. Its sigmoidal nonlinearity is expressed as,

$$F_i(R_i) = \frac{1}{1+\exp(-kR_i)} \tag{2.2}$$

Where R_i is the weighted sum of all synaptic inputs plus the bias of neuron i; k is the gain of sigmoid that varies monotonically from $-\infty$ to $+\infty$, and F_i is the output of neuron i.

BackPropagation is a supervised learning method, and is an implementation of the Delta rule, whose architecture belongs to feedforward network. Its actual algorithm can be illustrated as,

*1. **Initialize** the weights in the network*
*2. **Do***
    ***For each** example e in the training set do*
        1. O = neural-net-output (network, e)
        2. T = teacher output for e
        3. Calculate error (T - O) at the output units
        4. Compute and update new_wi for all weights to output layer
        *5. **For each** hidden layer*
            1. Calculate error at the hidden units
            2. Compute and update new_wi for all weights to hidden layer
*3. **Until** all examples classified correctly or stopping criterion satisfied*
*4. **Return***

There are no well-defined criteria for stopping its operation. One of sensible convergence criteria reported by Kramer and Sangiovanni-Vincentelli [1989] is: "The back-propagation algorithm is considered to have converged when the Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold". By considering the cost function or error measure as stationary at the local or global minimum, Simon Haykin [1999] suggested a different criterion of convergence: "The back-propagation algorithm is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small". He concluded that the rate of change in the average squared error should lie in the range of [*0.1%, 1%*] per epoch. Practical methods are also used by verifying if the network performance is apparently adequate.

Given the MultiLayer Perceptrons as the basic building block, three different architectural layouts of dynamic networks based upon the additional signals at the side of input can be further defined. The signals may come from historical input, output layer or hidden layers. Then, dynamic networks can simply be divided into two categories based on whether they are feedback-related, for example, those that have only feedforward connections (e.g. Focused Time-Delay Neural Network), and those that have feedback, or recurrent connections (e.g. Elman Networks). Back-propagation through time algorithm (BPTT) is a leading learning algorithm for training a recurrent network, which is an extension of the standard back-propagation algorithm. Another typical learning algorithm for recurrent networks is a real-time recurrent learning algorithm.

In the following sections, specific networks including Focused time-delay neural network, Elman Networks, and Probabilistic neural networks are discussed.

2.2.1 Focused time-delay neural network

The Focused Time-Delay Neural Network (FTDNN) consists of a feedforward network with a tapped delay line at the input. This is part of a general class of dynamic networks called focused networks, in which the dynamics appear only at the input layer of a static multilayer feedforward network.

This network is well suited to time-series prediction, which is stimulated through a short-term memory. Given an input signal consisting of the present value *x(n)* and the *p* past values, *x(n-1)*, ... , *x(n-p)*, stored in a delay line short-term memory of order *p*, a three layers structure with three hidden neurons and two outputs is shown in Figure 2.3.

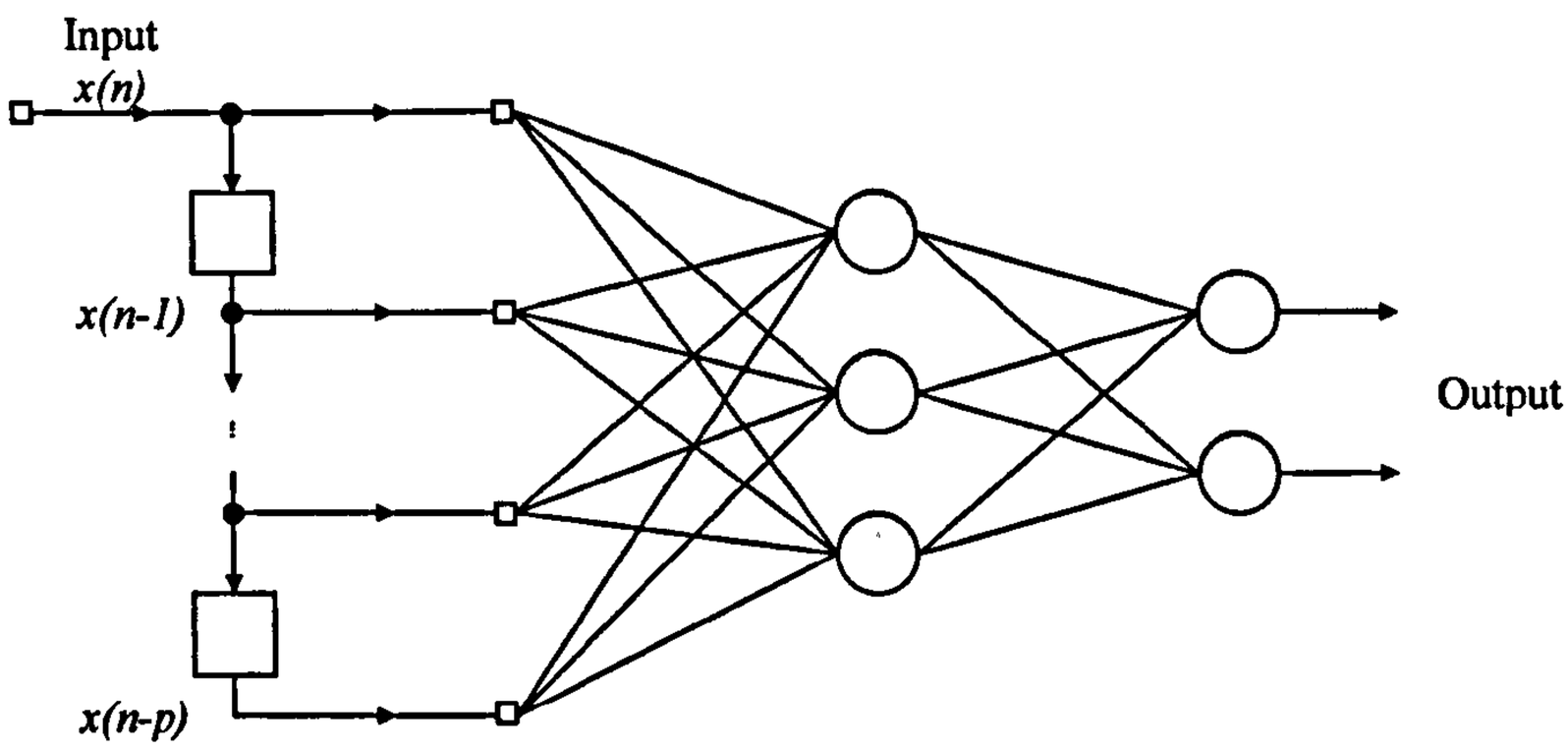


Figure 2.3 A three layer Focused Time-Delay Neural Network

The tapped delay line memory captures temporal information contained in the input signals, and neurons embed that information in their own synaptic weights. The standard BackPropagation algorithm can be used as a learning method of FTDNN network. It does not require dynamic BackPropagation to compute the

network gradient. This is because the tapped delay line appears only at the input of the network, and contains no feedback loops or adjustable parameters. For this reason this network trains faster than other dynamic networks.

The Focused Time-Delay Neural Network has been well applied to motion recognition used by surveillance system, multimodal human computer interface and traffic control system. For example, Woo [2000] proposed a FTDNN based scheme to extract information from dynamic gestures of dance performance and learn dancer's emotional intention. The experimental results demonstrated that consistent emotional analysis can be achieved using FTDNN based scheme, which maps between local features and symbolic representation of emotion in real-time. A Further development of FTDNN model based on both space and time domains was also proposed [Lin, 1999] for lip-reading. In the experiment, the space based FTDNN model was able to recognize the lip motions in a high performance based on the inputs of real image sequences.

2.2.2 Elman network

Compared to tapped delay method of Focused Time-Delay Neural Network in terms of input, Elman networks [Elman, 1990] are three-layer BackPropagation networks, with the addition of a feedback connection from the output of the hidden layer to its input. This feedback path allows Elman networks to learn to recognize and generate temporal patterns, as well as spatial patterns. Elman [1990] used neural network architecture shown in Figure 2.4 to explain the network processing.

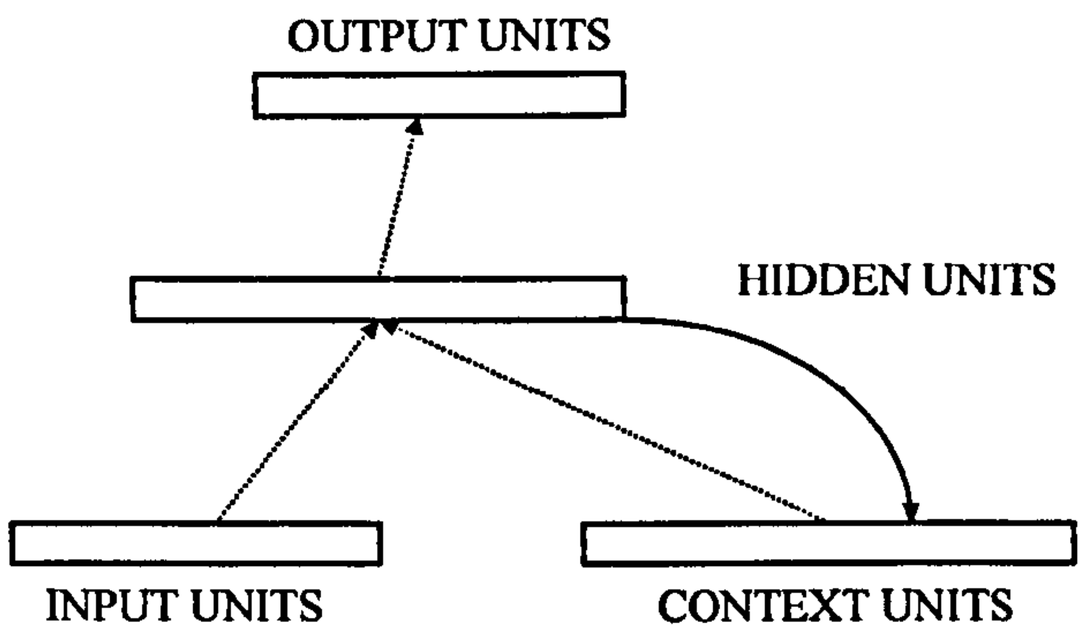


Figure 2.4 Elman neural networks architecture [1990]

As can be seen from Figure 2.4, Elman network's input accept two categories of data: one is from out sources to input units, and another is from hidden units of itself to context units.

Let's consider a sequential input with order sensitivity at time $t = 1$, the input units receive the first input and the context units are initialized to a certain value. Both input and context units activate hidden units. Then hidden units feed forward to output units and simultaneously feed back to context units. If a learning algorithm is applied, the output will be compared with target output and the delta error will be backpropagated to adjust all the weights. At the next time step $t+1$ the above sequence is repeated. Then at the time $(t+1)$, the context units contain values, which are exactly the hidden unit values at time t. These context units thus provide the network with memorial history.

Elman network can be used to recognize both spatial and temporal patterns. Schellhammer et al. [1998] used Elman network to learn sequences of word categories in a text. The grammar induced by the network was made explicit by cluster analysis. The output of *k*-means cluster analysis is converted to state-

transition diagrams which represent the grammar learned by the network. Elman network was also applied to human activity recognitions by head movement from a set of color image sequences [Henry, 2003]. A considerable high recognition rate of *92.5%* was achieved, compared to *85.5%* and *87%* obtained by the traditional k-NNR and HMM classifiers.

2.2.3 Probabilistic neural network

Probabilistic neural networks (PNN) are a type of radial basis network suitable for classification problems. It is a feedforward network built with three layers. They are derived from Bayes Decision Networks [Specht, 1988 & 1990]. The architecture of a PNN is shown in Figure 2.5.

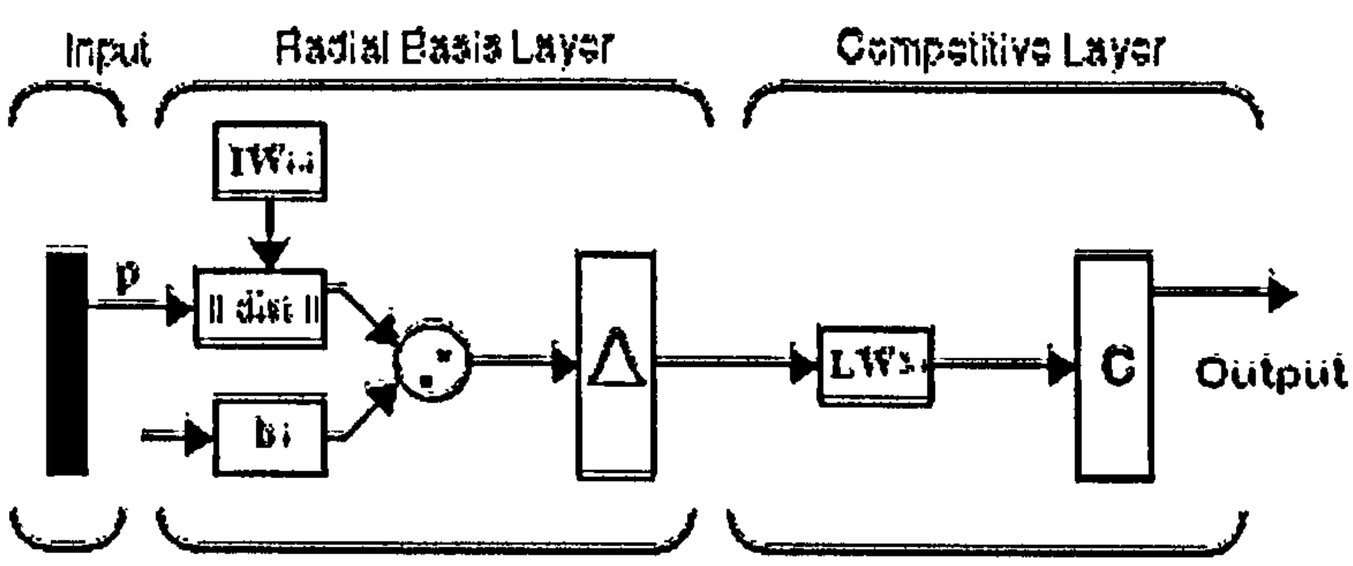


Figure 2.5 A representation of Probabilistic Neural Network architecture [The MathWorks, 2009]

When an input is presented, the first layer computes distances from the input vector to the training input vectors and produces a vector whose elements indicate how close the input is to a training input. The second layer sums these contributions for each class of inputs to produce a vector of probabilities. Finally,

a competing transfer function on the output of the second layer picks the maximum of these probabilities, and produces a '*1*' for that class and a '*0*' for the other classes.

The probabilistic neural networks offer the following advantages [Wasserman, 1993],

- *Rapid training speed and enables incremental training.*
- *Robustness to noisy examples.*
- *Guaranteed convergence if enough training examples are provided.*

As the BackPropagation algorithm is used for training multilayer neural networks, the probabilistic neural networks also possesses some other useful characteristics as presented below.

- ***Learning capacity.*** *It captures the relationships between given training examples and their given classification.*
- ***Generalization ability.*** *It identifies the commonalities in the training examples and allows performing classification of unseen examples from the predefined classes.*

Ganchev1 et al. [2002] applied Probabilistic Neural Networks (PNNs) as core classifiers to medium scale speaker recognition over fixed telephone networks. Two PNN-based open-set text-independent systems for speaker identification and speaker verification correspondingly were presented. The systems demonstrated a good reliability and robustness under noisy telephone conditions.

A Probabilistic Neural Network joint with fuzzy logic for human face identification [Anagnostopoulos 2003] was proposed to develop a computer-based face detection system. The Probabilistic Neural Network was trained for the identification of the facial areas, which were extracted using fuzzy logic rules. Experimental results showed that the overall identification performance was measured to be 83%. However, this performance level is achieved for frontal-parallel faces, since the classification performance deteriorates when extended to different views of a human face.

2.2.4 Structuring neural network

The number of layers and the number of processing elements per layer to a feedforward, back-propagation topology are important decisions. There is no universal answer to the layout of the network for any particular application. Some general rules have been followed as shown below.

Rule one: *As the complexity in the relationship between the input data and the desired output increases, the number of the processing elements in the hidden layers should also increase.*

Rule two: *If the process being modeled is separable into multiple stages, then additional hidden layer(s) may be required. If the process is not separable into stages, then additional layers may simply enable memorization of the training set, and not a true general solution effective with other data.*

Rule three: *The amount of training data available sets an upper bound for the number of processing elements in the hidden layer(s). The equation is,*

$$N_h = size(P) / (s \cdot (N_i + N_o))$$

Where P is the number of cases in the training dataset, N_i is the number of input neurons, N_o is the number of output neurons, s is a scaling factor between five and ten. Larger scaling factors are used for relatively less noisy data.

2.3 Statistical language modelling

The goal of Statistical Language Modelling (SLM) is to build a statistical language model to estimate the distribution of natural language as accurately as possible. A statistical language model is a probability distribution *P(s)* over strings *S* that attempts to reflect how frequently a string *S* occurs. It has been used in many natural language processing applications such as speech recognition, handwriting recognition and data compression. Simply, a probability to predict next symbol can be expressed as,

$$P(\textit{next symbol} \mid \textit{document so far}) \tag{2.3}$$

N-gram, Prediction by Partial Matching (PPM) and unbounded PPM are among those widely used statistical language models. N-gram models are a type of probabilistic model for predicting the next item in a sequence. An *n*-gram is a sub-sequence of *n* items from a given sequence. The items can be phonemes, syllables, letters, words or base pairs according to the application.

For example, a one-gram statistics about relative frequencies of letters in general English plain text [Lewand, 2000] is shown in Figure 2.6. The Letter '*e*' has been used most commonly and then the letter '*t*', while the usage of the letter '*z*' is considerably rare in general English plain text.

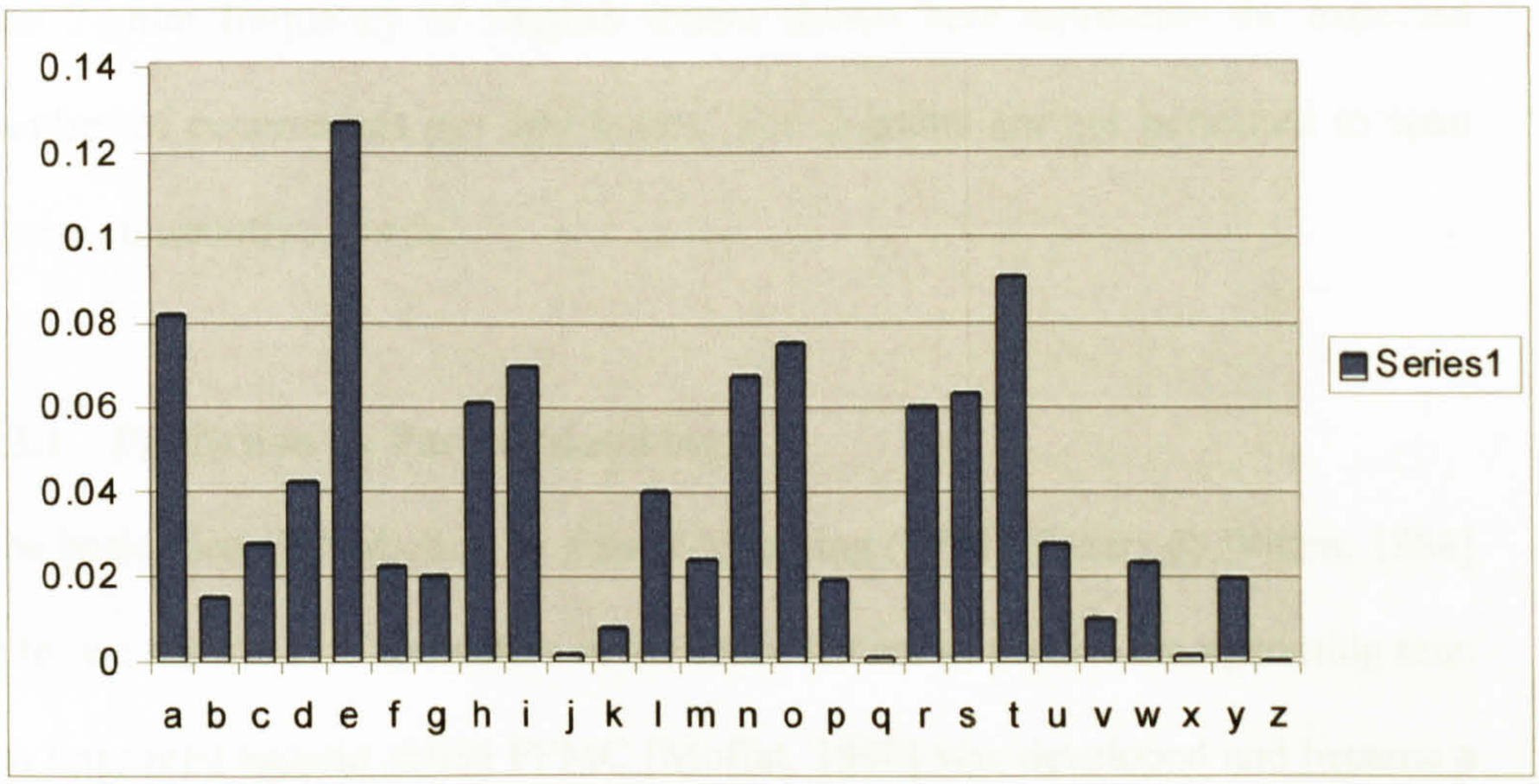


Figure 2.6 *1*-gram relative frequencies of letters in general English plain text

2-gram (i.e., digram or bigram) are groups of two written letters, two syllables, or two words, and are very commonly used as the basis for simple statistical analysis of text. The consecutive letters in one word has a strong relationship with each other. For example, a *2*-gram Frequency in the English language shows that the relationships between '*t*' and '*h*' is much closer than '*v*' and '*e*'. Here is the table of *2*-gram frequency of English letters,

th 50	*at 25*	*st 20*
er 40	*en 25*	*io 18*
on 39	*es 25*	*le 18*
an 38	*of 25*	*is 17*
re 36	*or 25*	*ou 17*
he 33	*nt 24*	*ar 16*
in 31	*ea 22*	*as 16*
ed 30	*ti 22*	*de 16*
nd 30	*to 22*	*rt 16*
ha 26	*it 20*	*ve 16*

The *2*-gram frequency of English letters shown here represents the expected number of occurrences per *200* letters. The *2*-grams are not permitted to span across consecutive words.

2.3.1 Prediction by Partial Matching

The basic idea of Prediction by Partial Matching (PPM) [Cleary & Witten, 1984] is to use the last few characters in the input stream to predict the upcoming one. An improved version called PPMC [Moffat, 1990] was developed and became a benchmark version. If given a sequence $S = \{s_1 ... s_n\}$, and finite-context models of order k (so-called k-gram), where k is the number of preceding symbols used and $0 \leq k \leq n$, Prediction by Partial Matching employs a suite of fixed-order context models with different values of k, ranging from *0* up to maximum n, to predict upcoming characters.

For each model k-gram, prediction probabilities are calculated from all characters occurrences that have followed every length k sub-sequence. Thus a predicted k-gram probability distribution is obtained. The PPM method can be viewed as blending several fixed-order context models together rather than any particular one to predict next symbol. An unbounded length algorithm (i.e. PPM*) [Cleary, 1995] to allow the context length to vary depending on coding situation was developed. The shortest deterministic context which produces one absolute prediction and context trie which is used as search structure are adopted in the design. The unbounded PPM has reliably achieved superior results to PPMC (one of the popular PPM applications). The testing results demonstrated an improvement of about *6%* over the old PPM method [Cleary, 1995].

One further development of language modelling is Factored Language Model (FLM) [Bilmes & Kirchhoff, 2003]. In a factored language model, a word is viewed as a vector of k factors, so that $w_t = \{f_t^1, f_t^2 ... f_t^k\}$ factor can be word classes (e.g. noun, verb) or semantic features as well as words themselves if one considers English Language. An FLM provides the probabilistic model where the prediction of a factor is based on n parents. For example in speech recognition, if w represents a word token and t represents a word class for English, the expression $p(w_i \mid w_{i-2} w_{i-1} t_{i-1})$ gives a model for predicting current word token based on a traditional *2*-gram model as well as the word class of the previous word.

2.3.2 Information Entropy

Shannon entropy [1951] or information entropy is a minimum message length necessary to communicate information. It is a measure of language modelling. Suppose that $\varepsilon = [E_i; i \in I]$ in some finite probability space *S*. The entropy of ε, denoted $H(\varepsilon)$, is

$$H(\varepsilon) = -\sum_{i \in I} P(E_i) \log P(E_i) \tag{2.4}$$

Shannon estimated the entropy of written English to be *1.0* and *1.5* bits per character (*bpc*) or as low as between *0.6* and *1.3* bits per character by having human subjects guess successive characters in a string of text selected at random

from various sources. He proved that if the probability of taking r guesses until the correct letter is guessed is p_r, then the entropy, H (in *bpc*) is

$$\sum_r r(p_r - p_{r+1}) \log_2 r < H < \sum_r p_r \log_2 \frac{1}{p_r} \qquad (2.5)$$

2.4 Data pre-processing

Although many factors affect the success of Machine Learning (ML) on a given task, the representation and quality of the instance data is first and foremost [Pyle, 1999]. Data preprocessing describes any type of processing performed on raw data to prepare for another processing procedure. For example, in a neural network, data preprocessing transforms the data into a format that will be more easily and effectively processed.

There are a number of different tools and methods used for Machine Learning preprocessing. Kotsiantis et al. [2006] have suggested well known algorithms for each step of data pre-processing of Machine Learning to achieve best performance for the training dataset. It consists of six steps,

- *Instance selection and outlier detection.*
- *Missing feature values.*
- *Discretization.*
- *Data normalization.*
- *Feature selection.*
- *Feature construction.*

Using instance selection (i.e. sampling) and outlier detection, irrelevant data as well as noise and/or redundant one can usually be removed. In addition, when a dataset is too huge, it may not be possible to run a Machine Learning algorithm. In this case, instance selection reduces data and enables a Machine Learning algorithm to function and work effectively with huge data.

Real word data tend to be incomplete. Missing data preprocessing function attempts to fill in missing values so as to allow the whole dataset to be processed smoothly. Discretization techniques can be used to reduce the number of values for a given continuous attribute, by dividing the range of the attribute into intervals. Data normalization organizes data for more efficient access. For example, within a feature there is often a large difference between the maximum and minimum values, then normalization can be performed on the value magnitudes to scale to appreciably low values.

Feature selection can pull out specified data that is significant in some particular context. It reduces the dimensionality of the data and may allow learning algorithms to operate faster and more effectively. On the other hand, by constructing new features from the basic feature set, it could provide a better discriminative ability than the best subset of given features and help improve the training data quality.

As examples, several most popular methods used in data transformation and data sampling are presented here in detail.

Data normalization [Han & Kamber, 2001] scale the attribute so as to fall within a small specified range, such as [*-1.0, 1.0*] or [*0.0, 1.0*]. Min-max normalization performs a linear transformation on the original data. Suppose that V_{min} and V_{max}

are the minimum and maximum values of an attribute V. Min-max normalization maps a value v of V to v' in the range $[V'_{\min}, V'_{\max}]$ by computing,

$$v' = (v - V_{\min}) * (V'_{\max} - V'_{\min}) / (V_{\max} - V_{\min}) + V'_{\min} \tag{2.6}$$

Min-max normalization preserves the relationships among the original data values.

Given attribute dataset $V = \{v_i \mid 1 \le i \le n\}$, define $\bar{V}$ and σ_V as the mean and standard deviation respectively of attribute V, then one has

$$\bar{V} = \frac{1}{n} \cdot \sum_{i=1}^{n} v_i \tag{2.7}$$

$$\sigma_V = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (v_i - \bar{V})^2} \tag{2.8}$$

The continuous probability density function of the normal distribution is a Gaussian function. If one simply uses μ and σ to represent the mean and standard deviation of the continuous probability density function, the normal distribution can be expressed as,

$$\varphi_{\mu,\sigma^2}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} = \frac{1}{\sigma} \varphi\left(\frac{x-\mu}{\sigma}\right), x \in \mathbb{R} \tag{2.9}$$

Where, $\varphi(x) = \varphi_{0,1}(x) = \frac{e^{-x^2/2}}{\sqrt{2\pi}}, x \in \mathbb{R}$ is the density function of the "standard" normal distribution (i.e., the normal distribution with $\mu = 0$ and $\sigma = 1$). Then, in z-score normalization (or zero-mean normalization) the values for an attribute V are normalized based on the mean and standard deviation of V. A value v of V is normalized to v' by computing,

$$v' = (v - \bar{V}) / \sigma_V \tag{2.10}$$

Where $\bar{V}$ and σ_V are the mean and standard deviation respectively of attribute V. After normalization, the mean of the transformed set of data points is reduced to zero. This method of normalization is useful when the actual minimum and maximum of attribute V are unknown, or when there are outliers that dominate the min-max normalization.

Normalization by decimal scaling normalizes by moving the decimal point of values of attribute V. The number of decimal points moved depends on the maximum absolute value of V. A value v of V is normalized to v' by computing,

$$v' = v / 10^j \tag{2.11}$$

Where j is the smallest integer such that $\max(|v'|) \leq 1$. For example, given a dataset $V \in [1,100]$, then the maximum absolute value of V is *100* and j=*2*, so that V is normalized to the range [*0.01, 1*].

Sampling can be used as a data reduction technique since it allows a large dataset to be represented by a much smaller random sample (or subset) of the data. The possible sampling methods [Han & Kamber, 2001] for *D* are listed as,

- *Simple Random Sampling Without Replacement* ***(SRSWOR)***
- *Simple Random Sampling With Replacement* ***(SRSWR)***
- *Cluster sampling*
- *Stratified sampling*

Suppose that a large dataset *D* contains *M* clusters, *N* records, and the size of sample is represented by *n*. Then the illustration with examples about those four sampling methods can be shown in Figure 2.7, where $D = \{T1, T2, \ldots, T20\}$, $N = 20$, $M = 4$.

SRSWOR samples are created by randomly picking data from source D without repetitive records. From Figure 2.7, a subset $n = 4$ records *{T12, T20, T3, T9}* are created; SRSWR samples are created by allowing repeated samples, for example in the dataset *{T5, T17, T5, T2}* created in the figure, the record T5 is drawn twice.

Cluster sampling randomly chooses samples. Figure 2.7 shows two cluster samples – *{T6→T10, T11→T15}* which are randomly picked up, where the whole number of cluster $m = 4$. If a certain number of records are selected from each cluster, then this sampling process is called stratified sampling. In Figure 2.7, stratified sample set *{T2, T9, T13, T17}* is created by selecting one record from each cluster.

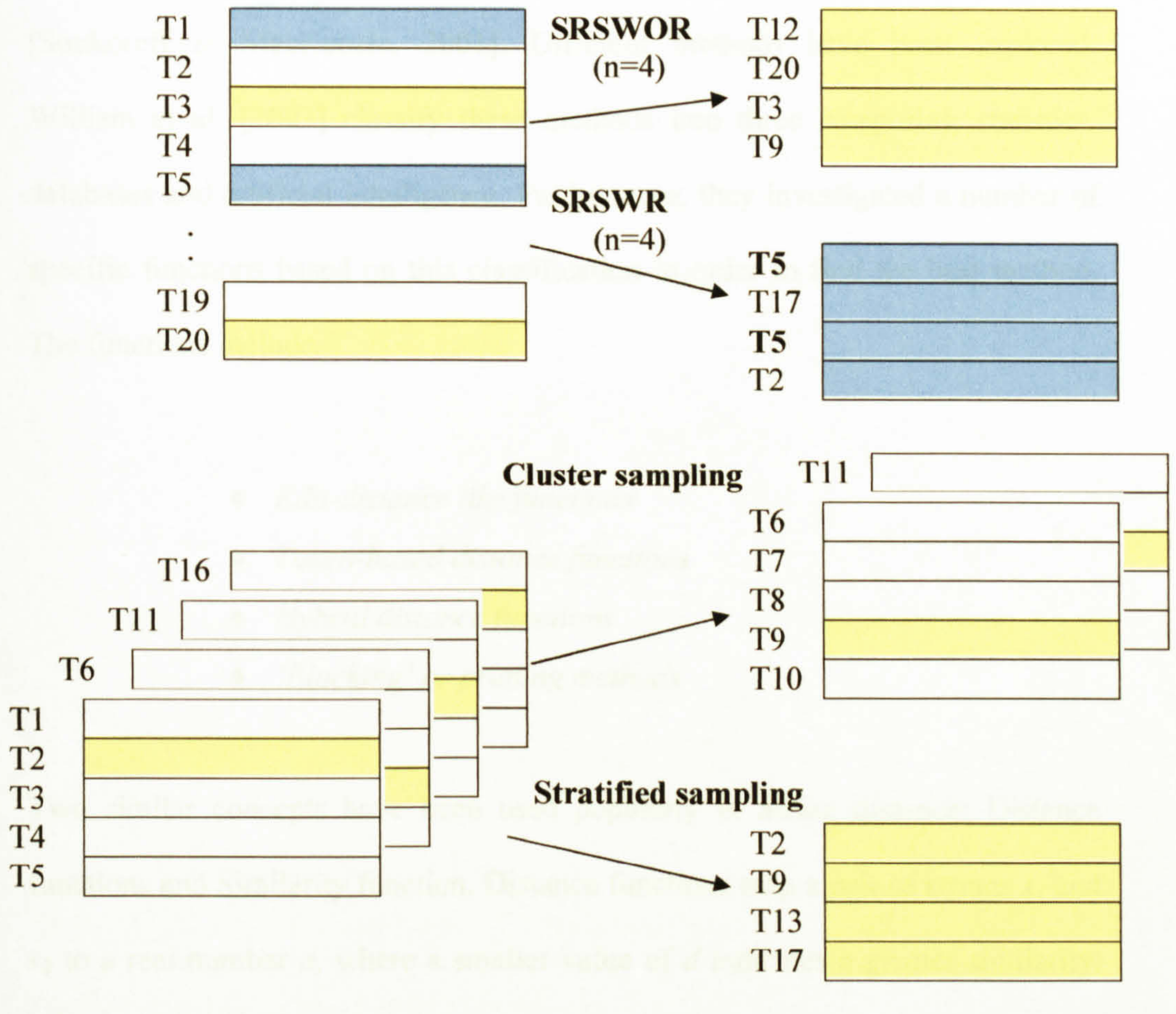


Figure 2.7 An example of preprocessing sampling methods [Han & Kamber, 2001]

When applied to data reduction, sampling is one of the most commonly used methods. It is possible using the central limit theorem to determine a sufficient sample size for estimating a given function within a specified degree of error [Han & Kamber, 2001].

2.5 String distance

String distance is a method to measure the amount of difference between two sequences, which emerged from theoretical work on self-correcting binary codes

[Soukoreff & MacKenzie, 2001]. Different methods have been explored. William et al. [2003] classify these methods into three categories: statistics, databases and artificial intelligence. Furthermore, they investigated a number of specific functions based on this classification in order to find the best method. The functions include,

- *Edit-distance like functions*
- *Token-based distance functions*
- *Hybrid distance functions*
- *'Blocking' or pruning methods*

Two similar concepts have been used popularly in String distance: Distance functions and Similarity function. Distance functions map a pair of strings s_1 and s_2 to a real number d, where a smaller value of d indicates a greater similarity, while similarity functions is on the contrary, a larger value indicates the greater similarity.

Levenshtein distance [Levenshtein, 1965] is a popular figure in Edit-distance like functions. The Levenshtein distance between two strings is given by the minimum number of operations needed to transform one string into another, where an operation is an insertion, deletion, or substitution of a single character. For example,

hellu → *hello* *(d = 1)* *// a substitution of 'o' for 'u'*
helloo → *hello* *(d = 1)* *// delete 'o'*
hat → *hello* *(d = 6)* *// delete 'at' and insert 'ello'*

Another similar method is Jaro metric [Jaro, 1995]. The Jaro distance metric states that given two strings s_1 and s_2,

1) Two characters a_i, b_j from s_1 and s_2 respectively, are considered matching only if,

$$i - \frac{\min(|s_1|,|s_2|)}{2} \le j \le i + \frac{\min(|s_1|,|s_2|)}{2} \tag{2.12}$$

then their distance d is calculated as,

2) $$d = \frac{1}{3}\left(\frac{|s_1^{'}|}{|s_1|} + \frac{|s_2^{'}|}{|s_2|} + \frac{|s_1^{'}| - t}{|s_1^{'}|}\right) \tag{2.13}$$

Where $|s_1^{'}|,|s_2^{'}|$ are the numbers of s_1 matching s_2 and s_2 matching s_1 characters respectively, t is the number of transpositions.

A variant of Jaro metric [Winkler, 1999] uses a prefix scale p, which is the longest common prefix of string s_1 and s_2. Let's define Jaro distance as d, then Jaro-Winkler distance can be defined as,

$$Jaro - Winkler(d + \frac{\max(p,4)}{10} * (1-d)) \tag{2.14}$$

A piece of open source code has been published by US Census Bureau. The result of the Jaro-Winkler distance metric is normalized into the range [*0, 1*]. It is designed and best suited for short strings.

2.6 Fitt's law

Fitts' law is a robust model of human psychomotor behaviour [Paul Fitts, 1954]. The model is based on time and distance. It enables the prediction of human movement and human motion based on rapid, aimed movement, not drawing or writing [Dov Te'eni et al., 2007].

It is discovered that movement time was a logarithmic function of distance when target size was held constant and that movement time was also a logarithmic function of target size when distance was held constant. Mathematically, Fitts' law can be stated as follows [MacKenzie & Buxton, 1992],

$$MT = a + b\log_2\left(D/W + 1\right) \tag{2.15}$$

Where MT represents the movement time, a and b represent the regression coefficients, D represents the distance of movement from start to target center, W represents the width of the target. If one takes a computer keyboard as an example, the equation was rewritten by Zhai et al. [2000] as,

$$MT = a + b\log_2\left(D_{ij}/W_j + 1\right) \tag{2.16}$$

Where D_{ij} is the distance moving from key i to key j, W_j is the width of key j. The equation is illustrated by the keyboard logical expression as shown in Figure 2.8.

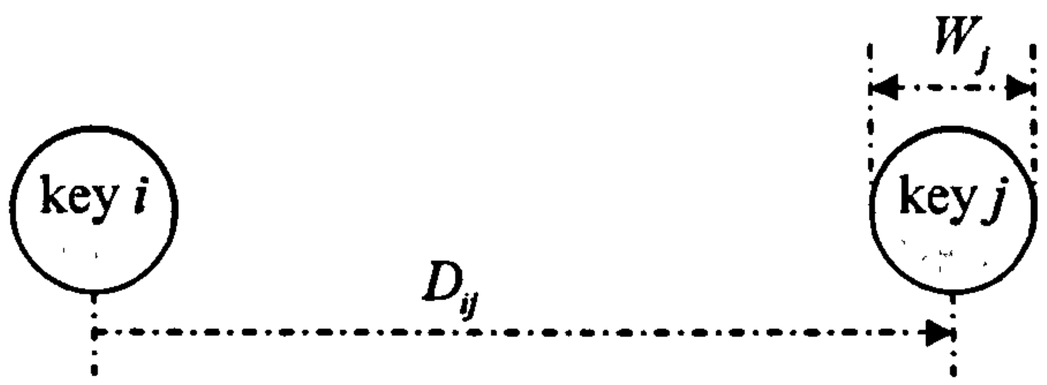


Figure 2.8 Fitt's law expression based on keyboard [Zhai et al. 2000]

Fitts' law has been used popularly to predict the performance of humans when using various input or output devices.

2.7 Text prediction and correction applications

Predictive text/word input technologies are some of the techniques that are often found useful by text entry users. There are many products available on the market to offer this support. Compared to pure prediction products, some efforts have also been made to reduce typing mistakes, although far few tools can intelligently identify new genre of mistakes. Here a full review based on both categories is given.

2.7.1 Text prediction applications

Using data compression methods such as PPM as language model to predict next letter, word or text are very popular technologies in prediction field. But it has been argued that the information available in the local context of each word based on statistics should be augmented by global sentence information. Jianhua Li & Graeme Hirst [2005] proposed a combination model by integrating

semantic knowledge with *n*-gram probabilities to predict semantically more appropriate words.

First, a semantic knowledge base was generated for English words, especially for nouns, by exploring the semantic relatives. Then, the semantic knowledge is used to measure the semantic association of completion candidates with the context. Those that are semantically appropriate to the context are promoted to the top positions in prediction lists due to their high association with context.

Figure 2.9 shows the combination model [Jianhua & Graeme, 2005].

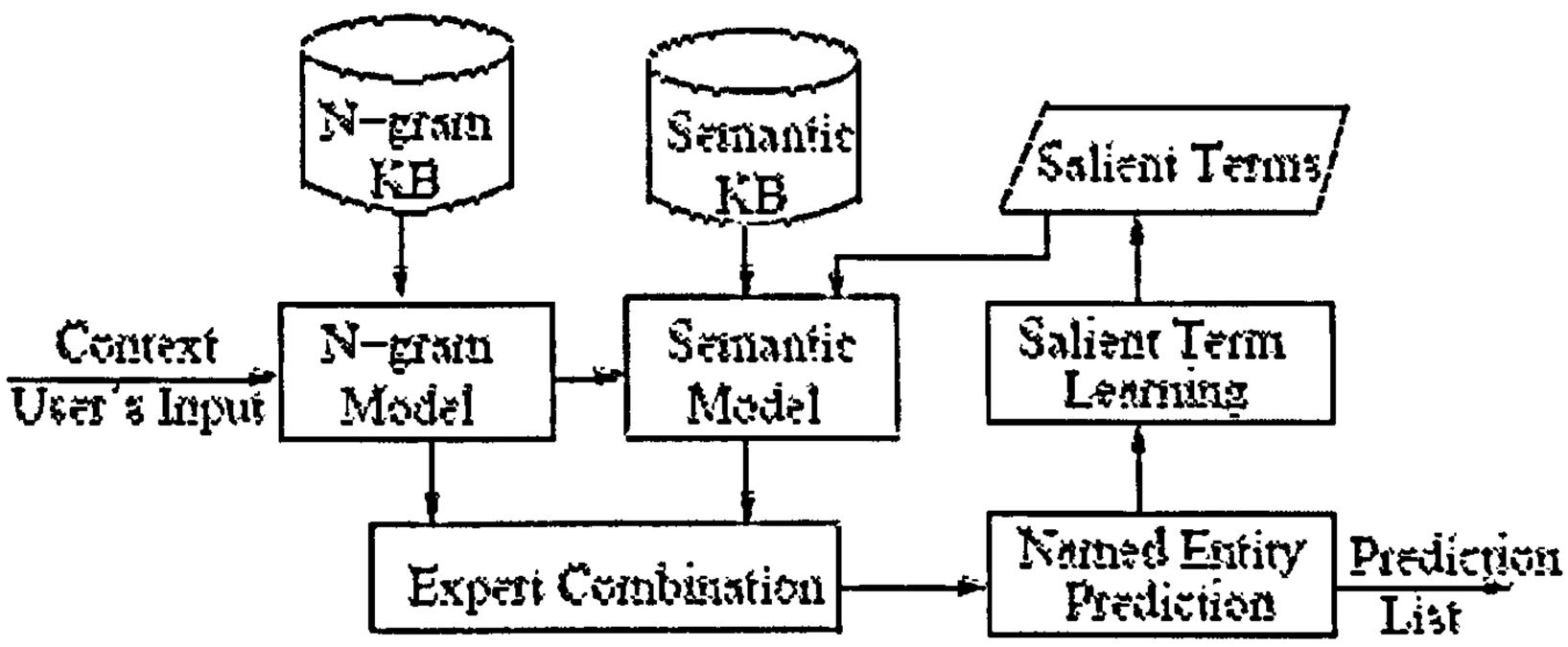


Figure 2.9 Jianhua & Graeme word prediction model [2005]

This model is a combination between *n*-gram model and semantic model. Semantically related words and their pointwise mutual information (PMI) are extracted from the British National Corpus World Edition (BNC). An algorithm that automatically determines the salient terms of a text during the prediction process was proposed to measure semantic association for a candidate whenever the candidates find no related words in the context.

By comparing the combination model with Fazly and Hirst's syntax-and-n-gram model [2003], where part-of-speech *n*-gram information was added to traditional *n*-gram model, the keystroke saving rate of the combination model increases to 6%.

Masaui & Nakayama [1994] proposed a simple and powerful predictive interface technique by making use of *dynamic macro*. When a user types a special "repeat" key after performing repetitive operations in a text editor, an editing sequence corresponding to one iteration is detected, defined as a macro, and executed at the same time. Although being simple, a wide range of repetitive tasks can be performed just by typing the repeat key. When another special "predict" key for conventional prediction techniques is used in addition to the repeat key, wider range of prediction schemes can be performed depending on the order of using these two keys. The complex rules for combination of two keys are given in Figure 2.10.

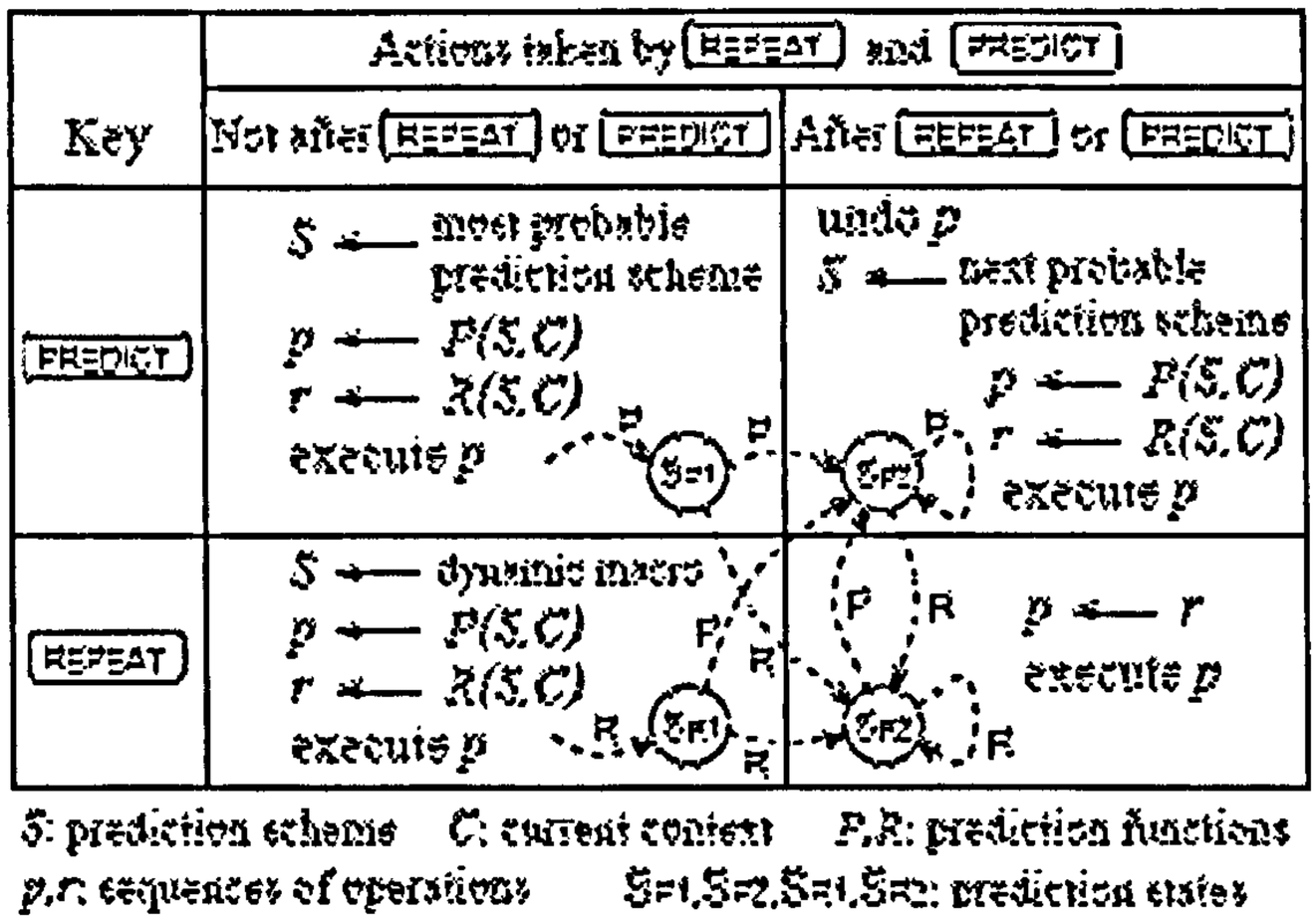


Figure 2.10 Repeat and Predict key composition rules [Masaui & Nakayama, 1994]

It combines editor and prediction function together by using two additional keys: Repeat and Prediction key. The emphasis of design focuses mainly on the interface design rather than prediction function itself. Although Repeat key provides words based on a simple typing history, the memory would vanish after the editor is closed.

An early work for text prediction is presented by Toshiyuki Masui [1999], called POBox, which consists of two steps for entering a word or a phrase. First, a user enters a small part of the word or some other attributes, and POBox dynamically searches a dictionary for candidate words and shows them to the user for selection. The user then selects the desired word from the candidate list, and POBox enters the word into the user's document.

POBox gives a simple and small-sized solution and has been developed for different languages. It can use the context of the user's document to help identify likely candidates. POBox and two-key technology are different in principle. The former gives a word of list for user's choice based on spelling, pronunciation or shape of characters. It is a useful tool for handheld or ubiquitous computers such as PDA or mobile phones.

Another application worth mentioning is T9, which stands for Text on *9* keys, a patented predictive text technology for mobile phones, developed by Alex Robinson [1998]. It combines the groups of letters on each phone key with a fast-access dictionary of words. It looks up all words in the dictionary which correspond to the sequence of key presses.

As it gains familiarity with the words and phrases the user commonly uses, it speeds up the process by offering the most frequently used words first and then

lets the user access other choices with one or more presses of a predefined Next Key. The dictionary can be expanded by adding missing words, enabling them to be recognized in the future. Its interface workflow is illustrated below.

- *Enter a word by tapping one key per letter.*
- *Display the word with first priority.*
- *Use Next key to choose or type further letter if the word is not the right one*
- *Add it to dictionary if no right word in the list*

A series of research based on words vocabulary which apply both neural network and language modelling methods are also worth being mentioned here. One of the major representatives is Bengio and Ducharme [2001] with their neural probabilistic language modeling. They suggest a model to learn a distributed representation for words that allows each training sentence to inform the model about an exponential number of semantically neighboring sentences. Schwenk and Gauvain [2002] further address a related problem that the *n*-gram space is highly sparse, by carrying out a probability estimation in a continuous space and enabling a smooth interpolation of the probabilities. However, due to the curse of dimensionality in the discrete space representation, they still have to narrow the vocabulary by using a shortlist which damages the prediction accuracy and fail to learn a long-span LM with $n >> 3$ gram, not to mention a broader word space with noises which may increase in a geometric degree.

The research illustrated above describes different input modelling techniques in an effort to develop efficient and high performance text input techniques. They

are largely developed based on statistical language models, natural language process, interface-oriented or combinations of these. Some of them are not specially targeted for QWERTY keyboard user. However the software can be tailored or refined to suit the audience.

2.7.2 Typing Correction Applications

There are many types of errors caused by users, roughly such as spelling errors, hitting adjacent key and cognitive difficulties. Prediction technology can foresee users' typing intention, but can't directly correct typing mistakes. Some efforts have been made to reduce these mistakes, although far few tools can intelligently identify new type of mistakes.

One way to improve accuracy is to install filters which modify the control signals generated by the device. Such filters can have a significant effect on the speed and accuracy with which a device can be used. For example, motor disabilities may have difficulties with a number of aspects of keyboard use, resulting in high error rates, fatigue and slow typing [Shari, 2003].

Most operating systems provide a suite of accessibility settings which can be used to configure the keyboard. Popular examples are MS Windows and the Apple's accessibility features. Some of these features directly address the problems with the keyboard usage. For example, the keyboard accessibility features available under Windows are,

- *Key Repeat Delay*
- *Key Repeat Rate*
- *Stick Keys*

- *De-bounce Time*
- *Key Acceptance Delay*
- *Key Guard (it can be either physical or a piece of software)*

These features address the problem to some extent, and it is generally seen that people tend to use them. However, the problem lies with their complexity since not all users are good enough to set them at a desired level. Their precise setting is a time-consuming and error-prone task, and generally takes some time before they are configured properly.

Attempts have also been made for example by IBM to devise intelligent mechanisms that could adjust the settings of the keyboard accessibility features by detecting the usage problems [Shari, 1998 & 2003]. They present a user model to examine the behaviour of a real computer user. The model encompasses four aspects of keyboard use which can present difficulties for people with motor disabilities, and bases its recommendations to configure the Accessibility Option of Windows on observation of users typing free English text.

Initial feedback from *978* active users indicates that the key repeat delay adjustment is acceptable to users, but the key repeat rate adjustment algorithm needs more work, and the debounce feature may not be appropriate for dynamic adjustment.

However, all they can control is only the accessibility features available under the belt of operating systems. Thus, they can only be viewed as a slightly intelligent layer over the set of accessibility features.

Another way is to use a spell checker to suggest corrections for wrongly spelled words. Customarily a spell checker consists of two parts: a set of routines for scanning text and extracting words, and a wordlist. Research aimed at correcting words in a text has focused on three problems [Karen, 1992] as follows,

- *Non word error detection*
- *Isolated-word error correction*
- *Context-dependent word correction.*

In response to non word error detection, it was suggested that in large samples of common English publication text, 42% of all digram combinations are unlikely to occur.

An alternative method called Dictionary Lookup, is a straightforward method to detect non-word, but response time and memory etc. become problems as vocabulary increases. So tailoring the vocabulary to set up a user-oriented dictionary is a good solution.

Computer-user interaction is also being considered, for example by capturing and analysing user's typing actions. That is one of the motivation of KeyCapture [Soukoreff & MacKenzie, 2003] development. But unfortunately, William Soukoreff & MacKenzie put more attention on language models creation, and then ignored the importance of feedback from users' word correction.

Other research such as Natural Language Processing modelling are also recommended. NLP tries to simulate specific language and generate a text by analysing specific language's words, syntax and semantics through symbolic,

statistical, connectionist approaches. For example, it can detect a misspelled word based on the context of the surrounding words.

Spell checker applications are either stand-alone applications capable of operating on a block of text, or as a feature of a larger application, such as a word processor, email client, or search engine. It also provides another function offering to users the right to add abbreviation or habitual mistyping along with its corresponding right pair into a database. Then the repetitive typing mistakes can be identified.

2.7.3 Dasher

Dasher [Ward & MacKay et al, 1997-2008] is an information-efficient text-entry interface, driven by natural continuous pointing gestures, designed by Inference Group of Cambridge University. Dasher has a zooming interface. It is based on language model prediction, through which the space of interface is determined to each piece of text. Compared to improbable pieces of text, probable pieces of text are given more space, so they are quick and easy to select. The language model of Dasher could learn all the time along with user's typing.

Figure 2.11 is quoted from Dasher's demonstration web page. It shows the state of the Dasher interface while the user is writing the word `objection'; alternative words that could easily be written at this point include `objective', `objects_', and `object_oriented'.

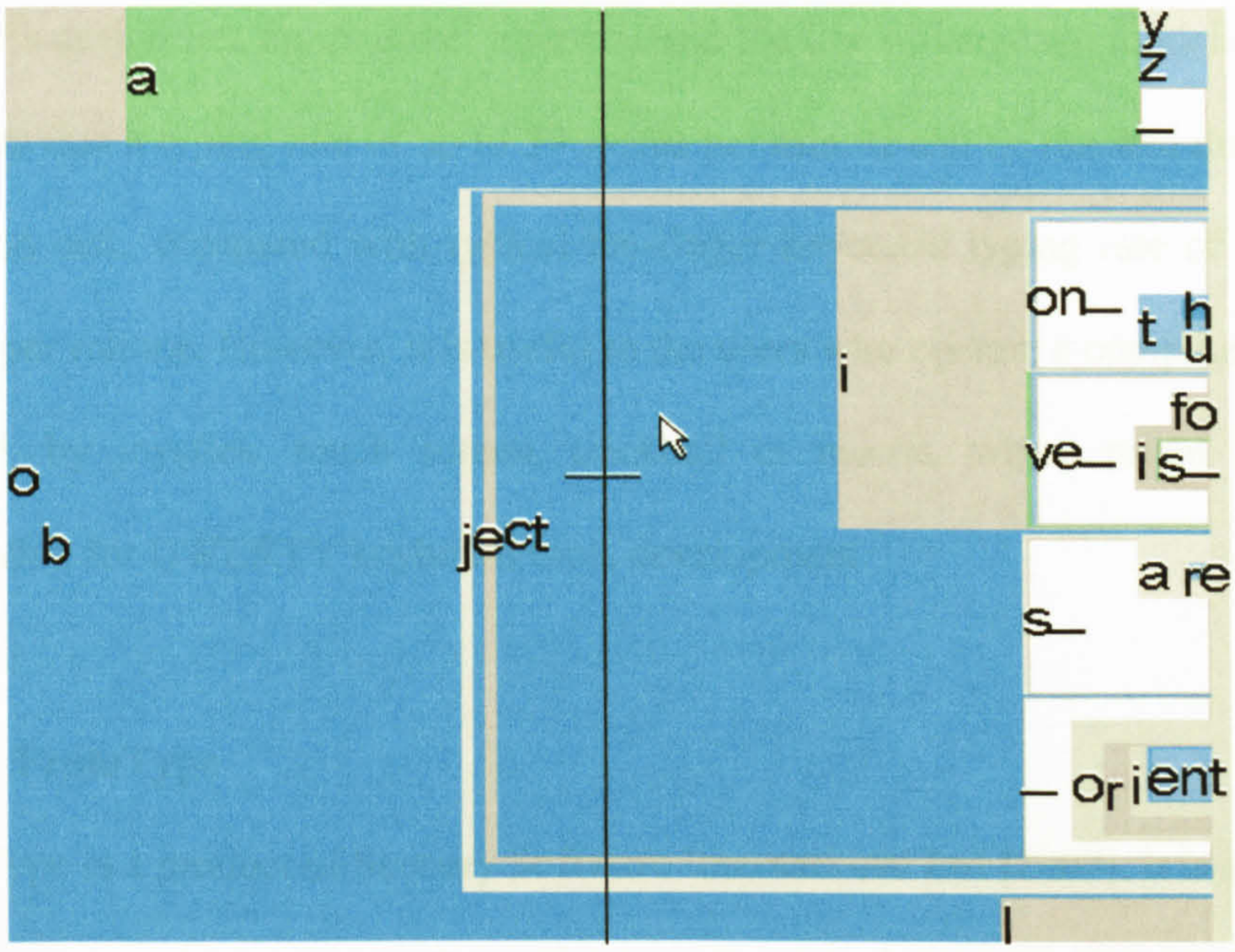


Figure 2.11 A Dasher interface example [Ward & MacKay et al, 1997-2008]

The users only need to move their cursors toward the right choice. Then the interface zooms in, the place under the cursor passes through the central cross and then the choice is made.

Dasher's screen layout is determined by probabilistic model and driven by continuously two-dimensional gestures: horizontal and vertical. Its division of right-hand is analogous to arithmetic coding. The real line [0, 1] is divided into I intervals of lengths equal to $P(x_i = a_i)$, where a_i stands for an alphabet [David et al., 2000].

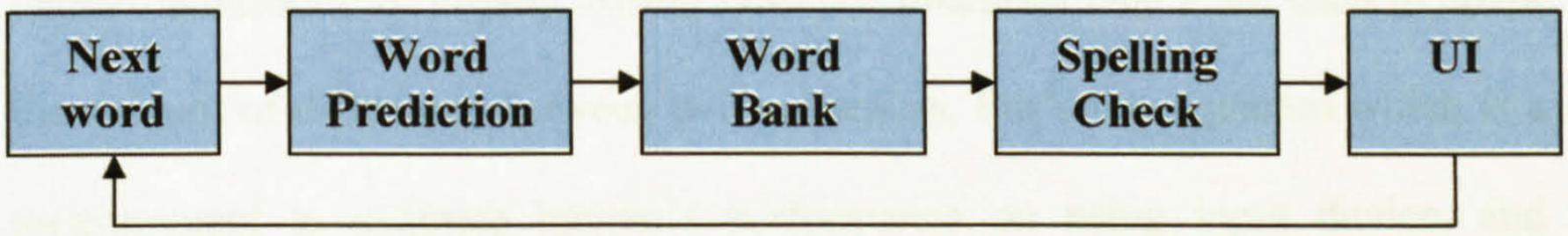
Dasher uses an algorithm called PPM5D+ to predict words, which can compress most English text to around *2* bits per character. By renormalizing the probability, the same algorithm is used to determine the intervals.

Dasher's design lies on its novel interface and the use of language modelling. It suggests that a typing rate of up to *39* words per minute can be reached during a dictation task, compared with typical ten-finger keyboard typing rate of *40-60* words per minute. However, it is useful to the users who operate a computer one-handed, by joystick, touch screen, trackball or mouse, which might be an inspiration for QWERTY keyboard tools development.

2.7.4 ProtoType

ProtoType is a product of Sensory Software International Ltd [2007]. It is a piece of software used to type text into other programs such as a word processor. It uses lists of words to help a user to type, which includes word prediction, spelling correction and word banks.

As a user is typing, some of the automated features can highlight spelling errors, capitalise words, replace common spelling mistakes or expand abbreviations. Its architecture is shown in Fig 2.12.



Figurer 2.12 ProtoType program flow diagram [Anon, 2006]

User's each touch would firstly trigger word predication list, and then Word Bank and Spelling Check function will be called in sequence. ProtoType is designed to improve spelling for people with dyslexia or spelling difficulties, but

it is unfunctional to correct the keystrokes mistakes made by most motor disabled people.

Apart from those technologies, alternatives such as hardware are also developed. For instance, Keyguard is a plate that fits over a keyboard, which users can rest their hand on and make key press. It is useful for motor disabled people to avoid typing mistakes. Other hardware such as BigKeys, Intellikeys Keyboard and VisiKey Multimedia keyboard are also on the market.

2.8 Summary

First, Neural Network models which include Focused Time-Delay Neural Network, Elman Network, and Probabilistic Neural Network, and Statistical Language Modelling which includes language Modelling rationale, Prediction by Partial Matching algorithm and Entropy concept are described. Then, a unique data preprocessing process mainly based on Kotsiantis' suggestion – the six steps of data preprocessing to achieve best performance for neural networks' training dataset is introduced. Finally, String Distance functions which are used to count the amount of difference between two sequences, Fitt's law equation which is a measurement to estimate human's performance on using input device, and several word prediction and correction applications such as Dasher and Prototype are reviewed.

The Focused Time-Delay Neural Network was well applied to motion recognition used by surveillance system, multimodal human computer interface and traffic control system etc. Elman network was used to recognize both spatial

and temporal patterns such as word category classification and human activity recognition. Probabilistic Neural Network was used by a computer-based face detection system as a core classifier. However, they are hardly seen to apply to noisy text entry processing such as user typing stream and its extracted sub-dataset, which implies all users' self-rectification actions, user's vocabulary, typing habits and typing performance. And also, although efforts have been made in multiple directions such as language modelling, natural language processing and user interface design, those technologies, if used alone, will fail to meet the user's particular needs. It is also worth arguing that those designs reviewed in the chapter (e.g. Figure 2.9) emphasize excessively on providing a global method, and lack 'user-oriented' feature. Furthermore they are short of self-adaptive ability (i.e. learning ability), and fail to fully recognize the right patterns from user's distinct performance. Finally the most impotence of current research is that it has neglected the significance of negative influence incurred by the text entry noises, which have badly affected the accessibility and usability in human computer interaction, and a systematic solution as a bridge between user and computer to filter noises and make text entry more effectively has never been on the agenda.

This research is motivated by these arguments and user requirements to find an answer to those existing weaknesses and fundamental gaps. A comprehensive solution to develop a user oriented hybrid framework with self-adaptive ability is required. It would provide a combination of models with multiple features such as prediction and correction functions based on a neural network language modelling research. These will be fully discussed in following chapters.

CHAPTER THREE

A NOVEL FRAMEWORK FOR NOISY LANGUAGE ANALYSIS

3.1 Introduction

Computer text entry is full of noises. For example, computer keyboard users inevitably make typing mistakes and their typing stream implies all users' self-rectification actions. These have produced a dramatically negative influence on the accessibility and usability of applications that need text entry from input devices. But the issue hasn't been addressed well. Therefore, a fundamental concept to develop an intermediate layer language modelling framework to analyze the language stream data with noises is required. The intermediate layer lies between computer input devices layer and applications layer. It requires the framework to be capable of reducing input errors significantly as well as increasing the input efficiency highly. This framework can be seen not only as a noisy language filter between input and output, but also as a bridge between a user and a computer, or among input devices. To illustrate the framework, in this chapter a specific case is also studied aiming to develop a user oriented hybrid system with self-adaptive function to help disabled people to use QWERTY keyboard more effectively.

3.2 A novel intermediate layer language framework

Via computer, an individual interacts with applications by producing events, which are triggered by appropriate input devices and then transformed into values that are expected by the target system. Typical input devices include keyboard, mouse, camera and so on.

Text input is one of the purposes of some typical input devices. For example, one of the facial recognition functions is to allow computer to interpret the speaker's speech and spot the text they intend to express along with their facial expression changes. As another example indicates, although for computer keyboard it is much easier to generate a text from symbols to phrases and then sentences, the typing stream shows a complicated and user related input process, where it is full of rectification, repetitive keystroke mistakes and almost hardly identified. These problems are reflected due to the flaw of input devices as well as human inevitable mistakes when interacting with a computer.

These prove language based text input is a fundamental function in human and user interaction, which imply the requirement to design a user friendly framework which is a communication interface acting as a mapping layer between input devices and applications. As a result of the design under a specific input device, it will become easier (or if necessary) to carry out further personalization with respect to adjusting the mapping between the input device and applications. It may also provide a platform for the cooperation between two input devices such as bimodal speech recognition's recorder and a camera.

In this research an intermediate layer framework called Adaptive Language Modelling Intermediate Layer (ALMIL) with two fundamental language modelling functions, namely, text prediction and text correction functions is presented. As a text input processing platform, ALMIL is transparent for both lower layer such as input devices (including related system drivers) and higher layer such as applications.

In the framework, the values produced by the input devices as a result of user interaction with computer, are represented by vectors. Each vector is composed of attributes, which include time, predecessors and IDs. For example, the word identified by speech recognition can be represented by its predecessors, genre (speech recognition) and probability of the word, which are then converted into a quantified vector.

ALMIL is mostly designed as a language model dealing with noisy input stream triggered from different sources. It makes recommendation both for wrong and uncompleted input. It combines with several technologies which include *n*-gram statistics, neural networks and human computer interaction technology, and then designed in units as shown in Figure 3.1.

The intermediate layer framework includes eight working units, namely, Vector Generation unit, Pre-processing unit, Prediction unit, Correction unit, Short-term Memory, Long-term Memory, Inference Engine unit and Interaction Port. Vector Generation unit, Pre-processing unit and Interaction Port deal with the interaction between input and output, while the rest of units compose the data processing centre and information storage centre.

The input of ALMIL can be a single symbol such as an alphabet, a logical set such as a word, or a context set such as a phrase or a sentence. The input stream also represents an input process evolving from a symbol to a complete word, then a sentence.

As shown in the dash circle of Figure 3.1, the logical set is composed of consecutive symbols while the context set is made up of consecutive logical sets.

Let's take computer QWERTY keyboard as an input device and English

language as a user's typing language. Then the symbol set is a subset of ASCII set. The English words are referred to as logical sets and the context is represented by English phrases and sentences.

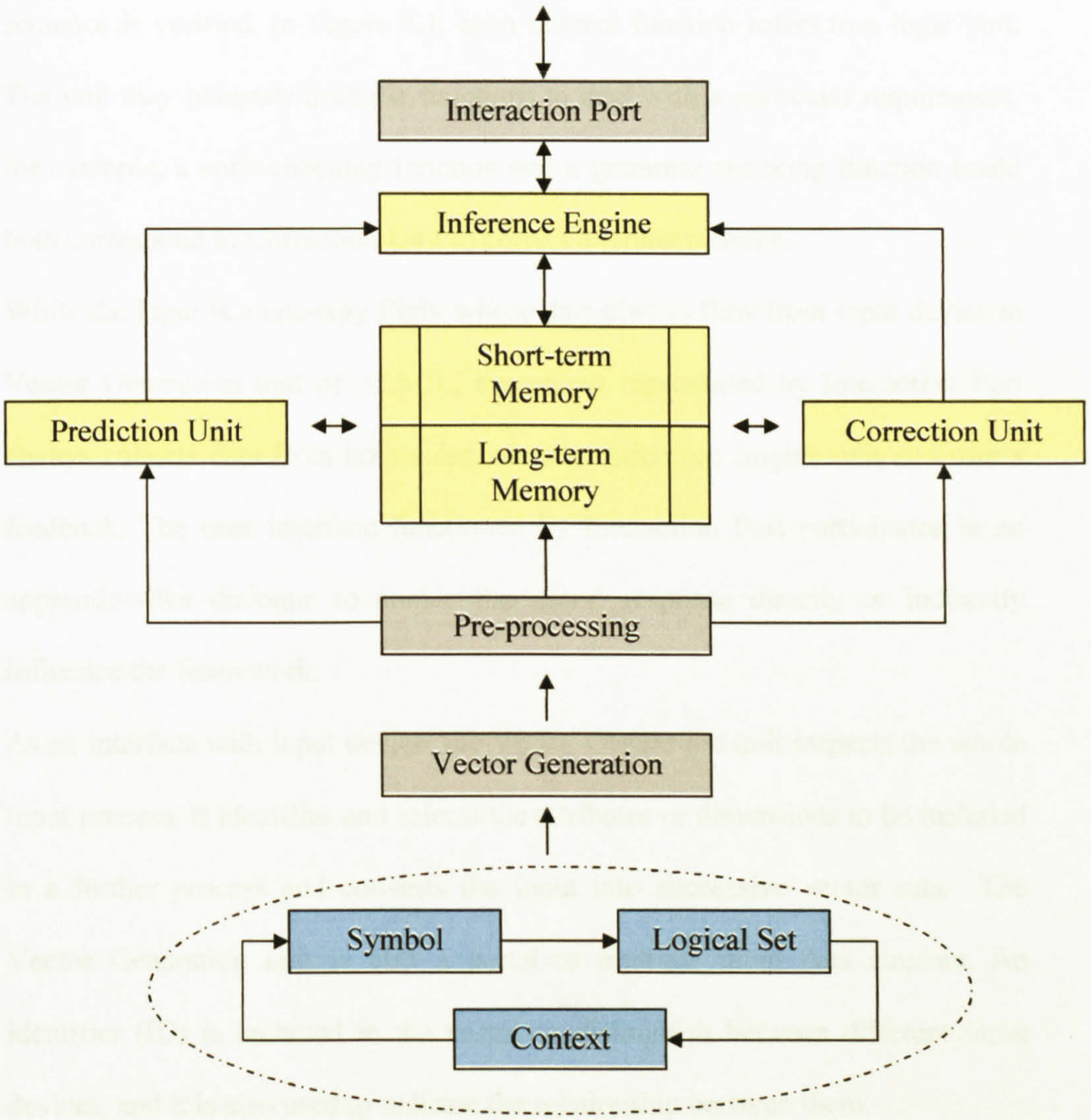


Figure 3.1 Adaptive Language Modelling Intermediate Layer Framework (ALMIL)

The different compositions trigger distinct logical process. For example, let's imagine a speech recognition process: an English user speaks word by word,

which can be considered as a logical set while phonemes are considered as symbols. Then each identified word triggers a process to verify correctness of the word. As soon as a sentence is composed, a distinct function (e.g. grammar checker) or several functions referred to a sentence is triggered and the whole sentence is verified. In Figure 3.1, each distinct function refers to a logic unit. The unit may integrate multiple functions to deal with a particular requirement, for example, a spell checking function and a grammar checking function could both correspond to Correction Unit to correct a typing mistake.

While the input is a one-way flight whose data always flow from input device to Vector Generation unit of ALMIL, the output represented by Interaction Port always collects data from both sides, namely, Inference Engine unit and user's feedback. The user interface functioned by Interaction Port participates in an apprentice-like dialogue to enable the users' response directly or indirectly influence the framework.

As an interface with input device, the Vector Generation unit inspects the whole input process. It identifies and selects the attributes or dimensions to be included in a further process and converts the input into successive vector sets. The Vector Generation unit is also a portal to multiple input data streams. An identifier (ID) is included in the vector to distinguish between different input devices, and it is also used to indicate the relationship between them.

The Pre-processing Unit handles the noisy, missing and inconsistent data based on the generated vector sets. The handling methods are derived from data mining preprocessing which includes data cleaning, data integration, data normalization and data reduction. This unit is set to be configurable both based on manual and

automatic work. Some user's distinct typing characters can be configured by setting up a related user profile.

The memory is designed based on the MHP architecture – Model Human Processor architecture, which was suggested as one of the human computer interaction theory to human cognition. As a simulation to human mind, the memory of ALMIL is divided into short-term memory and long-term memory. The long-term memory includes a knowledge base which is represented as a set of rules and facts which are used to match the rules. Each rule specifies a relation, strategy or heuristic with a certain structure such as IF…THEN structure.

The problem-specific information such as current input and recently used but not frequently used words which don't exist in long-term memory, are stored into the short-term memory. The frequently used terms such as words, phrases and production rules are stored into long-term memory. All other units are able to communicate with long-term memory and short-term memory directly.

The ALMIL is designed to provide users with two major functions, namely, prediction function and correction function. For example, let's consider that ALMIL is processing a user's typing stream generated from a computer keyboard. The user's typing stream includes all self-corrections of the user, the user's vocabulary, typing habits and typing performance. Based on the analysis of Inference Engine to the user's typing history and online feedback, the prediction unit is able to predict user's typing intention, while the correction unit is able to correct user's real-time typing mistakes.

The central unit of this framework is the Inference Engine. Its learning comes mostly from users' real-time feedback. Each feedback is converted into rewards

assigned to specific facts, weights, rules saved in the database. The rewards are partly generated based on specific intelligent learning algorithms. Their values are adjusted as soon as the Inference Engine receives feedback from the Interaction Port. The Inference Engine is also designed to learn from the offline analysis of historical data stream derived from input devices, and deduce the association rules. Then the rules are kept in the database and used to improve the functionalities of the prediction and correction units.

More specifically, an appropriate learning oriented technology such as neural network learning algorithm, generic algorithm or a hybrid algorithm can be applied to the inference engine. The design of hybrid inference engine can be based on specific requirements. It is also required to refer to the features of that particular technology. For example probabilistic reasoning mainly deals with uncertainty, while fuzzy logic and evolutionary computation mainly deal with imprecision and optimization respectively.

ALMIL has some substantial differences from the other models such as Jianhua model. First, it is designed as a generalized intermediate layer language modelling framework that lies between computer input devices and applications layer to analyze the noisy language stream such as typing or bimodal input stream, whereas Jianhua model is developed as a word prediction application based on a combination of n-gram model and semantic model with a simple ranking strategy. ALMIL as an input behaviour analyzer deals with the original data stream that are directly taken from input devices with full of noises rather than that Jianhua model does a half-processed structured data. ALMIL is a hybrid system that introduces a variety of state-of-the-art technologies such as

neural network, natural language processing and data mining, with high self-learning requirement, while Jianhua model is a two models combination only with a limited learning which is based on the pointwise mutual information. Moreover ALMIL consider HCI as its part of architecture which emphasizes user-oriented, short-memory and long memory, and user interface design, while Jianhua model focuses on the word prediction functionality only.

In order to illustrate the process of ALMIL, Bimodal Speech Recognition data stream is given as an example here. As human mouth cavity is part of vocal tract, lips and tongue, teeth mouth jaw and chin play a very important role on speech generation. Then bimodal Speech Recognition (visual and audio) was suggested to defy the noisy environment where the performance of audio speech recognition degrades drastically. It identifies a user's speech by analysing both their facial and voice phonemes. Given the ALMIL framework, the data stream marked with time stamps and IDs is transformed to distinct vectors within Vector Generation module, where the IDs are distinguished based on visual and audio phonemes. Then the preprocessing unit synchronizes the vectors based on time stamps and IDs, and tackles noise problems such as filling the missing values and removing the outliers.

The Correction Unit makes a quick decision on where correction is required. Correction solutions are generated when needed, and a cross-correction function is applied for the bimodal recognition if one considers this specific case. The Prediction Unit can process data stream simultaneously. Then both results are presented to the Inference Engine unit. Based on neural network or other inference algorithms, a comparison between Facial Expression Recognition

results and Speech Recognition results is carried out. Eventually, the results marked with ranking probabilities or a direct highest ranking presentation is presented to the user. In turn, the user's feedback is used to improve the inference algorithms performance.

The next two sections are a further demonstration of ALMIL. Firstly a comprehensive disabled user investigation is carried out, and then an Intelligent Keyboard directed towards analyzing disabled users' typing stream is presented.

3.3 Disabled keyboard users investigation

Computer users with motor disabilities or cognitive problems may have difficulties in accurately manipulating the QWERTY keyboard. The aim of this investigation is to offer researchers an opportunity to closely observe this group of users' typing behaviour. During the investigation, about twenty seven people have been interviewed. Both old and disabled people are involved. Their performance can be classified as below.

- **'Unfamiliar with computer' performance**

Difficult to find keys: *Especially function and punctuation keys (e.g. 'F12', '?', '/', '+').*

'Enter' key puzzle: *Some computers are with no "enter" or "shift" printed on the key surface, so it is difficult for old people to find where they are.*

Compound keys puzzle: *Due to different definitions in distinct software, compound keys cause confusion to many people.*

- **'Motor disability' performance** [Trewin & Pain, 1998]

Long key press: *These occur when an alphanumeric key is unintentionally pressed for longer than the default key repeat delay.*

Modifier keys: *For example, 'Shift'+ 'a'. One-hand typists in particular may find it difficult to press two keys at once.*

Additional keys: *Some users often press keys adjacent to the target keys.*

Bounce errors: *These occur when the user unintentionally presses a key more than once.*

Easily tired: *It is a hard task for some disabled people to type more than certain number of words in succession*

Prefer big keys: *Some users can't cope with laptop well because of the smaller keys. They prefer big keys, like "space bar"*

- **'Dyslexia people' performance**

Miss letters or add letters: *For instance, "student" -> "studnt"*

Reverse letters: *For instance, "student"->"studnet"*

Spelling errors: *For instance, leave vowel out of word, "magic"->"mjc"*

Mix up similar words: *For instance, "dose"->"does"*

Phonetic form: *For instance, "shud"->"should"*

- **Other typing performance**

Miss words: *Leave out words in the typed sentences.*

Reverse words: *Reverse words in a sentence.*

Mix lines: *If there are some similarities (for example, include some same words) between two or among more lines, users could mix the lines.*

Non-sense sentences: *From the context of paragraph, the sentences which user typed are not what they intend to type.*

Additional words: *User could add additional words to a structured and fully meaningful sentence.*

One-hand users: *There are unclear different difficulties for left hand and right hand user in using the same kind of keyboard.*

- **User special characters**

The font and size influence: *The font and size of a document could affect user's cognitive ability.*

Environment influence: *Noise, music could lead a positive or negative influence while disabled people are typing.*

Background color influence: *Some users prefer for example, to have a yellow color background on both of screen and keyboard.*

Image influence: *Dyslexics are usually more comfortable with computer images than words.*

Capital errors: Typing *capital letters is a difficulty for some disabled people. The occurrence of errors is high.*

Habit mistakes: *For example, one may type SPACEBAR after each word, but sometime the required may be a punctuation mark.*

- **Reflected questions**

More efficient typing: *Generally their typing speed is far slower than average. If the document isn't finished on time, the frustration could further badly affect their typing performance.*

Higher typing error tolerance: *The MS word does have certain tolerance to spelling errors, but typing mistakes made by disabled people may vary.*

- **Required solution from user**

Learning and evolving: *These require a system lying between input device and applications, which is able to analyze users' typing behaviour, and then predict user's typing intention and correct mistakes accordingly. The system should have the knowledge of user's characters and be able to learn and recognize the new patterns over time.*

3.4 Intelligent Keyboard framework

As illustrated in the last section, computer users with motor disabilities or cognitive problems may have difficulties in accurately manipulating the QWERTY keyboard. As for motor disability this may be seen in a form of tremor owing to a certain disease such as Parkinson's or any other factor, for instance reduced range of hand motions due to Arthritis. Cognitive problems usually are caused by loss of the ability to process, learn and remember information. For example, Dyslexia can cause significant problems in remembering even short sequences of numbers in the correct order.

This section is a further demonstration of the designed ALMIL framework by analyzing and developing a particular case. Based on the disabled user investigation on computer QWERTY keyboard in section 3.3, an innovative framework – Intelligent Keyboard (IK) hybrid framework is designed to analyze disabled users' typing stream, and accordingly correct typing mistakes and predict users' typing intention.

Also, this section intends to give a user-oriented solution to help disabled people to use keyboard more efficiently. Based on the design, user's input typing stream can be checked in sequence by each module along with user's typing process.

Assumption: *The hybrid framework has a hierarchy structure. The functions in the hybrid architecture are categorized into two levels: the first level is named as* ***'unit*** *'(e.g. Error correction unit); under the first level (say sublevel, e.g. Motor Checker function) functions or unified functions (e.g. Noise process function) are named as* ***'module'****.*

3.4.1 Intelligent Keyboard framework and Rationale

For an implementation of cognitive tasks, it is shown that rather than seek solutions based on symbolic Artificial Intelligence e.g. neural networks alone, a more potentially useful approach would be to build structured connectionist models or hybrid system. Then it is able to combine more functions for some specific purposes based on machine learning model, which includes four fundamental elements, namely, Environment Analysis, Learning Elements, Knowledge Base and Performance element [Simon Haykin, 1999]. All of these units could be divided into more subdivisions, to form a highly efficient hybrid framework, while the whole framework would be taken as a noisy language modelling layer (named as Intelligent Keyboard) between keyboard and applications to analyze users' typing stream and filter possible noises.

Intelligent Keyboard hybrid framework which combines neural network, statistics and natural language model together is designed and intends to provide users with two fundamental functions, namely, text prediction and typing correction. User's typing data stream can be checked, rectified, and predicted in sequence by going through each unit following user's typing process. Through this way, the noises in typing stream are filtered significantly and the language interaction between user and computer becomes smoother.

Multiple units and a database (long-term memory and short-term memory) have been presented according to distinct technologies. The designed Intelligent Keyboard architecture is shown in Figure 3.2, which is thoroughly explained through an example in the next section. The architecture includes four processing units: Text prediction unit, Inference unit, Natural Language Processing unit

(*marked in dash line, which is currently not considered given the scope of this project*), and Error correction unit; and two additional modules, namely, User interface module and Noise process module to enable the interaction of the user with outside environment such as computer keyboard.

The two additional modules function as data pre-processing, post-processing and interaction interface. They correspond to machine learning model's Environment Element and part of Performance Element respectively. The Knowledge Base element is represented by Long-term Memory and Short-term Memory. The rules inferred from Inference Engine and some other facts such as user profile and frequently used texts, are saved in the Long-term Memory. Other facts such as recently used new words are stored in the Short-term Memory which will be transferred to the long-term memory if a certain threshold is reached.

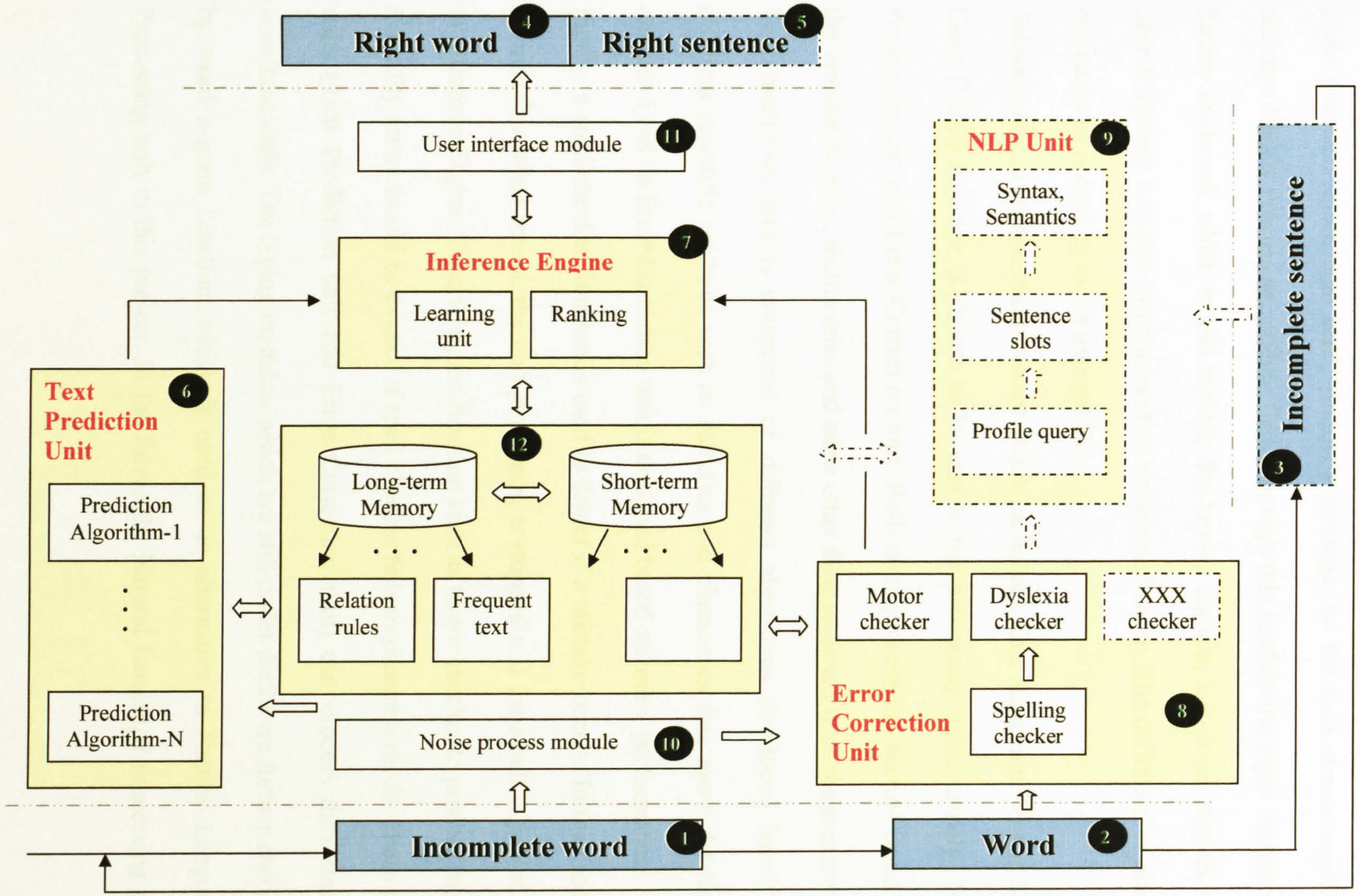


Figure 3.2 The architecture of Intelligent Keyboard

Intelligent keyboard is invoked by user's key strokes. As much of the typing data stream could be un-preprocessed, incomplete and noisy, for example, a long key press generates more than one Window's message, so the data stream needs to undergo Noises Processing module first. Through this module the input vectors are further exploited, which would include the key-up signals, key-down signals, the time difference between two consecutive strokes and so on. The definition of noises can be given according to the user profile.

Subsequently, a representation vector which includes a time stamp and Virtual Key Code (VKC) message is chosen to be sent to the processing units, namely, Text Prediction unit and Error Correction unit. Both units process the vectors based on the association rules, dictionaries and some other facts retrieved from the memory.

Text prediction unit is composed of different algorithms developed based on different scientific methods such as statistics and phonemics (*see more details in chapter 5*) while Error Correction unit is designed based on users' performance.

Firstly, a spell checker function is used to detect if a mistake occurs. In the case of no mistakes being traced, the unit processing is stopped and the result is passed on to Inference Engine. Otherwise, the function such as motor checker to process motor disability errors would be evoked if spell checker fails to present a result. These two units (Text Prediction unit and Error Correction unit) can process data stream simultaneously. The typing mistakes which are still under doubt are further checked by word *n*-gram function, which is used as an alternative to Natural Language Processing unit in this project. In the future the Natural Language Processing unit

could be implemented and used to check syntax and semantics errors. Finally, the results are refurbished and shown to the user by User Interface unit.

The results (e.g. a list of words) generated from Text Prediction unit and Error Correction unit, which are usually more than one, are presented to Inference Engine unit. The Inference Engine unit ranks the results based on their probabilities to generate a word-list or directly presents a highest ranking presentation to the user. The user's feedback such as selections and correction actions is recorded by the inference algorithms (here is a neural network algorithm), and transferred to rules or rewards to be stored into the memory.

From Figure 3.2, blue boxes and their connections represent the system's input and output process. An input of a sentence is a process passing through different structure status from letter, word to sentence, during which distinct units are evoked up according to the structure status changes.

The processing units from left to right, which has been marked as light yellow, are named as Text prediction unit (No. 6), Inference engine unit (No. 7), Error correction unit (No. 8), and Natural Language Processing (NLP) unit (No. 9). Data storage (No. 12) is divided into short-term memory and long-term memory, where the temporary and permanent information are stored. There are two additional modules: Noise process module (No. 10) and User interface module (No. 11), which are responsible for the interaction with the outer environment such as keyboard. (*Further details of all units and modules can be found in Appendix E*).

3.4.2 Intelligent Keyboard framework demonstration

Let's assume a one-hand user is typing the sentence: **Tomorrow shall we go to the park?** Capital letter is a big obstacle to one-hand users, as they have to type twice when inputting a capital 'T' ('Shift'->release-> 't' ->release).

After receiving these two Windows Messages, the Noise Process module analyses and assembles them to one letter - 'T', then send it to Text Prediction unit to provide the user with a list with ranked words, as shown in Example 3.1,

T_
1) Talk 2) That 3) The 4) Them 5) They 6) This

Example 3.1 Words List Interface -1

Since the target word did not appear in this list, the user continues to type the next letter "o", and the following list of words starting with "t-o" appears, as shown in Example 3.2,

To_
1) Today 2) Tomorrow 3) Tonight

Example 3.2 Words List Interface -2

Then the user can choose the word "Tomorrow" at once. Dyslexia users often reverse letters, e.g. 'shall' to 'sahll', as shown in Example 3.3. As a complete word (i.e. a string between two non-alphabets), it is sent to Error correction unit for

verification. Some errors are filtered out after spell checking. For the remaining errors, one of these modules, namely, 'motor checker' or 'dyslexia checker' is chosen as a further solution. Eventually 'sahll' reaches Dyslexia checker module and is corrected to 'shall'.

Tomorrow sahll _
1) shall 2) shell 3) should

Example 3.3 Words List Interface -3

The typing moves on, as shown in Example 3.4,

Tomorrow shall we go park? _
Tomorrow shall we go to park?

Example 3.4 Words List Interface -4

No other units could cope with such error (e.g. not 'go park', but 'go to park') (***attention**: the Inference Engine unit could learn and recognize after rounds of training.*), until the sentence is sent to word *n*-gram module to conduct a syntax analysis.

During the whole process, each unit frequently communicates with memory database to fetch user profiles, association rules and system configuration. Meanwhile the results of units' analysis, user's keystrokes (e.g. correction action 'wrong letter'-> user press 'Del'->'right letter'), and user's responses are sent back

to Inference Engine unit. According to user's feedbacks, Inference Engine unit again optimizes its model and related vectors. In the mean time, it also re-ranks the display list before sending it to User interface module.

In some way Text Prediction unit and Noise Process module are responsible for the minimum input unit (i.e. **letter**) errors' correction; Error correction unit is in charge of **word** checking; word *n*-gram helps to wipe off the rest of errors based on the analysis of **sentences**' syntax and semantics. User's input context is checked in sequence by each unit with user's typing process, corresponding to the blue boxes which circles the architecture, and are marked as 'Incomplete word', 'Word' and 'Incomplete sentence'.

3.5 Summary

This chapter develops a novel intermediate layer noisy language modelling framework – ALMIL, which lies between computer input devices layer and software applications layer, to analyze language data stream with noises and provide a user with data prediction and correction functions. This framework highly emphasizes on its adaptability, learning ability and compatibility, which can be used by text entry applications such as computer keyboard and bimodal related applications as a standard intermediate layer noisy language processor.

Following a disabled user investigation, a demonstration of ALMIL through Intelligent Keyboard framework aiming to design a user oriented hybrid framework

with self-adaptive function to help disabled people in using QWERTY keyboard more effectively is developed.

The Intelligent Keyboard framework provides disabled people with a comprehensive solution to use QWERTY keyboard more effectively. It can learn from user's typing history and feedback based on neural network algorithm. The user's typing intention is predicted based on user's input history, and the typing errors in data streams are gradually corrected as the data stream goes through each module. Through this way, the noises in typing stream can be filtered significantly and the language interaction between a user and a computer becomes smoother.

The modules of Intelligent Keyboard architecture are extendable according to distinct user profiles. It is developed as a practical demonstration of ALMIL framework. Multiple technologies such as statistics and neural network are applied to the framework and multiple modules such as motor checker and dyslexia checker are integrated into this framework. It fills the gap between input device (i.e. keyboard) and user applications as a noisy language filter. Hence, the ALMIL framework builds a foundation for this research. In the next chapter an intensive neural network modelling based on the analysis of both plain text and user typing stream is fully presented.

CHAPTER FOUR

NEURAL NETWORK AND LANGUAGE MODELS DEVELOPMENT

4.1 Introduction

The previous chapter proposes an intermediate layer noisy language modelling framework called ALMIL to analyze language data stream with noises. In order to demonstrate and simulate the noisy language modelling process, a comprehensive neural network models development based on the analysis of both plain text and user typing stream is carried out. Firstly, an amount of datasets including a part of a book and two user typing stream logs are determined and collected. Furthermore, a preprocessing tool is developed, and data extraction and coding method are proposed. Then, a Focused Time-Delay Neural Network Model with extendable input and hidden units is designed and performed with noise-free, noisy and typing stream datasets respectively. Distinct numbers of grams with distinct numbers of hidden neurons are cross-experimented. Based on different noise rates, the noisy language text is modeled with a *27* symbol set, while the typing stream model is designed and implemented based on a larger symbol set.

Following a general language modelling on noise-free, noisy and typing stream datasets, several distinct neural network models are developed based on a specific dataset extracted from user typing stream, for example, the influence of Time Gap on user's typing performance is studied through Time Gap modelling and Prediction with Time Gap modelling. Finally, a novel Probabilistic Neural Network model is developed to simulate the 'Hitting Adjacent Key errors' based on Key Distances, Time Gap and Error Margin Distance elements.

4.2 Experimental datasets

One of the objectives in this research is to design neural network models and test these models by applying experimental noisy datasets, to estimate their prediction and correction accuracy rates. The approach adopted here is to construct distinct and extendable intelligent models according to the varied data samples and related features. Various neural network models are trained by the minimization of an appropriate error function defined with respect to the training dataset. Several pieces of data samples are collected from different sources. The main experimental datasets used by this research are described below,

- **DATASET ONE:** a novel – 'Far from the Madding Crowd' was written by Thomas Hardy [1874], which is his fourth novel and first major literary success. It has been used as a testing sample by some compression algorithms such as PPM*. The version used here is extracted from Calgary Corpus [2009] with a size of *751kb*. An example is shown below.

> *"I don't think it is for you, sir,' said the man, when*
> *he saw Boldwood's action. "Though there is no name*
> *I think it is for your shepherd.'*

Due to system performance restriction, a continuous *100k* segment is extracted from dataset one as a testing sample, and is subsequently divided into training dataset, validation dataset and testing dataset in a proportion of

70%, *20%* and *10%* respectively. The training subset is used to estimate weights and thresholds of neural network models; the validation subset is used to determine when to stop training. Error estimates using training and validation data will be biased as both are applied to design the neural network, so the testing subset is involved to obtain an unbiased estimate of the generalization error.

- **DATASET TWO:** it is extracted from Disability Essex helpline keystroke log. The associated computer is used as a question recording, database query and email writing tool by a disabled volunteer. From the reflected keystroke log, the typing mistakes are predominantly about adjacent key press and prolong key press errors. The keystroke recording tool used in this research is KeyCapture software [Soukoreff & MacKenzie, 2008], which has been modified and adjusted for the purpose of this research. It runs in background under Windows environment to collect keystrokes without interfering with user's work. A typical structure of generated log is demonstrated in Figure 4.1.

```
01929 KeyPress    20080605-132149-593 "T" Status=(down) Key(84) Extra(0x14) KeyDistance(3.500000) TimeGap(307)
01930 KeyPress    20080605-132149-655 "T" Status=(up) Key(84) Extra(0xc014) KeyDistance(0.000000) TimeGap(62)
01931 KeyPress    20080605-132149-658 "H" Status=(down) Key(72) Extra(0x23) KeyDistance(2.500000) TimeGap(3)
01932 KeyPress    20080605-132149-694 "H" Status=(up) Key(72) Extra(0xc023) KeyDistance(0.000000) TimeGap(36)
01933 KeyPress    20080605-132149-804 "A" Status=(down) Key(65) Extra(0x1e) KeyDistance(5.000000) TimeGap(110)
01934 KeyPress    20080605-132149-992 "A" Status=(up) Key(65) Extra(0xc01e) KeyDistance(0.000000) TimeGap(188)
```

Figure 4.1 A piece of KeyCapture log sample

KeyCapture can answer the following questions: What editing keys are really used when entering text? How much time does a typical user spend using the mouse, as opposed to entering text? Or, what applications does a user use, and for how long? [Soukoreff et al., 2003]. In Figure 4.1 each line contains nine columns as illustrated below.

Column 1: sequence number
Column 2: key press – used to distinguish from mouse action
Column 3: action date and time (ms)
Column 4: key pressed
Column 5: key status (up or down)
Column 6: the value of Virtual Key Code
Column 7: key press information from lParam
Column 8: distance between two keys of a standard keyboard
Column 9: time gap between two consecutive key presses.

According to Windows system, each key stroke evokes two messages, namely, key-pressed (either WM_KEYDOWN or WM_SYSKEYDOWN) and key-released (either WM_KEYUP or WM_SYSKEYUP), which are associated with two 32 byte parameters, *wParam* and *lParam*. In Figure 4.1, every couple of lines marked with status equal to 'up' or 'down' represents a complete key press. The time gap is the margin value of each two rows in *Column 3*. The key distance is the sum of horizontal and vertical distance from one key to another. A half key (*0.5*) distance is counted if two keys' distance is not an integer multiple of one-key distance.

The Helpline dataset is collected within a period of twenty three days with a capacity of *6.5* mega bytes. As an example generated in a real-time, the occurrences of alphabet (recorded in virtual key codes) counted in keys up and keys down are shown in Figure 4.2.

	40	41	42	43	44	45	46	47	48	49	4a	4b	4c	4d	4e	4f
UP	0	839	242	379	3801267	231	246	487	799		18	134	519	287	657	820
DOWN	0	839	242	388	3801267	231	246	487	802		18	134	519	287	657	819

	50	51	52	53	54	55	56	57	58	59	5a	5b	5c	5d	5e	5f
UP	271	11	650	741	831	355	115	219	40	238	7	23	0	0	0	0
DOWN	271	11	650	741	830	355	115	219	40	238	7	865	0	0	0	0

Figure 4.2 A statistic of alphabet occurrences in typing stream

- **DATASET THREE:** typing samples from people with Parkinson disease and motor disability are also gathered. The segment chosen and used by intelligent models is as follows,

ORIGIN – *'the quick brown fox jumped over the lazy dog'*
TYPED – *'hthe quick brrooownn fgow jummppefd*
iobverethe lwqazy doooggfg'

The 'ORIGIN' refers to a typing reference, and the 'TYPED' refers to a user's typing sample shown in a Notepad editor. These are recorded in a plain text format. Its associated keystrokes are also tracked and saved by the software KeyCapture.

Some other testing datasets are also used in this project. They will be explained along with the description of associated intelligent models.

All the datasets collected here lie in two categories: non-noisy or noisy dataset, which are closely related to the research hypotheses made previously. Dataset one as a clean dataset (i.e. non-noisy) having a considerably large size will be used by the general noisy language modeling process. The noises can be added gradually into the clean dataset to model noise-prediction rates. Dataset two will be taken as a specific research case and applied to type stream models. However, user typing behaviours in dataset two are ambiguous that may be difficult to be classified clearly into a specific category, therefore, a separate typing stream research case aiming at processing the 'hitting adjacent errors' (i.e. dataset three) is collected. Certain factors such as Time Gap and Key Distance will be addressed in the related model development.

4.3 System environment

System environment including computational capability and memory capacity may have a great influence on the intelligent models' testing process. Therefore, it is necessary to fully present the related system environment of all testing. In this research, all experiments are carried out based on Lenovo T60 (IBM) platform,

Windows XP operating system, and MATLAB and its Neural Network Toolbox. A detailed system environment description is given as follows,

- **Operating System:**

```
Microsoft  Windows  XP  Professional  Version  2002
Service Pack 3
```

- **Hardware Configuration:**

```
Intel® Core™2 CPU T5600 @1.83GHz, 3.00GB of RAM
Hard disk 120GB
```

- **Toolbox and Configuration:**

```
MATLAB Version 7.4.0 (R2007a) Neural Network toolbox
Physical Memory (RAM):              Total - 3070 MB
Page File (Swap space):             Total - 4445 MB
Virtual Memory (Address Space):     Total - 2047 MB
```

4.4 Data processing tools

Data collection procedure has been followed in two ways: hand sampling from sources such as compression websites and wikipedia, and automatic collection using KeyCapture software (see Figure 4.3). As manual extraction and pre-processing of data fed to intelligent models from different sources is tedious and the chance of human error is high, all gathered raw data need to be further processed prior to being used by intelligent models as input vectors. However, no particular software or

functions have been found to perform the extraction and pre-processing. To overcome this problem, a piece of software called Enstatistics is developed that is capable of providing a platform to pre-process dataset by reading the raw data from different sources and transforming them into text files to meet the requirements of intelligent models. The overall process is shown in Figure 4.3.

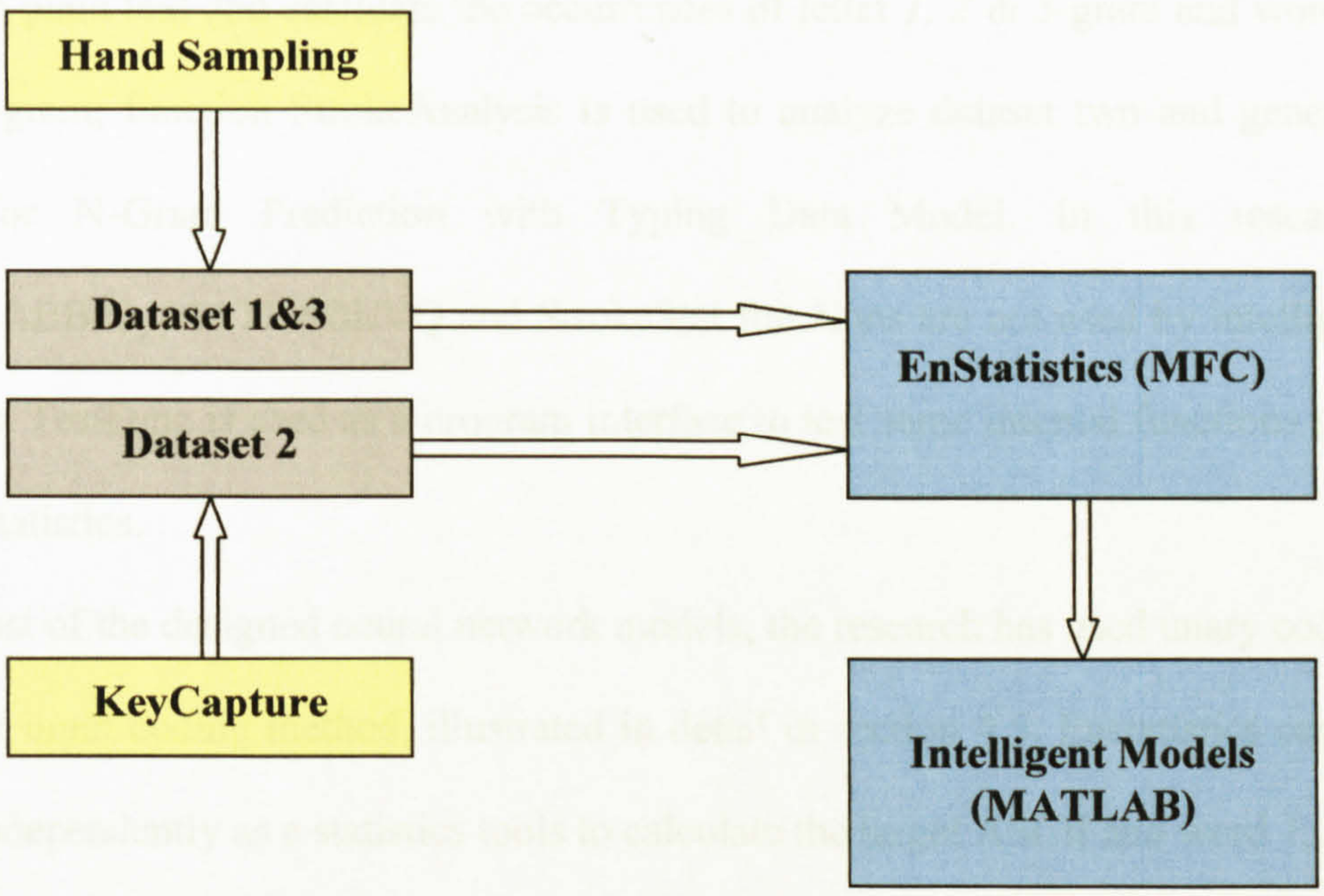


Figure 4.3 Schematic representation of intelligent modelling

Enstatistics is developed based on Microsoft Foundation Class (MFC) Library to convert the datasets to an appropriate input format for neural network models. The software interface within a data pre-processing description diagram is shown in

Figure 4.4. Each function will be illustrated following the latest introduction of specific intelligent models.

Figure 4.4 gives a clear demonstration of data preprocessing related to this research. The datasets are interpreted by distinct functions of Enstatistics based on specific requirements. Each function may correspond to a particular neural network model or a particular requirement. For instance, function TextAnalysis is used to analyze the English plain text and calculate the occurrences of letter *1, 2 & 3*-gram and word *1, 2 & 3*-gram; function StrokeAnalysis is used to analyze dataset two and generate data for N-Gram Prediction with Typing Data Model. In this research, MATLABBP2, MATLABLVQ and StrokeStat functions are not used by intelligent models. TestFunc is used as a program interface to test some internal functions used by Enstatistics.

For most of the designed neural network models, the research has used unary coding as their input coding method, illustrated in detail in section 4.5. Enstatistics can be used independently as a statistics tools to calculate the target ASCII and word *1, 2 & 3*-gram. Also it can be used for any neural network model for data conversion with a minimal alteration of the existing code. The performance and the accuracy of the software have been verified within this research.

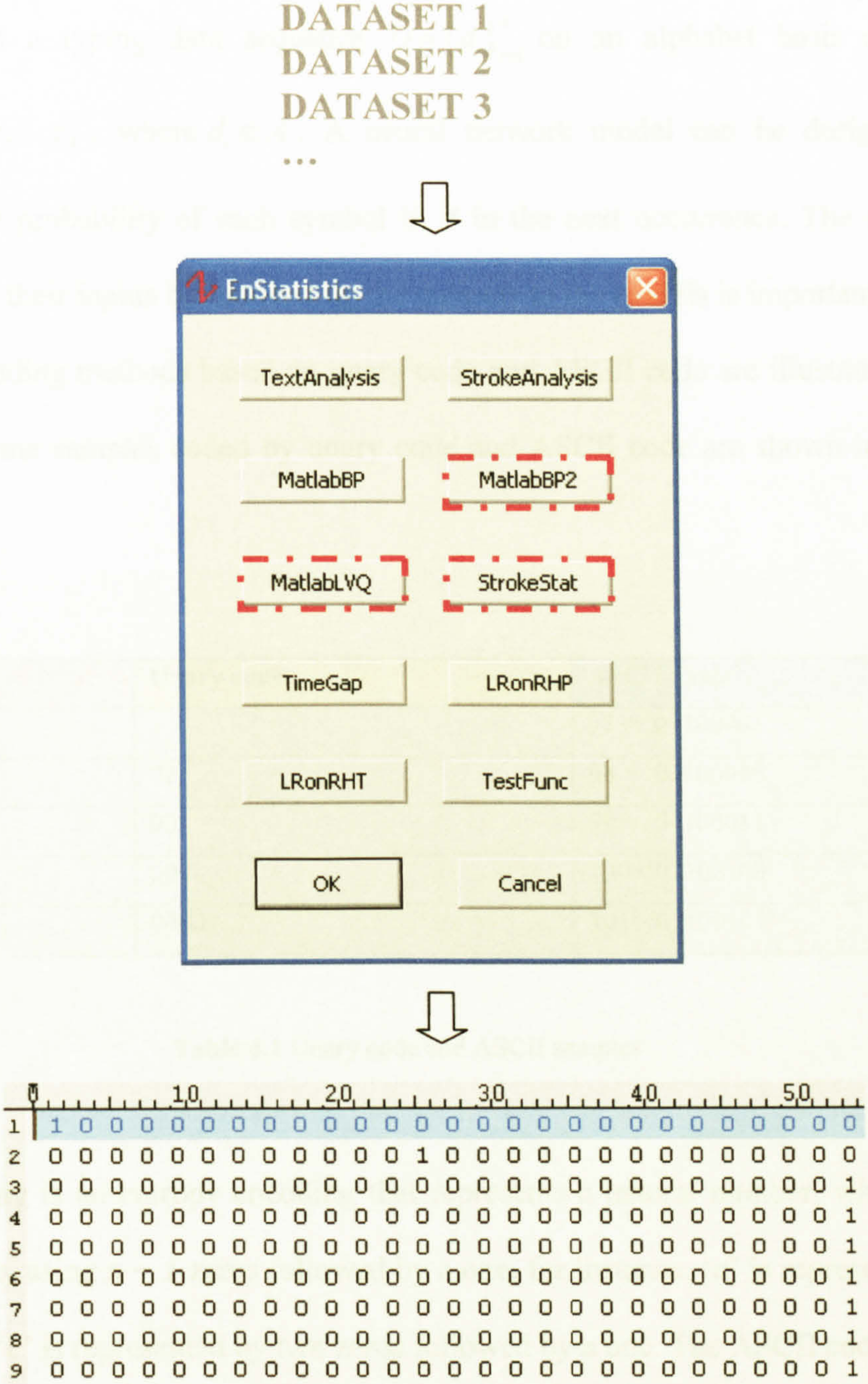


Figure 4.4 Interface representation of intelligent models data pre-processing

4.5 Input coding

Let's model a typing data sequence $D = \{d_i\}_{i=1}^{n}$ on an alphabet basis of size space $A = \{a,...,z\}$, where $d_i \in A$. A neural network model can be designed to compute the probability of each symbol in A in the next occurrence. The method how to code their inputs before feeding the datasets to the models is important. Here, two input coding methods based on unary code and ASCII code are illustrated. For example, some samples coded by unary code and ASCII code are shown in Table 4.1.

Alphabet	Unary code	ASCII Code
a	1	97 = '01100001'
b	01	98 = '01100010'
c	001	99 = '01100011'
d	0001	100='01100100'
e	00001	101='01100101'

Table 4.1 Unary code and ASCII samples

Unary coding is an entropy encoding that represents a natural number, which is a symbol here, using $n - 1$ zeros followed by a one, for instance, '*a*' is represented by a '*1*' while '*c*' is represented by two zeros followed by a one. The ASCII code uses a fixed length (e.g. *8* bits) with its value to represent a symbol as shown in table 4.1, where '*a*' is represented by '*01100001*' which consists of three ones and five zeros.

The unary codes can be adapted to a fixed length to fit the requirements of neural networks without changing the number of input neurons. For example, let's consider a dataset within a three symbols {*a, b, c*} space, these three symbols can be coded as fixed unary codes {*100, 010, 001*} or a ASCII style code {*01, 10, 11*}. Let's suppose there is a training iteration of neural network where the input is a symbol '*c*' and the target output is '*b*', then the designed neural network architecture with unary coding and ASCII coding with two and three hidden layer neurons respectively is illustrated in Figure 4.5 and Figure 4.6.

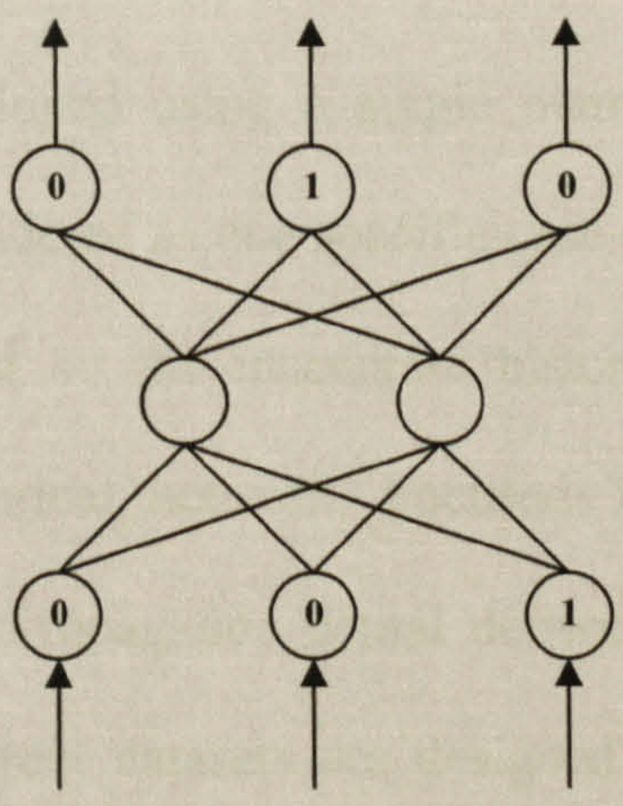


Figure 4.5 Neural Network with unary coding

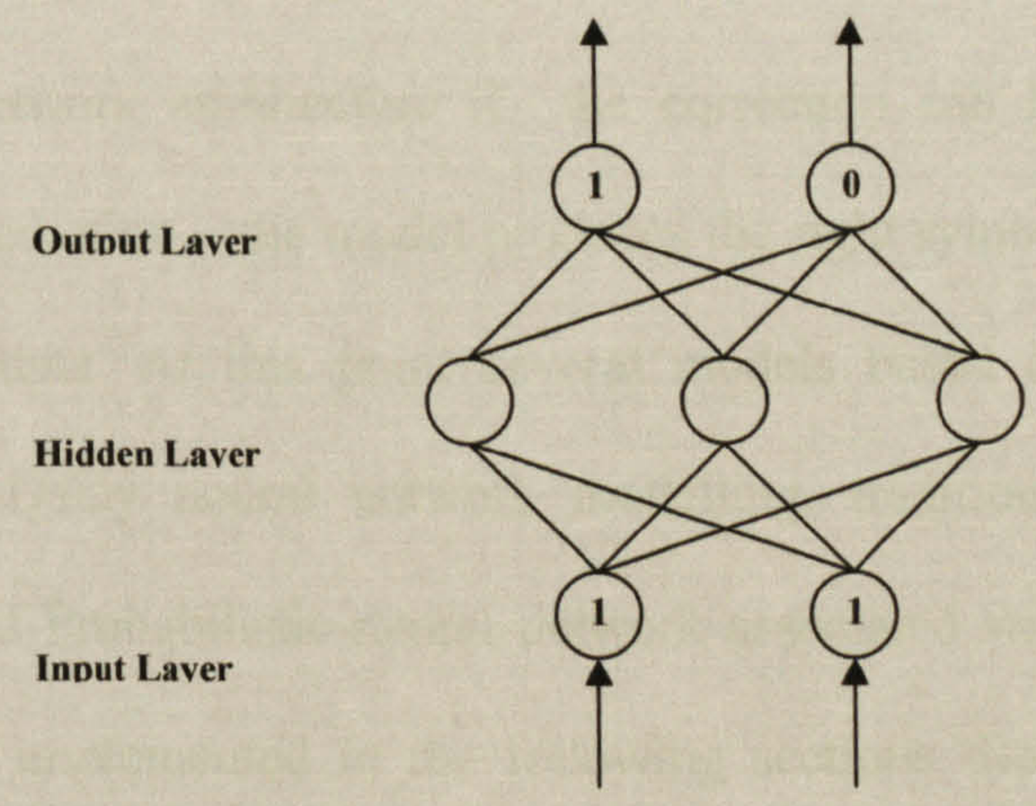


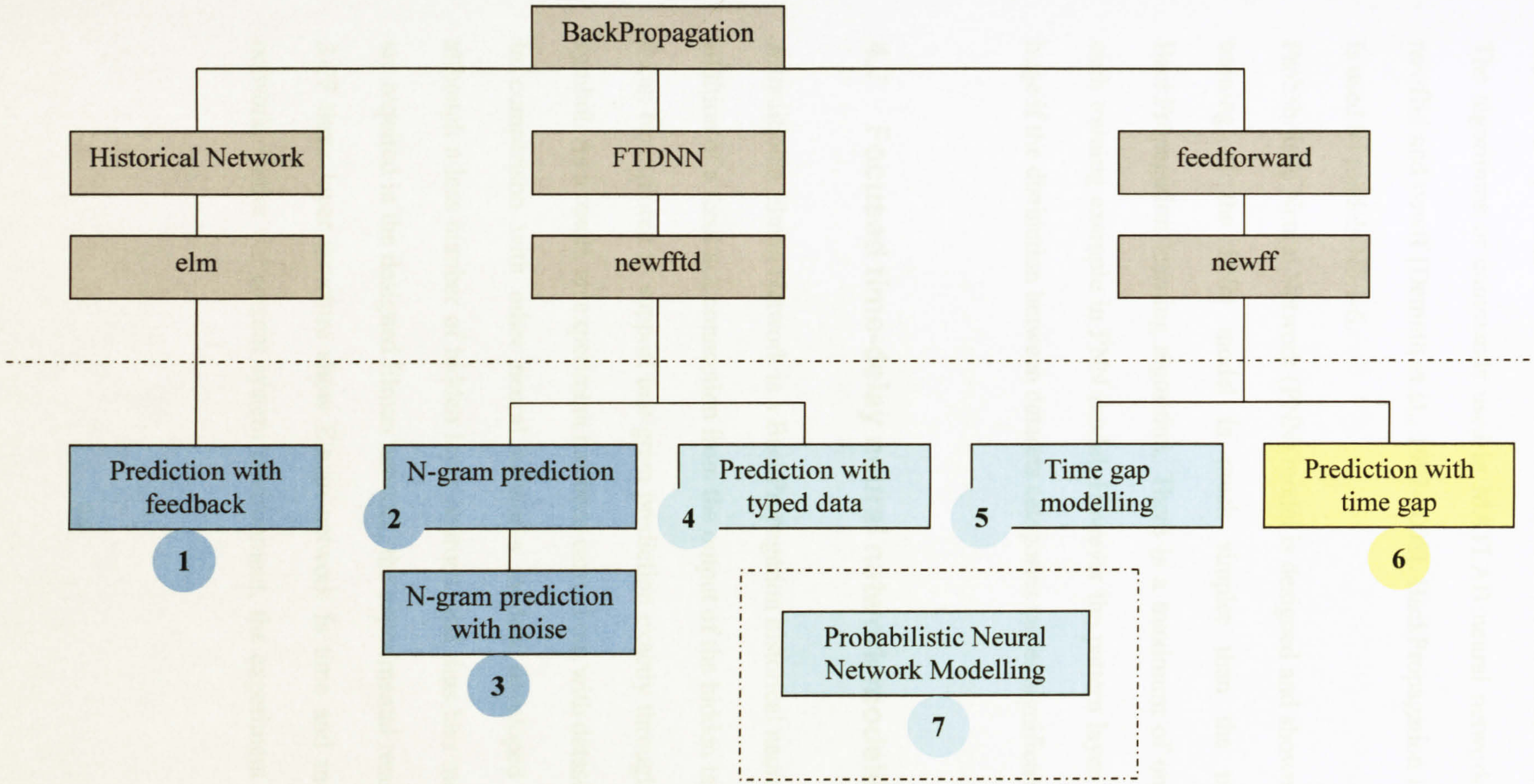
Figure 4.6 Neural Network with ASCII coding

At the input layer of Figure 4.5, only one neuron is activated once at a time while the rest of the inputs are set to zeros. Figure 4.6 shows neural network architecture with ASCII style coding, which is able to use only two neurons at the input layer to represent a dataset. For a neural network with ASCII coding, less neurons are

required than unary coding at input layer, but more neurons are required at hidden layer, and the time cost of training based on ASCII coding is higher. This research has applied the unary coding both to the input neurons and output neurons of neural network models where it is necessary. The determination of number of hidden layer neurons is based on adjustable, experimental and heuristics methods.

4.6 Neural network models development outline

The text entry prediction and correction using neural network models can be achieved based on historical data through designing two types of models; or can be developed using a single neural network architecture if the correction can be considered as one specific case of prediction – the model produces the right symbol based on the inaccurate historical data. At this point several models based on Historical network, Focused Time-Delay neural network modelling, traditional BackPropagation neural network and Probabilistic neural network associated with different datasets are designed and implemented in the following sections. Both general languages based modelling using Focused Time-Delay neural network and specific data extraction based modelling using traditional BackPropagation Neural network and Probabilistic neural network are tested. Figure 4.7 demonstrates the experimental procedure with datasets one, two & three.



elm: Elman historical network
newfftd: Focused time-delay neural network
newff: feed-forward network of propagation

Figure 4.7 The procedure of experiments with datasets one, two & three

Altogether seven models (marked as No.1 – 7) are designed. They are trained separately by using three different datasets colored in blue, turquoise and yellow. The algorithms or commands used in MATLAB neural network toolbox are elm, newfftd and newff [Demuth et al., 1992-2008]. BackPropagation learning algorithm is used by models No.1-6.

Probabilistic Neural Network (PNN) model is designed and shown in box No.7. The training of the PNN model is much simpler than the models based on BackPropagation learning algorithm. There is a maximum of one pattern unit for each training example in PNN model. However the pattern layer can become quite huge if the distinction between datasets categories varies significantly.

4.7 Focused time-delay neural network modelling

A traditional Elman Network is a BackPropagation historical neural network with an addition of a feedback connection from the output of the hidden layer to its input, so it can be explored to support one gram prediction exactly through tracing back one symbol. As a result, an experiment has been carried out with dataset one.

In comparison with other neural networks models developed in this research, although a less number of hidden layer neurons, and thus less memory in practice, are required in the designed Elman network, the experimental results based on a *27-3-27* three layer structure show Elman network is time and memory consuming network. Under the current system environment, the experiment demonstrates that

training using dataset one with >=*1500* symbols (contrast to a *100K* dataset used by FTDNN) has failed to reach a final result with an 'Out of memory' error. A further test with a training data of *1000* symbols and a testing data of *500* symbols is performed. The test shows the time cost is significant. It takes about *42* minutes (*2557* seconds) for the designed Elman network to complete a training process. The result shows that a prediction rate lies at *45.09%* of First Three (FT) Hitting Rate and at *28.26%* of First Hitting Rate.

Focused Time-Delay Neural Network belongs to dynamic network, which consists of a feedforward network with a tapped delay line at the input. In this research Focused Time-Delay Neural Network is selected because it can represent and explain the unclear and complex relationship between current typed sequence and its preceding one, and this is reflected by the associated probabilities.

Focused Time-Delay Neural Network is suitable for time-series prediction. Studying user's typing behaviour would require the network to study user's history and trace back length of context to some extent (so called n-gram) to predict the next probable occurrence of symbols. Adding one more gram requires one more time delay. Simon Haykin [1999] has demonstrated that the FTDNN is more reliable in response to time and memory requirement, while the design using Elman network to support *n*-gram prediction is complicated. However, a comprehensive research on Focused Time-Delay Neural Network language modelling has never occurred. In the following sections an extendable FTDNN n-gram prediction is developed to predict noise-free, noisy and typing stream datasets.

4.7.1 FTDNN N-Gram Prediction

- **FTDNN N-gram prediction definition:** let's assume existing string $S=\{s_1..s_i..s_j..s_k..s_m \mid i\le j\le k\le m\}$ and $(j-i)=n,(k-j)=l$, where s_i,s_j,s_k,s_m are symbols and *i, j, k, l, m, n* are natural numbers, if one builds a relation $R_n=\{x,y \mid x=(s_i...s_j)_n \rightarrow y=(s_{j+1}->s_k)_l\}$, then the relation is defined as *n*-gram's l-prediction; if one considers the special case $l=1$, then the relation is called *n*-gram's one-prediction, or *n*-gram prediction for short. For example, given string *S= {student}*, some *2*-gram prediction cases are,

'st' → *'u'*
'tu' → *'d'* ...
'en' → *'t'*

- **Symbol-Distribution Definition:** Given a certain ranking level *m* and a symbol set $A=\{a,...,z,space\}$, one defines the n-gram Symbol-Distribution in ranking level *m* is $D_n^m=\{x,y \mid x_i->y_i\}$, where $x_i \in$ symbol set A, and y_i is the level *m* Hitting Rate corresponding to each symbol.

Due to the system environment limitations in this research, rather than adopting the whole dataset, a chunk of data ranging from zero to *100k* is selected from dataset one in order to train and test the designed neural network models. The dataset is subsequently divided into training data, validation data and testing data. A symbol

set with twenty-seven elements, $A = \{a...z, space\}$, is applied to simplify the dataset one. The corresponding function in EnStatistics is MATLABBP2 whose processing logic followed by a sample of results is illustrated below,

for each symbol $s_i \in$ *context C, where* $C = \{s_1...s_n\}$

{

if 'a' $< s_i <$ *'z' then write unary code to file*

else if 'A' $< s_i <$ *'Z' then convert to* $\{a,...,z\}$ *and write unary code to file*

else convert to blank and write unary code to file

}

```
%a b c d e f g h  i j  k l m n o p q r  s t u v w x y z ''
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0      %'t'
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0      %'h'
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0      %'e'
```

First, all of the capital letters are converted to their corresponding lower case. The other symbols which do not belong to symbol set *A* are converted to *space*. Then each symbol is converted to a twenty seven length unary code. As shown in the example above, the word 'the' is successfully converted.

For this n-gram prediction model, a three layer FTDNN network shown in Figure 4.8 with twenty-seven input neurons, twenty-seven output neurons, extendible numbers of hidden layer neurons and extendible numbers of time delays is designed.

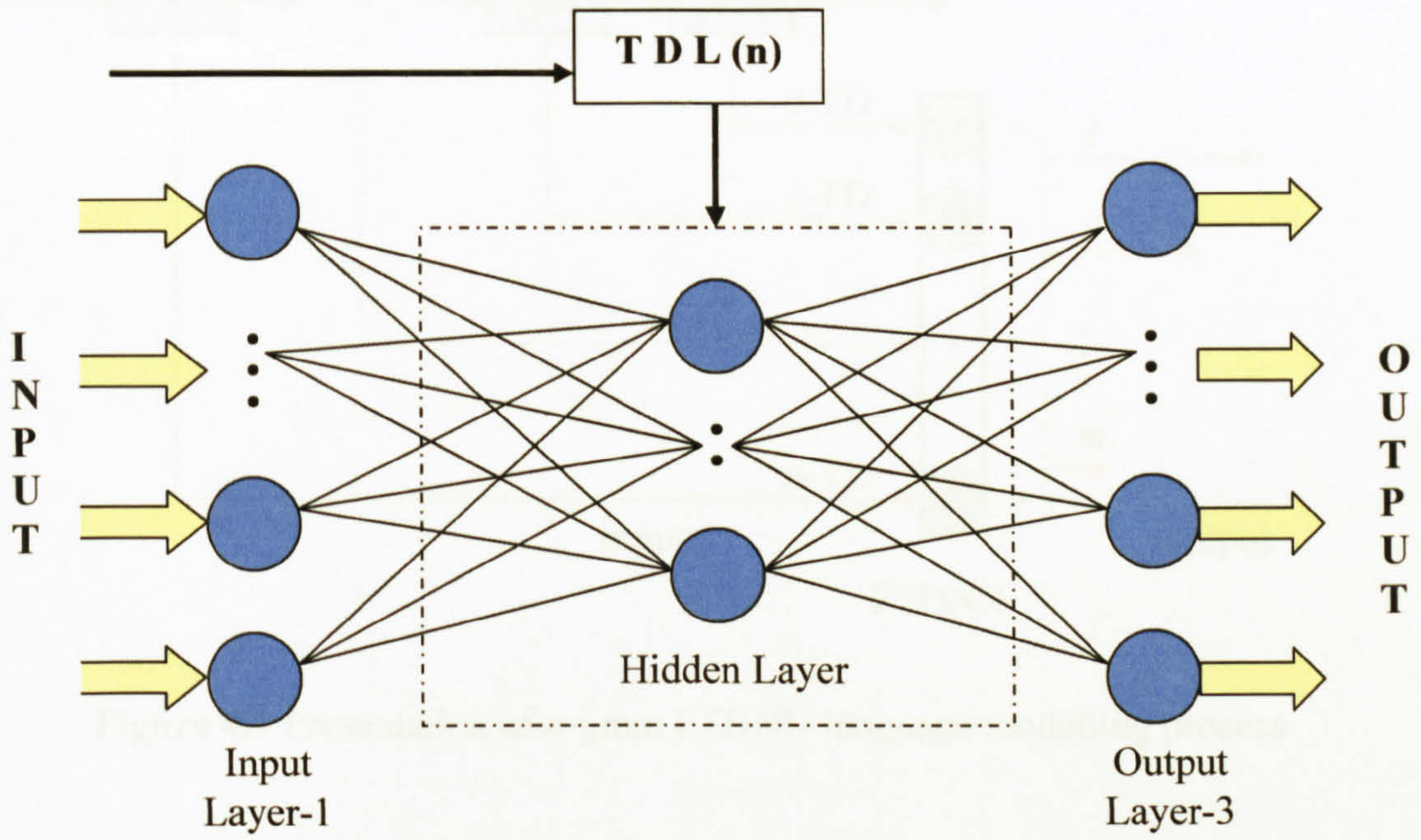


Figure 4.8 Architecture of Focused Time-Delay Neural Network

Figure 4.9 further demonstrates FTDNN *n*-gram language modelling process and the relation between *n*-gram, FTDNN and level *m* Hitting Rate. Studying user's typing behaviour requires the network to study user's history and trace back certain length of context (i.e. *n*-gram) to predict the next probable occurrence. Here, *n* time delays (i.e. *n*-TD) correspond to *n*-gram. Adding one more gram requires one more time delay. Variable *m* represents the number of the language symbol set as well as the

number of output neurons, then a level *m* Hitting Rate set related to the symbol set is generated in a testing iteration.

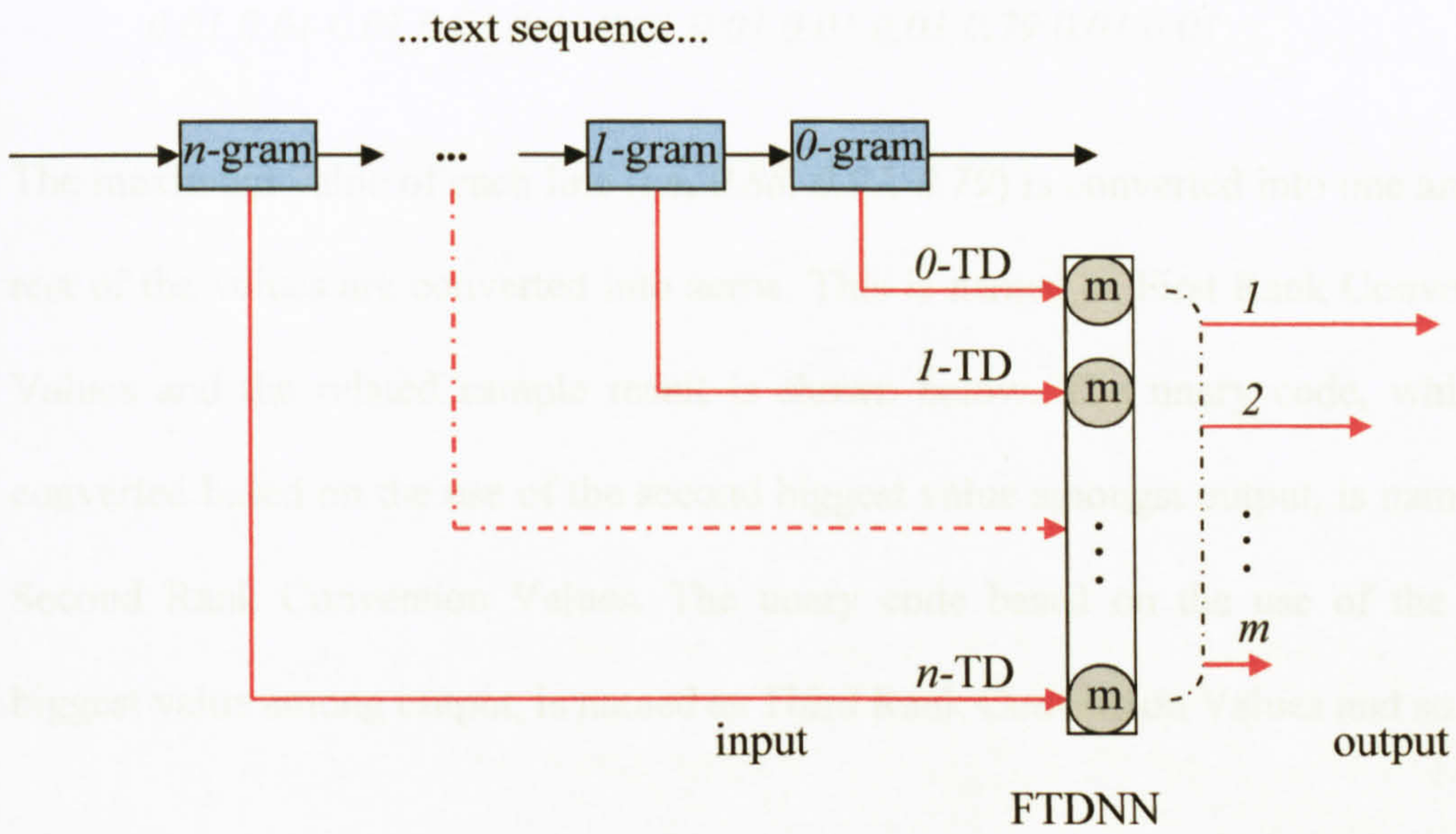


Figure 4.9 Presentation of *n*-gram FTDNN language modelling process

Both input and output are encoded in unary code. The '*purelin*' and '*logsig*' activation functions are applied to the input and output respectively. A post processing function which ranks the twenty-seven output of '*logsig*' in a descending order has been used to produce the unary code results: the maximum value is converted into one and the rest of the values are converted into zeros. For instance let's consider those three letters sample ('the') shown above, which produces the following outputs,

0.02 0.03 *0.01 0.01 0.01 0.01 0.01* ***0.88*** *0.01 0.01 0.01 0.01 0.01 0.01 0.01*
0.01 0.01 0.01 0.01 0. 01 0.01 0.01 0.01 0.01 0.01 0.01 0.01

0.02 0.04 *0.01 0.01* ***0.97*** *0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01*
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01

0.03 *0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01* ***0.02*** *0.01 0.01*
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 ***0.79*** *0.01 0.01*

The maximum value of each line (i.e. *0.88, 0.97, 0.79*) is converted into one and the rest of the values are converted into zeros. This is named as First Rank Conversion Values and the related sample result is shown below. The unary code, which is converted based on the use of the second biggest value amongst output, is named as Second Rank Conversion Values. The unary code based on the use of the third biggest value among output, is named as Third Rank Conversion Values and so on.

```
%a b c d e f g h i j k l m n o p q r s t u v w x y z ''
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0      %'h'
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0      %'e'
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0      %'y'
```

Then as a *2*-gram prediction model, the generated relationship between input and output is as follows,

```
't'   →   'h'
'h'   →   'e'
'e'   →   'y'
```

In order to weight the experimental results, two concepts are introduced here, namely, **Hitting Rate** and **First Three (FT) Hitting Rate**. If given a testing metrics *P*, a target metrics *T* and a testing result metrics *R* with their numbers of lines and

columns equal and expressed as n, m respectively, then the Hitting Rate is $HR = \{hr_i \mid hr_i = \frac{zeros(T - R_i)}{n}, i \in m\}$, where R_i is a vector of the i^{th} Rank Conversion Value of R, and *zeros()* is the function to compute the numbers of zero vectors included in the metrics. For instance, the second Hitting Rate is the second best option for the symbol prediction in all output neurons of FTDNN, while the third Hitting Rate is the third best option etc. Obviously the sum of all Hitting Rates is $HR = \sum_{i=1}^{k} HR_i = 100\%$.

Let's define the FT Hitting Rate as the sum of hr_1, hr_2, hr_3, where $k = 3$. Based on the previous example, the testing metrics P = *{'the'}*, R = *{'hey'}*, n = *3* and m = *27*. Let's assume the target dataset T = *{'hem'}*, while dataset R is the so called 1st Rank Conversion Value. Then the first Hitting Rate $hr_1 = \frac{zeros(T - R_1)}{n} = \frac{zeros('hem' - 'hey')}{3} = \frac{2}{3} = 66.67\%$, where two letters (i.e. *'h'* and *'e'*) are predicted correctly by the result R, that is, $zeros('hem' - 'hey') = 2$. In a similar way, the 2nd and 3rd Rank Conversion Values based on the previous sample are,

--2nd rank conversion values

```
%a b c d e f g h i j k l m n o p q r s t u v w x y z ''
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    %'b'
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    %'b'
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0        %'a'
```

--3rd rank conversion values

```
%a b c d e f g h i j k l m n o p q r s t u v w x y z ''
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     %'a'
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     %'a'
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0     %'m'
```

Where $hr_2 = \frac{zeros('hem'-'bba')}{3} = \frac{0}{3}$ and $hr_3 = \frac{zeros('hem'-'aam')}{3} = \frac{1}{3}$. Then the FT Hitting Rate is $\frac{2}{3}+\frac{0}{3}+\frac{1}{3}=100\%$, which is an ideal case. In this research its average rate based on FTDNN model is just below 50%.

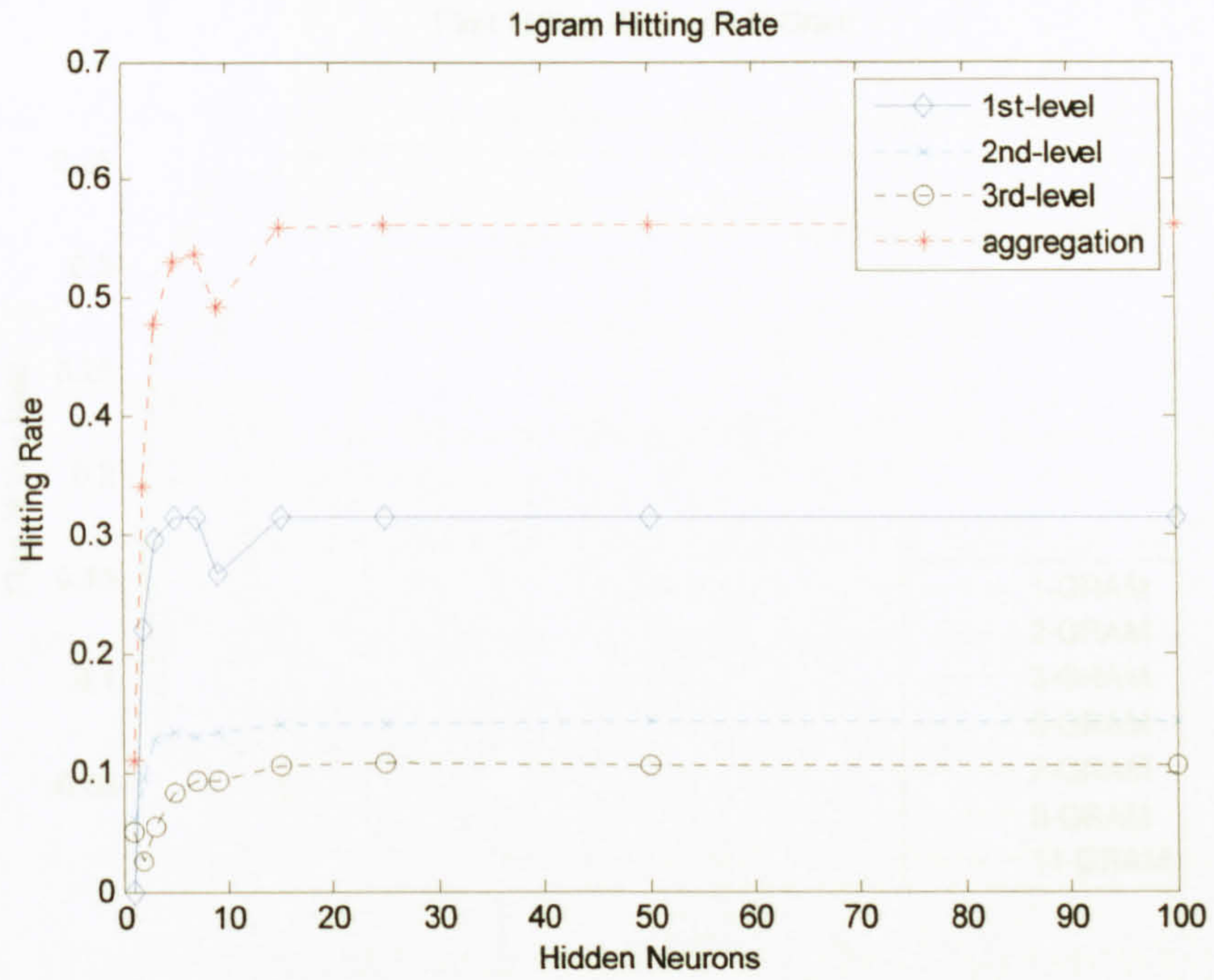
During the FTDNN model training and testing using dataset one, the numbers of grams – [*1, 2, 3, 5, 7, 9, 11, 13*] which are represented by time delays, and the numbers of hidden neurons – [*1, 2, 3, 5, 7, 9, 15, 25, 50, 100*] are cross-designed and implemented. Thereinto, as the gram reaches *11* and the number of hidden neurons reaches *100*, or as the gram reaches *13* and the number of hidden neurons reaches *15* or onwards, the memory of current system is beyond its limit. Therefore, the experimental results are abandoned from *G-11 & H-100* onwards.

Based on the various grams and hidden neurons, several types of plots have been designed. Figure 4.10 shows one type of design with two examples, namely, *1*-gram Hitting Rate and *11*-gram Hitting Rate plots. Both plots illustrate four different ranking level Hitting Rate curve, these are, First Hitting Rate – hr_1 (in blue), second Hitting Rate – hr_2 (in cyan), third hitting Rate – hr_3 (in black) and their aggregation – FT Hitting Rate – $hr_1 + hr_2 + hr_3$ (in red).

In Figure 4.10, the X-axis represents the increase of hidden neurons numbers with a maximum value of *100*, and Y-axis represents Hitting Rate, whose maximum value is *100%*. The first plot of Figure 4.10 shows that the Hitting Rate reaches a stable point from twenty-five hidden neurons onwards. On the contrary it is more difficult to reach a stable point for a neural network prediction model with more grams as shown in *9*-gram plot. The figures with [*1, 2, 3, 5, 7, 9, 11*] gram shown in Appendix D, clearly demonstrate that from seven-gram onwards the *n*-gram prediction results become more diverse, and the uncertainty also becomes higher under the current training dataset.

As shown in the first plot, *1*-gram Hitting Rate quickly convergences towards a best Hitting Rate from one hidden neuron to twenty five hidden neurons where it reaches first level HR = *31.5%*, second level HR = *14.0%*, third level HR = *10.8%* and FT Hitting Rate = 56.2%. The second plot – *9*-gram Hitting Rate Curve shows unstable Hitting Rate with an increase of hidden neurons. It reaches a maximum at fifty hidden neurons, where first level HR = *33.5%*, second level HR = *9.4%*, third level HR = *5.8%* and FT Hitting Rate = *48.7%*.

In order to demonstrate the effect of different grams on the hitting rate, a new type of plots are produced based on the same experimental results, as shown in Figure 4.11. It has [*1, 2, 3, 5, 7, 9, 11*] grams associated with various number of hidden neurons. It is evident that *2, 3 & 5*-gram give the best three First Hitting Rates while *1, 2 & 3*-gram give the best three FT Hitting Rates (in a small margin *2*-gram gives the best FT hitting rates and *3*-gram gives the best First hitting rate).



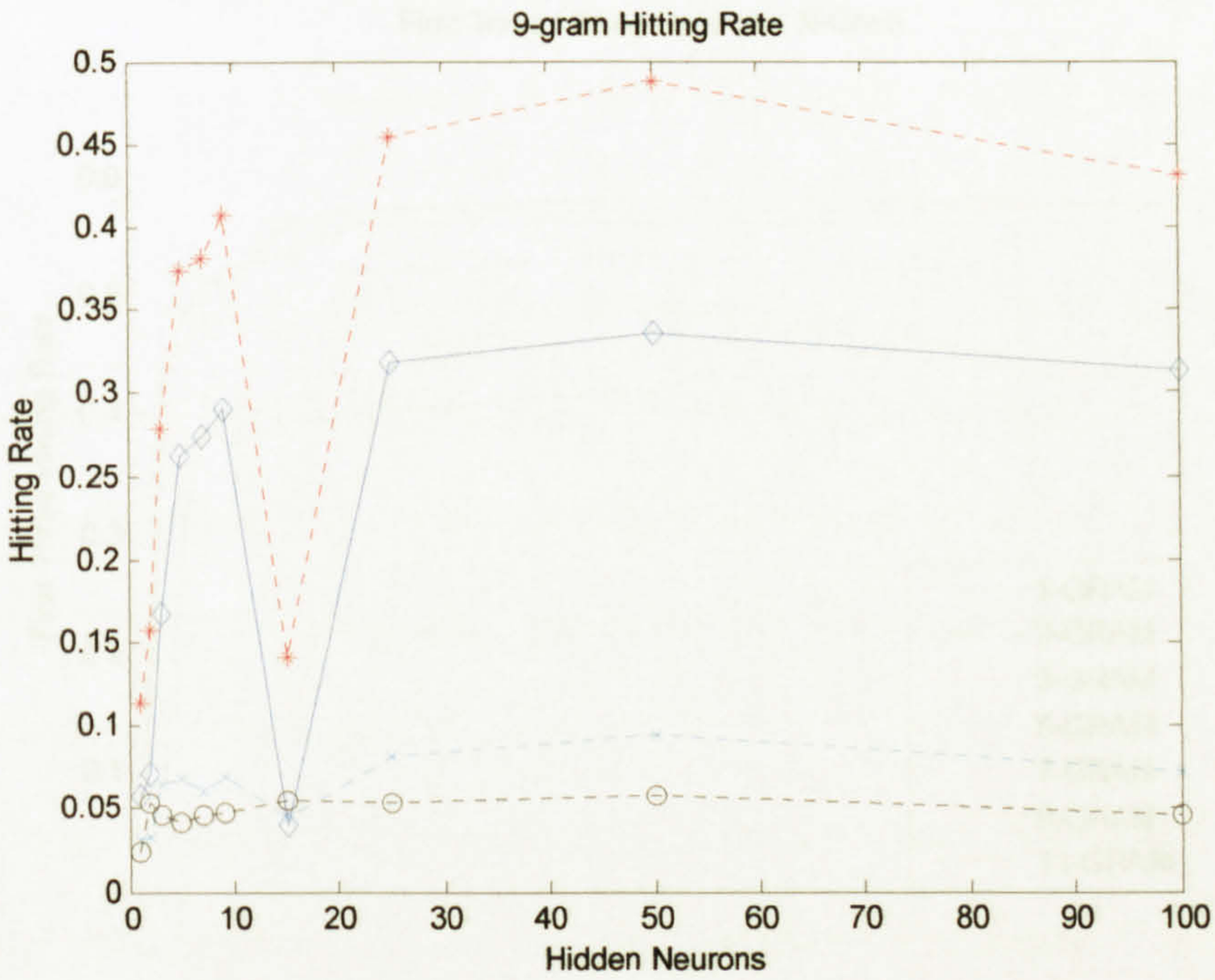


Figure 4.10 One & nine-gram Hitting Rate curves

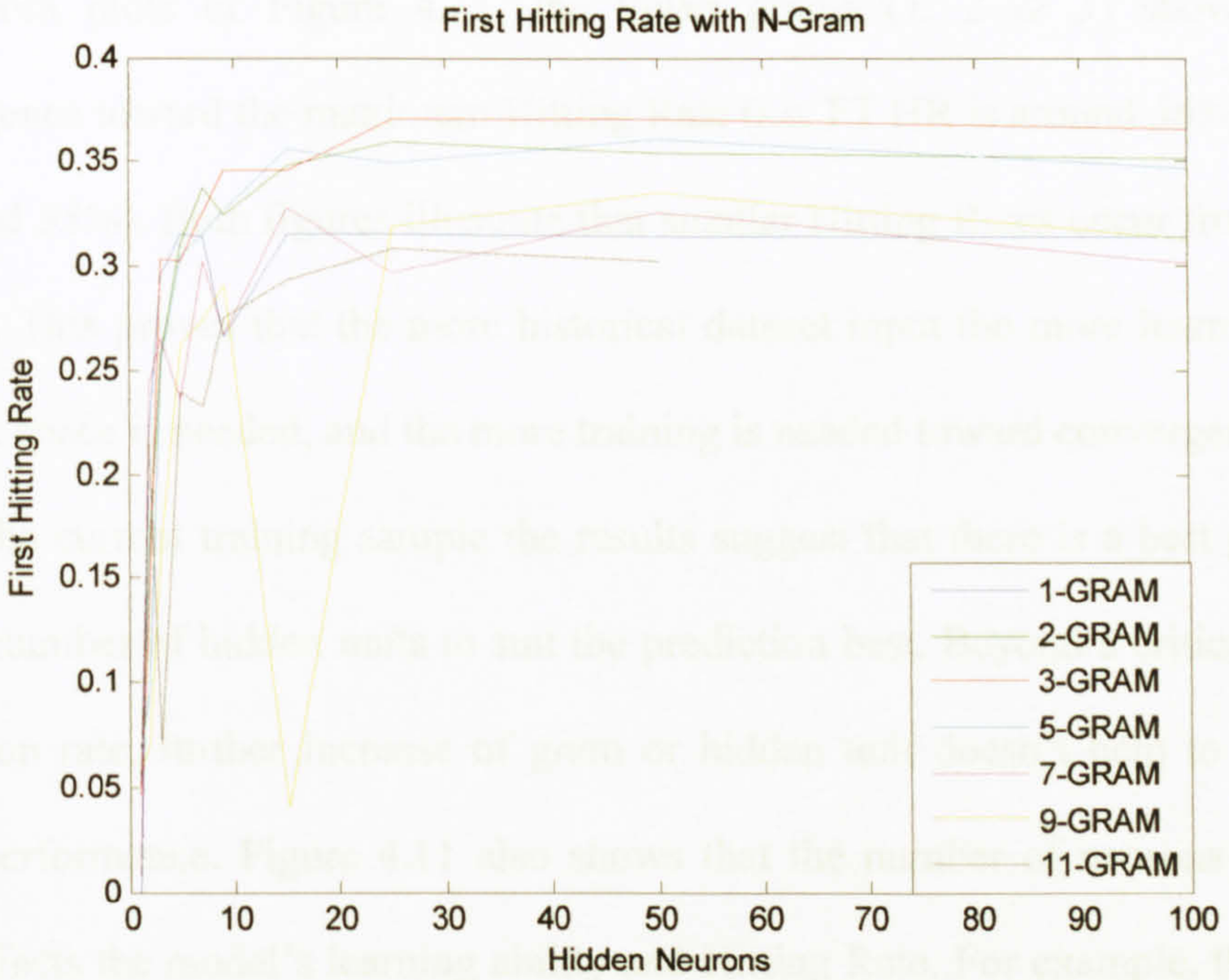


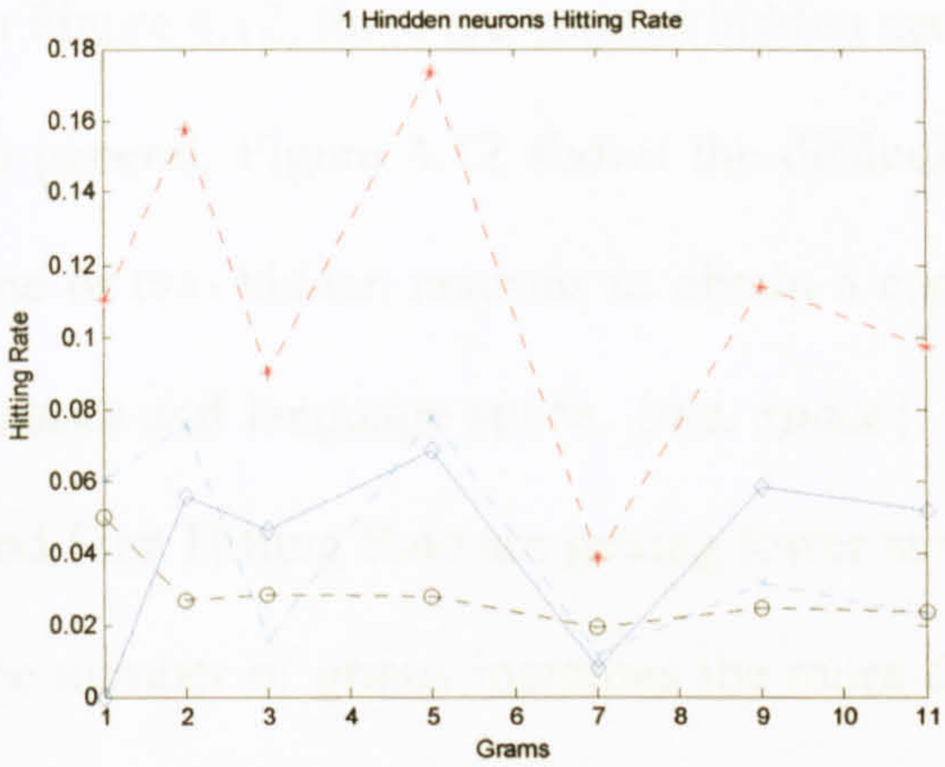
Figure 4.11 N-gram First and First Three Hitting Rate curves

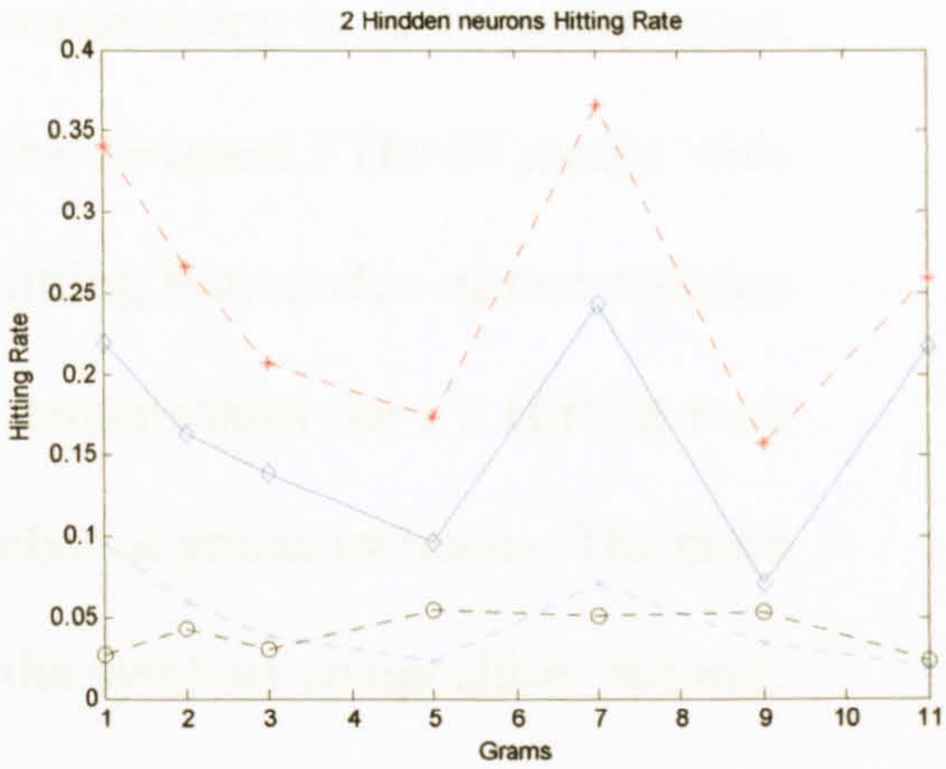
From both plots of Figure 4.11, the lower grams (*1, 2 & 3*) show a better convergence toward the maximum Hitting Rate (i.e. FT HR is around *56%*, First HR is around *33%*). Both figures illustrate that smaller Hitting Rates occur from *4*-gram onward. This proves that the more historical dataset input the more learning neural network space is needed, and the more training is needed toward convergence.

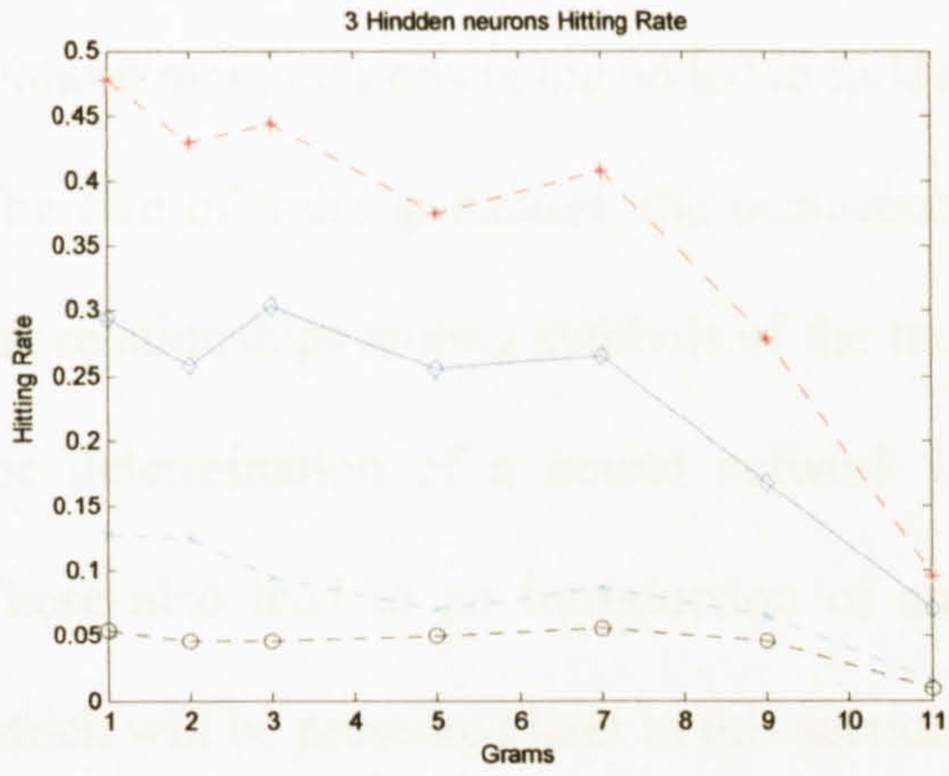
Under the current training sample the results suggest that there is a best gram with certain number of hidden units to suit the prediction best. Beyond a critical point of prediction rate, further increase of gram or hidden unit doesn't help to achieve a better performance. Figure 4.11 also shows that the number of neurons in hidden layer affects the model's learning ability and Hitting Rate. For example, the number of neurons in hidden layer should not be too small to a structured symbol set {*a, ... , z, space*} distribution; otherwise, it would be difficult for neural network to reach a good hitting rate.

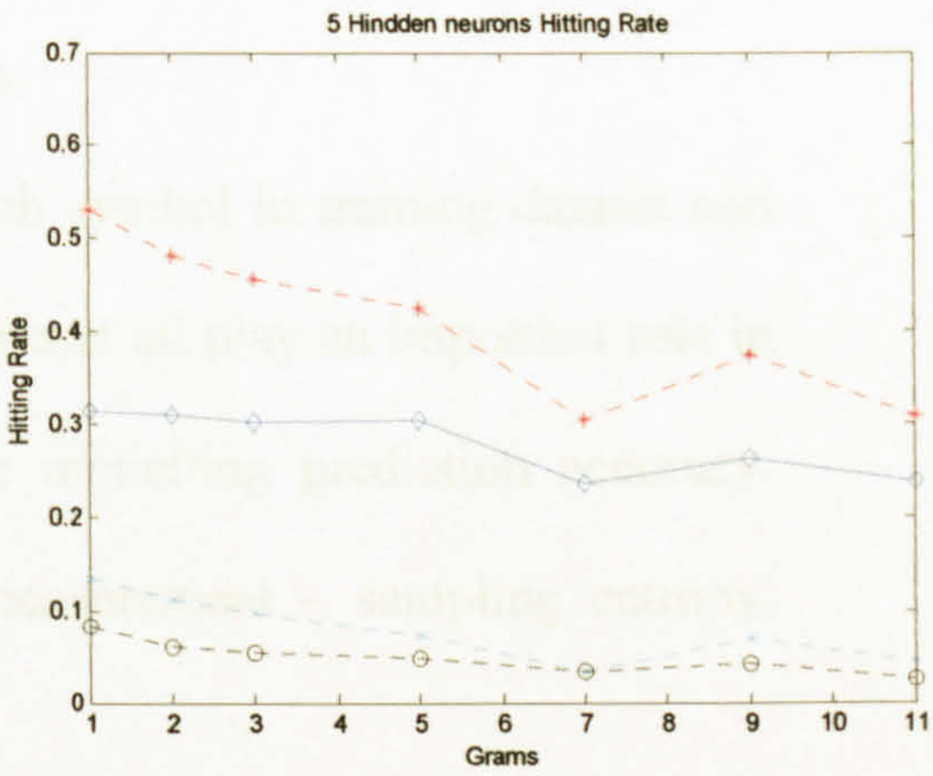
Looking at the black curve generated in Figure 4.11, the *11*-gram testing stops at fifty neurons; a *27-100-27* three layer FTDNN model has failed to complete the training process under the current system environment as it runs out of memory error. The hierarchy hitting rates generated by this model can be used by the prediction ranking functions.

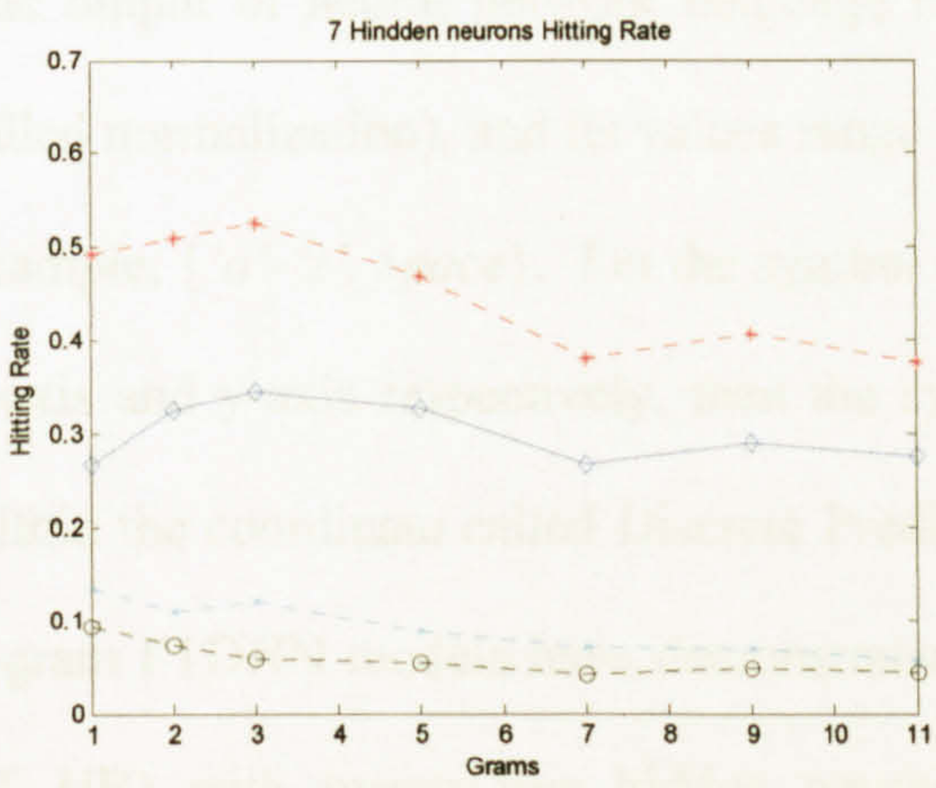
Another type of plots are produced in order to illustrate the variation with increasing grams and fixed number of hidden neurons. The plots are shown in Figure 4.12. The testing results of full Hidden neurons vector [1, 2, 3, 5, 7, 9, 15] with larger figures is displayed in Appendix D.











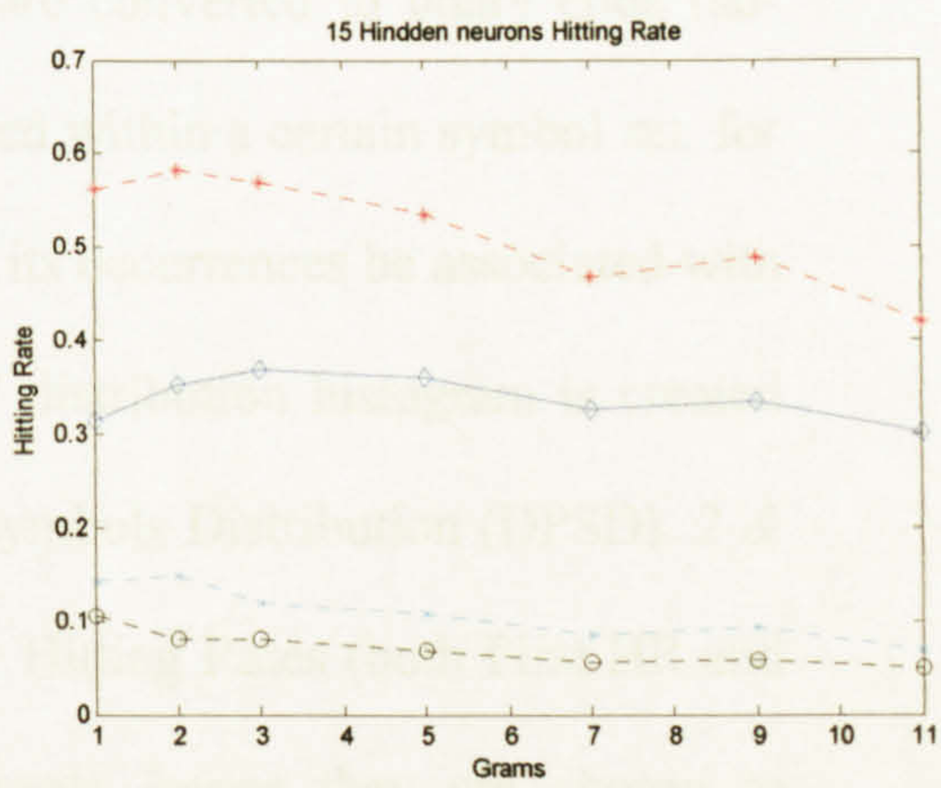


Figure 4.12 [1, 2, 3, 5, 7, 9, 15] hidden neurons Hitting Rate curves

In Figure 4.12, there are various hidden neurons corresponding to each specific gram. In general, Figure 4.12 shows the difficulty for the designed FTDNN model with one or two hidden neurons to obtain a constant Hitting Rate under current training dataset and language space, {*a-z, space*}. It also demonstrates the FT Hitting Rate and First Hitting Rate are getting lower as the number of grams increases. The more the number of grams increases the more diverse the symbols composition become. Then it becomes more difficult for a neural network language model to learn fully without more neurons being added in hidden layers.

The size of training dataset, the occurrence of each symbol in training dataset and the relationships among symbols of the training dataset all play an important role in the determination of a neural network language modelling prediction accuracy. These also lead to an introduction of another measurement – sampling entropy, which will be presented later in this section.

The output of neural network language models are converted to unary code (so-called normalization), and its values range is limited within a certain symbol set, for example, {*'a'-'z', space*}. Let the symbol set and its occurrences be associated with x-axis and y-axis respectively, then the symbols' distribution histogram is created within the coordinate called Discrete Prediction Symbols Distribution (DPSD). *2 & 3*-gram FTDNN models have demonstrated better Hitting Rates (both First HR and FT HR) with twenty-five hidden neurons onwards, hence they are chosen to generate the Discrete Prediction Symbols Distribution under First Hitting Rate.

Figure 4.13 shows the Discrete Prediction Symbols Distribution, which displays a comparison between prediction symbols occurrences and target symbols occurrences.

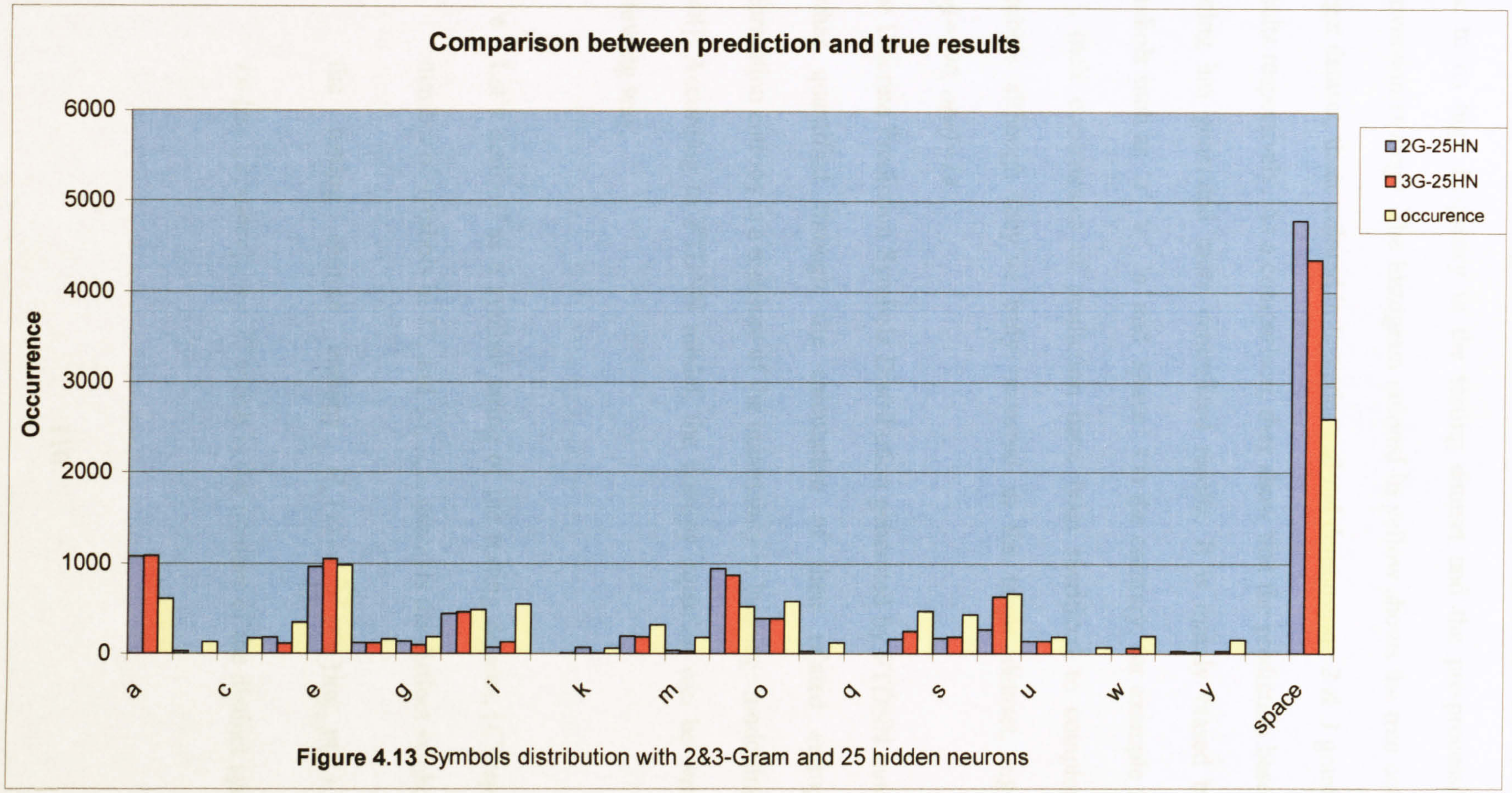


Figure 4.13 Symbols distribution with 2&3-Gram and 25 hidden neurons

In total, *10k* symbols have been tested and predicted. From Figure 4.13, symbol 'space' has shown the highest occurrence under all circumstances, which is mainly due to its high frequency in the testing dataset and the pre-processing symbol conversion strategy. The histogram colored in yellow shows the true occurrence of target dataset; those colored in lavender and red demonstrate *2 & 3* gram prediction results respectively. As a comparison, they show that the prediction based on unary coding has generated more centralized results. It is mainly biased towards the symbols such as *'a'*, *'e'*, *'n'* and *'space'*. On the contrary, for example for *'u'*, *'v'*, *'w'*, their occurrences of prediction have been sacrificed to complement other symbols although they've truly occurred in the target dataset, e.g. *o(u)=72, o(v)=196, o(w)=14.*

The Discrete Prediction Symbols Distribution generated by FTDNN models can be further quantified through the calculation of their related entropy values. Information entropy is a measure of the uncertainty in language modelling [Shannon, 1948]. According to FTDNN model, the entropy equation can be written in the following way,

- Let's define C as a symbol string of the testing dataset, $|C|$ represents the number of symbols in C, and $\{a,...,z,space\}$ is the distinct symbol set, then the testing sample entropy $H_{ts}(x) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)$, where $i \in \{a,...,z,space\}$, $n = 27$ which is the number of the distinct symbols, x_i is

the occurrence of symbol i in C and $p(x_i)$ is the probability occurred of x_i compared with $|C|$.

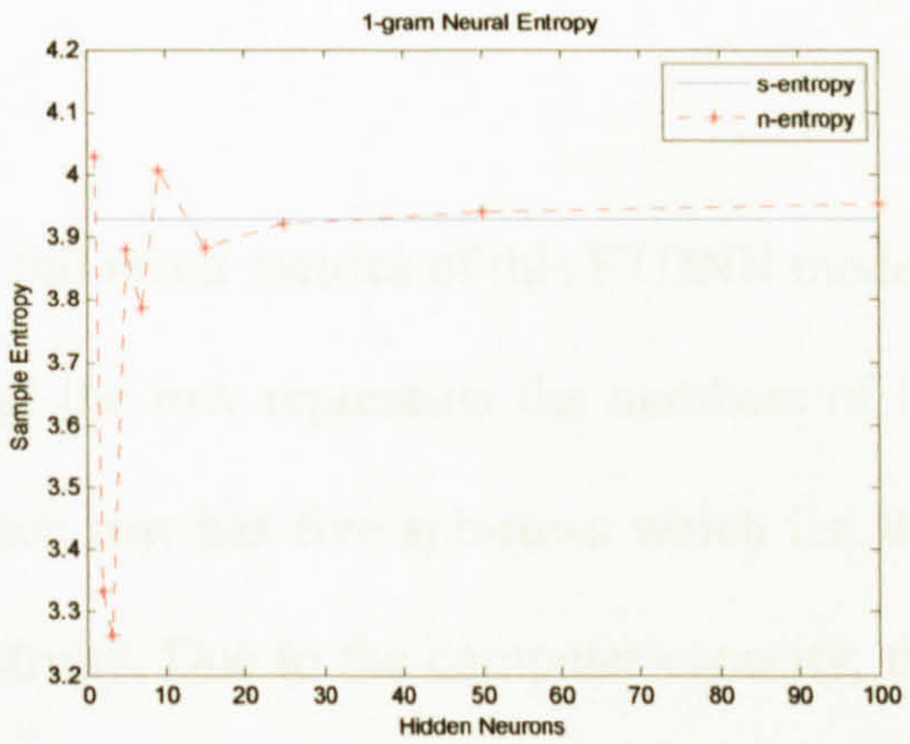
- Given a constant 'GRAM' which represents the number of grams used by FTDNN model, let's calculate its Neural Network-entropy (n-entropy) H_{nn} of the testing sample, $H_{nn}(y) = -\sum_{i=1}^{n} p(y_i)\log_2 p(y_i)$, where the probability of the sum of the output neuron i is $y_i = (\sum_{j=1}^{m} a_{ij})/Y$, $m = |C| - GRAM$, a_{ij} is the j th prediction conversion value of output neuron a_i, and the sum $Y = \sum_{i=1}^{n} y_i$.

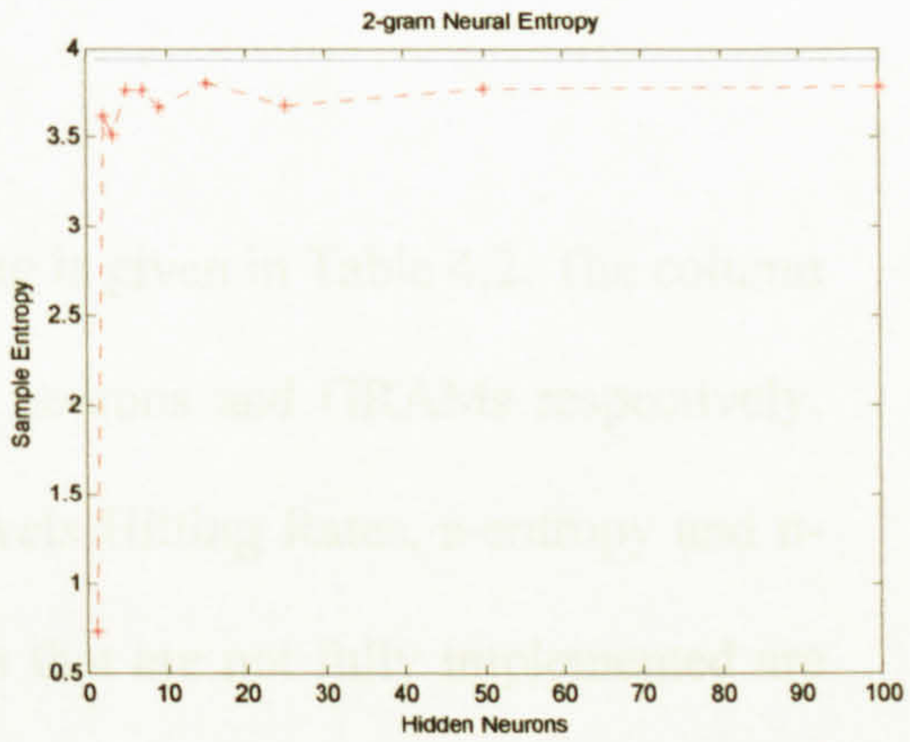
Here sample entropy (s-entropy) is defined as the entropy of testing dataset. Given a testing sample, the sample entropies are calculated according to the number of grams. If one considers a small gram range (e.g. < *11*), then those sample entropies only have a very small difference in comparison with the capacity of the testing sample which is valued as less than *0.001* and can be ignored.

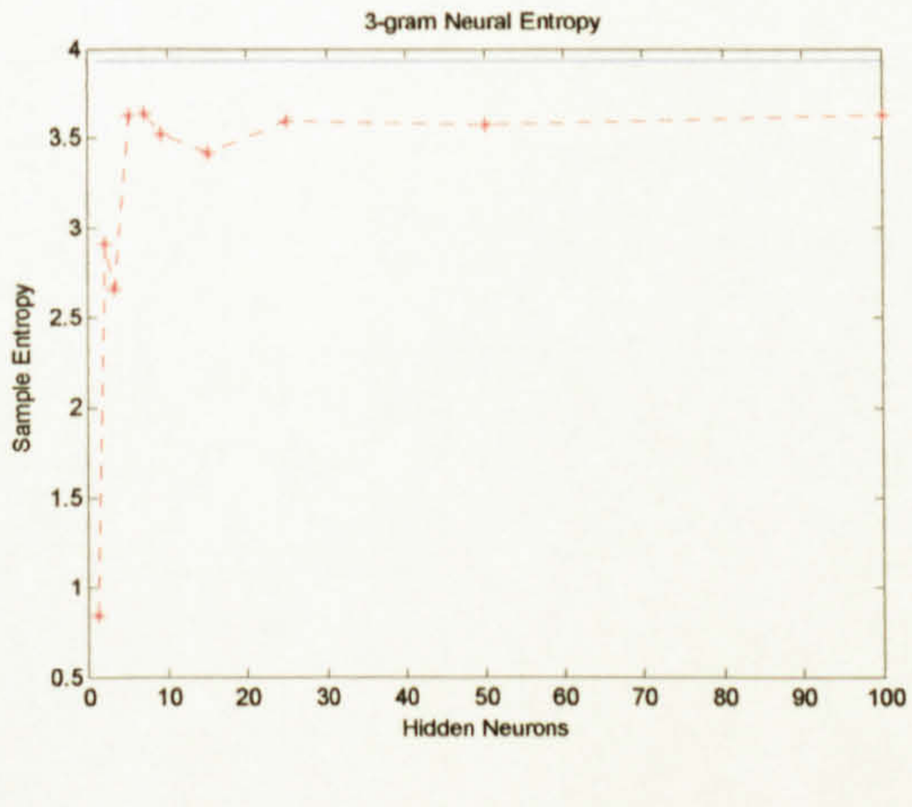
A comparison between s-entropy (in blue) and n-entropy values (in red) based on various grams and hidden neurons is shown in Figure 4.14. It indicates that *1*-gram n-entropy has gradually reached the same value as s-entropy with an increase of hidden neurons, and that *2 & 3*-gram are well below s-entropy. However, it becomes more difficult for *7 & 9*-gram to reach a stable value despite the dramatic increase of

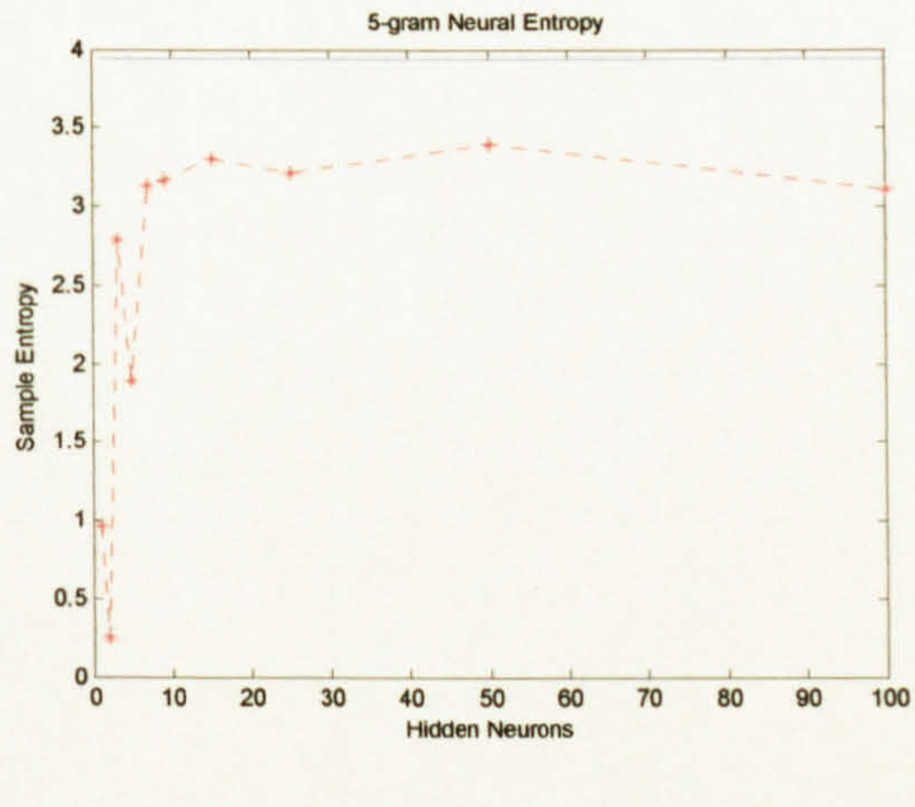
hidden neurons. On the other hand, these also demonstrate that *1, 2 & 3*-gram generate better results than other grams.

It would be interesting to use a larger size training dataset with a heuristic content selection strategy to estimate the natural entropy of English language. But this will require much more computational ability and memory capacity of a computer. The learning algorithm could also be required to adapt to a parallel computation algorithm for efficiency consideration.









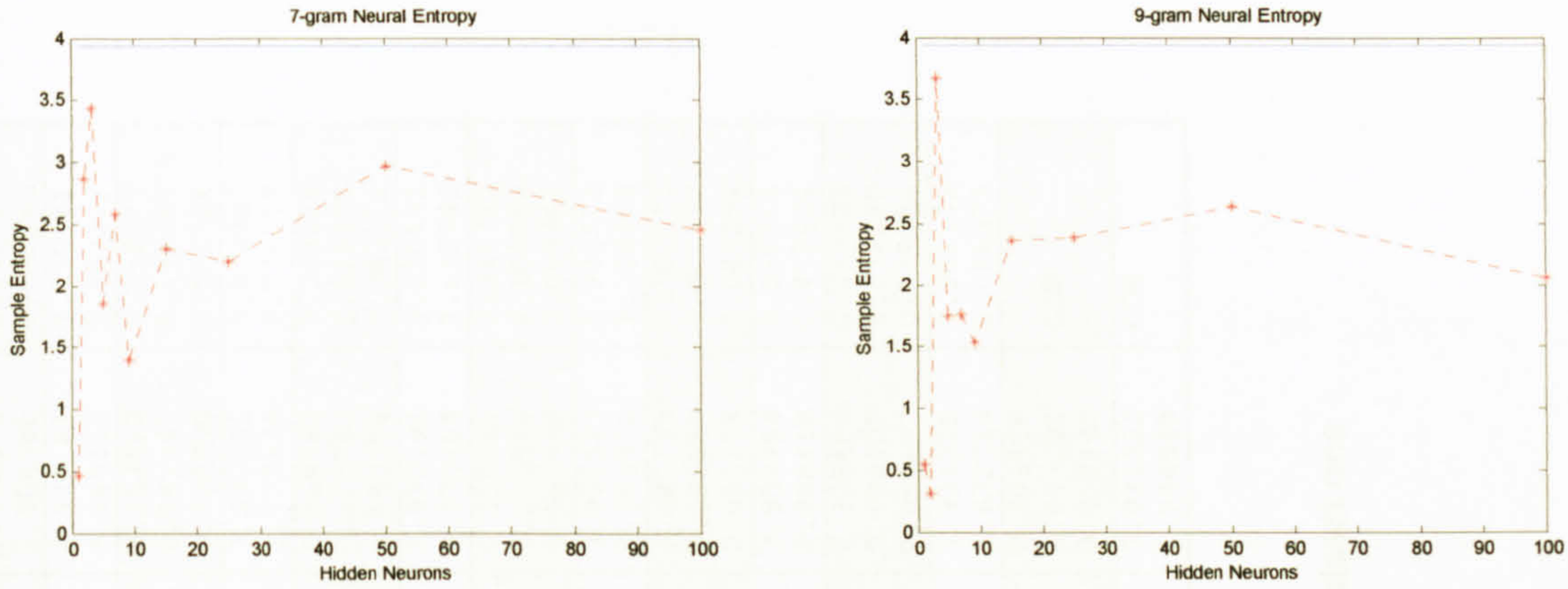


Figure 4.14 [1, 2, 3, 5, 7, 9] gram s-entropy curves

A full result metrics of this FTDNN model testing is given in Table 4.2. The column and the row represents the numbers of hidden neurons and GRAMs respectively. Each row has five sub-rows which list three levels Hitting Rates, s-entropy and n-entropy. Due to the computer capacity, the tests that are not fully implemented are marked as OOM (out of memory).

GRAM		HN. 1	HN. 2	HN. 3	HN. 5	HN. 7	HN. 9	HN. 15	HN. 25	HN. 50	HN. 100
1	L1	0.000000	0.219119	0.294991	0.314715	0.314618	0.266087	0.314715	0.314715	0.314618	0.314618
	L2	0.060443	0.095401	0.127820	0.134167	0.130749	0.133190	0.140221	0.140221	0.142076	0.142076
	L3	0.049897	0.026560	0.055073	0.083000	0.092374	0.093350	0.104482	0.107704	0.104580	0.105166
	E1	3.930181	3.930181	3.930181	3.930181	3.930181	3.930181	3.930181	3.930181	3.930181	3.930181
	E2	4.028716	3.332937	3.262079	3.878624	3.786988	4.005435	3.881985	3.921311	3.940613	3.953522
2	L1	0.055762	0.163574	0.257813	0.310840	0.337891	0.324023	0.347949	0.359375	0.353613	0.350977
	L2	0.075293	0.059473	0.125586	0.111523	0.117773	0.110352	0.140527	0.138184	0.147461	0.152051
	L3	0.026465	0.042871	0.045605	0.060254	0.060840	0.073340	0.077441	0.077148	0.080859	0.074316
	E1	3.930001	3.930001	3.930001	3.930001	3.930001	3.930001	3.930001	3.930001	3.930001	3.930001
	E2	0.732437	3.616409	3.509968	3.759661	3.755713	3.664871	3.804349	3.679014	3.772426	3.776077
3	L1	0.045903	0.138490	0.303350	0.302569	0.325715	0.345151	0.346128	0.370446	0.369763	0.365172
	L2	0.015627	0.038383	0.094443	0.098056	0.103233	0.120715	0.117004	0.124329	0.118762	0.128431
	L3	0.028421	0.030081	0.045708	0.054009	0.068561	0.059283	0.073249	0.077156	0.080574	0.083211
	E1	3.930191	3.930191	3.930191	3.930191	3.930191	3.930191	3.930191	3.930191	3.930191	3.930191
	E2	0.844693	2.914490	2.663493	3.626991	3.634807	3.523757	3.426460	3.591218	3.570806	3.630942
5	L1	0.068086	0.096513	0.254762	0.304874	0.314643	0.327440	0.355768	0.349516	0.360848	0.346781
	L2	0.077757	0.022272	0.070724	0.072287	0.090945	0.087135	0.108723	0.105695	0.106574	0.100908
	L3	0.027645	0.054410	0.049722	0.047279	0.060369	0.054117	0.064765	0.065937	0.067598	0.059392
	E1	3.930573	3.930573	3.930573	3.930573	3.930573	3.930573	3.930573	3.930573	3.930573	3.930573
	E2	0.961402	0.249557	2.779414	1.887143	3.132063	3.159051	3.309605	3.210364	3.393972	3.106408
7	L1	0.007621	0.243087	0.264778	0.235955	0.302589	0.267416	0.327504	0.295945	0.325745	0.300733
	L2	0.011236	0.071324	0.087152	0.034783	0.074939	0.070151	0.065071	0.084123	0.085589	0.077577
	L3	0.019248	0.050611	0.055691	0.034001	0.044846	0.042404	0.036932	0.060186	0.055594	0.047191
	E1	3.930954	3.930954	3.930954	3.930954	3.930954	3.930954	3.930954	3.930954	3.930954	3.930954
	E2	0.459946	2.864593	3.437550	1.852380	2.577396	1.393357	2.300555	2.197835	2.968418	2.452404
9	L1	0.057950	0.071338	0.167595	0.262973	0.273722	0.290433	0.040164	0.318382	0.335483	0.313105
	L2	0.031369	0.033421	0.065474	0.068504	0.061272	0.068992	0.045148	0.081501	0.094107	0.072120
	L3	0.024528	0.052868	0.045343	0.042119	0.046418	0.047884	0.055214	0.054041	0.057657	0.045734
	E1	3.931335	3.931335	3.931335	3.931335	3.931335	3.931335	3.931335	3.931335	3.931335	3.931335
	E2	0.548535	0.310431	3.670541	1.749527	1.758109	1.539157	2.354459	2.377240	2.632714	2.056379
11	L1	0.051119	0.217281	0.070472	0.238686	0.231747	0.275340	0.293520	0.307888	0.301926	
	L2	0.022383	0.019646	0.014759	0.044766	0.097644	0.055322	0.059525	0.060600	0.069201	OOM
	L3	0.023458	0.022578	0.009970	0.024729	0.063044	0.044473	0.040172	0.036653	0.050435	
	E1	3.931332	3.931332	3.931332	3.931332	3.931332	3.931332	3.931332	3.931332	3.931332	OOM
	E2	0.188492	0.080103	1.918713	1.685054	1.367051	1.337522	1.830799	1.874695	1.946945	

HN – No. of Hidden Neurons.
L1 – Hitting Rate Level 1.
E1 – Entropy calculated from sample .
E2 – Entropy calculated from Neural Network.
OOM – Out of Memory.

Table 4.2 Experimental results of FTDNN modelling with dataset one

4.7.2 N-Gram Prediction with noise

To test FTDNN neural network models' prediction ability, a noisy randomization method within the data pre-processing function is designed and applied to the training dataset and testing dataset. Generally speaking, some noises are randomly distributed into the dataset one after data unary code conversion.

In practice, an *iRand (n, m)* function is developed, where *n* is the range of random values, *m* is the required number of noisy symbols. Then, the noise rate or mistakes rate can be expressed as $mRate = m/|C|$ where $|C|$ is the number of symbols in C. During the training and testing, first, *iRand(*$|C|$*, m)* is applied to locate the symbols that need to be randomized, then, after resetting the status of random generator, function *iRand(27, m)* is used to assign each symbol located by *iRand(*$|C|$*, m)* to a new value. The process is illustrated below.

NoiseSymbolArraySequenceNumber[m] = iRand($|C|$*, m)*

Set the state of random generator

NoiseSymbolArray[m]= iRand(27, m)

For zero to m

Do

x = NoiseSymbolArraySequenceNumber[m]

y= NoiseSymbolArray[m]

set dataset1[x] to zeros

dataset1[x][y] = 1

End for

For example, given $m = 1$ and unary code dataset below,

```
%a b c d e f g h i j k l m n o p q r s t u v w x y z ''
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    %'h'
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    %'e'
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0    %'y'
```

then the number of dataset's symbols $|C| = 3$. If *NoiseSymbolArraySequenceNumber[1],* that is, *iRand(3, 1) = 2* and *NoiseSymbolArray[1],* that is, *iRand(27, 1) = 26*, then, the dataset is randomized into,

```
%a b c d e f g h i j k l m n o p q r s t u v w x y z ''
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    %'h'
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0    %'z'
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0    %'y'
```

According to this randomization strategy, several FTDNN models with *2, 3, 5 & 7*-gram and extendable hidden units to study the noise prediction based on dataset one are developed. All the training, validation and testing data are preprocessed and mixed with randomized noisy data. Considering correction as a special case of prediction, the FTDNN modelling process is a symbol prediction as well as noise correction process tracing back n-gram history.

Let's define the noise rate range as $[\alpha, \beta]$, and the lower boundary and upper boundary used in this test as $\alpha = 0.001$ and $\beta = 0.1$ respectively. Three concrete values $mr_i\,(i = 1,2,3)$, namely, *0.001, 0.01* and *0.1* are used in this test, and the numbers of randomized symbols can be computed based on the equation, $m_i = mr_i * |C|$. Based on the same FTDNN structure, the First Hitting Rate and FT Hitting Rate with various grams and hidden neurons are shown in Figure 4.15.

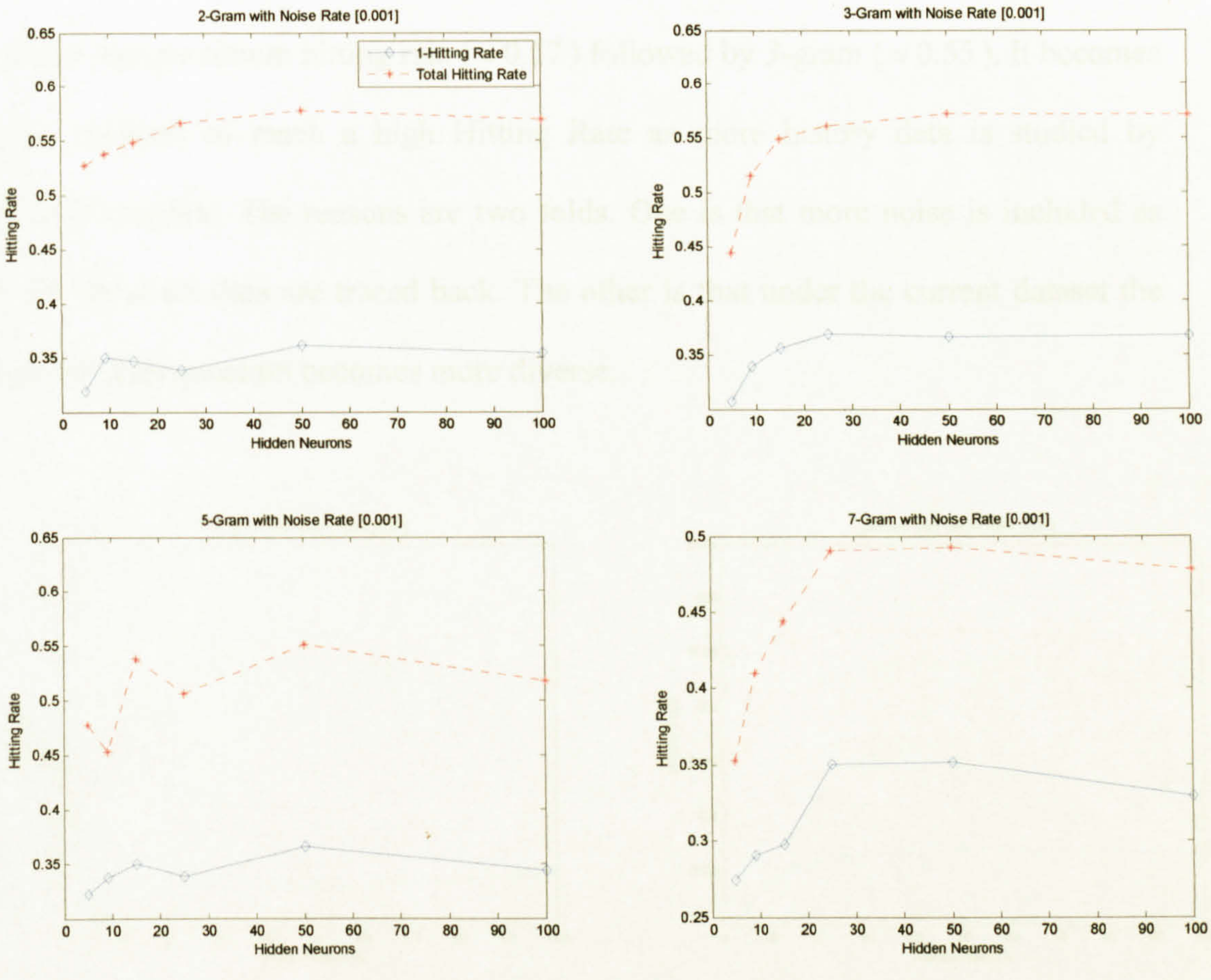
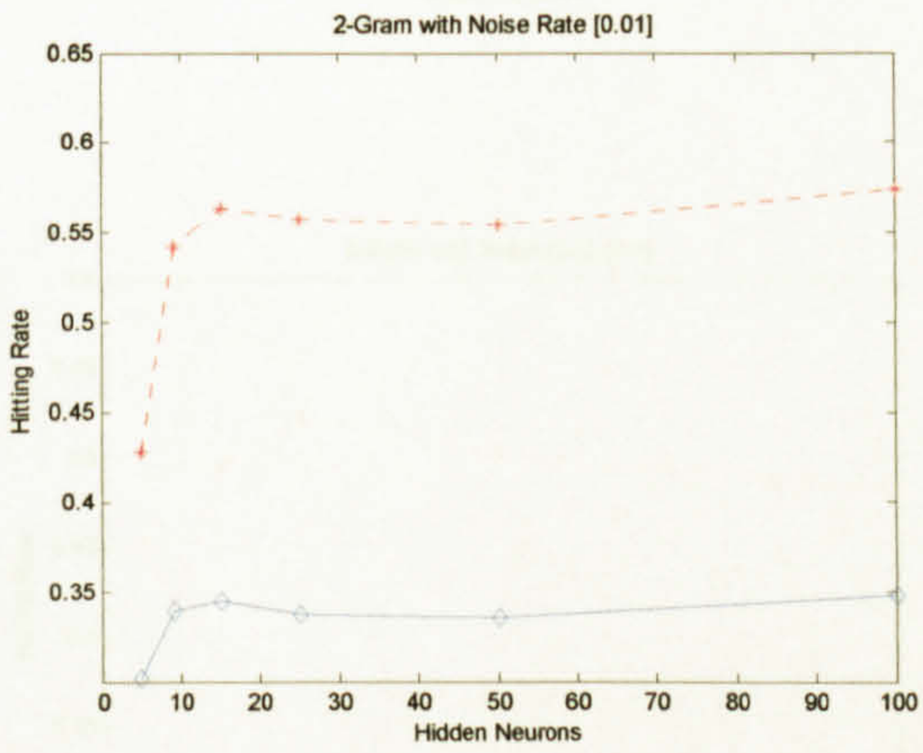
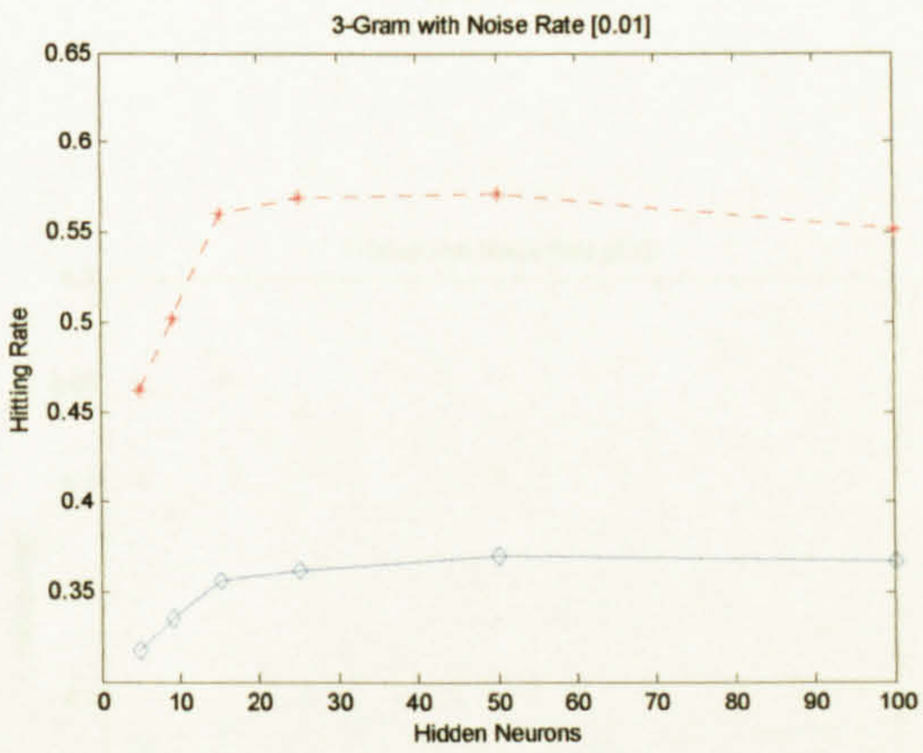


Figure 4.15 N-gram prediction with Noise Rate = 0.001

The blue curve represents the First Hitting Rate and the red one represents the FT Hitting Rate. The *2 & 3*-gram show the best FT Hitting Rate with fifty neurons under Noise Rate of *0.001*. Their Hitting Rates tend to change less as the number of hidden neurons grows. On the contrary, it becomes difficult for *5 & 7*-gram to reach a steady value. All four figures show that the significant increase of hidden neurons (from fifty neurons onward) does not improve the prediction rates significantly.

The plots of the *n*-gram predictions under noise rate of *0.01* and *0.1* are also displayed in Figure 4.16 and Figure 4.17. Similarly to *0.001* noise rate, *2*-gram reaches the maximum hitting rate (≈ 0.57) followed by *3*-gram (≈ 0.55). It becomes more difficult to reach a high Hitting Rate as more history data is studied by FTDNN models. The reasons are two folds. One is that more noise is included as more historical data are traced back. The other is that under the current dataset the symbol determination becomes more diverse.





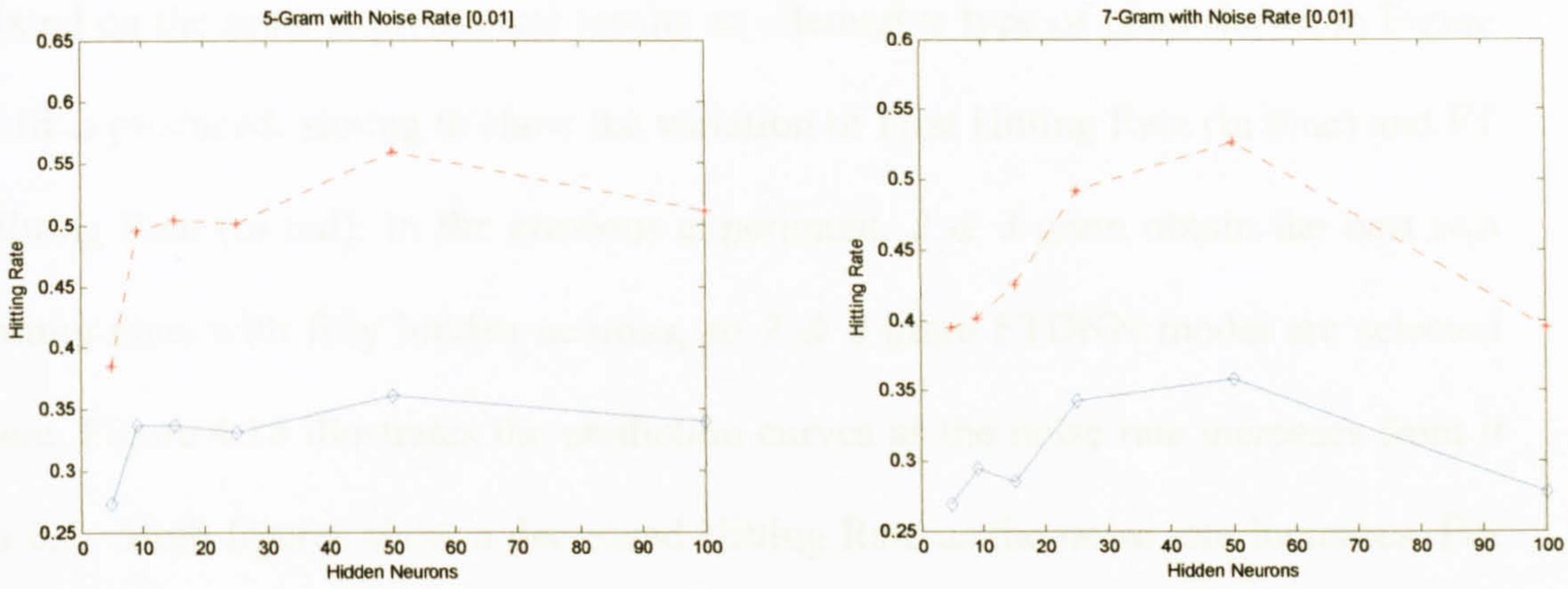
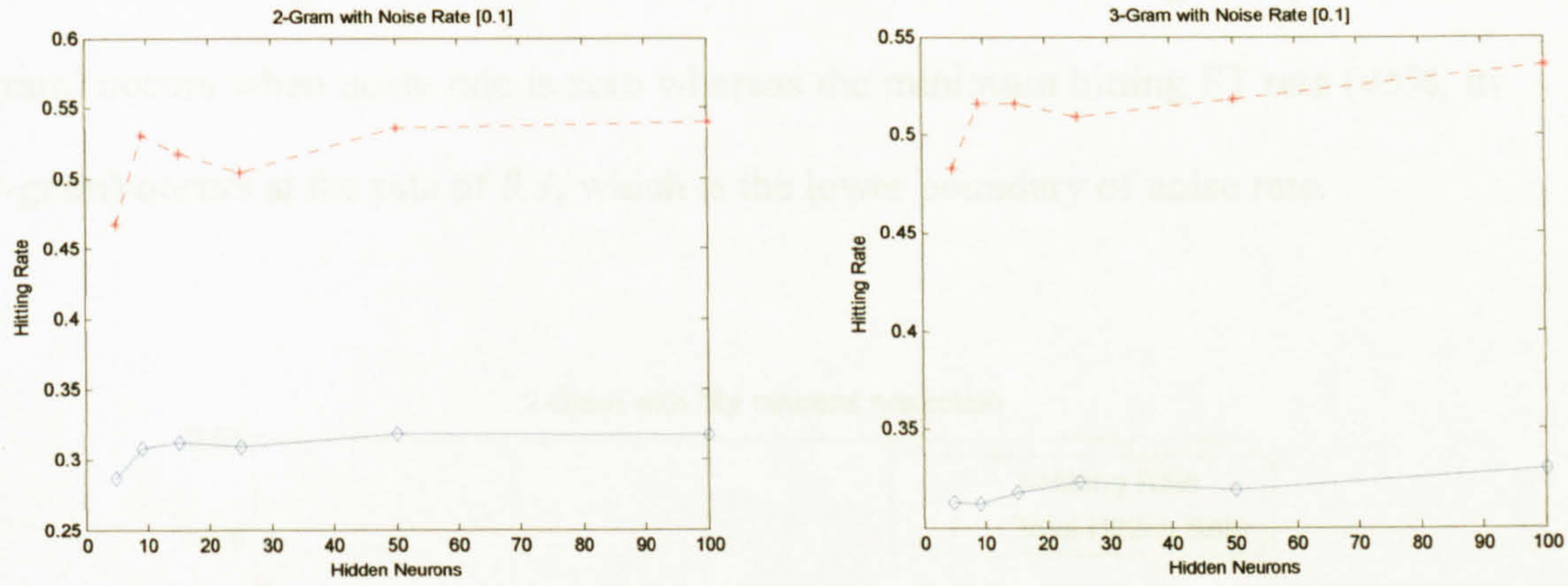


Figure 4.16 N-gram prediction with Noise Rate = 0.01



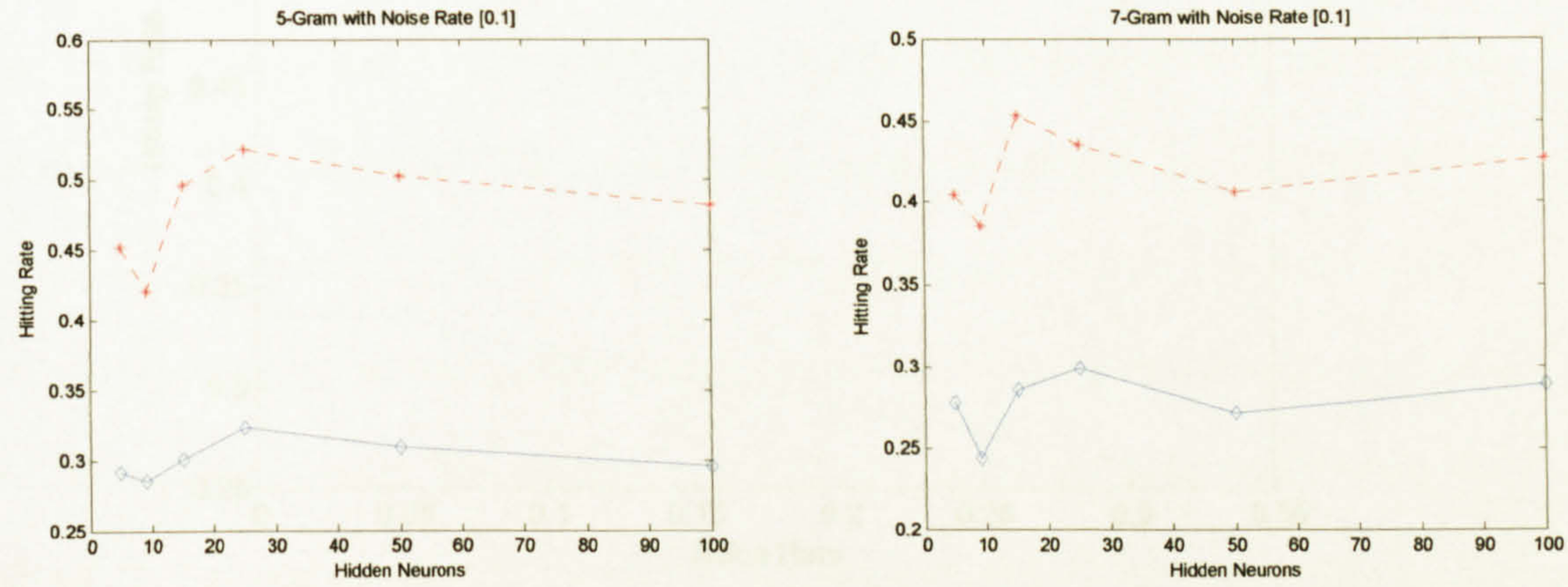
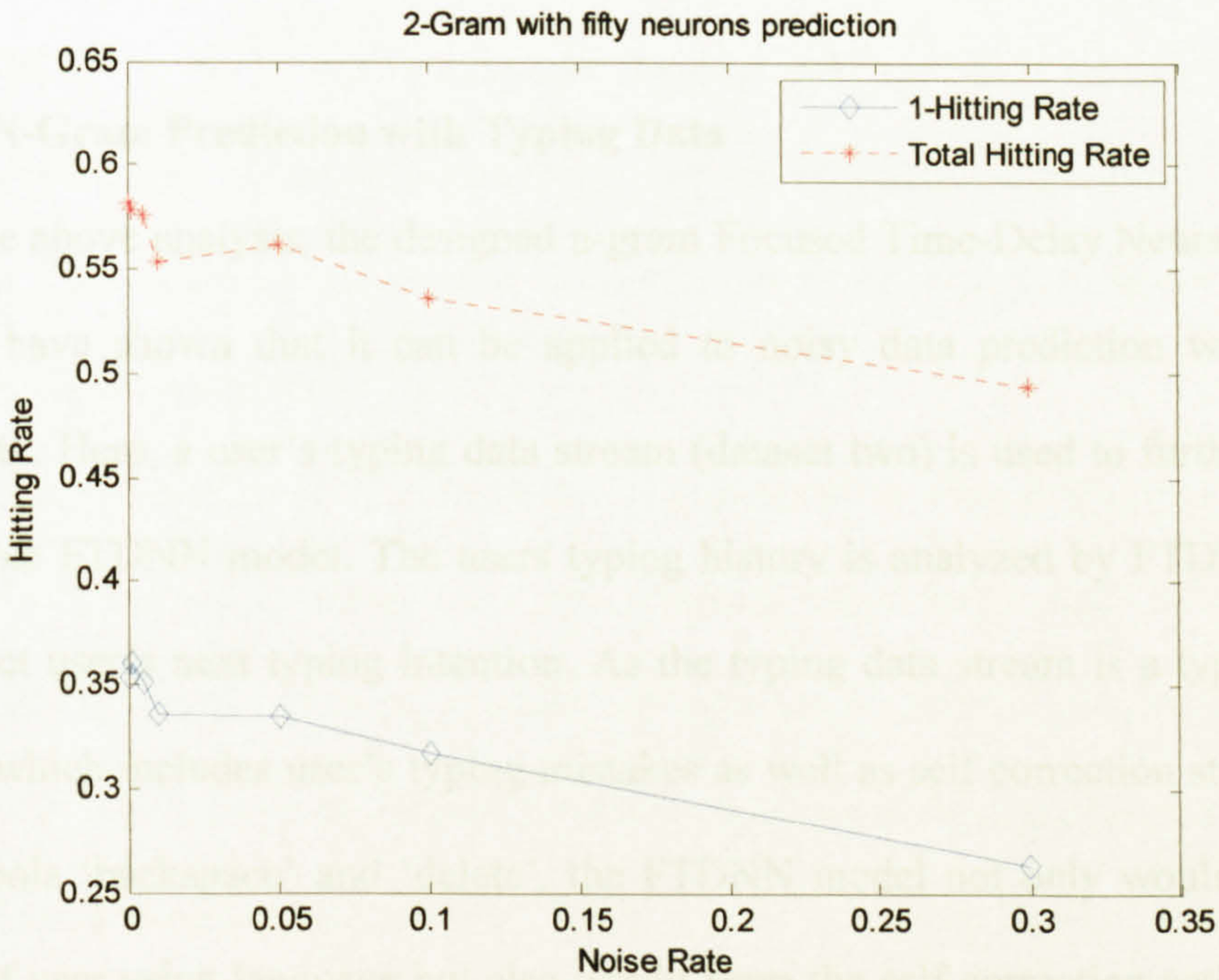


Figure 4.17 N-gram prediction with Noise Rate = 0.1

Based on the same experimental results an alternative type of plots shown in Figure 4.18 is produced, aiming to show the variation of First Hitting Rate (in blue) and FT Hitting Rate (in red). In the previous experiment, *2 & 3*-gram obtain the best two hitting rates with fifty hidden neurons, so *2 & 3*-gram FTDNN model are selected here. Figure 4.18 illustrates the prediction curves as the noise rate increases from *0* to *0.3*. Both figures show a decreased Hitting Rate as the noise rate increases. For example, when the noise rate reaches the value of approximately *30%*, its corresponding First Hitting Rate is only *27%* compared to the correction rate of *37%* without noise. Figure 4.18 indicates that the maximum FT hitting rate (*58%*, in *2*-gram) occurs when noise rate is zero whereas the minimum hitting FT rate (*45%*, in *3*-gram) occurs at the rate of *0.3*, which is the lower boundary of noise rate.



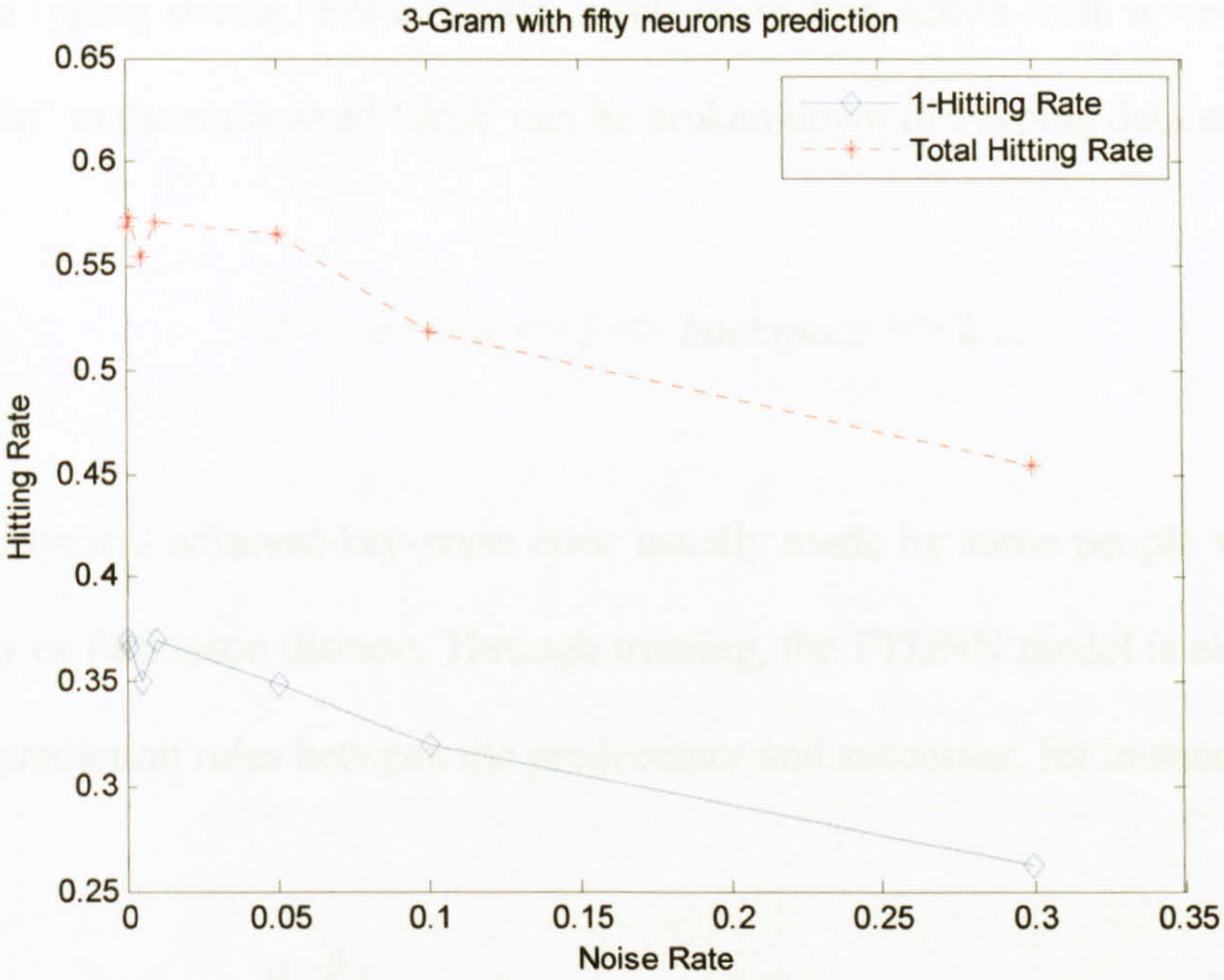


Figure 4.18 2 & 3-gram Hitting Rate curves under noise rates

4.7.3 N-Gram Prediction with Typing Data

From the above analysis, the designed n-gram Focused Time-Delay Neural Network models have shown that it can be applied to noisy data prediction with a high capability. Here, a user's typing data stream (dataset two) is used to further test the extendible FTDNN model. The users typing history is analyzed by FTDNN model to predict user's next typing intention. As the typing data stream is a typical noisy dataset which includes user's typing mistakes as well as self correction strokes such as symbols 'backspace' and 'delete', the FTDNN model not only would learn the habits of user using language but also would learn the self-correction actions which

occurs in typing stream. For example, a self-correction action from a wrong typing word '*desj*' to the right word '*desk*' can be broken down in a typing data stream as,

$$d => e => s => j => backspace => k \ldots$$

This is a typical adjacent-key-press error usually made by some people with motor disability or Parkinson disease. Through training, the FTDNN model is able to learn *2*-gram prediction rules between the predecessor and successor, for instance,

$$d \rightarrow e$$
$$e \rightarrow s$$

From the typing stream shown above, the model will learn not only the existing noises such as 's' → 'j', but also the correction actions such as 'j' → 'backspace'. In practice, users just continue their typing without stopping in spite of the possible mistakes. The model should be able to correct the mistakes automatically or specify recommendations later on.

The collected data stream in dataset two is expressed in Virtual Key Codes. In this research only editing virtual keys are adopted, other keys such as arrows are discarded. Then, the size of symbol set originally used by FTDNN model is extended into fifty three individual symbols, which apart from alphabet also include some other symbols such as,

VK_BACK => *BACKSPACE key*
VK_RETURN => *ENTER key*
VK_SHIFT => *SHIFT key*
VK_DELETE => *DEL key*

(*A full list with fifty three editing virtual key codes is illustrated in the Appendix F*). Based on the original design of FTDNN model, an extension to fifty-three units both at the input and output layer has been made. The dataset two has recorded both the key press 'down' status and 'up' status. Considering some disabled people specific typing behaviour such as prolonged key press which would generate more 'down' keys corresponding to one 'up' keys, the keystrokes with 'down' status are chosen by the pre-processing function for neural network training and testing. For example, a converted dataset sample from typing stream: 'j'→'backspace'→'k' after preprocessing can be presented as,

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ***%j***

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ***%backsapce***

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ***%k***

Owing to the previous experimental results, the test with *n*-gram array used by training and testing is *GRAMarray = [1, 3, 5, 7, 9]*; the hidden neurons array used is *HiddenNeuronsArray = [3, 5, 7, 9, 15, 25, 50, 100]*. Part of testing results is shown in Figure 4.19. (*Full detail can be found in Appendix D*).

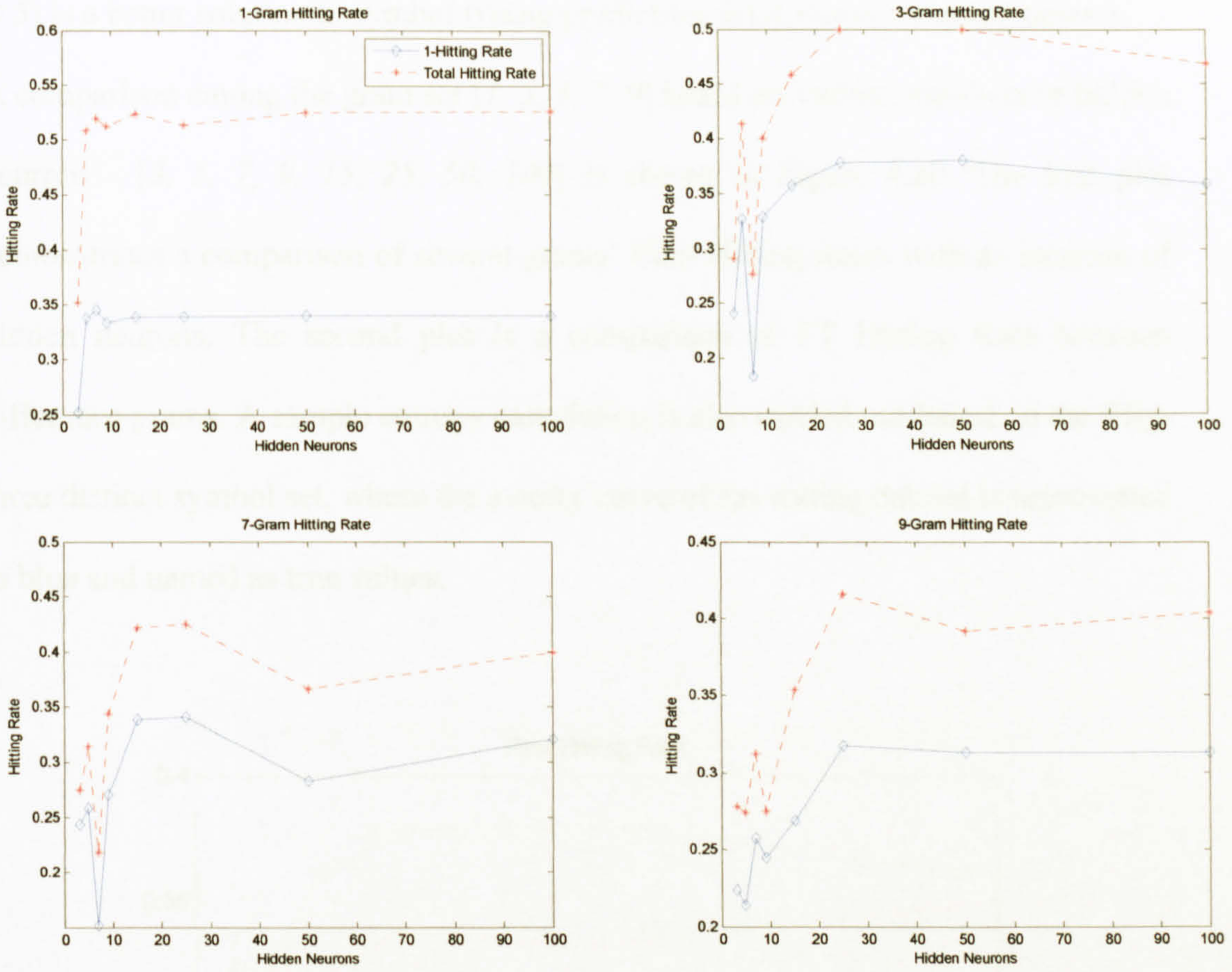
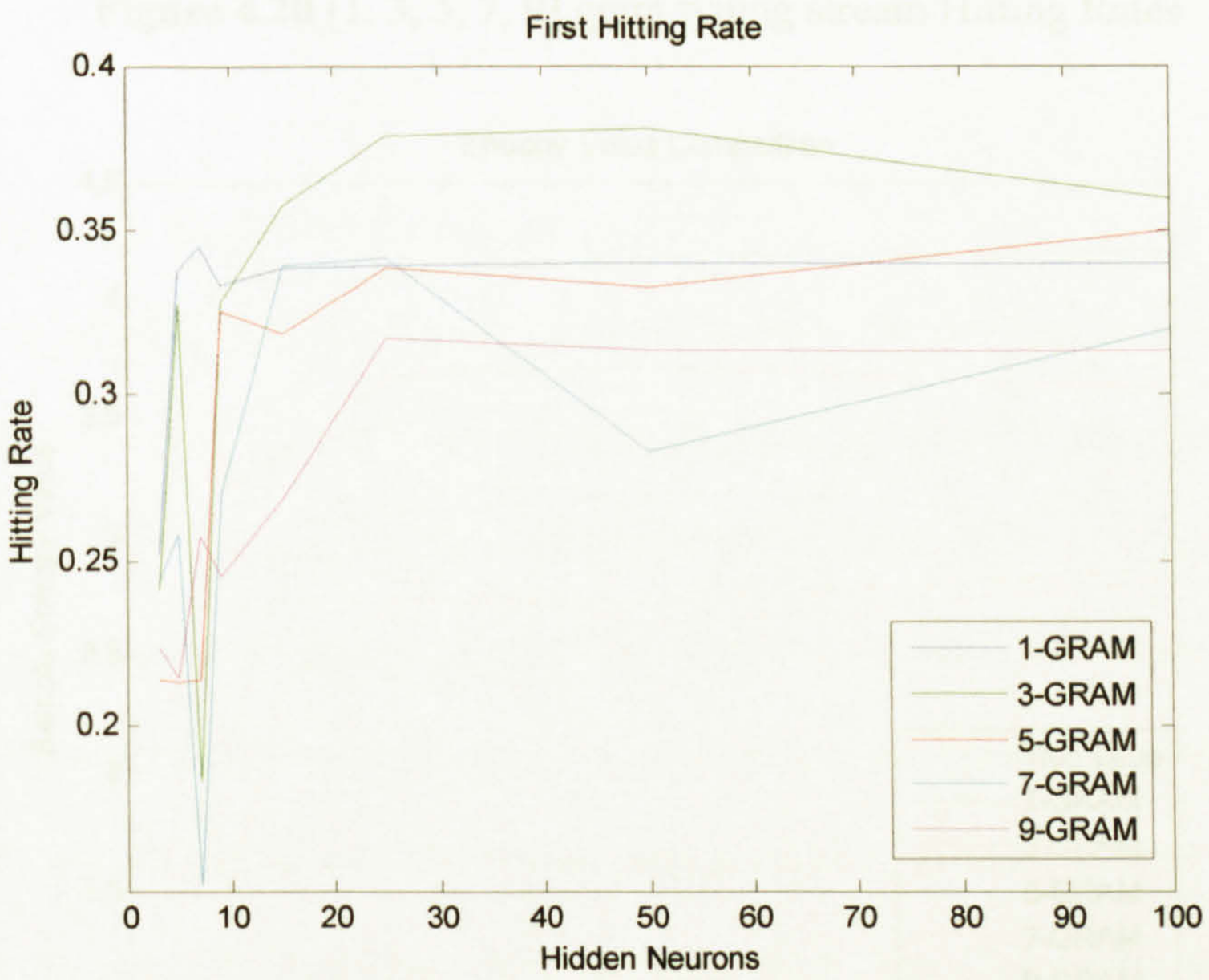


Figure 4.19 [1, 3, 7, 9] gram typing stream individual Hitting Rates

The figure illustrates the variation of First Hitting Rates and FT Hitting Rates with the increase of hidden neurons. In general, it shows that *1 & 3*-gram generate a better Hitting Rate than *7 & 9*-grams. The experiment indicates that from twenty five hidden neurons onwards, the First Hitting Rates and First Three Hitting Rate of *7 & 9*-gram are changing more sharply than the case with *1 & 3*-gram. It further

confirms the previous experimental results which show that a lower gram (e.g. gram < 5) is a better solution to symbol typing prediction using current training dataset.

A comparison among the gram set [*1, 3, 5, 7, 9*] based on various numbers of hidden neurons - [*3, 5, 7, 9, 15, 25, 50, 100*] is shown in Figure 4.20. The first plot demonstrates a comparison of several grams' First Hitting Rates with an increase of hidden neurons. The second plot is a comparison of FT Hitting Rate between difference grams. A sample entropy calculation is also carried out based on the fifty-three distinct symbol set, where the s-entry curve of the testing dataset is represented in blue and named as true values.



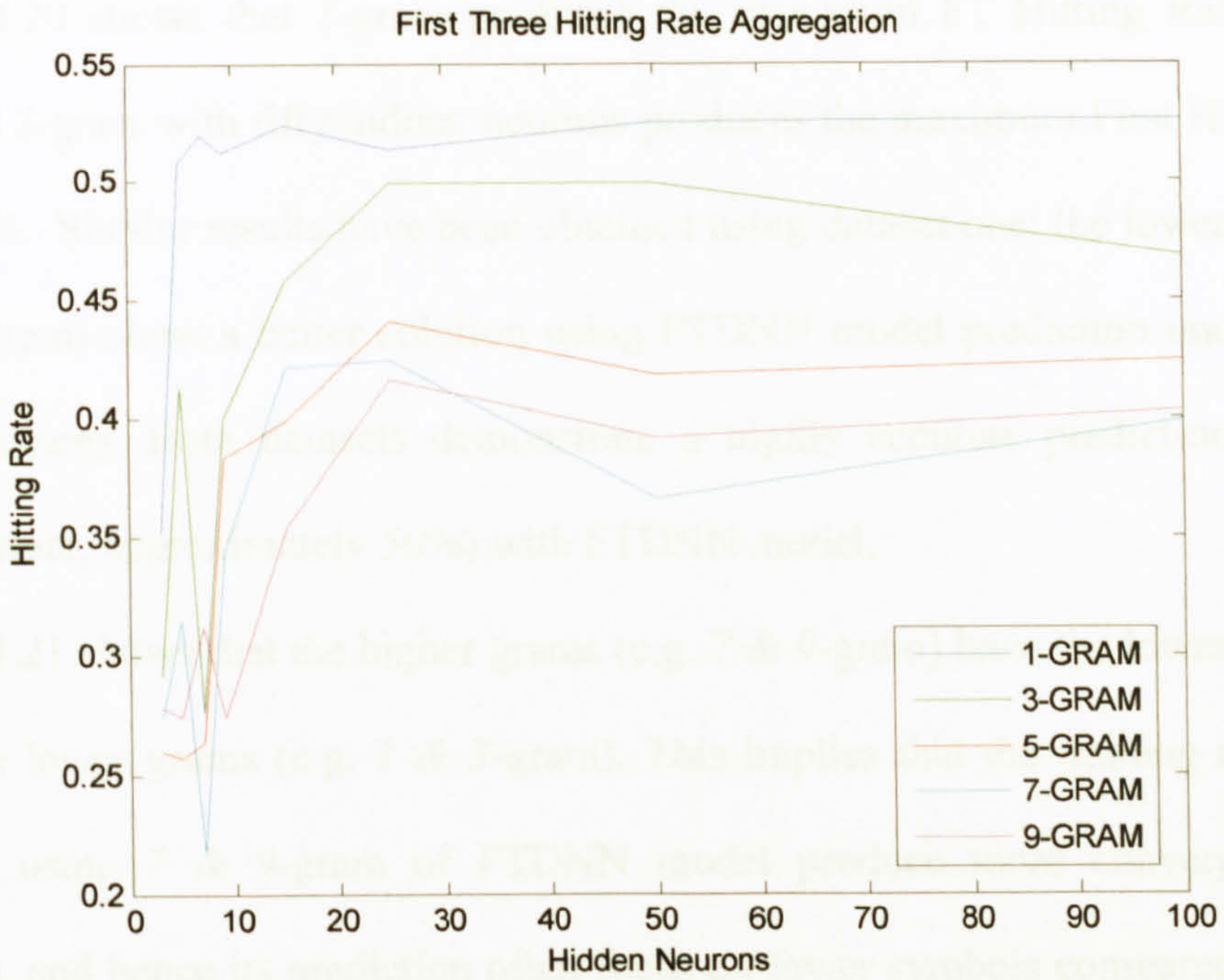


Figure 4.20 [1, 3, 5, 7, 9] gram typing stream Hitting Rates

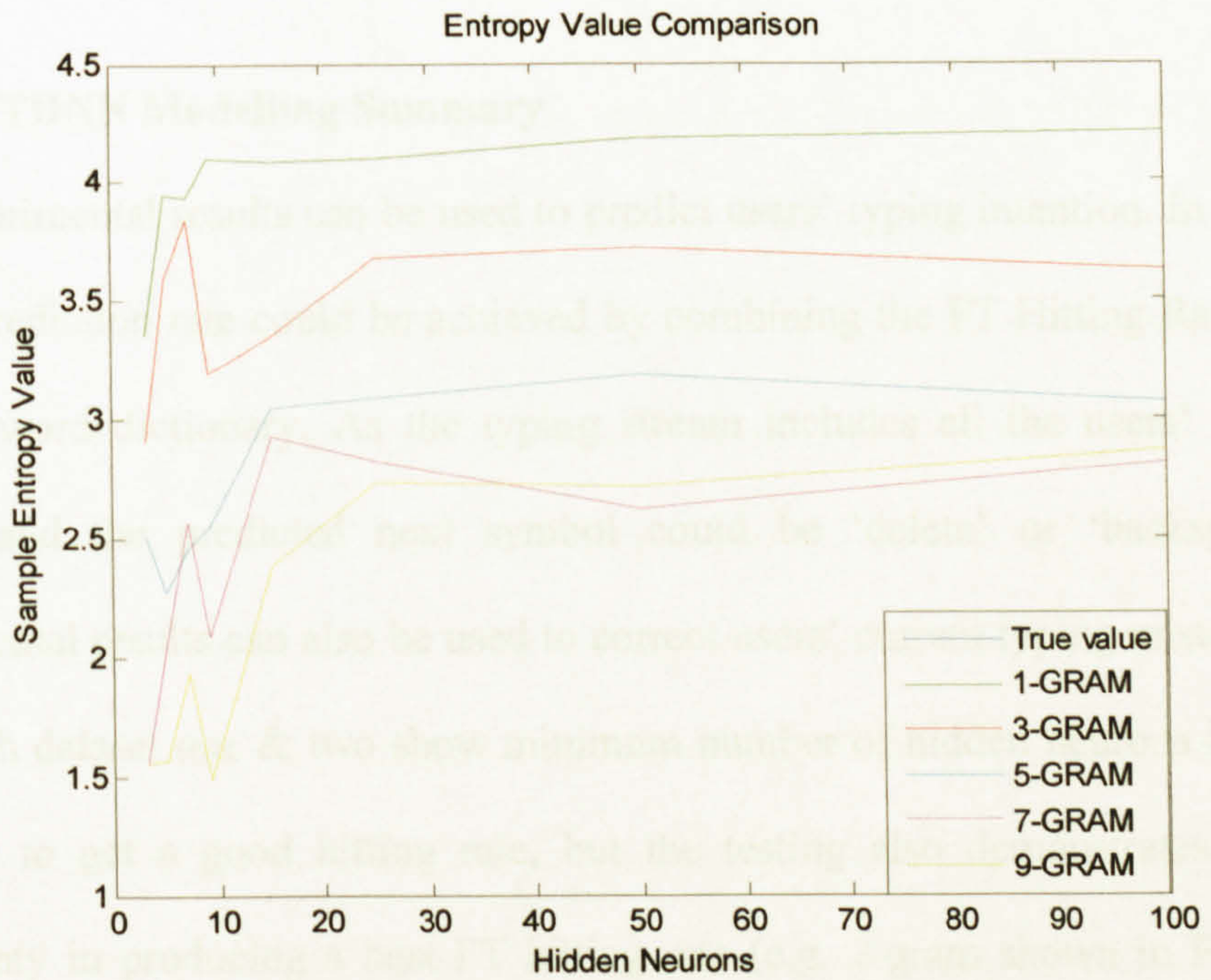


Figure 4.21 [1, 3, 5, 7, 9] gram typing stream s-entropy curves

Figure 4.20 shows that *1*-gram produces the maximum FT Hitting Rate of *53%* whereas *3*-gram with fifty hidden neurons produces the maximum First Hitting Rate of *38.1%*. Similar results have been obtained using dataset one: the lower grams (*1, 2 & 3*-gram) show a better solution using FTDNN model prediction under current circumstances. Both datasets demonstrate a highly accurate prediction rate (FT Hitting Rate, approximately *50%*) with FTDNN model.

Figure 4.21 shows that the higher grams (e.g. *7 & 9*-gram) have the lower n-entropy than the lower grams (e.g. *1 & 3*-gram). This implies that the training and testing process using *7 & 9*-gram of FTDNN model produce more convergent result datasets, and hence its prediction often focus on fewer symbols compared to what *1 & 3*-gram produce.

4.7.4 FTDNN Modelling Summary

The experimental results can be used to predict users' typing intention. In practice a higher prediction rate could be achieved by combining the FT Hitting Rate with an English word dictionary. As the typing stream includes all the users' correction actions and the predicted next symbol could be 'delete' or 'backspace', the experimental results can also be used to correct users' current typing mistakes. Both tests with dataset one & two show minimum number of hidden neurons is required in order to get a good hitting rate, but the testing also demonstrates the gram uncertainty in producing a best FT hitting rate (e.g. *2*-gram shown in Figure 4.11 and *1*-gram shown in Figure 4.20). Therefore, a combination of *1, 2 & 3*-gram is an optimum solution to keep a considerably high and stable hitting rate.

This research develops a Focused Time-Delay Neural Network model with extendible numbers of hidden layer neurons and extendible numbers of time delays to analyze noise-free, noisy and user's historical typing data. Approximately *50%* FT Hitting Rate has been obtained from experimental results. In practice, the results can be applied to symbol prediction and correction.

Further research which includes a distributed representation method to preprocess the typing symbols and applying FTDNN model to predict *l*-length string based on n-gram's *l*-prediction is worthwhile, as more accurate prediction hitting rate can be achieved and more symbols can be predicted once at a time.

4.8 Time gap modelling

From Fitts'law [Paul Fitts, 1954], users input performance ***IP*** in bits per second is proportional to the variable movement time ***ID***, which has a direct relation with the moving distance from one point to another. Let's consider a standard keyboard layout, the time gap between two consecutive strokes directly depends upon the distance between those two keys. As observed, the last key's position represented by the distance and angle with the target typing key could affect some of the disabled users' judgment on their typing accuracy and speed, which would be reflected by the time gap recorded on the computer log. Given the user's typing history, a *1*-gram neural network model named as Time Gap Neural Network (TGNN) is designed here to simulate and predict the two consecutive typing letters' time gap, which uses

dataset two as its experimental dataset. A function - *OnBnClickedsuggesttimegap* is programmed to pre-process dataset two. A fifty-four virtual key codes set is considered, which includes all fifty-three symbols used in 'N-Gram Prediction with Typing Data' section such as alphabets and space. The other symbols which appear in dataset two but do not belong to the fifty-three symbols set are classified as a designed symbol - 'Other'.

OnBnClickedsuggesttimegap function only extracts the keystrokes whose time gaps is in a range of [*0, 3000*] ms. The rest which have been considered as either out of range or computer system related problems are ignored. *2*-gram dataset is created with their corresponding time gaps. This requires *108* (NumberOfSymbols * Gram) neurons in the input layer. All the time gap values are normalized into a range of [*-1, 1*] according to Min-Max Normalization before they are used by Time Gap Neural Network model. The normalization equation is shown below,

$$v' = (v - V_{\min}) * (V'_{\max} - V'_{\min}) / (V_{\max} - V_{\min}) + V'_{\min} \qquad (4.1)$$

Where $V'_{\max} = 1, V'_{\min} = -1$ and variable v is the time gap value extracted from dataset two. The results of TGNN model will be reversed to their natural values based on the same equation.

A traditional BackPropagation neural network is designed with a *108-7-1* three layer structure. The input includes two consecutive symbols represented by unary codes, and the output is the expected time gap between these two consecutive symbols. The

'*tansig*' and '*purelin*' functions are considered as the hidden and output layer's activation function (*see Appendix G*).

A reconstructed dataset extracted from dataset two is used as neural network's training dataset; another two datasets, 'abcdefghijklmnopqrstuvwxyz' (in an alphabetical order) and 'qwertyuiopasdfghjklzxcvbnm' (in a QWERTY keyboard layout order) are used as two testing cases. The experimental results generated by TGNN model based on these two datasets are show in Figure 4.22 and 4.23.

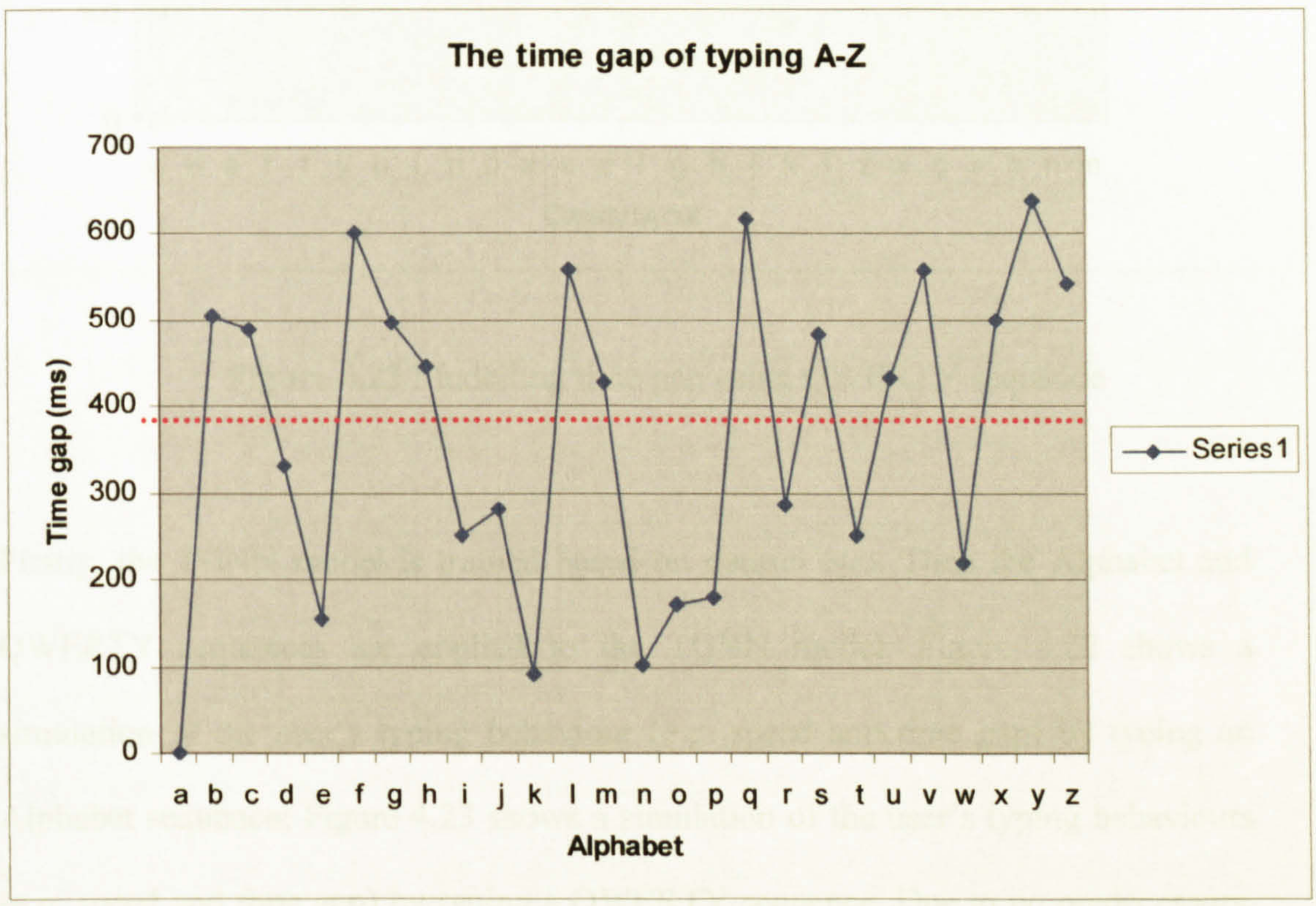


Figure 4.22 Modelling time gap using A→Z sequence

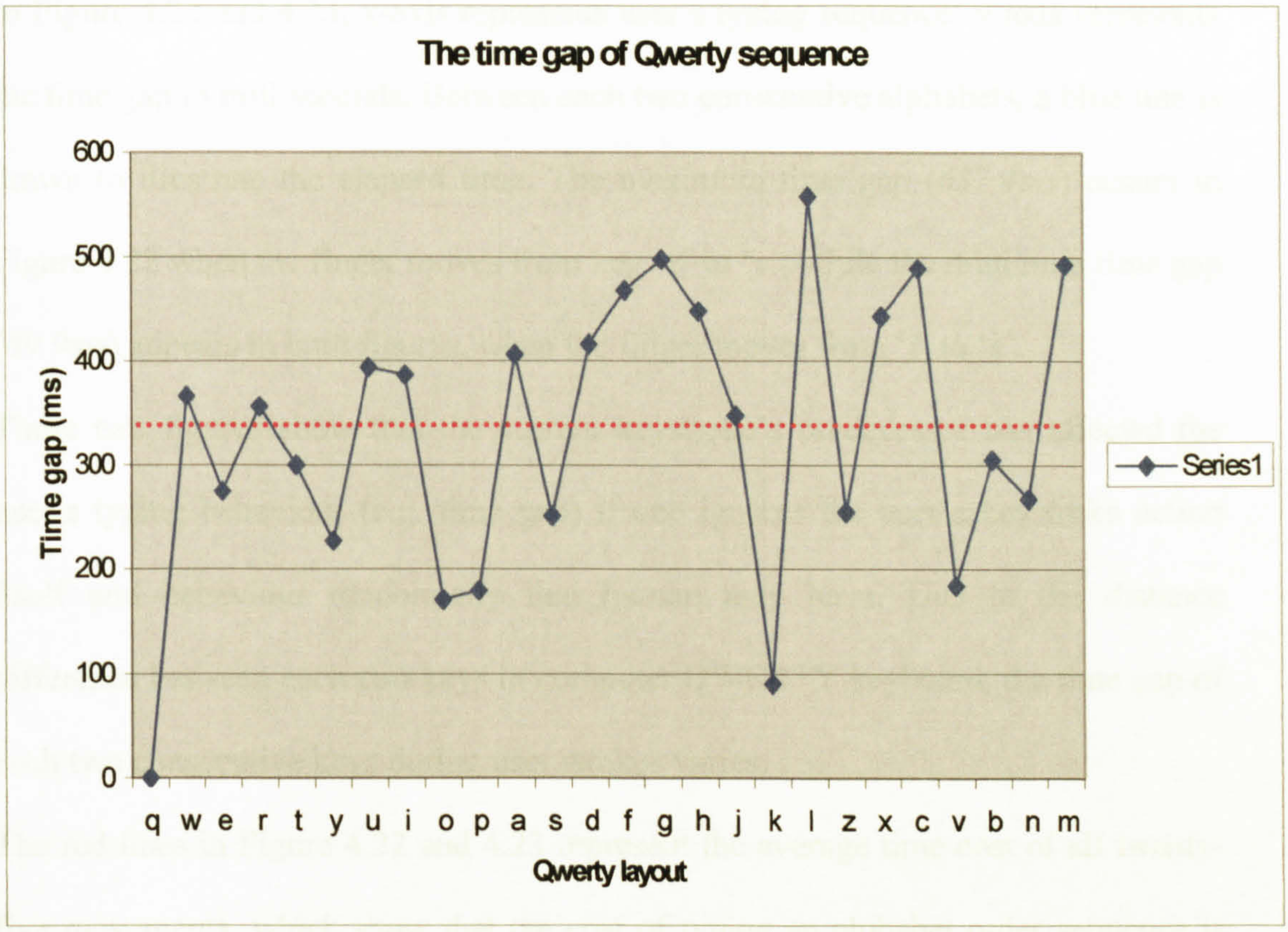


Figure 4.23 Modelling time gap using QWERTY sequence

Firstly, the TGNN model is trained based on dataset two. Then the Alphabet and QWERTY sequences are applied to the TGNN model. Figure 4.22 shows a simulation of the user's typing behaviour (e.g. speed and time gap) by typing an Alphabet sequence; Figure 4.23 shows a simulation of the user's typing behaviours (e.g. speed and time gap) by typing a QWERTY sequence. Due to no predecessors, both corresponding time gaps of the first keystrokes in sequence (in Figure 4.22 is '*a*'; and in Figure 4.23 is '*q*') are counted as zero.

In Figure 4.22 and 4.23, x-axis represents user's typing sequence; y-axis represents the time gap in milliseconds. Between each two consecutive alphabets, a blue line is drawn to illustrate the elapsed time. The maximum time gap (*637.4ms*) occurs in Figure 4.22 when the finger moves from key '*x*' to '*y*'; while the minimum time gap (*89.9ms*) appears in both figures, when the finger moves from '*j*' to '*k*'.

These two figures show that the current keystroke's predecessor has affected the user's typing behaviour (e.g. time gap) if one ignores the user's keystroke action itself and behaviour randomicity that human may have. Due to the distance difference between each two keys in computer QWERTY keyboard, the time gap of each two consecutive keys during user strokes varies.

The red lines in Figure 4.22 and 4.23 represent the average time cost of all twenty-five movements, which show that the cost of typing an alphabet order sequence is *384.44ms* (see Figure 4.22), whereas the cost of typing a QWERTY order sequence is *342.50ms* (see Figure 4.23). The test shows typing an Alphabet sequence is more time consuming based on a standard keyboard. This can be explained by movement cost, meaning that an alphabet order sequence would require more time for user to locate the keys from one to another.

This research gives a glance at the idea that the Time Gap between two consecutive keystrokes is influenced by current symbol's predecessor. A further research tracing back more than one gram history accompanied with a larger dataset is necessary. The physical mobility control and energy cost can be involved in order to find the right patterns among movement direction, typing symbols composition and

keyboard layout. Subsequently researchers may be able to find a convenient, energy saving, fixed or adaptive keyboard layout for those users with special needs.

4.9 Prediction using time gap

People with motor disability or Parkinson disease using keyboard may press adjacent keys or stick keys. These can be shown from the time gap between each two consecutive key strokes. For example, a time gap between the windows keyboard messages caused by sticking keys can be much smaller than the user's normal typing speed; the opposite case may also happen when more time can be spent by disabled people aiming at the target before making up their mind. From observation, interestingly enough it is rare for those people to completely miss typing a symbol. According to these distinct behaviours, a neural network model using BackPropagation (*newff*) is designed by adding an extra Time Gap variable in the input layer, called Prediction using Time Gap (PTG). Here, a small sample typed by a Parkinson person is used to demonstrate the idea. The target typing sample is,

the quick brown fog jumped over the lazy dog

The user's true typing sample is,

hthe quick brrooownn fgow jummppefd iobverethe lwqazy dooggfg

The typed sample is reconstructed for preprocessing,

@the quick br@o@@wn@ @@f@ox@ jum@p@e@d @@o@ver the l@@azy do@g@@@

Where the symbol '@' represents an error or a NULL, compared to the right sample which should be recognized by PTG model. During preprocessing, the time gap value which is one of the input parameters is categorized into three levels and converted into three bits unary codes. In this case,

'<= 10 milliseconds'	*over-fast => 001*
'10< && <=1000 milliseconds'	*user-Speed => 010*
'>1000 milliseconds'	*over-slow => 100*

The user's typing has been recorded both by Notepad and KeyCapture software.

Prediction using Time Gap model is designed with three layers *30-7-28* structure, where the input requirement of PTG model is twenty seven length unary coding symbol {*'a'... 'z', space*} and three length unary coding time gap, and the output requirement is twenty eight length unary coding limited in symbol set {*'a'... 'z', space, '@'*}, where the symbol '@' is added to represent an additional or missed symbol.

The correction rate distribution within one hundred times training is shown in Figure 4.23, which has a mean value of *0.8480* and a deviation of *0.0501*. The x-axis represents the correction rate based on the comparison between the target dataset and PTG generating dataset; the y-axis represents the absolute frequency of the one

hundred times training results, which illustrates the number of times a particular outcome occurs.

Figure 4.24 demonstrates the range that PTG model's correction rate lies on. It shows that the results lie predominantly between *65%* and *90%*. Under this test sample there is about twenty-seven times where the correction rate has reached near *90%* and only once the correction rate happens to be less than *65%*.

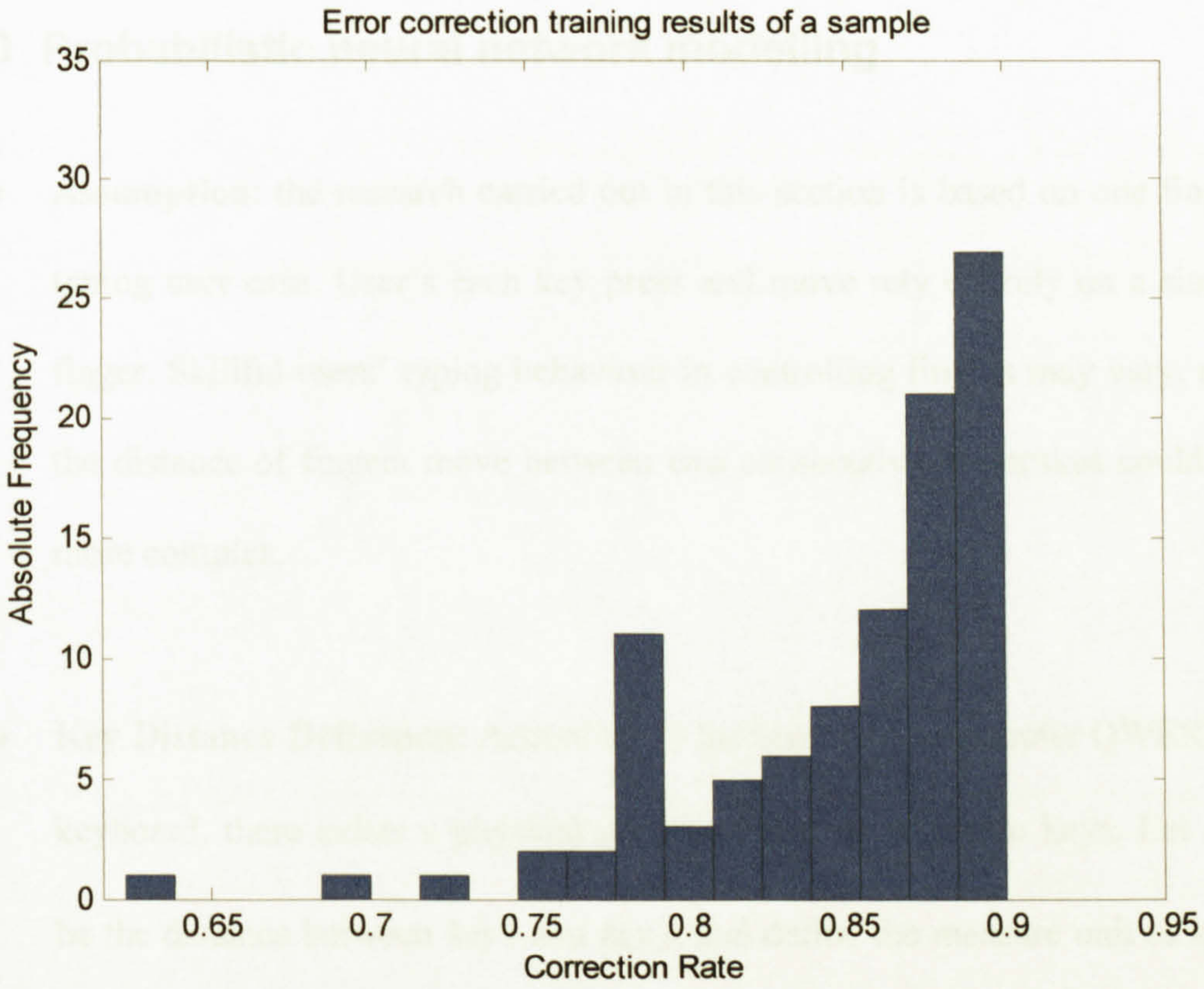


Figure 4.24 Absolute Frequency of PTG model Correction Rate

This test indicates that the time gap can be considered as an input element used by neural network model to correct wrong typed symbols. Due to no gram consideration and the size limitation of training dataset, the relationship built between input and output is a pure right-wrong relationship. This will lead to a further research on the *n*-gram language modelling with larger training and testing dataset.

4.10 Probabilistic neural network modelling

- **Assumption:** the research carried out in this section is based on one finger typing user case. User's each key press and move rely entirely on a single finger. Skillful users' typing behaviour in controlling fingers may vary, and the distance of fingers move between two consecutive keystrokes could be more complex.

- **Key Distance Definition:** According to the layout of a computer QWERTY keyboard, there exists a physical distance between each two keys. Let $d_{i,j}$ be the distance between *key i* and *key j*, and define the measure unit as key-distance. Then, $d_{a,s}=1$ shows that the distance between key '*A*' and key '*S*' is one key-distance; $d_{a,f}=3$ means there are three key-distances between key '*A*' and key '*F*'. Users move their fingers toward the next key as soon as

they finish current key press. The distance between two keys affects a user's typing performance.

- **Error Margin Distance (EMD) Definition:** Based on Key Distance, a variable $\Delta d_{s,f}$ is further defined as a distance between a user's typed key - key_s and target key - key_f and called Error Margin Distance. The Error Margin Distance is mainly caused by the user's 'hitting adjacent key error'.

- **Key Distance Class Definition:** Let's define a class, $C_{key_i,J} = \{\widehat{key_{ij}} \mid key_i\}$, by giving $key_i, \widehat{key}_{ij} \in \{key_1,...,key_n\}$, where $i, j \le n$, n is the number of keys related to a computer QWERTY keyboard, $\widehat{key_{ij}}$ represents a key set around key_i within j key-distances. For instance, a one key-distance set corresponding to key 's' is, $C_{s,1} = \{\widehat{s_1} \mid s\} \approx \{'D','E','W','A','Z','X'\}$.

Noisy data prediction models such as FTDNN not only can be **generally** used to analyze a language text, but also can be explored to analyze some **specific** problems. For example, let's take the Helpline data as a real scenario. As observed, a typist is frequently making 'hitting adjacent key errors'[1] mistakes. Therefore, all the typing

[1] A Toy Problem - Three letters to determine a word: in a QWERTY layout, a typing word with hitting adjacent key errors can be separated into groups based on time gap similarity, where the consecutive letters associated with a certain shorter time gaps are assigned to the same group. Assume the number of letters in each group is more than one, then the toy question is: by picking up a letter from each group randomly to form a letters composition, how many maximum letters, that is groups, are required to determine one and only one English word? (My estimation is *3*)

mistakes are extracted from dataset two and used to identify the possible rules. A sample of 'hitting adjacent key errors' is shown below.

```
"Q" Status=(*) Key(*) Extra(*) KeyDistance(*) TimeGap(*)
"S" Status=(*) Key(*) Extra(*) KeyDistance(*) TimeGap(*)
"BACK" Status=(*) Key(*) Extra(*) KeyDistance(*) TimeGap(*)
"D" Status=(*) Key(*) Extra(*) KeyDistance(*) TimeGap(*)
```

This is a typical 'hitting adjacent key errors' typing mistake that occurred within a user's typing stream. The user's intention is to type a letter '*d*' following letter '*q*', but the letter '*s*' is mistakenly pressed. So the user has to go back and make a correction by pressing 'backspace' key shortly after the mistake is made (in virtual key code, the 'backspace' is represented by 'BACK'). Both Key Distance and Time Gap are calculated and recorded in the log.

The user investigation shows users' hitting adjacent key behaviour is related to the positions of both the last key and the current key if one ignores the stroke randomicity that users' symptoms may cause. It also shows that a user's typing speed moving from one key to another also plays an important role in making such errors. For example, although a faster typing speed than a user's normal speed increases the occurrence of 'hitting adjacent key errors', the users' hesitation which leads to much slower typing speed does not always help to an increase of right typing rate, as observed.

Here, the idea is to use these essential parameters, namely, Key Distance, Time Gap and Error Margin Distance to discover the fundamental rules behind uses' typing

mistakes. Let's start with the introduction of the most popular keyboard layout – QWERTY keyboard, and consider Figure 4.25 and 4.26,

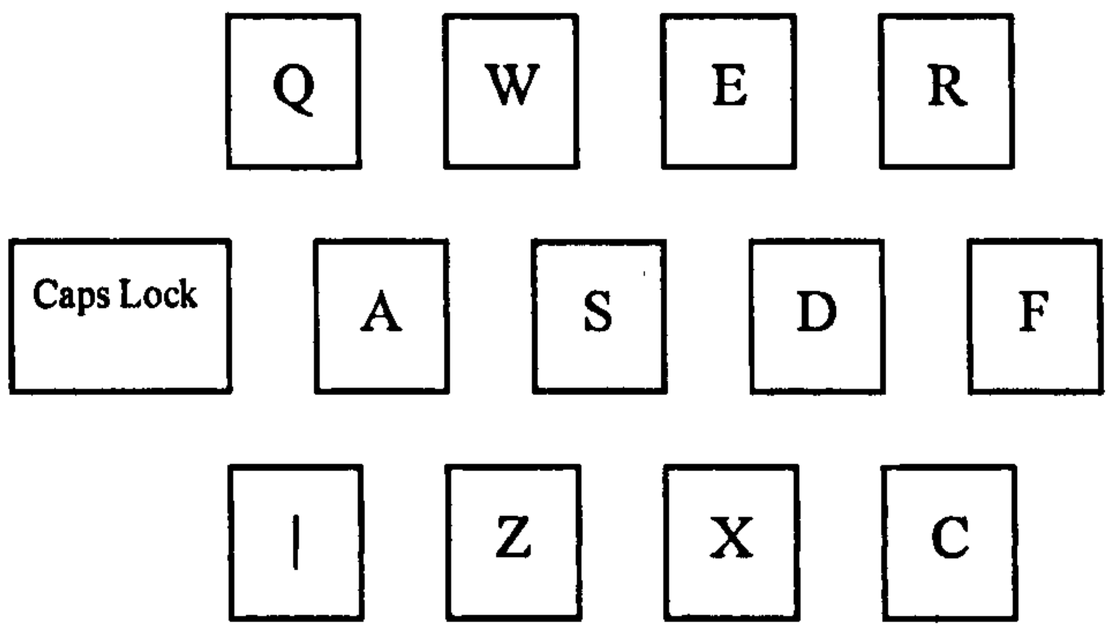


Figure 4.25 A QWERTY keyboard layout sample

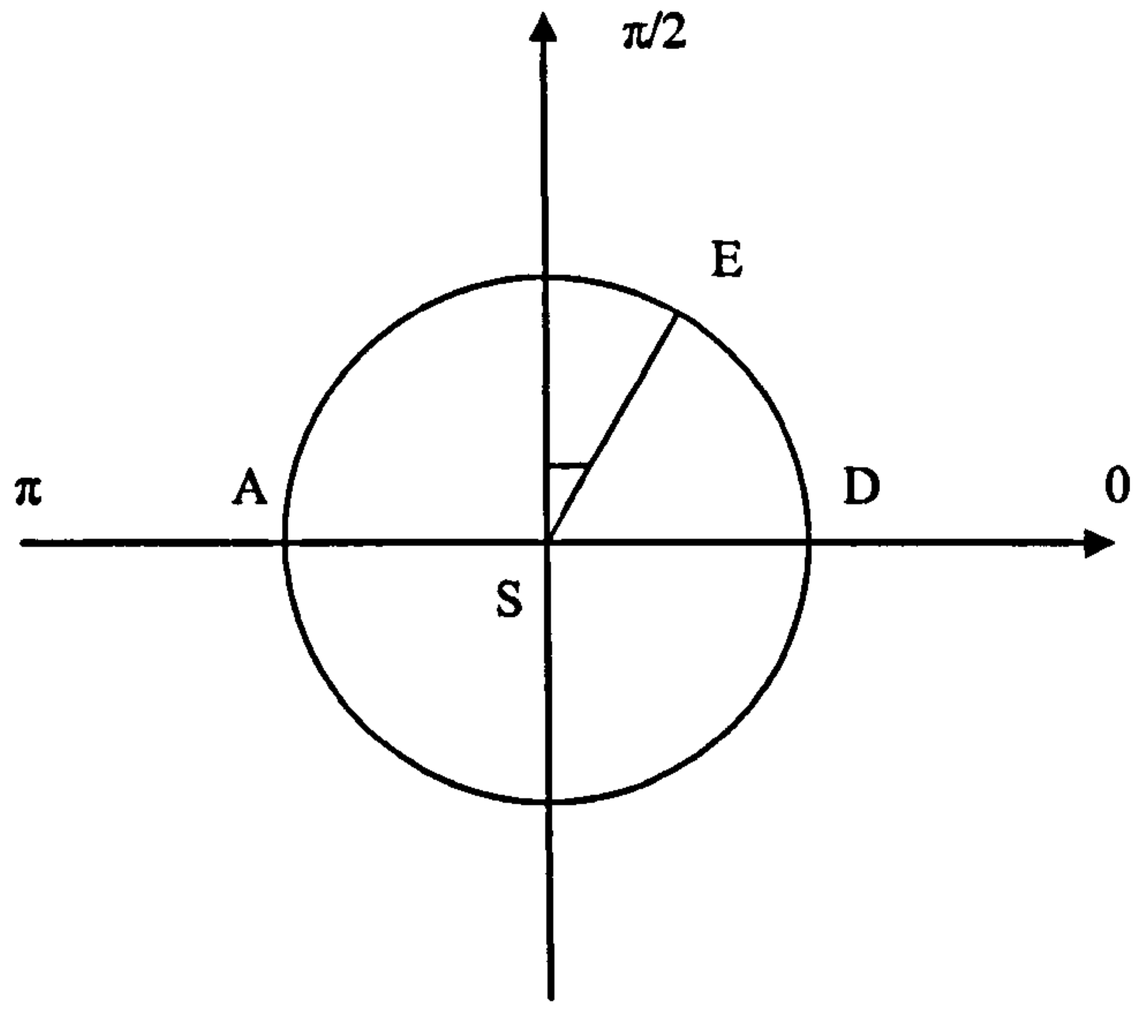


Figure 4.26 Relationship – angle between keys and its surrounding keys D,E,A

In Figure 4.25, key '*S*' is surrounded by one key-distance dataset {'*W*', '*E*', '*A*', '*D*', '*Z*', '*X*'} and two key-distance dataset {'*Q*', '*R*', '*caps lock*', '*F*', '|', '*C*'}. Given

certain inputs, if one requires the neural network model to be able to produce the right symbol that a user intends to type, the designed model not only need to deduce the dataset which the right symbol belongs to, but also the right angle the user intends to move towards. This is illustrated in Figure 4.26. All keys surrounding '*s*' are positioned with different angles. Let's assume the circle starts from right-hand side of '*s*' and turns in an anticlockwise direction. Then the key '*D*' can be expressed by a three dimensions vector, key_d = *{key='s', distance=1, angle=0}*, where *key='s'* illustrates the dataset surrounding key 's', *distance=1 & angle =0* represent the key which is one key-distance away from key 's' with an angle of zero degree. The key '*A*' can be expressed as key_a ={ *key='s', distance=1, angle=π}*, *distance=1, angle=π* means the key is one key-distance away from key '*s*' with an angle of π degree.

The key distance and time gap between last two grams could determine the error margin between the wrong key and the right key. In order to prove this hypothesis, a Neural Network Topology (DAT model) with Distance, Angle and Time Gap vectors in the input layer, and the Error Margin Distance vector between the typed key and target key in the output layer is designed. These require a precise measurement on both input and output parameters. However, given the difficulty of QWERTY keyboard and its associated operating system to respond to an accurate simulation of users' movement and the difficulty of a neural network to provide a precise output, this solution, as it stands, is not practical. For example, the difference in angle between key '*S*' → key '*E*' and key '*S*' → key '*R*' is not

significant. This high precision requirement raises the design difficulty of a neural network model.

In order to overcome these obstacles, a more robust neural network model with re-designed vectors on both input and output layers is developed in this research. The input of neural network model uses (x, y) coordinate expression instead of distance and angle, where x represents x-axis key-distance (i.e. horizontal distance), and y represents y-axis key-distance (i.e. vertical distance). X-axis key-distance refers to a user's horizontal move toward the typed key; y-axis key-distance refers to a user's vertical move toward the typed key (see Figure 4.27). The time gap parameter is kept unchanged, which represents the time difference (*ms*) between two consecutive key strokes. When the error margin is calculated, the coordinate centre lies at the current typed key. When the distance of last typed key and current typed key is calculated, the coordinate centre lies at the last typed key. The sign of key distance will be determined as soon as the coordinate centre is fixed.

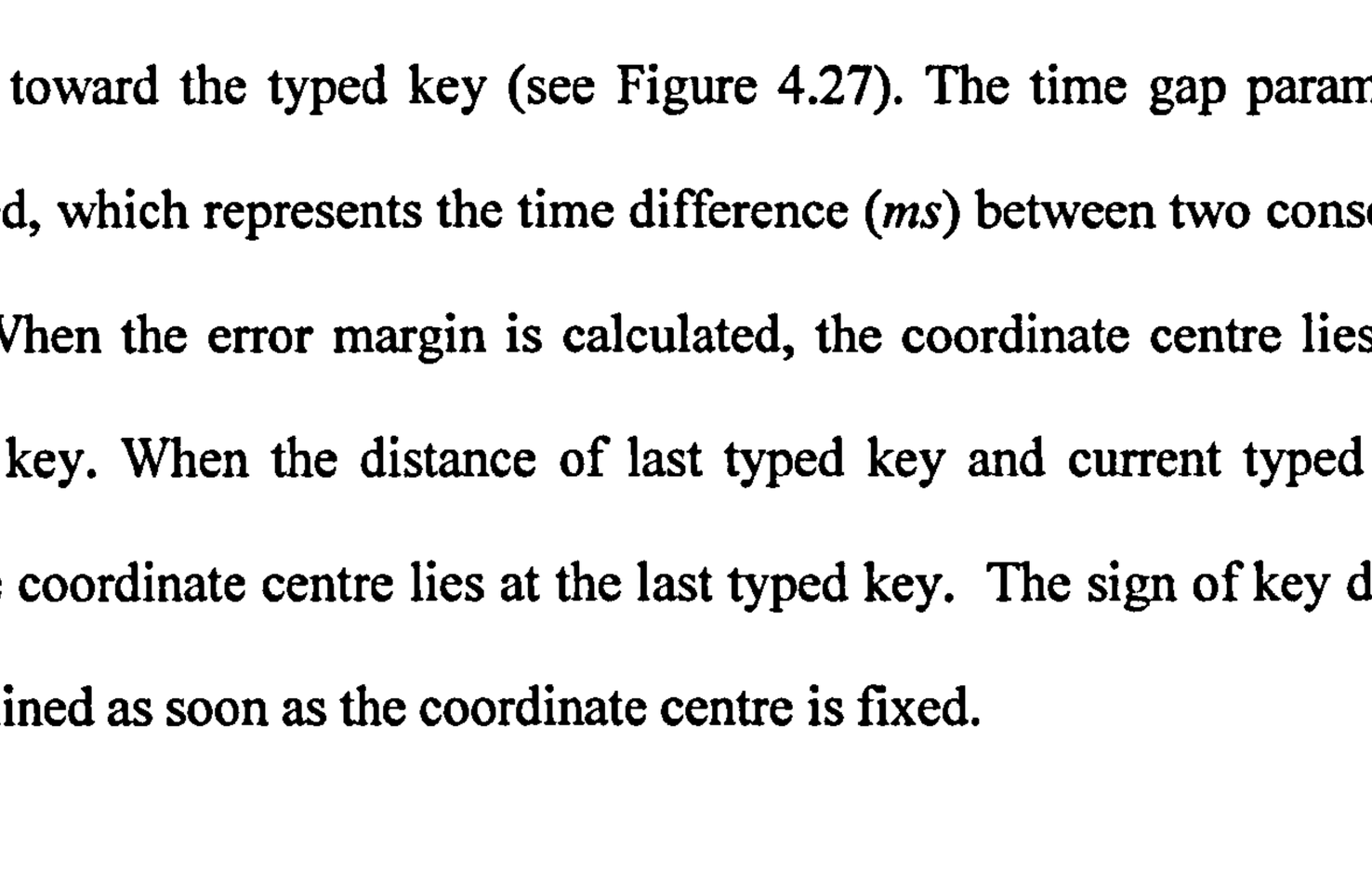


Figure 4.27 Key distances coordinate for PNN classification

In QWERTY keyboard there are maximum of six one key-distance keys around each key. The user investigation records suggest that most of 'hitting adjacent key errors' occur in an area where the keys are equal or less than one key-distance away from the target keys. Therefore, instead of computing a precise error margin $\Delta d_{t,f}$, the output of neural network model can be designed as a six-classes classifier. If one counts the class in a wise-clock direction according to traditional coordinate, then, from Figure 4.26, '*d*' belongs to class one, '*e*' belongs to class two and so on. Thus the question can be interpreted as finding an appropriate neural network model to solve a classification issue associated with input vectors: Distance, Angle and Time Gap.

It is well known that radial basis networks can require more neurons than standard feedforward BackPropagation networks, but quite often they can be designed in a fraction of the time it takes to train standard feedforward networks. One of Radial basis networks is Probabilistic Neural Networks (PNN) which can be used for classification problems. As PNN is a time-efficient and classification-solving solution, in this research a *3-N-1* structure model (DATP model) is designed based on PNN to predict where the target key could possibly lie against the wrong key press.

The DATP model consists of three layers, input layer, hidden layer and output layer. The hidden layer – radbas layer compute the distance between the input vector and the hidden weights vector, and then produces a distance vector which indicates how

close the input is against the correct letter. The third layer would classify the results of radbas layer and produces the right class.

In this experiment, thirty three 'hitting adjacent key errors' are identified from the file 'Helpline20080605-103126-843.txt' of dataset two, and are converted into the format training dataset manually. At the same time another ten samples are extracted from the file 'Helpline20080627-160526-312.txt' as test samples. Here an example is given to show the pre-processing procedure,

```
"C" Status=(*) Key(*) Extra(*) KeyDistance(*) TimeGap(78)
"J" Status=(*) Key(*) Extra(*) KeyDistance(*) TimeGap(108)
"BACK" Status=(*) Key(*) Extra(*) KeyDistance(*) TimeGap(78)
"H" Status=(*) Key(*) Extra(*) KeyDistance(*) TimeGap(923)

➔       3.5      1       108       4
```

The first four lines are extracted from 'Helpline20080605-103126-843.txt'. The line following an arrow is the data transformed manually from the lines above, which has four parameters, namely, horizontal distance, vertical distance, time gap between two consecutive keystroke, and class.

The first line shows that the horizontal distance from '*C*' to '*J*' is *3.5* key-distances, however, if the move are from '*J*' to '*C*', the key-distance would be *-3.5*, the rationale has been shown in Figure 4.27; the vertical distance is one key-distance; the time gap from '*C*' to '*J*' is *108ms* (shown in red) and the class is '*4*' as the key '*H*' is at the left hand side of key '*J*'. In the case of overlapping keys, a half key-distance can be counted. For example,

```
"D" Status=(*) Key(68) Extra(*) KeyDistance(*) TimeGap(93)
"G" Status=(*) Key(71) Extra(*) KeyDistance(*) TimeGap(218)
"H" Status=(*) Key(72) Extra(*) KeyDistance(*) TimeGap(3)

➔       2.5      0       218       4
```

This is a typical key press with overlapped key '*G*' and key '*H*'. The time gap between '*G*' press and '*H*' press is *3ms*, which is much less than the user's usual typing speed. This has been proved by the user's correction which happened afterwards, as shown in dataset two. The horizontal key-distance between key '*D*' and key '*G*' is two key-distances, however, another *0.5* key-distance is added in pre-processing by taking into consideration the overlapping. The vertical distance between these two keys is zero, while the time gap is *218ms* and the output class is *4*. The experimental results show a correction rate of *50%* which is five out of the ten testing samples. However, due to the highness of user's typing disorder and the small size of training dataset, a random training and testing dataset selection strategy is further adopted. The thirty three samples from file 'Helpline20080605-103126-843.txt' and ten samples from file 'Helpline20080627-160526-312.txt' are mixed up and the random function *iRand* is applied to randomly pick up the training dataset and testing dataset in a proportion of *2/3* and *1/3* respectively. Two groups of trials are carried out, and each group of them includes ten training and testing samples. The corresponding plots are shown in Figure 4.28.

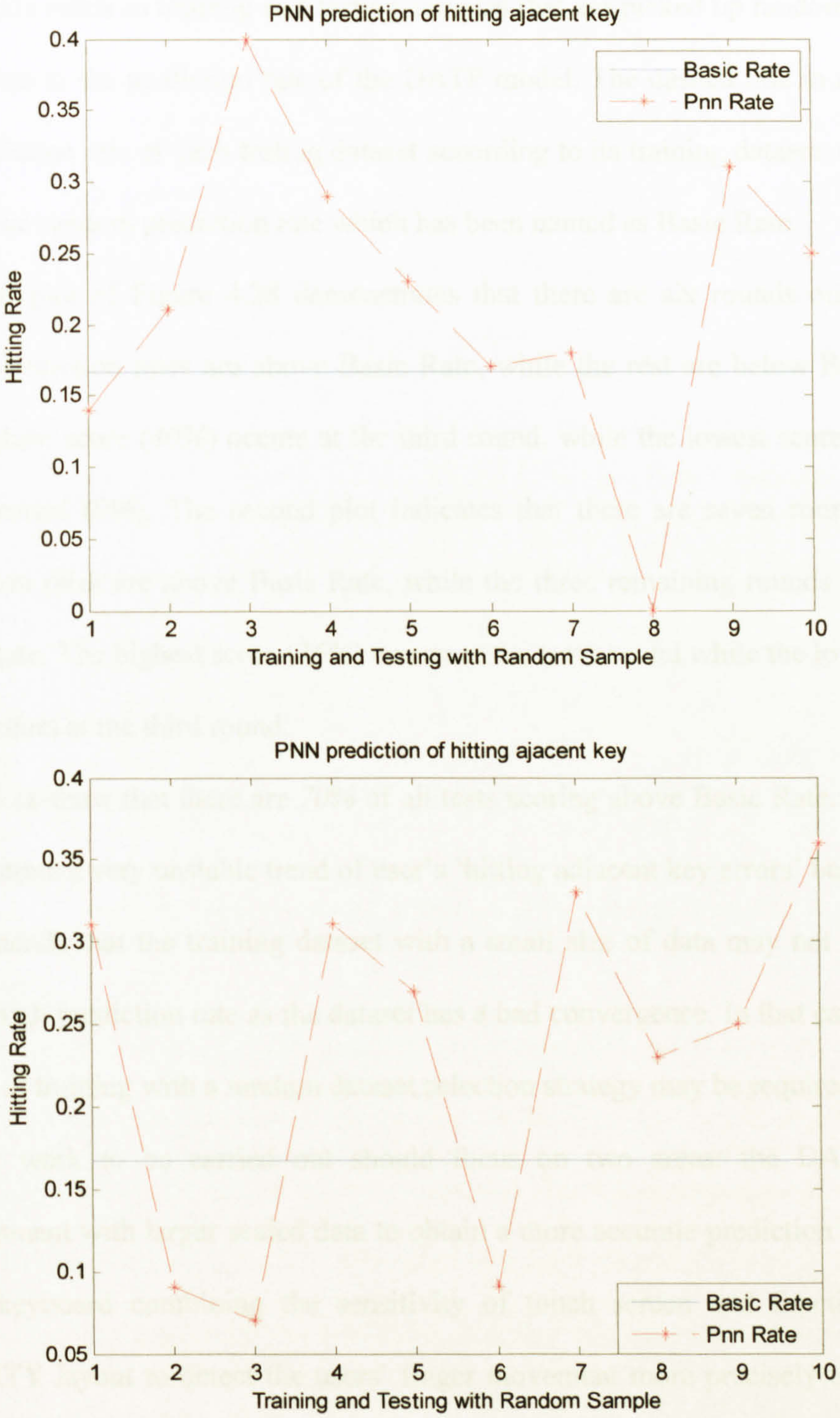


Figure 4.28 Hitting adjacent key prediction rates based on PPN network

The x-axis refers to training and testing samples that are picked up randomly; the y-axis refers to the prediction rate of the DATP model. The dashed line in red shows the prediction rate of each testing dataset according to its training dataset; the line in blue is the random prediction rate which has been named as Basic Rate.

The first plot of Figure 4.28 demonstrates that there are six rounds out of eight whose prediction rates are above Basic Rate, while the rest are below Basic Rate. The highest score (*40%*) occurs at the third round, while the lowest score occurs at eighth round (*0%*). The second plot indicates that there are seven rounds whose prediction rates are above Basis Rate, while the three remaining rounds are below Basic Rate. The highest score (*36%*) occurs at the tenth round while the lowest score (*7%*) occurs at the third round.

Both plots show that there are *70%* of all tests scoring above Basic Rate. They also demonstrate a very unstable trend of user's 'hitting adjacent key errors' behaviour. It recommends that the training dataset with a small size of data may not be able to give a high prediction rate as the dataset has a bad convergence. In that case, several rounds of training with a random dataset selection strategy may be required.

Further work to be carried out should focus on two areas: the DATP model development with larger scaled data to obtain a more accurate prediction rate, and a touch keyboard combining the sensitivity of touch screen and functionality of QWERTY layout to detect the users' finger movement more precisely to calculate the accurate $\Delta d_{s,f}$.

4.11 Summary

In this chapter an intensive Neural Network language modelling process is applied. Following testing datasets collection and data processing tools development, several novel neural network models are developed, whose functionalities, performances and related testing datasets are concluded in Table 4.3.

	Model	Dataset	Noisy	Performance
1	Elm prediction with feedback	Dataset 1	No	First HR = *28%*, FT HR = *45%*[1]
2	FTDNN *n*-gram prediction	Dataset 1	No	First HR = *33%*, FT HR = *56%*[2]
3	FTDNN *n*-gram prediction with noise	Dataset 1	Yes	Noise Rate*[0, 0.3]* → *[37%, 27%]*[3]
4	FTDNN prediction with typed data	Dataset 2	Yes	First HR = *38%*, FT HR = *53%*[4]
5	TGNN time gap modelling	Dataset 2	No	A = *384.44ms*, Q = *342.50ms*[5]
6	Prediction using time gap (PTG)	Dataset 2	Yes	correction rates ∈ *[65%, 90%]*
7	Probabilistic neural network modelling (PNN)	Dataset 3	Yes	*70%* >= Basic Rate[6]

Table 4.3 Neural Networks modelling and the related performances

First, an innovative FTDNN language model is designed and performed with noise-free, noisy and typing stream datasets (i.e. model 2, 3 & 4). It is developed with extendible numbers of hidden layer neurons and extendible numbers of time delays. Based on user's typing history, a *38%* First Hitting Rate and a *53%* FT Hitting Rate are obtained (see model 4). The results suggest that this can be practically applied to

[1] Time & memory consuming
[2] The best performance
[3] First Hitting Rate (2-gram with fifty neurons)
[4] The best performance: *38%* under *3*-gram; *53%* under *1*-gram
[5] The cost of typing an alphabet order sequence is *384.44ms*, while *342.50ms* in a QWERTY order
[6] *70%* of all tests score above Basic Rate

symbol prediction and correction. In contrast, the Elm network (i.e. model 1) shows an un-extendible gram prediction character and a time & memory consuming performance, thus, is not adopted in the research.

Second, the influence of time gap on user's typing performance is studied, and a unique Time Gap model (i.e. TGNN-model 5) is developed. Experimental results show that the current keystroke's predecessor affected the user's typing behaviour, and the Time Gap between two consecutive keystrokes is influenced by current symbol's predecessor. Inspired by this conclusion, a fundamental PTG model (i.e. model 6) is developed. Its experimental results indicate that the correction rates predominantly lie in between *65%* and *90%* with the current testing sample.

Furthermore, an innovative model based on Probabilistic Neural Network (i.e. PNN-model 7) is developed to simulate a specific user typing behaviour – 'hitting adjacent key errors' based on key distances. Results demonstrate that about *70%* of all tests score above Basic Correction Rate. Results also show a very unstable trend of user's 'hitting adjacent key errors' behaviour, which suggest that several training trials with a random dataset selection strategy could be applied.

On the whole, the seven models developments build a foundation for further demonstrating the designed ALMIL framework and developing its related research case, i.e., Intelligent Keyboard.

CHAPTER FIVE

INTELLIGENT KEYBOARD FURTHER DEVELOPMENT

5.1 Introduction

Chapter 3 introduced a fundamental concept which is an intermediate layer noisy language modelling framework called ALMIL, and its demonstration called Intelligent Keyboard (IK), to tackle the noisy text entry originating from difference sources. Chapter 4 carried out a comprehensive neural networks modelling process to explore the methodologies and demonstrate the feasibility of the IK framework by generally analyzing both plain text and user typing stream, and specifically studying particular user typing behaviour. This chapter carries out a further development of Intelligent Keyboard framework. Different users may have distinct typing behaviours and requirements. Therefore, specific and multiple correction and prediction functions can be integrated into the framework based on distinct language features and user characters. However each function may rarely generate a single answer, let alone multiple functions which may produce a larger list of suggestions. This requires developing an evolutionary and adjustable approach to prioritize the suggestions in this list. A word list online ranking approach (WLR) based on neural network BackPropagation learning algorithm can be an optimum solution to meet this requirement. Initial research adopts three distinct algorithms, namely, Levenshtein word distance algorithm, Metaphone algorithm, and Two-Gram word algorithm, with their word distance and evolving frequency difference parameters as input factors for the WLR model. Both time element and words similarity rate should be covered in the design. Furthermore, a pilot application – English Input

Method under Windows XP environment is developed for the purpose of the framework demonstration. It mainly consists of two fundamental functions, typing prediction and correction. Both are capable of real-time learning from user's feedback and evolving along with user's typing toward a more accurate prediction and correction rate.

5.2 Further framework development

As illustrated in chapters 3 & 4, Error correction unit of IK framework is designed intending to combine different algorithms based on distinct scientific methods to predict typing intention and correct typing errors. Several methods based on statistics and phonemics are integrated in this application. Metaphone [Philips, 1990] is a phonetic algorithm indexing words by their sound, which can be adjusted to correct typing errors. Two examples are given below.

able -> APL
hello-> HL

The right side of the arrow is words' phonetic keys. Let's assume that a user intends to type a word 'hello' but mistakenly typed 'hallo' instead, whose phonetic keys (HL) are identical. Subsequently, the system is able to index and retrieve possible words from the database based on the phonetic key, and present them to user for selection.

Levenshtein distance [Levenshtein, 1965] is another function that needs to be explored. It is designed based on the calculation of minimum number of operations required to transform one string into another, for instance,

hello <-> hallo *// the string distance is one*
hello <-> all *// the string distance is three*

After a comparison with each string stored in the memory, the pair with the least distance can be considered as having the highest similarity, and then the one or the group with the least distance can be presented through the user interface module.

Mistakes Recording Function (MRF) is an alternative solution. User's typing mistakes and correction are recorded in a real-time. It builds relationships between the mistake and the correctness. The related occurrences of these relationships are also counted. This method is extremely useful to handle habitual typing errors.

The same design can be applied to Text Prediction unit. N-gram symbol counting and *n*-gram word counting algorithm combined with the existing statistical results can be integrated together. A ranking function would assign a weight to each specific function used in the word prediction or correction. The weights would be adaptable by applying a BackPropagation Neural Network learning algorithm.

5.3 Word list neural network ranking

Multiple solutions can be integrated into Intelligent Keyboard (IK) framework to correct user's typing mistakes and foresee user's typing intention as illustrated in section 5.2. But these solutions rarely produce a single answer or share common results. Therefore, this requires a word-list with word priority (PRI) rather than a single word to be generated. For instance, a user intends to type a word 'hello' but mistakenly typed 'hallo' instead. Let's assume that two functions, namely, Metaphone method and Levenshtein distance, have been integrated into the IK framework as correction functions. Suppose that the results are produced as follows,

Metaphone method generates words: *'hello' and 'hall'.*
Levenshtein distance generates words: *'hello', 'all', and 'allow'.*

Then a words list with 'hello', 'hall', 'all' and 'allow' is made available to the user. It is evident that a ranking algorithm computing each individual's priority is demanded before it is presented to the user interface module.

As IK framework is a real-time model, it requires that the word-list priority computation is able to adapt itself timely based on the user behaviour and some other factors. In this research, this can be simplified by considering the word-list priority computation as a function of three variables, which are Time Change, Context Change and User Feedback. Therefore, a ranking algorithm, which is able to learn from user's selection and adjust the weights assigned to related objects such

as algorithms, words or specific attributes in a real-time, can be developed and deployed.

In this research, the three variables can be further quantified and represented by **frequency increase, word *2*-gram statistic** and **a supervised learning algorithm** respectively. Subsequently, a novel Word List Ranking neural network model associated with the variables is proposed and developed. First, let's give the definition of Word-list, N-formula prediction, Word-List Success Prediction Rate and Simulation Rate.

- **Word-list and N-formula prediction definition:** Let's assume that one has distinct algorithms set $A=\{a_1...a_i...a_n\}$, where $1\le i\le n$ and i,n are positive integers. To process a sequence *s*, if there exists a one-to-many mapping $\{s\to O_i\}$ associated with algorithm a_i between input and output, where $O_i=\{o_i^j \mid 1\le j\le m_i\}$, o_i^j is a generated sequence from the algorithm, j,m_i are positive integers, then one has $\sum_{i=1}^{n} m_i$ sequence generated, and the sequence set is defined as Word-list. The process based on the use of *n* algorithms to generate a word-list is called *n*-formula prediction.

- **Word-List Success Prediction Rate Definition:** Given a word list generated by several algorithms to correct a wrong typing, if the intended word is in the word list, then it is a Success Prediction. If there is a set of

wrong typing, the proportion between the number of Success Prediction and wrong typing is called Word-List Success Prediction Rate (*SP* Rate).

Let's define the number of Success Prediction as o_1 and the number of wrong typing as o_2, then one has $o_1 \leq o_2$ and *SP Rate* = o_1/o_2.

- **Simulation Rate Definition:** Given natural numbers *i, m, n,* where $i \leq n$ and $m \leq n$, let's simulate a testing dataset $p_1...p_i...p_n$ with a trained neural network, and its target dataset $t_1...t_i...t_n$, if output $r_1...r_i...r_n$ has m elements which are $r_i = t_i$, then the Simulation Rate (*SM* Rate) is *m/n*. Given Word-List Success Prediction Rate SP and Simulation Rate SM, then the *First Hitting Rate = SP * SM.*

As illustrated above, the word prediction function involves multiple algorithms. All algorithms would produce their self-interpreted results independently, which is the so-called Word-list *n*-formula prediction. These results could be rarely similar while user may require only one of them if Success Prediction is fulfilled. Then, a functional ranking model will play a major role to present an efficient word list with priority (PRI). If one considers the learning factor required by a word list and variability of its related dataset, a neural network model is a good choice with the dataset updated constantly.

To test and implement this approach, Typing Correction function is adopted as a practical case in this research (*Typing Prediction function can adopt the same*

approach). Three Typing Correction algorithms, namely, Levenshtein word distance algorithm, Metaphone algorithm, and Two-Gram word algorithm (referred to as L.M.T) are chosen to predict right words based on the wrong typing. Levenshtein word distance algorithm is used to calculate the similarity between two words. Metaphone algorithm is used to retrieve the possible words from database against the current typed word. Two-Gram word algorithm is used to retrieve the context-related words from database against the last typed word. Based on the definition of Word-list *n*-formula prediction illustrated above, the correction can be defined as Word-list *3*-formula prediction. Let's use the example introduced in section 3.4.2, where the word 'shall' is wrongly typed as '*sahll*'.

Tomorrow sahll we go to the park?

and assume the database, which includes a *1*-gram & *2*-gram table, has been initialized by a sentence,

Out of your shell! Tomorrow all of us shall start a new training.

Then, the correction result of word '*sahll*' based on Two-Gram word algorithm is '*all*'; the two correction results based on Metaphone algorithm is '*shall*' and '*shell*'; and the two correction results based on Levenshtein word distance algorithm is '*all*' and '*shall*' respectively.

Let's suppose that corresponding to every wrong typing each algorithm generates a maximum of two words in a descending order of frequency. Each word is represented by its two features: frequency and word similarity values. In a real-time database, the word frequency will be updated along with the user typing. Both, frequency and word similarity datasets are normalized before the neural network training and testing start.

Based on the above analysis, a neural network model with *12-3-6* three layer structure is developed and shown in Figure 5.1, where the number of neurons in input layer is determined by the expression: ***Number of Algorithms*** (=3) * ***Number of Words predicted*** (=2) * ***Number of Features of each word*** (=2). The model having a competitive output layer is named as word list neural network ranking (WLR) model. BackPropagation algorithm is adopted as its learning algorithm. Its inputs are wrongly typed words. Each algorithm generates two predictions based on the input. Each prediction is presented by two features, namely, Jaro-Winkler distance and word frequency. In Figure 5.1, the circles in blue are neurons of WLR model; the circles in grey are predicted words; the three rectangles represent the three algorithms; the shapes in yellow show the input and output of WLR model.

Generally speaking, the WLR model is designed to predict a highest ranked word amongst every six recommendations. Then a ranking matter is converted to a neural network classification question solving issue. At the output layer of WLR model, there is only one neuron fired once at a time. To normalize the difference between the typed word and the predicted word, Jaro-Winkler metric method [Jaro, 1995; Winkler, 1999] is applied. It normalizes the words difference, also called words

similarity value, into a range of [*0, 1*]. The dataset of another parameter called word frequency is normalized by Normal Probability Density function based on their mean value and standard deviation.

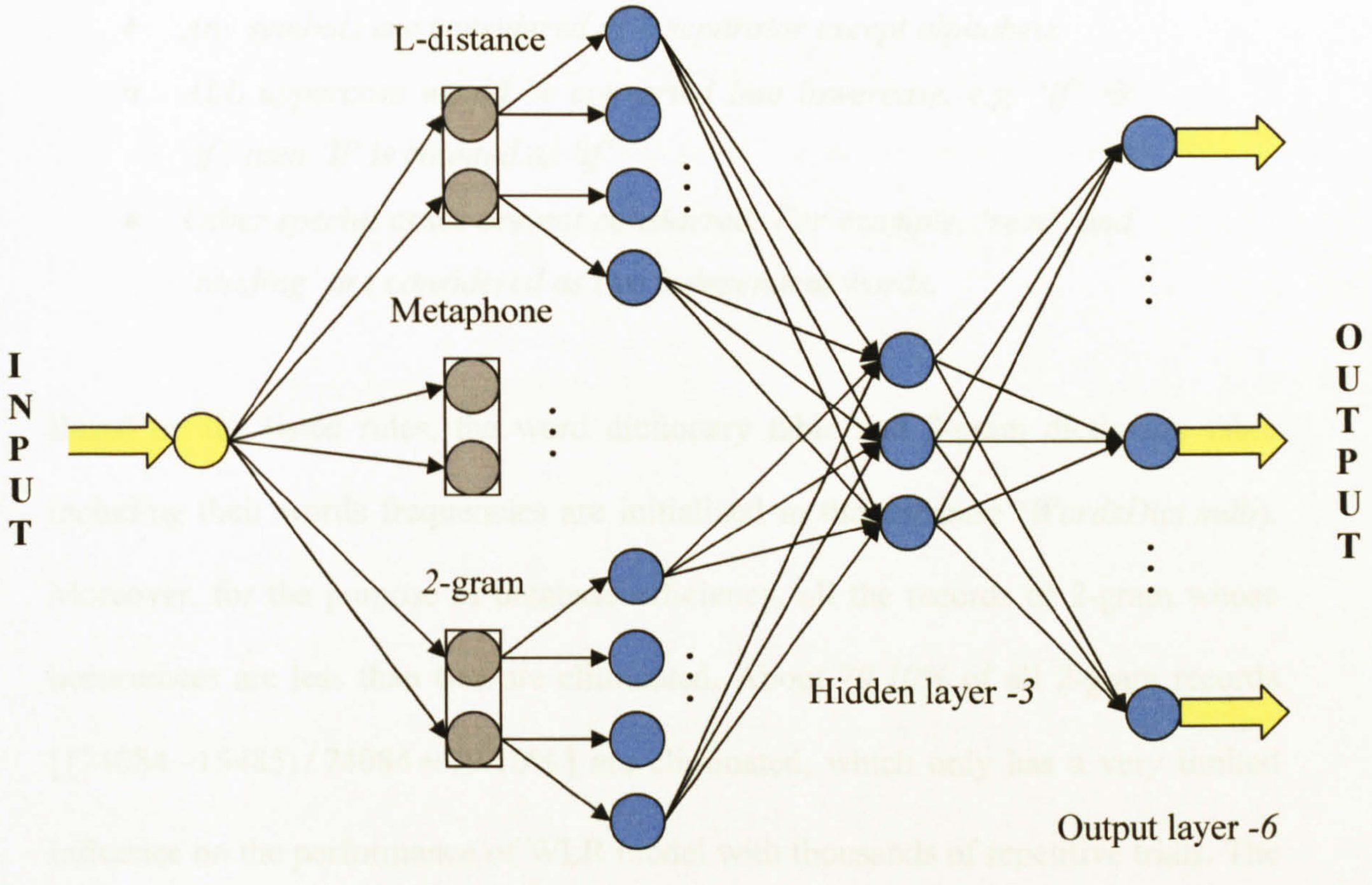


Figure 5.1 Presentation of word list neural network ranking model (WLR)

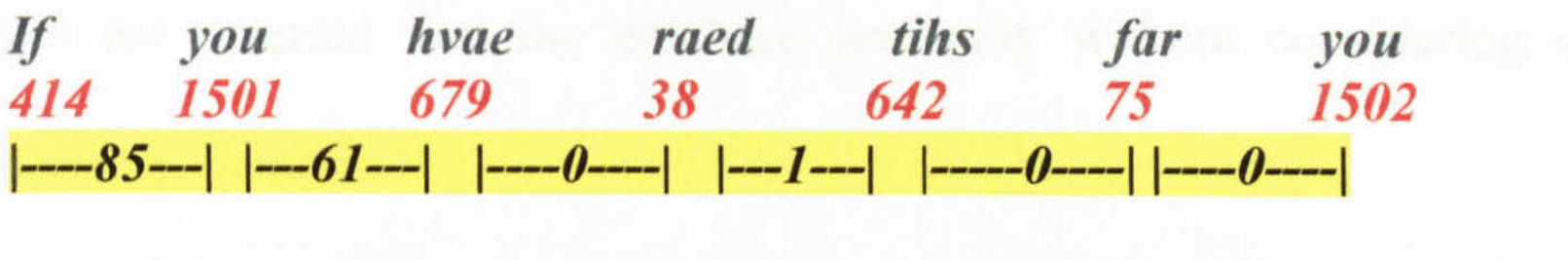
A pilot application introduced in section 5.3 – '*edpa.ime*' and its related Access database – '*WordsDict.mdb*' are used as a software tool to generate an experimental dataset for WLR model. The related database has been initialized by the words' *1* & *2*-gram frequency statistics of dataset one ['Far from the Madding Crowd (1874)']

before the experimental dataset is generated. The database initialization has followed these rules,

- *A word is defined as a sequence of alphabets between two separators.*
- *Any symbols are considered as a separator except alphabets.*
- *ALL uppercase would be converted into lowercase, e.g. 'If' → 'if', then 'If' is counted as 'if'*
- *Other special cases are not considered. For example, 'read' and 'reading' are considered as two independent words.*

Based on the listed rules, the word dictionary table and *2*-gram dictionary table including their words frequencies are initialized in the database (*WordsDict.mdb*). Moreover, for the purpose of database efficiency, all the records of *2*-gram whose occurrences are less than two are eliminated. About *79.10%* of all *2*-gram records $[(74084-15485)/74084 \approx 79.10\%]$ are eliminated, which only has a very limited influence on the performance of WLR model with thousands of repetitive trials. The occurrences of the words' *1 & 2*-gram are kept updated along with user's typing progress (The new *2*-gram words and their occurrences – 'initially, *=1*' are inserted into the database if they happen to be typed). Therefore, these updated frequencies can well represent a user's temporal typing state captured and stored in a database.

As a good simulation to dyslexic's typing, an extra testing sample [Davis, 2003] is used as an experimental dataset for the designed WLR model. It is shown below.



Some words within sentences are wrongly typed, such as ‘hvae’ (should be ‘have’) and ‘raed’ (should be ‘read’). The numbers which are right under each word (in red) indicate the frequency of the word after the database initialization. For example, the frequency of the word ‘If’ is *414* and the frequency of the word ‘you’ is *1501* in the database. The numbers in black indicate the two-gram frequency between two consecutive words. For example, the frequency between the first two words ‘If’ and ‘you’ is eighty-five, shown as ‘|---*85*---|’.

Let’s assume the frequencies of the words shown above gradually increases in the database while other words are rarely typed. Consequently, the change of other words’ frequencies will not have a big effect on the algorithms. Therefore, a simulation can be performed by using the testing dataset which has ignored the influence brought by other words’ frequency changes. In this research, *5505* trials of

test samples are inserted into the database gradually without considering other words' frequency changes.

Let's define a sampling point as a starting point of sampling in these *5505* trials, and define a sampling step as a gap between two consecutive sampling actions. Twenty five sampling points are set up to collect the three algorithms' prediction results in this research. Only those wrongly typed and completed words are considered at every sampling point. For example, the prediction results for words such as 'hvae' and 'raed' are collected; while the prediction results for right words such as 'if', 'you' and uncompleted words such as 'hva' of 'hvae' are ignored. At each sampling point, the whole dataset are gathered and called a sample. Then, twenty five samples are gathered. The determination of sampling points and sampling step is based on a heuristics method, which shows that the influence of initial frequency updating is essential while further updating influence is waning.

Figure 5.2 illustrates the sampling procedure. The x-axis refers to the frequency of the whole sample; the y-axis refers to the numbers of sampling. The sampling points are classified in four categories [*0→5, 10→50, 55→505, 1505→5505*]. As illustrated above, the influence of frequency updating is waning from one category to another although the sampling steps are actually increasing. In Figure 5.2, the red line shows these four categories. For example, five samples have been collected with the frequency being changed from zero to five (i.e. the sampling step is one), and ten samples are collected when the frequency changed from *55* to *505* (i.e. the sampling step is *50*).

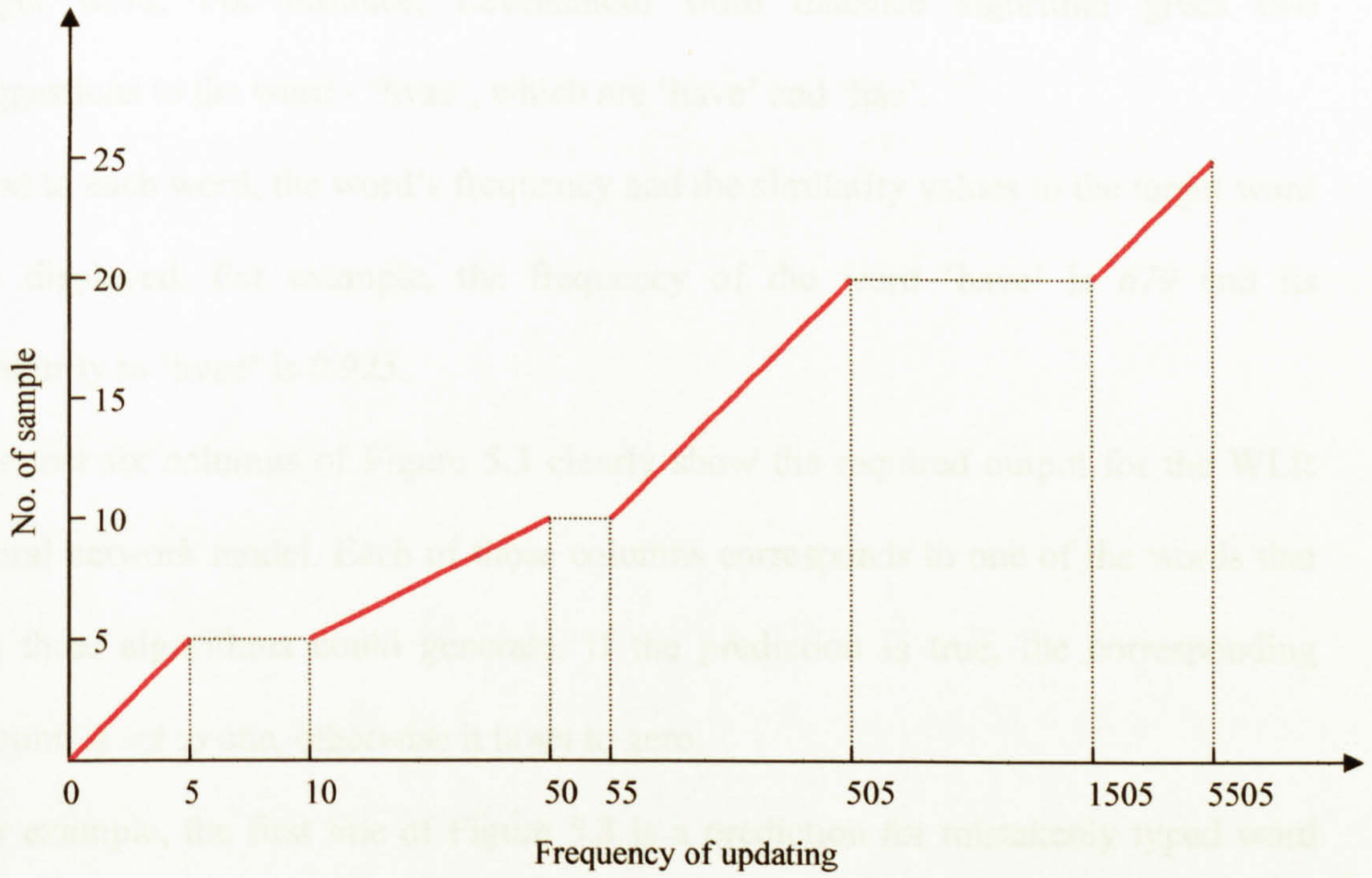


Figure 5.2 Sampling Points representation of WLR modeling

The first two subsets of sample one generated by program – *edpa.ime* are shown in Figure 5.3.

```
%Levenshtein word distance          Metaphone                          Two-Gram word                          %output
have 679 0.925000 |hae   3 0.925000 |ve    93 0.583333 |feei  1 0.500000 |are   86 0.527778 |know 65 0.000000 |1  0  0  0  0
red   43 0.925000 |read 35 0.925000 |right 65 0.483333 |road 64 0.850000 |been 112 0.500000 |a    27 0.750000 |0  1  0  0  0
```

Figure 5.3 A sample of WLR model experimental dataset

Figure 5.3 lists the predicted results of two mistakenly typed words, 'hvae' and 'raed'. The first line marks the three algorithms name and 'output'. Each of the three algorithms has generated two words shown in columns as two predictions of the

target word. For instance, Levenshtein word distance algorithm gives two suggestions to the word - 'hvae', which are 'have' and 'hae'.

Next to each word, the word's frequency and the similarity values to the target word are displayed. For example, the frequency of the word 'have' is *679* and its similarity to 'hvae' is *0.925*.

The last six columns of Figure 5.3 clearly show the required output for the WLR neural network model. Each of those columns corresponds to one of the words that the three algorithms could generate. If the prediction is true, the corresponding column is set to one, otherwise it is set to zero.

For example, the first line of Figure 5.3 is a prediction for mistakenly typed word 'hvae'. Among the six predictions generated by the three algorithms, only the first result of Levenshtein word distance algorithm is predicted correctly. Therefore, the first column of the output is set to one while others are set to zeros. By default, the processing would stop at the first '*1*', and subsequently, the others will be set to zeros. So the output would have a maximum of one '*1*'. Hence, a competitive layer can be appropriately applied to WLR model.

The data shown in Figure 5.3 still can not be used by WLR model directly, as further data processing is required. Thus, the following procedures are applied,

- *Delete the redundancy such as the words of each line.*
- *Normalize all frequencies by applying Normal probability density function*

- *Apply missing data processing rules where it is needed – If some algorithms' prediction results are less than two items, then the frequency and similarity values of the missing items will be set to zeros instead; if none of the algorithms are able to generate results, then this line will be deleted.*

The sampling points are set up according to a heuristic method which analyzes the frequency distribution of the database. For example, the first five frequency updating procedures are considered to be more influential than the case when the frequency changes significantly (e.g. *>1000*). So, the sampling step of the first five is set to one while the rest are sparser.

In this experiment, a vector [*5, 5, 10, 5*] of samples are collected from the four categories and their sampling steps are set to [*1, 10, 50, 1000*]. For example, the first five samples are collected in a step distance of one, the third ten samples are collected in a step distance of fifty.

The dataset is further separated into training dataset [*4, 4, 7, 3*], and testing dataset [*1, 1, 3, 2*]. The post-processing of WLR model follows a 'winner takes all' rule – the neuron which has the biggest value among the six outputs are set to one while others are set to zeros.

After the training process, the Hitting Rates of the testing dataset associated with each category are shown in Figure 5.4 (*A full MATLAB program can be found in Appendix A*).

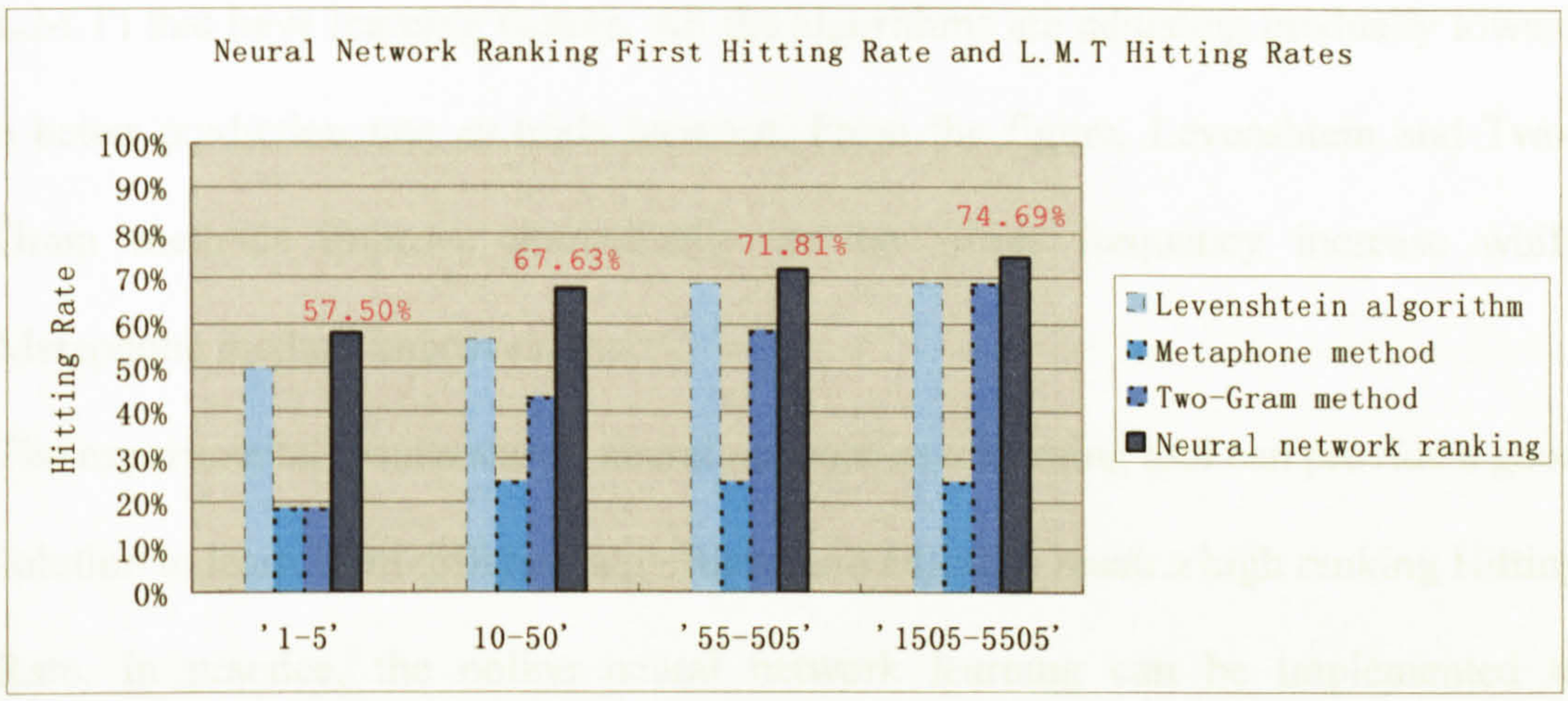


Figure 5.4 The comparison of Neural network ranking First Hitting Rate and L.M.T Rates

From Figure 5.4, the x-axis refers to the increase of words frequency difference, and it is also evident that the samples are separated into four categories based on the step distance, [*1, 10, 50, 1000*]; the y-axis refers to the hitting rate of the algorithms. The bars in dark blue from left to right represent the evolution of neural network ranking first hitting rate. For example, the first histogram shows a *57.50%* Ranking First Hitting Rate with the samples of category one; the fourth histogram shows a best achievement of *74.69%* Ranking First Hitting Rate with more samples collected between frequency *1505* and *5505* in five separated sampling points. The other bars colored in pale blue, sky blue and light blue represent Levenshtein, Metaphone and Two-Gram prediction methods respectively.

Figure 5.4 shows an increase of ranking hitting rates as words frequency difference and the amount of testing samples increase, and that the WLR model achieves the best results in all stages, which is partly influenced by the three algorithms (i.e

L.M.T) that have learning factors. All the algorithms are adjusting gradually toward a better prediction rate as trials increase. From the figure, Levenshtein and Two-Gram methods improve dramatically as the words frequency increase while Metaphone method improves less.

The experimental results shows neural network as a learning tool can provide a good solution to learn from different algorithms and adjust to reach a high ranking Hitting Rate. In practice, the online neural network learning can be implemented to propagate rewards to each algorithm and word.

Currently WLR model adjusts its ranking based on the change of frequency and word similarity. More parameters can be explored such as time element (e.g. 'the most recently used'), and more algorithms can be integrated such as Abnormal Table function, which records the relationship between the wrong and the right word. Then, an improved WLR model with larger scale dataset collection can be developed.

5.3 A pilot application

For the purpose of further Intelligent Keyboard framework demonstration, a pilot application has been developed in this section. The pilot application is named Essex Disabled People Association (EDPA) English Input Method (EIM), which provides a user with two main functions, namely, Text Prediction and Typing Correction. It is designed specially for people with special needs. The user's typing intention (i.e.

shown on computer screen as a list of words) is predicted based on user's input history. The user's typing errors in data streams are gradually checked and corrected when the typing stream goes through each module.

The EIM is developed based on the design of IK framework. Input Method Editor (IME) API and VC are used as developing tools within Windows XP environment. It provides EIM with a way to communicate with most of the Windows applications. The database is designed based on Windows sharing memory and MS Access software and separated into long-term and short-term memory. The relationship between Windows, applications and EIM is illustrated in Figure 5.5.

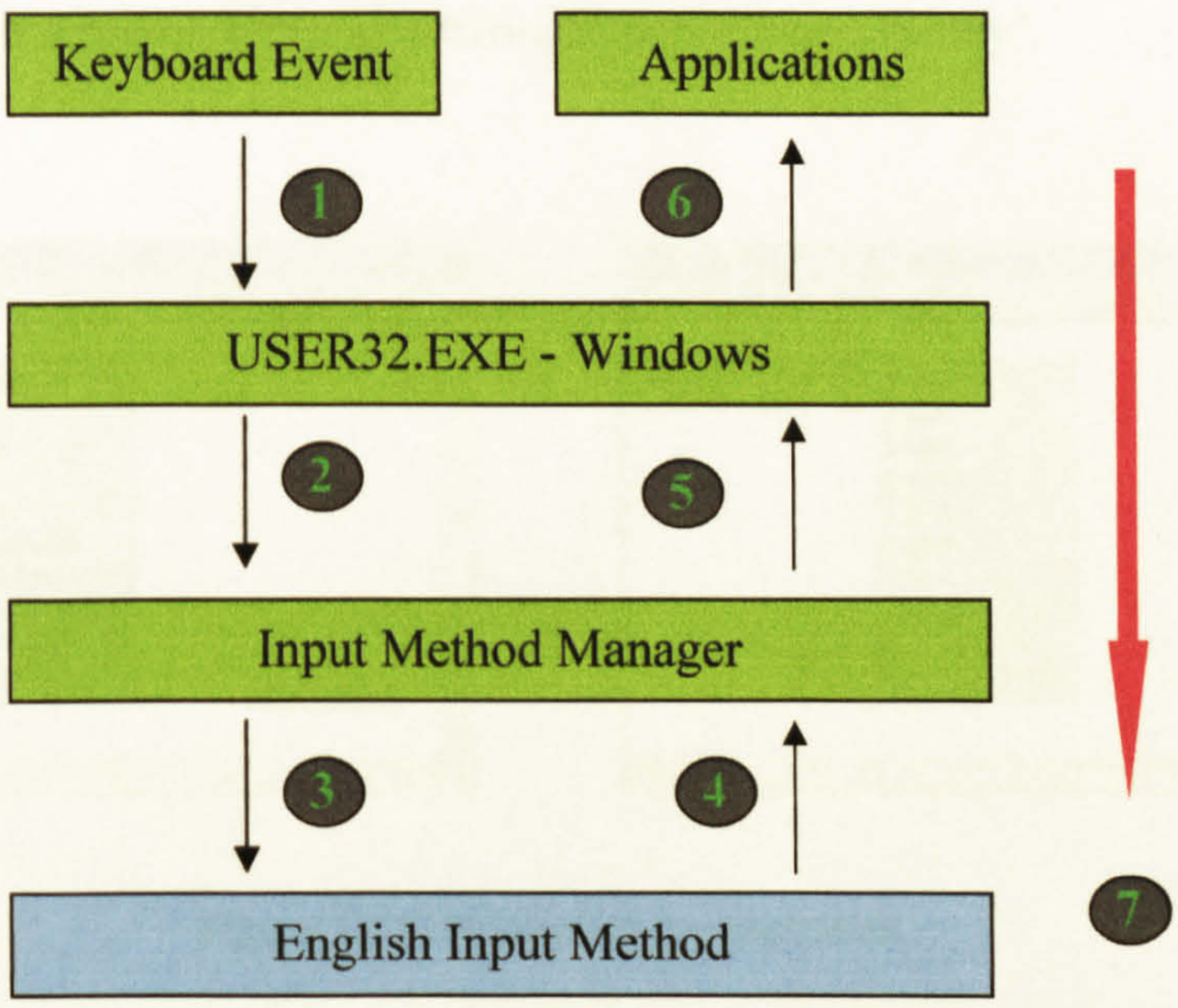


Figure 5.5 Relationship of Windows, applications and EIM

The keyboard events are re-represented by *user32.exe* and then sent to Input Method Manager (IMM). English Input Method is the intermediate language layer which is responsible for receiving and analyzing Windows messages and its related context. Its result is a word-list with ranked prediction or correction words. Via Windows, applications can send commands to English Input Method on their own initiative as indicated in process No. 7.

The interface examples of EDPA EIM are shown in Figure 5.6, which has used Notepad editor as an application. The two figures demonstrate two major functions of the IK application: Text Prediction and Typing Correction. The first figure shows a word list which is a response to the user typing, 'he'. The second figure shows a word list which is a prediction to the wrongly typed word, 'sutdy'.

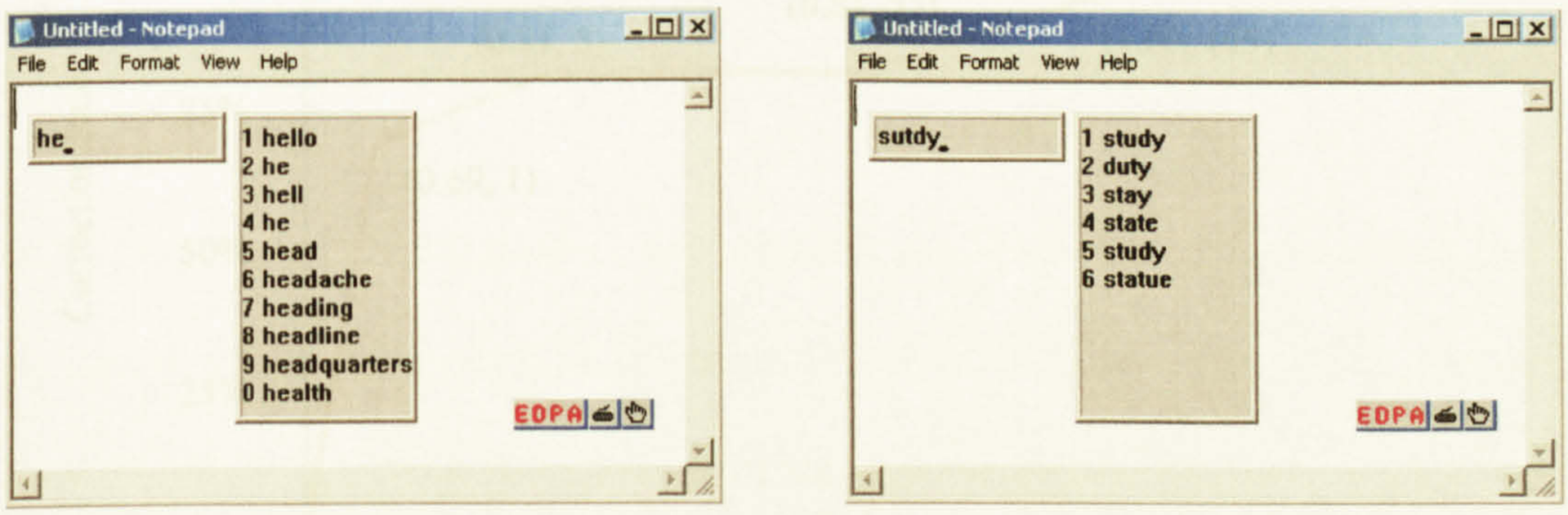


Figure 5.6 EIM interface demonstration

A correction measurement has been used to evaluate the correction rate of EIM. Let's assume the maximum number of prediction words of a word list is fixed. Then

given n words mistakenly typed, if m words are correctly predicted (no matter which position these words lie in the list), then the correction rate c is expressed as,

$$c = m / n \tag{5.1}$$

An experiment is carried out with the designed EIM application to correct user's typing mistakes with L.M.T algorithms adopted in this application. The same data and the same number of trials as in section 5.2 are used. Figure 5.7 shows the evolution of EIM correction rate.

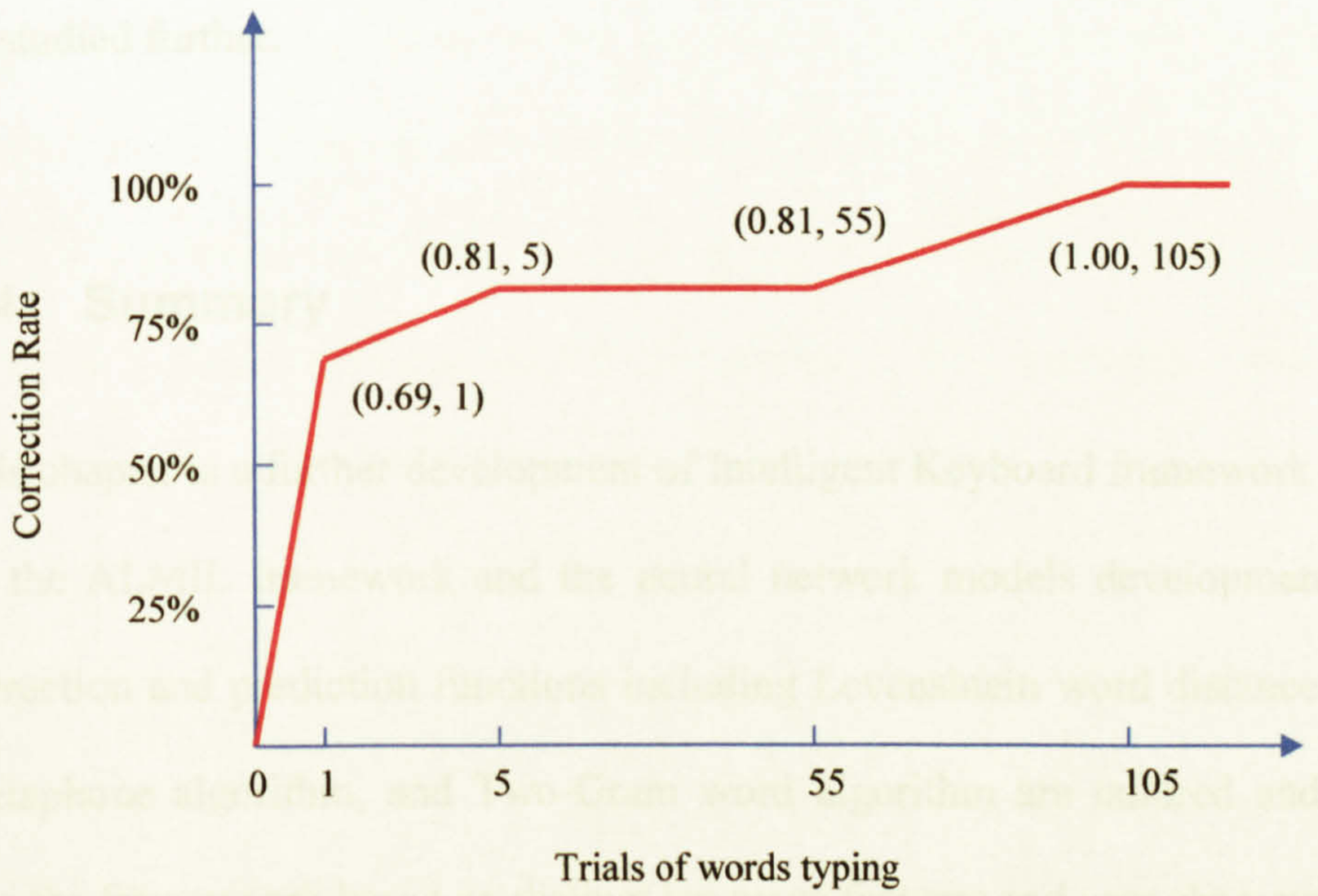


Figure 5.7 Sampling – Evolution of Correction Rates

It shows that the correction rate has improved as the typing trials increase. The first several trials have generated a sharp influence on the improvement. Under current testing dataset, *69%* correction rate is achieved in the first trial. The *100%* correction rate can be reached under ideal condition, that is, without having new mistakes generated. From Figure 5.7, the *100%* correction rate is met after 105 trials.

The reason behind typing mistakes is complex. It depends on numerous factors such as, user mobility, computer environment and typing context, which require specific strategies to deal with. Further development of this application can focus on individual typing behaviour study and more featured algorithms integration. As the correction rate *c* is not a constant in the IK framework, its learning ability *l* need to be studied further.

5.4 Summary

This chapter is a further development of Intelligent Keyboard framework both based on the ALMIL framework and the neural network models development. Multiple correction and prediction functions including Levenshtein word distance algorithm, Metaphone algorithm, and Two-Gram word algorithm are tailored and integrated into the framewoprk based on distinct language features and user characters.

A word list online ranking method – WLR to prioritize words in the word list based on neural network BackPropagation algorithm is developed. A preprocessing method using Jaro-Winkler metric and Normal Probability Density function is

applied to the model's input vectors. All three elements, namely, Time Change, Context Change and User Feedback are principal vectors in WLR model development. In this research, they are quantified and represented by frequency increase, word *2*-gram statistic and a supervised learning algorithm respectively. Experimental results show that *57.50%* Ranking First Hitting Rate with the samples of category one and a best Ranking First Hitting Rate of *74.69%* within category four are achieved. It shows that as a learning tool, neural network can provide a good solution to learn from different algorithms and adjust to reach a high ranking Hitting Rate. In practice, the implementation capable of real-time learning based on propagating rewards to each algorithm and word is a potential research.

For IK framework demonstration purpose, a pilot application – English Input Method is also developed under Windows XP environment. The experiment shows that the correction rate is improved as typing trials increase. The first several trials generate a sharp influence on the improvement. The correction rate of *69%* is achieved in the first trial with the current testing dataset.

CHAPTER SIX

DISCUSSION, CONCLUSION AND RECOMMENDATION FOR FUTURE WORK

6.1 Discussions and conclusions

The goal of Statistical Language Modelling (SLM) is to build a statistical language model that can estimate the distribution of natural language as accurately as possible, which has been widely used to predict the next item in a sequence. The associated algorithms include *n*-gram prediction and prediction by partial matching [Cleary & Witten, 1984] etc. However, language text entry such as typing stream are full of noises, and this issue hasn't been addressed well both in its related research and application area. In this study, an ALMIL framework, a related practical research case called Intelligent Keyboard and a comprehensive neural network noisy language modelling process to analyze noisy text entry including typing stream are developed. Other useful software tools are also implemented.

6.1.1 Adaptive language modelling intermediate layer framework

It is known that the text entry interaction between human and computer could be noisy. However present research such as Dasher [Ward & MacKay et al., 1997-2008] and Windows Control Panel mainly focused on either general or specific user requirements, and failed to meet both cases. Moreover, these research and related applications are rarely compatible with each other. In contrast, this research develops a novel intermediate layer language modelling framework called ALMIL, which is a communication language layer between user and computer applications to analyze the noisy language stream. The advantage of this development is its

immense functionalities and transparency. Through ALMIL the noises of a language stream are filtered significantly and the text entry interaction between user and computer becomes smoother without considering input devices in detail. This framework simplifies and speeds up the process of Human-Computer text entry Interaction, which may lead to a simplistic HCI language development.

6.1.2 Intelligent Keyboard and its pilot application

Predictive text input technologies are some of the techniques that are often found useful by text entry users. Compared to pure prediction products, some efforts such as Prototype [Sensory Software International Ltd, 2007] have been made to reduce typing mistakes, although far few tools can intelligently identify new genre of mistakes. Furthermore, they are short of self-adaptive ability and fail to fully recognize the right patterns from user distinct performance. In contrast this research develops an Intelligent Keyboard framework derived from ALMIL to offer a user-oriented hybrid system with self-adaptive function to help people, disabled people in particular, using QWERTY keyboard more effectively.

Currently, researchers mostly intend to find a unique solution to correct typing mistakes. Although some distinct functions such as *n*-gram and Metaphone [Philips, 1990] have been developed, an optimum solution is hardly identified and a combination of those multiple solutions is never on the agenda. Moreover, these solutions such as T9 [Tegic 1993] rarely produce a single answer or share common results; the answers may change within different context; and the solutions are also required to evolve based on user's feedbacks. All these require a development of

hybrid solution which put all merits of those distinct functions together, and then to produce an optimal prediction with learning and evolutionary factors. Therefore in comparison with traditional single solution, in this research a hybrid solution is designed and integrated into Intelligent Keyboard framework to correct user's typing mistakes as well as foresee user typing intention. Simultaneously a novel word list online ranking method based on neural network BackPropagation algorithm is developed. This novel approach takes advantage of three distinct factors, namely, Time Change, Context Change and User Feedback to learn from different algorithms in order to find an optimum ranking solution. The advantage of this method is that it fully considers the factors that influence the ranking mechanism in a nonlinear learning modal instead of the traditional linear method. The experimental results show that *57.50%* Ranking First Hitting Rate using category one samples and a best Ranking First Hitting Rate (*74.69%*) within category four are achieved.

The research also results in the production of a dictionary database and a piece of software – EnStatistics that is capable of providing a platform to pre-process dataset to meet intelligent models requirements. The software can be independently used as a statistics tools to calculate the target's ASCII and words' *1, 2 & 3* grams. It can be expanded further into *n*-gram letters, *n*-gram words and natural entropy computation assisted with internet crawling.

6.1.3 FTDNN language modelling

Current language modelling research to estimate the probabilities of a set of symbols or words are mainly based on the clean text with statistics techniques such as PPM

[Cleary & Witten, 1984] and PPM* [Cleary, 1995], while noisy language modelling such as user typing stream modelling using neural network are hardly traced. In this research, a novel Focused Time-Delay Neural Network model with extendable hidden neurons and input vectors is developed to analyze a plain text extracted from a novel. Subsequently, noises are added to the plain text with uprising NoiseRates of *0.001, 0.01 and 0.1* before a user typing stream is applied to FTDNN model. Approximately *50%* FT Hitting Rate has been obtained from the typing stream testing. The key production is that the research pioneers a comprehensive neural network language modelling process based on the cross-experiments between the extendible inputs, hidden neurons and noise rates, while statistics based traditional methods have failed to consider the possible noises of a language stream. This approach also produces a significant contribution in the area of neural networks application.

6.1.4 Specific typing behaviours analysis using neural networks

Current research [Karen, 1992] on user typing mistakes correction mostly originates from spell checking techniques. No typing stream research coupled with neural network technology has been seen. Most applications simply consider a typing string as an input without recognizing the influence from other factors such as keyboard layout, Time Gap between two consecutive keys and a user finger moving route. Instead of developing a global application alone, in this research, user's particular behaviours are extracted from dataset and studied through developing several distinct neural network models. An innovative Time Gap model is designed

to study the influence of time gap on user's typing performance. Its experimental results demonstrate that the time gap between two consecutive keystrokes is influenced by the current symbol's predecessor. This has led to a development of an advanced Prediction using Time Gap neural network model. The related experimental results show that the model's correction rates lie predominantly between *65%* and *90%* using current test sample.

'Hitting adjacent key errors' occurs frequently in motor disabled people's typing stream. Traditionally [Abbott, 2004], they are tackled based on the keyboard layout coupled with an English dictionary. All possible compound character sequences have to be checked by conforming to the dictionary even it is considerably computation and memory consuming. In comparison with such an exhaustion method, this research develops a time-efficient and classification-solving solution based on Probabilistic Neural Network model using Key Distance, Angle and Time Gap factors. A distinct approach which suggests adding more training trials using a random dataset selection strategy is adopted during the neural network testing. The experimental results show that about *70%* of all tests have scored above Basic Correction Rate.

6.1.5 Conclusions

Overall this research has fulfilled the listed hypotheses made prior to the main investigation. It originally brings forth a novel concept, intermediate layer language modelling framework called ALMIL for noisy language processing, which fills the gap between input device (i.e. keyboard) and user applications as a noisy language

filter, and develops numerous innovative neural network models which include Focused Time-Delay Neural Network models, Time Gap model, Prediction with Time Gap model, Probabilistic Neural Network Model and Word-List Neural Network Ranking model. The experimental results produced by neural network models have shown some very high language stream prediction and correction performances. Through neural networks modelling, several vital factors which may have an influence on user input behaviours and subsequently affect ALMIL and Intelligent Keyboard framework's functionality, accuracy and efficiency are identified.

This research pioneers a comprehensive FTDNN language modelling process using noise free, noisy data with distinct NoiseRates, and user typing stream both on a general and specific analysis basis. It develops a hybrid solution to combine multiple correction functions based on an evolutionary ranking approach. All these produce a significant contribution in the area of neural networks application, and show a direction for Human-Computer noisy language interaction research. Also this work generates a full report on the disabled people typing behaviour using computer QWERTY keyboard, develops an Intelligent Keyboard framework and its related pilot application named as EDPA EIM for typist, and a universal pre-processing tool with statistical function for all neural networks modelling and *n*-gram estimation. The research and its generated software package build a foundation for further computer noisy text entry research.

6.2 Contributions

- **A novel intermediate layer language modelling framework called ALMIL is developed.** As a communication language layer between user and computer applications, ALMIL is designed to analyze noisy language stream data. It learns from historical data and makes recommendation both for wrong typed and incomplete input. It combines several technologies which include *n*-gram statistics, neural networks and human computer interaction technologies. ALMIL provides a language modelling platform lying between computer users or input devices layer and software applications layer, which lead to a clean text entry. It also provides a platform for the cooperative text entry justification among input devices.

- **An innovative noisy language model using Time-Delay Neural Networks (FTDNN) is developed.** A comprehensive language modelling process using FTDNN model is carried out. Plain text dataset, automatically generated noisy dataset and user typing stream are tested in sequence based on extendable input, hidden neurons and uprising NoiseRates. The results indicate that Time-Delay Neural Network is a capable tool to model language plain text and noisy data. It suggests that a combination of *1, 2 &* *3*-gram is an optimum solution to keep a considerably high and stable Hitting Rate. In practice, the results can be applied to symbol prediction and correction.

- **An Intelligent Keyboard framework is developed.** Intelligent Keyboard is derived from ALMIL. It is developed toward a user oriented hybrid framework with self-adaptive function to help disabled people to use QWERTY keyboard more effectively. Through Intelligent Keyboard, user's typing data stream can be checked, rectified, and predicted in sequence by going through each of its units and modules along with user's typing process. Multiple algorithms are integrated into the framework both based on users' specific characters and related language features.

- **Several neural network models to analyze user typing streams are developed.** Inspired by Fitt's law, a Time Gap Neural Network model (TGNN) is developed to simulate and predict a user's two consecutive typing letters' time gap. The experimental results suggest that the time gap between two consecutive keystrokes is influenced by current symbol's predecessor. This has led to a development of Prediction using Time Gap model (PTG). Time Gap between two consecutive keystrokes has influenced user typing behaviour. TPTG model is developed to predict right symbols based on an extra Time Gap variable at the input layer. Its experiments have proved that the Time Gap can be considered as an input element by neural network model to correct typing mistakes. This would also lead to a better understanding on the mobility control ability and energy cost of those people who have difficulties in using computer keyboards.

User investigation shows that user 'hitting adjacent key errors' behaviour is heavily related to the positions of both the last stroked key and the current key. So a Probabilistic Neural Network (PNN) based model is developed, which adopts Key Distance, Time Gap and Error Margin distance parameters to identify the possible rules behind uses' typing mistakes. About *70%* of all test datasets have scored above the Basic Rate. It also demonstrates a relatively high randomicity of the user's 'hitting adjacent key errors' behaviour.

- **An innovative approach on Word List real-time Ranking (WLR) using Neural Network BackPropagation algorithm is developed.** As a combination typing correction function may generate multiple predictions, WLR model is developed to prioritize these predictions. Three distinct algorithms with word distance and evolving frequency difference parameters are used as WLR model's input vectors. The results indicate that neural network is a suitable learning tool to provide a good combination solution and reach a high ranking Hitting Rate.
- **Several useful 'by-products' are developed along with the project.** An investigation report on disabled user using computer QWERTY keyboard is generated. About twenty-seven people who are elderly or disabled have been interviewed. Their performances are classified into five major categories. As least as three distinct typing behaviours has been concluded in each category. Reflected questions and required solution from users are

also recorded. This report builds a foundation for research on people with distinct typing behaviours so as to seek an alternative solution for improving computer's usability and accessibility.

A pilot application called English Input Method is developed for disabled people community. This software is a demonstration of Intelligent Keyboard framework, and specially designed to meet disabled people's needs. It provides users a way to communicate with most of Windows applications by filtering typing errors. It presents two distinct functions for users, namely, typing prediction and typing correction. The experiment shows the correction rate is improved as the number of typing trials increases. The *100%* correction rate can be reached without new type of mistakes generated.

A universal statistics and preprocessing tool called Enstatistics is developed, which aims to provide a platform to pre-process dataset by reading raw data from different sources and transforming the raw data into text files to meet the requirements of all intelligent models. This software can be independently used as a statistics tool to calculate ASCII and word one, two and three grams. It can also be used for any neural network models for data conversion with a minimal modification on the existing code.

6.3 Recommendations for future work

In this final section, an outline of research opportunities for future work is presented below.

- **Increase the scale of disabled user investigation, data collection and computer capacity to enhance Intelligent Keyboard processing.**

One limitation of this research is down to the external environments. The research has predominantly focused on motor disability, but typing behaviours are various, even motor disability typing behaviours may include wider range of features than the current cases addressed in this research. Hence, a more comprehensive disabled user investigation with typing stream log is required. Furthermore, an intensive research on users' hands motor control can be conducted.

A statistics of QWERTY keyboard user key strokes and related analysis also need to be explored further. Nestor Prediction Measurement (NPM) [Nestor 1997], which used keystrokes to calculate the typing occurrence saving, can be used to combine with Fitts law or other energy computing methods to calculate and evaluate user's typing effort saving rate.

As presented in chapter 4, neural network language modelling demands not only large training dataset but also large capacity of memory and computing. An improved FTDNN model along with an implementation of parallel

processing on large-scale computing machine can be investigated and conducted.

With the larger computer capacity, one may integrate more complex algorithms based on NLP such as grammar checking into Intelligent Keyboard framework. This will be able to deal with the errors which are not well solved by the Correction unit using the incompetent word *n*-gram function illustrated in chapter 3. Then the performance of Intelligent Keyboard can be further improved.

- **Focused Time-Delay Neural Network modelling expansion**

A distributed representation method to preprocess the typing symbols, where each symbol is represented by several features such as key distance, time stamp and symbols can be applied to the FTDNN models. In such a case, the prediction will not be solely based on the symbols themselves but also on the related *n*-gram features. Another potential research is to apply FTDNN model to predict *l*-length string based on *n*-gram's *l*-prediction. Therefore with the same *n*-gram input as presented in chapter 4, more symbols can be predicted.

- **Prediction based on an online Markov chain method – an alternative to neural network models.**

In a Hidden Markov model, the state is not directly visible, but variables influenced by the state are visible. Each state has a probability distribution

over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states.

Let's consider a user typing sequence as output tokens or a so-called emission and the intended typing sequence as series of states, then HMM can be used to calculate the maximum likelihood estimate of the transition and emission. Based on the computed probability of transition and emission matrix, the Viterbi algorithm can be applied to compute the most likely sequence of states given a sequence of emission, and then the most likely next symbol is predicted.

Given a symbol set $\Omega = \{x_1, ..., x_n\}$ and a temporal sequence $s_1 ... s_i ... s_m$, where $1 \le i \le m, s_i \in \Omega$, to build a symbol prediction model based on HMM, the following work can be carried out,

- *Collect and categorize massive typing stream according to each individual user.*
- *Pre-process typing stream (for example, a particular symbol can be added to represent some conditions happened during typing, e.g. an extra symbol which exists in emission sequence but stats sequence).*
- *Collect and calculate several general probabilities of stats between symbols x_i and x_j based on particular dataset, user's typing history and previous research [Jones & Mewhort, 2004].*
- *Design step function to generate weight parameters $W_j = \{w_{j1} ... w_{jm}\}$ based on users' typing temporal and pervious weight set W_{j-1}. Then fuzzy logic method can be used to blur the boundary between each two steps.*
- *Use Viterbi algorithm [Andrew Viterbi, 1967] to deal with the new generated typing sequence.*
- *Estimate results and go back to 3.*

(For future work, a relevant pseudo code written in Matlab code is given in Appendix A).

- **Word-List Ranking neural network model (WLR) improvement**

Current WLR model adjusts its ranking based on the change of word frequency and similarity. Further research can focus on more parameters such as time element and more algorithms integration such as Abnormal Table function, which records the relationship between the wrong and the right words. An extendable hidden units experiment could be applied, and the First and FT hitting rates can be calculated along with the frequency-difference evolution. Furthermore, WLR model related application can be developed to propagate the rewards based on user feedbacks to all algorithms and words stored in the database.

- **An innovative design of touch keyboard towards a more precise user hand movement detection.**

Based on Fitts' law, a study [Card et al., 1978] that compared use of a touch screen (i.e. finger pointing), mouse, joystick and keystrokes, and used position time as a performance measure, shows that finger pointing with a touch screen is optimal, with text keys being the second slowest, while function keys is the slowest of all. Further research on a development of an efficient touch keyboard combined with the EIM application to precisely detect user hand movement is strongly recommended. This will lead to an

achievement of a more accurate prediction and correction rates. A radial basis network model with a linear layer at the output can be designed to compute the accurate $\Delta d_{s,f}$.

- **English Context Dictionary via Internet**

The EIM application needs a proper dictionary with a statistics of ASCII and word *n*-gram. Based on the statistics, entropy can be calculated and evaluated through a comparison with other statistics methods. Specifically, a further research aiming to build a word dictionary with words occurrences and semantic-related *n*-gram through searching vast amount of web pages is recommended. The relationship such as semantic similarity and grammatical relationship among the words should be highlighted. The dictionary will be integrated into EIM application. This function is an extension of the Enstatistics software package.

REFERENCES

Andrew Golding and Dan Roth (1999) 'A winnow-based spelling correction algorithm', *Machine Learning*, 34, pp.107-130

Andrew McCallum, *SRILM - The SRI Language Modelling Toolkit, SRI International*, Sep 08, 2008, available: http://www.speech.sri.com/projects/srilm/ [accessed 05 February 2009]

Bengio Y, Ducharme R et al (2003) 'A neural probabilistic language model'. *The Journal of Machine Learning Research*, volume 3, pages: 1137 - 1155

Bengio Y., Simard p., and Frasconi P. (1994). 'Learning Long-Term Dependencies with Gradient Descent is Difficult'. *IEEE Transactions on Neural Networks 5*:157-166

Bill Winkler, George McLaughlin and Matt Jaro [online], *strcmp95.c - Version 2*, available: http://www.census.gov/geo/msb/stand/strcmp.c [accessed 23 January 2009]

Charles Bloom, *PPMZ–High Compression Markov Predictive Coder* [online], http://www.cbloom.com/src/ppmz.html and ftp://ftp.cpsc.ucalgary.ca/pub/projects /text.compression.corpus/text.compression.corpus.tar.Z [accessed 18 January 2009]

Charles Petzold, (1998) *Programming Windows, 5th Edition,* Microsoft Press

Chin-Teng Lin, Hsi-Wen Nein and Wen-Chieh Lin (1999) 'A Space-Time Delay Neural Network for Motion Recognition and Its Application to Lipreading' *International Journal of Neural Systems*, Vol. 9, No. 4, Aug 1999, pp. 311-334.

Cleary. J.G. and Witten. I.H. (1984) 'Data compression using adaptive coding and partial string matching', *IEEE Transactions on Communications*, 32(4), 396-402

Cohen, William W., Ravikumar, Pradeep and Fienberg, Steve. (2003). 'A Comparison of String Distance Metrics for Name-Matching Tasks', *IIWeb 2003*: 73—78

C.E. Shannon (1951). 'Prediction and entropy of printed English'. *Bell Systems Technical Journal*, 30:50-64, January, 1951

Dan Roth. (1998) 'Learning to Resolve Natural Language Ambiguities: A Unified Approach', *Proceedings of the National Conference on Artificial Intelligence*, pp.806-813

Daniel Fallman. (2002) *The penguin: using web as a database for descriptive and dynamic grammar and spell checking* [online], available: http://citeseer.ist.psu. edu/fallman02penguin.html [accessed 03 March 2008]

Darragh, J. J., Witten, I. H. at el. (1990). 'The reactive keyboard: A predictive typing aid'. *Computer*, vol.23, no.11, pp.41-49
Darragh, J. J. and Witten, I. H. (1991) 'Adaptive predictive text generation and the reactive keyboard', *Interacting with Computers,* Vol.3, no.1, pp.27-50

Darrel Hankerson, et al. (2003), *Introduction to Information Theory and Data Compression 2nd ed.* CRC Press LLC, US

David J. Ward, Alan F. Blackwell at el. (2000) *Dasher-a Data Entry Interface Using Continuous Gestures and Language Models*, [online], available: http://www.inference.phy.cam.ac.uk/djw30/papers/uist2000.html [accessed 03 March 2008]

David J. Ward. (2001) *Thesis - Adaptive Computer Interfaces* [online], available: http://www.inference.phy.cam.ac.uk/djw30/papers/thesis.html [accessed 03 March 2008]

Disability Essex, available: http://www.disabilityessex.org [accessed 18 January 2009]

Dov Te'eni et al. (2007) *Human Computer Interaction: Developing Effective Organizational Information Systems,* John Wilen & Sons, Inc

Fazly, A. and Hirst, G. (2003) 'Testing the efficiency of part-of-speech information in word completion', *Proceedings of the Workshop on Language Modelling for Text Entry Methods, 11th Conference of the European Chapter of the Association for Computational Linguistics*, 9–16

Felipe S. Santos, Maria Graça Pimentel and Cesar A. C. Teixeira (2006) 'An architecture to improve the generalization of interacting device developments for accessibility', *Proceedings of the 12th Brazilian symposium on Multimedia and the web*, Pages: 53 – 60

Golding, A. R. (1995) 'A Bayesian hybrid method for context-sensitive spelling correction', *Proceedings of the Third Workshop on Very Large Corpora*, pp.39-53

Henry C. C. Tan and Liyanage C. De Silva (2003) 'Human Activity Recognition by Head Movement using Elman Network and Neuro-Markovian Hybrids' *Image and Vision Computing* New Zealand 2003

Hojjat Adeli and Shih-Lin Hung (1995) *Machine Learning – Neural Networks, Genetic Algorithms, and Fuzzy Systems*, John Wiley & Sons, Inc

Houman Pournasseh, Writing Win32 Multilingual User Interface Applications [online], Microsoft, available: http://www.microsoft.com/globaldev/handson/dev/muiapp.mspx [accessed 25 January 2009]

Howard Demuth, Mark Beale and Martin Hagan (1992–2008) *Neural Network Toolbox™ 6 User's Guide*, The MathWorks, Inc

Ingo Schellhammer, Joachim Diederich, Michael Towsey (1998) 'Knowledge extraction and recurrent neural networks: An analysis of an Elman network trained on a natural language learning task', *Proceedings of the Joint Conference on New Methods in Language Processing and Computational Natural Language Learning: NeMLaP3/CoNLL98', Association for Computational Linguistics*

I. Anagnostopoulos et al. (2003) 'A Probabilistic Neural Network for Human Face Identification based on Fuzzy Logic Chromatic Rules' *11th Mediterranean Conference in Control and Automation MED*, Rhodes, Greece (in CD-proceedings) 2003, June 18-20, 2003.

I. Scott MacKenzie and William A. S. Buxton (1992). 'Extending Fitts' law to two-dimensional tasks', *Proceedings of ACM CHI 1992 Conference on Human Factors in Computing Systems*, pp. 219–226

J Bilmes and K Kirchhoff (2003). "Factored Language Models and Generalized Parallel Backoff", *Human Language Technology Conference*

Jaro-Winkler distance [online], 11 January 2009, Wikipedia, available: http://en.wikipedia.org/wiki/Jaro-Winkler [accessed 23 January 2009]

Jeffrey L. Elman (1990), 'Finding structure in time' *Cognitive Science*, Vol. 14, 1990, pp. 179-211

Jianhua Li & Graeme Hirst. (2005) 'Semantic Knowledge in Word Completion', *Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, pp.121 – 128

Jiawei Han & Micheline Kamber (2001) *Data Mining – Concepts and Techniques* Academic Press

Johan A. Du Preez, E. Barnard, D. M. Weber (1998). 'Efficient High-order hidden Markov modelling', *Proceedings of the International Conference on Spoken Language Processing*

John G. Cleary, W. J. Teahan, et al. (1995) 'Unbounded length contexts for PPM', *IEEE Computer Society Press*

[5] John McCarthy. (2007) *What is Artificial Intelligence?* [Online], 2007-11-12, available: http://www-formal.stanford.edu/jmc/whatisai/node1.html [accessed 19 January 2008]

Juan Antonio Pérez-Ortiz, Jorge Calera-Rubio and Mikel L. Forcada (2001) 'Online Symbolic-Sequence Prediction with Discrete-Time Recurrent Neural Networks', *Proceedings of the International Conference on Artificial Neural Networks*, p719 – 724

Juan Antonio Perez-ortiz et al (2002) 'Improving Long-Term Online Prediction with Decoupled Extended Kalman Filters' Artificial Neural Networks — ICANN 2002, Volume 2415/2002, Page 134

Jun Li, Karim Ouazzane and Marielle Brouwer (2008). 'A hybrid framework towards the solution for people with disability effectively using computer keyboard', IADIS International Conference Intelligent Systems and Agents 2008, pp. 209-212

Justin Boyan, Dayne Freitag and Thorsten Joachims (1996) 'A machine learning architecture for optimizing web search engines', *AAAI Workshop on Internet-based Information Systems*

J. G. Carbonell (1989) *Machine Learning – Paradigms and Methods*, Elsevierr Science Publishers B.V., Amsterdam, The Netherlands

J.Ross Quinlan (1993) *C4.5 Programs for Machine Learning*, Morgan Kaufmann Publishers

Kane, S.K. and Wobbrock, J.O. (2007) 'Automatically correcting typing errors for people with motor impairments'. *Adjunct Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '07).* Newport, Rhode Island (October 7-10, 2007). New York: ACM Press, pp. 59-60

Kane, S.K., Wobbrock, J.O., Harniss, M. and Johnson, K.L. (2008) 'TrueKeys: Identifying and correcting typing errors for people with motor impairments', *Proceedings of the ACM Conference on Intelligent User Interfaces (IUI '08).* Maspalomas, Gran Canaria, Spain (January 13-16, 2008). New York: ACM Press, pp. 349-352

Karen Kukich. (1992) 'Technique for automatically correcting words in text', *ACM Computing Surveys (CSUR)*, Vol.24, no.4, pp.377-439

Kevin Gurney (1997) *An introduction to neural networks*, UCL press

Laura R. Novick and Steven J. Sherman(2004) 'Type-based bigram frequencies for five-letter words', *Behaviour Research Methods, Instruments, & Computers 2004*, 36(3), 397-401

Luz Abril Torres Méndez, *Viterbi Algorithm in Text Recognition* [online], available: http://www.cim.mcgill.ca/~latorres/Viterbi/va_main.html [accessed 26 January 2009]
L. Allison, 24 January 2009 , *Tries* [online], Faculty of Information Technology (Clayton), Monash University, Australia, available: http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Tree/Trie/ [accessed 23 January 2009]

Mark Nelson. *Arithmetic Coding + Statistical Modelling = Data Compression* [online], September 2007, Mark Nelson's HomePage, available: http://www.dogma.net/markn/articles/arith/part1.htm [access 24 January 2008]

Masami Nakamura et al (1990), 'Neural network approach to word category prediction for English texts', *Helsinki University* 213—218

Matthew H. Austern (2002) *Generic Programming and the STL – Using and Extending the C++ Standard Template Library*, Addison-Wesley, Addison Wesley Longman, Inc

Matt Davis, *reading jumbled texts* [online], 30 October 03, available: http://www.mrc-cbu.cam.ac.uk/~mattd/Cmabrigde/ [accessed 26 January 2009]

Michael John Collins (1996). 'A New Statistical Parser Based on Bigram Lexical Dependencies'. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*

Michael J.A. Berry and Gordon S.Linoff (2004) *Data Mining Techniques – For Marketing, Sales and Customer Relationship Management 2nd edition*, Wiley Publishing, Inc

Michael Negnevitsky (2005) *Artificial Intelligence – A Guide to Intelligent Systems 2nd edition* Addison-Wesley, Pearson Education

Michael N. Jones and D. J. K. Mewhort (2004) 'Case-sensitive letter and bigram frequency counts from large-scale English corpora', *Behaviour Research Methods, Instruments, & Computers 2004*, 36(3), 388-396

Michael Towsey et al (1998), 'Natural language learning by recurrent neural networks: a comparison with probabilistic approaches', *Proc. Joint Conf. on New Methods in Language Processing and Computational Natural Language Learning*

Moffat. A. (1990) 'Implementing the PPM data compression scheme', *IEEE Transactions on Communications*, 38(11), 1917-1921

Moses Liskov, *Finite Automata and Theory of Computation* [online], available: http://www.cs.wm.edu/~mliskov/f07_cs423/index.html [accessed 26 January 2009]

Möller S, Kriventseva EV, Apweiler R. (2000) 'A collection of well characterised integral membrane proteins', Bioinformatics, 2000 Dec, 16(12):1159-60, and dataset website – www.ebi.ac.uk

Nestor Garay-Vitoria and Julio González-Abascal (1997) 'Intelligent word-prediction to enhance text input rate (a syntactic analysis-based word-prediction aid for people with severe motor and speech disability)', *Proceedings of the 2nd international conference on intelligent user interfaces*, Pages: 241 – 244

Nikolay Y. Nikolaev , 1 October 2008, *Probabilistic Neural Networks* [online], Goldsmiths, University of London. Available: http://homepages.gold.ac.uk/nikolaev/311pnn.htm [accessed 23 January 2009]

Paul M. Fitts (1954). 'The information capacity of the human motor system in controlling the amplitude of movement', *Journal of Experimental Psychology*, volume 47, number 6, June 1954, pp. 381-391

Premit Patel (2007) 'An Automated Visual Speech Reading System', *PhD Thesis – London Metropolitan University*

Robert Edward Lewand, *Relative Frequencies of Letters in General English Plain text* [online], Available: http://pages.central.edu/emp/LintonT/classes/spring01/cryptography/letterfreq.html [accessed 23 January 2009]

Schneier, B (1996). *Applied Cryptography, Second edition*, page 234. John Wiley and Sons

Schwenk H and Gauvain J, 'Connectionist Language Modeling for Large Vocabulary Continuous Speech Recognition'. *Proceedings of ICASSP*, pages 765-768, Orlando, May 2002

Serengul Smith-Atakan (2006) *Human-Computer Interaction*, Thomson Learning

Shannon, Claude E. (1950). *Prediction and entropy of printed English.* The Bell System Technical Journal, 30:50-64, 1950

Shari Trewin. (2002) 'An invisible keyguard', *ACM SIGACCESS Conference on Assistive Technologies*, pp.143 – 149

Shari Trewin. (2003) 'Automating Accessibility: The Dynamic Keyboard', *ACM SIGACCESS Accessibility and Computing, SESSION: Accessibility infrastructure and supporting tools*, no.77-78, pp.71-78

Shari Trewin and Helen Pain (1998) 'A Model of Keyboard Configuration Requirements', *International {ACM} Conference on Assistive Technologies*, pp. 173-181

Simon Haykin, (1999) *Neural Networks – A comprehensive Foundation 2nd ed.* Tom Robbins

Soukoreff, R. W., & MacKenzie, I. S. (2001). 'Measuring errors in text entry tasks: An application of the Levenshtein string distance statistic', *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems - CHI 2001*, p. 319-320. New York: ACM

Soukoreff, R. W., & MacKenzie, I. S. (2003). 'Input-based language modelling in the design of high performance text input techniques'. Proceedings of Graphics Interface 2003, pp. 89-96

Stamatios V. Kartalopoulos (1996) *Understanding Neural Networks and Fuzzy Logic*. The Institute of Electrical and Electronics Engineers, Inc

Steve Abbott 'EDBASE 2 – Intelligent Keyboard System, Inclusive Access to Database Applications' [online] available: http://www.disability.bcs.org.uk /docs/icat03/EDBASE2.pdf [accessed 17 April 2009]

S. Kotsiantis, D. Kanellopoulos, P. Pintelas (2006), 'Data Preprocessing for Supervised Leaning', *International Journal of Computer Science*, 2006, Vol 1 N. 2, pp 111-117

S.K. Card, W.K. English, & B.J. Burr (1978), 'Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT', Ergonomics, 21, 601-613

Tejpal Singh Chhabra, 28 Mar 2006, *Back-propagation Neural Net* [online], The Code Project, available: http://www.codeproject.com/KB/recipes/BP.aspx [accessed 23 January 2008]

Todor Ganchev1 et al. (2002) 'probabilistic neural networks combined with gmms for speaker recognition over telephone channels' *14th international conference on digital signal processing (DSP2002)*, Volume II, pp.1081-1084 July 1-3, 2002, Santorini, Greece 1081

Tom M. Mitchell (1997). *Machine Learning*, McGraw-Hill, Singapore

Toshiyuki Masui. (1999) 'POBox: An efficient text input method for handheld and ubiquitous computers', *Lecture Notes in Computer Science (1707), Handheld and Ubiquitous Computing*, pp. 289-300

Toshiyuki Masaui and Ken Nakayama (1994) 'Repeat and Predict: Two keys to Efficient Text Edition', *Proceedings of the Conference on Human Factors in Computing Systems*, pp. 118—123

Victoria J. Hodge and Jim Austin (2003) 'A Comparison of Standard Spell Checking, Algorithms and a Novel Binary Neural Approach', *IEEE Transactions on Knowledge and Data Engineering*, vol.15, no.5, pp.1073-1081

William Soukoreff and Scott MacKenzie, n.d. *KeyCapture* [online], available: http://dynamicnetservices.com/~will/academic/textinput/keycapture/ [accessed 18 January 2009]

William J. Palm III (2001) *Introduction to MATLAB 6 for Engineers*, McGraw-Hill/Irwin

W. Woo, J. Park and Y. Iwadate (2000) 'Emotion Analysis from Dance Performance Using Time-Delay Neural Networks' *Proc. JCIS-CVPRIP'00*, vol.2, pp. 374-377, Feb. 2000.

Zhai, S., Hunter, M., Smith, B.A. (2000), 'The Metropolis Keyboard: An Exploration of Quantitative Techniques for Virtual Keyboard Design'. *Proceedings of ACM Symposium on User Interface Software and Technology (UIST 2000)*, pp. 119-128

Computer Access Resources 2007 Catalogue, pp. 4-15. www.keytools.co.uk

Artificial neural network [online], 8 April 2009, Wikipedia, available: http://en.wikipedia.org/wiki/Artificial_neural_network [accessed 11 April 2009]

Artificial Neural Networks Technology [online], n.d. ITT Corporation, available: https://www.dacs.dtic.mil/techs/neural/neural7.php [accessed 23 January 2009]

BackPropagation [online], 10 January 2008, Wikipedia, available: http://en.wikipedia.org/wiki/Back_propagation [accessed 20 January 2008]

Basic Concepts for Neural Networks [online], last revised October 2003, Cheshire Engineering Corporation, available: http://www.cheshireeng.com/Neuralyst/nnbg.htm [accessed 23 January 2008]

Bigram [online], 19 September 2008, Wikipedia, available: http://en.wikipedia.org/wiki/Bigram [accessed 12 March 2008]

Bow: A Toolkit for Statistical Language Modelling, Text Retrieval, Classification and Clustering [online], 12 September 1998, available: http://www.cs.cmu.edu/~mccallum/bow/ [accessed 05 February 2009]

Cheat Sheet: Unicode-enabling Microsoft C/C++ Source Code [online] 13 November 2003 available: http://www.i18nguy.com/unicode/c-unicode.html [accessed 26 January 2009]

C – ODBC from C Tutorial Part 1 and Part 2 [online] n.d. available: http://www.easysoft.com/developer/languages/c/ [accessed 25 January 2009]

Data compression theory and algorithms [online], 18-December-2007, Maximum Compression, Available: http://www.maximumcompression.com /algoritms.php [accessed 24 January 2008]

Data preprocessing [online], 01 April 2005, SearchSQLServer.com, available: http://searchsqlserver.techtarget.com/sDefinition/0,,sid87_gci810056,00.html [accessed 23 January 2009]

Data Pre-processing [online], 16 January 2009, Wikipedia, available: http://en.wikipedia.org/wiki/Data_Pre-processing [accessed 23 January 2009]

Documentation - Neural Network Toolbox [online], The MathWorks, available: http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/ [accessed 23 January 2009]

English Text compression test [online], 18 December 2007, Maximum Compression-Lossless data compression software benchmarks / comparisons, Available: http://www.maximumcompression.com/data/text.php [accessed 25 January 2008]

Entropy encoding [online], 7 September 2008, Wikipedia, available: http://en.wikipedia.org/wiki/Entropy_encoding [accessed 18 January 2009]

Error [online], 18 January 2009, Wikipedia, available: http://en.wikipedia.org/wiki/Error [accessed 18 January 2009]

Factored language model [online], 2 August 2008, Wikipedia, available: http://en.wikipedia.org/wiki/Factored_language_model [accessed 23 January 2009]

Focused Time-Delay Neural Network (newfftd) [online], The MathWorks, available: http://www.mathworks.com/access/helpdesk/help/toolbox/nnet /dynamic3.html [accessed 23 January 2009]

Hidden Markov model [online], 25 January 2009, Wikipedia, available: http://en.wikipedia.org/wiki/Hidden_Markov_model [accessed 26 January 2009]

Knowledge Transfer Partnership, available: http://www.ktponline.org.uk/ [accessed 18 January 2009]

Language modelling [online], The Inference Group, available: http://www.inference.phy.cam.ac.uk/is/papers/ [accessed 25 January 2009]

Levenshtein algorithm [online], available: http://www.levenshtein.net/ [accessed 23 January 2009]

Machine learning [online], 15 January 2008, Wikipedia, available: http: //en.wikipedia.org/wiki/Machine_learning [accessed 19 January 2008]

Markov chain [online], 20 January 2009, Wikipedia, available: http://en.wikipedia.org/wiki/Markov_chain [accessed 23 January 2009]

Metaphone [online], 18 October 2008, Wikipedia, available: http://en.wikipedia.org/wiki/Metaphone [accessed 23 January 2009]

Neural net language models [online], 19 April 2008, Wikipedia, available: http://www.scholarpedia.org/article/Neural_net_language_models [accessed 23 January 2009]

Neural Networks Classification [online], available: http://www.resample.com/ xlminer/help/NNC/NNClass_intro.htm [accessed 07 January 2008]

N-gram [online], 26 November 2008, Wikipedia, available: http://en.wikipedia.org/wiki/N-gram [accessed 23 January 2009]

PAQ, [online], 3 January 2008, Wikipedia, Available: http://en.wikipedia.org /wiki/PAQ [accessed 25 January 2008]

Prototype, [online], n.d., available: http://www.sensorysoftware.com/ prototype.html [accessed 03 March 2008]

Spell checker [online], 03 March 2008, Wikipedia, Available: http://en.wikipedia.org/wiki/Spell_checker [accessed 03 March 2008]

The Dasher Project, [online], 14 Nov. 2007, Inference Group of Cambridge, available: http://www.inference.phy.cam.ac.uk/dasher/ [accessed 03 March 2008]

Thomas Hardy [online], 21 January 2009, Wikipedia, available: http://en.wikipedia.org/wiki/Thomas_Hardy [accessed 22 January 2009]

T9 (predictive text) [online], 03 March 2008, Wikipedia, Available: http://en.wikipedia.org/wiki/T9_%28predictive_text%29 [accessed 03 March 2008]

What is cross-entropy, and why use it? [online], School of Science and Engineering, UNSW, available: http://www.cse.unsw.edu.au/~billw/cs9444/ crossentropy.html [accessed 23 January 2009]

What is Statistical Language Modelling (SLM) [online], School of Informatics, The University of Edinburgh, available: http://homepages.inf.ed.ac.uk /s0450736/slm.html [accessed 23 January 2009]

Win32 Multilingual IME Overview for IME Development [online], 11 April 2003, available: http://www.osronline.com/ddkx/appendix/imeimes_0h2s.htm [accessed 25 January 2009]

Virtual key codes [online], available: http://api.farmanager.com/en /winapi/virtualkeycodes.html [accessed 05 February 2009]

APPENDICES

Appendix A

MATLAB SOURCE CODE

MATLAB – Backpropagation2.m

```
% ------------------------------------------------------------- %
% Function: NN with n-gram delay for key press prediction       %
% Author:   Jun Li                                              %
% Create:   02/08/2008                                          %
% Comment:  Programming is referred to LongKeyPress.m           %
% ------------------------------------------------------------- %

function Backpropagation2(GRAM, HiddenNeurons)
% Turn on echoing of commands inside the script-file.
%echo off
% Clear command window.
%clc

%clear all
%close all

% Define GRAM
%GRAM = 2-1;
InputNeurons = 27;
OutputNeurons = 27;
%HiddenNeurons = 5;

%Initialization
load ./data/BPdataP.txt;
Z = BPdataP';
si = size(Z, 2);
clear BPdataP;

load ./data/BPdataPValidation.txt;
V = BPdataPValidation';
Vsi = size(V, 2);
VV.P = con2seq(V(:,GRAM:(Vsi-1)));
VV.T = con2seq(V(:,(GRAM+1):Vsi));
clear BPdataPValidation;
load ./data/BPdataPTest.txt;
Test = BPdataPTest';
Tsi = size(Test, 2);
TV.P = con2seq(Test(:,GRAM:(Tsi-1)));
TV.T = con2seq(Test(:,(GRAM+1):Tsi));
clear BPdataPTest;

PR = zeros(InputNeurons, 2);
for i = 1: InputNeurons
    PR(i, :) = [0, 1];
end

ID = zeros(1, GRAM);
for i = 1: GRAM
    ID(i) = i-1;
end

%Pi - the initial inputs for which the network will not have a target
%P - the inputs for which the network will have a target
%T - the targets
```

```
%Pi = con2seq(Z(:,1:(GRAM-1)));
%P = con2seq(Z(:,GRAM:(si-1)));
%T = con2seq(Z(:,(GRAM+1):si));

net = newfftd(PR,ID,[HiddenNeurons OutputNeurons],{'purelin'
'logsig'}, ...
    'trainrp');

net.trainparam.epochs=50;
%net = train(net,P,T,Pi);
net = train(net,con2seq(Z(:,GRAM:(si-1))),con2seq(Z(:,(GRAM+1):si)),...
con2seq(Z(:,1:(GRAM-1))), [], VV, []);

%Y=sim(net,con2seq(Z(:,GRAM:(si-1))),con2seq(Z(:,1:(GRAM-1))));
%T = Z(:,(GRAM+1):si);

%if use Test sample
Y=sim(net,TV.P);
si = Tsi;
T = Test(:,(GRAM+1):Tsi);

% Strike any key to continue...
%pause;

%Post-processing...
SymbolFreq = zeros(1, OutputNeurons);
for i = 1: OutputNeurons
    SymbolFreq(i) = 0.00;
end

Ybackup = Y;
%---------Hit-Rate-1----------
for i = 1: (si - GRAM)
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    for j = 1:OutputNeurons
        SymbolFreq(j) = SymbolFreq(j) + Y{i}(j);
        Y{i}(j) = 0;
    end
    Y{i}(Idx) = 1;
end
%Convert Cell to Numeric Array
Y = [Y{:}];
%Comparison between Y and T
Result = Y - T;
%Number of right Prediction
Num1 = 0;

%alphabetY1,2,3 will receive discrete results.
alphabetY1 = zeros(1, 27);
alphabetY2 = zeros(1, 27);
alphabetY3 = zeros(1, 27);
alphabetT = zeros(1, 27);
for i  = 1:(si-GRAM)
    if(any(Result(:, i)) == 0)
        Num1 = Num1 + 1;
    end
```

```
    Idx = find(Y(:,i)==1);
    alphabetY1(Idx) = alphabetY1(Idx) + 1;
    Idx = find(T(:,i)==1);
    alphabetT(Idx) = alphabetT(Idx) + 1;
end

%---------Hit-Rate-2----------
Y = Ybackup;
for i = 1: (si-GRAM)
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    Y{i}(Idx) = 0;
    %find second maximum
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    for j = 1:OutputNeurons
        Y{i}(j) = 0;
    end
    Y{i}(Idx) = 1;
end
%Convert Cell to Numeric Array
Y = [Y{:}];
%Comparison between Y and T
Result = Y - T;
%Number of right Prediction
Num2 = 0;
for i  = 1:(si-GRAM)
    if(any(Result(:, i)) == 0)
        Num2 = Num2 + 1;
    end
    Idx = find(Y(:,i)==1);
    alphabetY2(Idx) = alphabetY2(Idx) + 1;
end

%---------Hit-Rate-3----------
Y = Ybackup;
for i = 1: (si-GRAM)
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    Y{i}(Idx) = 0;
    %find second maximum
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    Y{i}(Idx) = 0;
    %find third maximum
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    for j = 1:OutputNeurons
        Y{i}(j) = 0;
    end
    Y{i}(Idx) = 1;
end
%Convert Cell to Numeric Array
Y = [Y{:}];
%Comparison between Y and T
Result = Y - T;
%Number of right Prediction
```

```
Num3 = 0;
for i  = 1:(si-GRAM)
    if(any(Result(:, i)) == 0)
        Num3 = Num3 + 1;
    end
    Idx = find(Y(:,i)==1);
    alphabetY3(Idx) = alphabetY3(Idx) + 1;
end

%Write file...
fid = fopen('Backpropagation2.txt', 'at');
fprintf(fid, '\n------%d-GRAM------%d-HiddenNeurons-------\n', ...
            GRAM, HiddenNeurons);
%Hit Rate level-1,2,3
fprintf(fid, 'Num1=[%d]\nNum2=[%d]\nNum3=[%d]\n', Num1, Num2, Num3);
fprintf(fid, 'Num =[%d]\n', (si-GRAM));
fprintf(fid, 'Num1/Num=[%f]\nNum2/Num=[%f]\nNum3/Num=[%f]\n',...
            Num1/(si-GRAM), Num2/(si-GRAM), Num3/(si-GRAM));
%Alphabet Hit Rate level-1,2,3
for i  = 1:27
    fprintf(fid, 'alphabetY1-[%d]=      %d\n', i, alphabetY1(i));
end
for i  = 1:27
    fprintf(fid, 'alphabetY2-[%d]=      %d\n', i, alphabetY2(i));
end
for i  = 1:27
    fprintf(fid, 'alphabetY3-[%d]=      %d\n', i, alphabetY3(i));
end
Entropy1 = 0;
for i  = 1:27
    Entropy1 = Entropy1 + (alphabetT(i)/(si-GRAM)) * ...
                log2(alphabetT(i)/(si-GRAM));
    fprintf(fid, 'alphabetT-[%d]=       %d\n', i, alphabetT(i));
end
SFaddup = 0;
for i = 1: OutputNeurons
    SFaddup = SFaddup + SymbolFreq(i);
    fprintf(fid, 'SymbolFreq(%d)=       %f\n', i, SymbolFreq(i));
end
fprintf(fid, 'SFaddup=[%f]\n', SFaddup);
Entropy2 = 0;
for i = 1: OutputNeurons
    Entropy2 = Entropy2 + (SymbolFreq(i)/SFaddup) * ...
            log2(SymbolFreq(i)/SFaddup);
    fprintf(fid, 'SymbolFreq(%d)/SFaddup=       %f\n', i,
SymbolFreq(i)/SFaddup);
    fprintf(fid, 'alphabetT(%d)/(si-GRAM)=      %f\n', i,
alphabetT(i)/(si-GRAM));
end
Entropy1 = abs(Entropy1);
Entropy2 = abs(Entropy2);
fprintf(fid, 'Entropy1=[%f]\nEntropy2=[%f]\n', Entropy1, Entropy2);
fclose(fid);

% Create a menu, so the user can select a test set.
% K = MENU('Choose a file resolution','Test A','Test 0','Test 5','Test
L','Test V','Test W','Test H','Test 1','Test GB');
```

MATLAB – CallBackpropagation2.m

```
% ------------------------------------------------------------------- %
% Function: Call Backpropagation2                                     %
% Author:   Jun Li                                                    %
% Create:   24/11/2008                                                %
% Comment:                                                            %
% ------------------------------------------------------------------- %

% Turn on echoing of commands inside the script-file.
echo off;
% Clear command window.
%clc
clear all;
close all;

%MaxNumberGRAM = 10;
%GRAMarray = [1, 2, 3, 5, 7, 9, 13, 17, 25, 35];

%MaxNumberHiddenNeurons = 10;
%HiddenNeuronsArray = [1, 2, 3, 5, 7, 9, 15, 25, 50, 100];

%start GRAM-3, HN-15
MaxNumberGRAM = 1;
GRAMarray = [11, 13];

MaxNumberHiddenNeurons = 10;
HiddenNeuronsArray = [1, 2, 3, 5, 7, 9, 15, 25, 50, 100];

% Define GRAM and HiddenNeurons
for i = 1: MaxNumberGRAM
    GRAM = GRAMarray(i);
    for j = 1: MaxNumberHiddenNeurons
        HiddenNeurons = HiddenNeuronsArray(j);
        disp(sprintf('Backpropagation2...GRAM-[%d]...HiddenNeurons-
[%d].....',...
            GRAM, HiddenNeurons));
        Backpropagation2(GRAM, HiddenNeurons);
    end
end

%Backpropagation2(2, 5);
```

MATLAB – GenerateNoise.m

```
% ------------------------------------------------------------- %
% Funciton: Generate Noisy Data in { BPdataP.txt,               %
%                                    BPdataPValidation.txt      %
%                                    BPdataPTest.txt }          %
% Author:   Jun Li                                             %
% Create:   28/11/2008                                         %
% ------------------------------------------------------------- %
clear all
close all

rand('twister',sum(100*clock));
load ./data/BPdataP.txt;
si1 = size(BPdataP, 1);
load ./data/BPdataPValidation.txt;
si2 = size(BPdataPValidation, 1);
load ./data/BPdataPTest.txt;
si3 = size(BPdataPTest, 1);

%Noise Rate
NR = 0.3;

n = (si1+si2+si3);
m = round(NR*(si1+si2+si3));
r1 = iRand(n, m);

rand('twister',sum(100*clock));
%27 the number of symbols = input neurons
r2 = iRand(27, m);
for i = 1:m
    if r1(i) <= si1
        Idx = find(BPdataP(r1(i), :) == 1);
        BPdataP(r1(i), Idx) = 0;
        BPdataP(r1(i), r2(i)) = 1;
    elseif r1(i) <= (si1+si2)
        Idx = find(BPdataPValidation((r1(i)-si1), :) == 1);
        BPdataPValidation((r1(i)-si1), Idx) = 0;
        BPdataPValidation((r1(i)-si1), r2(i)) = 1;
    else
        Idx = find(BPdataPTest((r1(i)-si1-si2), :) == 1);
        BPdataPTest((r1(i)-si1-si2), Idx) = 0;
        BPdataPTest((r1(i)-si1-si2), r2(i)) = 1;
    end
end

%Write file...
dlmwrite('B2NoiseP.txt', BPdataP, ' ');
dlmwrite('B2NoiseV.txt', BPdataPValidation, ' ');
dlmwrite('B2NoiseT.txt', BPdataPTest, ' ');
```

MATLAB – B2Noise.m

```
% ------------------------------------------------------------- %
% Function: NN with n-gram delay for key press prediction       %
% Author:   Jun Li                                              %
% Create:   02/08/2008                                          %
% Comment:  Programming is referred to LongKeyPress.m           %
% ------------------------------------------------------------- %

function B2Noise(GRAM, HiddenNeurons)
% Turn on echoing of commands inside the script-file.
%echo off
% Clear command window.
%clc

%clear all
%close all

% Define GRAM
%GRAM = 2-1;
InputNeurons = 27;
OutputNeurons = 27;
%HiddenNeurons = 5;

%Initialization
load ./data/BPdataP.txt;
load ./B2NoiseP.txt;
B2NoiseP = B2NoiseP';
Z = BPdataP';
si = size(Z, 2);
clear BPdataP;

load ./data/BPdataPValidation.txt;
load B2NoiseV.txt;
B2NoiseV = B2NoiseV';
V = BPdataPValidation';
Vsi = size(V, 2);
VV.P = con2seq(B2NoiseV(:,GRAM:(Vsi-1)));
VV.T = con2seq(V(:,(GRAM+1):Vsi));
clear BPdataPValidation;

load ./data/BPdataPTest.txt;
load ./B2NoiseT.txt;
B2NoiseT = B2NoiseT';
Test = BPdataPTest';
Tsi = size(Test, 2);
TV.P = con2seq(B2NoiseT(:,GRAM:(Tsi-1)));
TV.T = con2seq(Test(:,(GRAM+1):Tsi));
clear BPdataPTest;

PR = zeros(InputNeurons, 2);
for i = 1: InputNeurons
    PR(i, :) = [0, 1];
end

ID = zeros(1, GRAM);
```

```
for i = 1: GRAM
    ID(i) = i-1;
end

%Pi - the initial inputs for which the network will not have a target
%P - the inputs for which the network will have a target
%T - the targets
%Pi = con2seq(Z(:,1:(GRAM-1)));
%P = con2seq(Z(:,GRAM:(si-1)));
%T = con2seq(Z(:,(GRAM+1):si));

net = newfftd(PR,ID,[HiddenNeurons OutputNeurons],{'purelin'
'logsig'}, ...
    'trainrp');

net.trainparam.epochs=100;
%net = train(net,P,T,Pi);
net = train(net,con2seq(B2NoiseP(:,GRAM:(si-
1))),con2seq(Z(:,(GRAM+1):si)),...
con2seq(B2NoiseP(:,1:(GRAM-1))), [], VV, []);

%Y=sim(net,con2seq(Z(:,GRAM:(si-1))),con2seq(Z(:,1:(GRAM-1))));
%T = Z(:,(GRAM+1):si);

%if use Test sample
Y=sim(net,TV.P);
si = Tsi;
T = Test(:,(GRAM+1):Tsi);

% Strike any key to continue...
%pause;

%Post-processing...
SymbolFreq = zeros(1, OutputNeurons);
for i = 1: OutputNeurons
    SymbolFreq(i) = 0.00;
end

Ybackup = Y;
%---------Hit-Rate-1----------
for i = 1: (si - GRAM)
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    for j = 1:OutputNeurons
        SymbolFreq(j) = SymbolFreq(j) + Y{i}(j);
        Y{i}(j) = 0;
    end
    Y{i}(Idx) = 1;
end
%Convert Cell to Numeric Array
Y = [Y{:}];
%Comparison between Y and T
Result = Y - T;
%Number of right Prediction
Num1 = 0;

%alphabetY1,2,3 will receive discrete results.
```

```
alphabetY1 = zeros(1, 27);
alphabetY2 = zeros(1, 27);
alphabetY3 = zeros(1, 27);
alphabetT = zeros(1, 27);
for i  = 1:(si-GRAM)
    if(any(Result(:, i)) == 0)
        Num1 = Num1 + 1;
    end
    Idx = find(Y(:,i)==1);
    alphabetY1(Idx) = alphabetY1(Idx) + 1;
    Idx = find(T(:,i)==1);
    alphabetT(Idx) = alphabetT(Idx) + 1;
end
```

```
%---------Hit-Rate-2----------
Y = Ybackup;
for i = 1: (si-GRAM)
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    Y{i}(Idx) = 0;
    %find second maximum
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    for j = 1:OutputNeurons
        Y{i}(j) = 0;
    end
    Y{i}(Idx) = 1;
end
%Convert Cell to Numeric Array
Y = [Y{:}];
%Comparison between Y and T
Result = Y - T;
%Number of right Prediction
Num2 = 0;
for i  = 1:(si-GRAM)
    if(any(Result(:, i)) == 0)
        Num2 = Num2 + 1;
    end
    Idx = find(Y(:,i)==1);
    alphabetY2(Idx) = alphabetY2(Idx) + 1;
end
```

```
%---------Hit-Rate-3----------
Y = Ybackup;
for i = 1: (si-GRAM)
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    Y{i}(Idx) = 0;
    %find second maximum
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    Y{i}(Idx) = 0;
    %find third maximum
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    for j = 1:OutputNeurons
        Y{i}(j) = 0;
```

```
    end
    Y{i}(Idx) = 1;
end
%Convert Cell to Numeric Array
Y = [Y{:}];
%Comparison between Y and T
Result = Y - T;
%Number of right Prediction
Num3 = 0;
for i  = 1:(si-GRAM)
    if(any(Result(:, i)) == 0)
        Num3 = Num3 + 1;
    end
    Idx = find(Y(:,i)==1);
    alphabetY3(Idx) = alphabetY3(Idx) + 1;
end

%Write file...
fid = fopen('B2Noise.txt', 'at');
fprintf(fid, '\n------%d-GRAM------%d-HiddenNeurons-------\n', ...
            GRAM, HiddenNeurons);
%Hit Rate level-1,2,3
fprintf(fid, 'Num1=[%d]\nNum2=[%d]\nNum3=[%d]\n', Num1, Num2, Num3);
fprintf(fid, 'Num =[%d]\n', (si-GRAM));
fprintf(fid, 'Num1/Num=[%f]\nNum2/Num=[%f]\nNum3/Num=[%f]\n',...
            Num1/(si-GRAM), Num2/(si-GRAM), Num3/(si-GRAM));
%Alphabet Hit Rate level-1,2,3
for i  = 1:27
    fprintf(fid, 'alphabetY1-[%d]=      %d\n', i, alphabetY1(i));
end
for i  = 1:27
    fprintf(fid, 'alphabetY2-[%d]=      %d\n', i, alphabetY2(i));
end
for i  = 1:27
    fprintf(fid, 'alphabetY3-[%d]=      %d\n', i, alphabetY3(i));
end
Entropy1 = 0;
for i  = 1:27
    Entropy1 = Entropy1 + (alphabetT(i)/(si-GRAM)) * ...
                log2(alphabetT(i)/(si-GRAM));
    fprintf(fid, 'alphabetT-[%d]=       %d\n', i, alphabetT(i));
end
SFaddup = 0;
for i = 1: OutputNeurons
    SFaddup = SFaddup + SymbolFreq(i);
    fprintf(fid, 'SymbolFreq(%d)=       %f\n', i, SymbolFreq(i));
end
fprintf(fid, 'SFaddup=[%f]\n', SFaddup);
Entropy2 = 0;
for i = 1: OutputNeurons
    Entropy2 = Entropy2 + (SymbolFreq(i)/SFaddup) * ...
            log2(SymbolFreq(i)/SFaddup);
    fprintf(fid, 'SymbolFreq(%d)/SFaddup=       %f\n', i,
SymbolFreq(i)/SFaddup);
    fprintf(fid, 'alphabetT(%d)/(si-GRAM)=      %f\n', i,
alphabetT(i)/(si-GRAM));
end
```

```
Entropy1 = abs(Entropy1);
Entropy2 = abs(Entropy2);
fprintf(fid, 'Entropy1=[%f]\nEntropy2=[%f]\n', Entropy1, Entropy2);

fclose(fid);

% Create a menu, so the user can select a test set.
% K = MENU('Choose a file resolution','Test A','Test 0','Test 5','Test
L','Test V','Test W','Test H','Test 1','Test GB');
```

MATLAB – CallB2Noise.m

```
% ------------------------------------------------------------------- %
% Function: Call Backpropagation2                                     %
% Author:   Jun Li                                                    %
% Create:   24/11/2008                                                %
% Comment:                                                            %
% ------------------------------------------------------------------- %

% Turn on echoing of commands inside the script-file.
echo off;
% Clear command window.
%clc
clear all;
close all;

MaxNumberGRAM = 4;
GRAMarray = [2, 3, 5, 7];

MaxNumberHiddenNeurons = 2;
HiddenNeuronsArray = [50, 100];

NR = 0.05;
fid = fopen('B2Noise.txt', 'at');
fprintf(fid, '\n\n<< << << << << << << NOISE RATE [%f] >> >> >> >> >> >>\n',
NR);
fclose(fid);
% Define GRAM and HiddenNeurons
for i = 1: MaxNumberGRAM
    GRAM = GRAMarray(i);
    for j = 1: MaxNumberHiddenNeurons
        HiddenNeurons = HiddenNeuronsArray(j);
        disp(sprintf('Backpropagation2...GRAM-[%d]...HiddenNeurons-
[%d]......',...
            GRAM, HiddenNeurons));
        B2Noise(GRAM, HiddenNeurons);
    end
end

%Backpropagation2(2, 5);
```

MATLAB – B2Helpline.m

```
% ------------------------------------------------------------------ %
% Function: NN with n-gram delay for Helpline Data prediction        %
% Author:   Jun Li                                                   %
% Create:   03/12/2008                                               %
% ------------------------------------------------------------------ %

function B2Helpline(GRAM, HiddenNeurons)
% Turn on echoing of commands inside the script-file.
%echo off
% Clear command window.
%clc

%clear all
%close all

InputNeurons = 53;
OutputNeurons = 53;

%Initialization
load ./Stroke.txt;
Z = Stroke';
si = size(Z, 2);
clear Stroke;

load ./StrokeTest.txt;
Test = StrokeTest';
Tsi = size(Test, 2);
TV.P = con2seq(Test(:,GRAM:(Tsi-1)));
TV.T = con2seq(Test(:,(GRAM+1):Tsi));
clear StrokeTest;

PR = zeros(InputNeurons, 2);
for i = 1: InputNeurons
    PR(i, :) = [0, 1];
end

ID = zeros(1, GRAM);
for i = 1: GRAM
    ID(i) = i-1;
end

%Pi - the initial inputs for which the network will not have a target
%P - the inputs for which the network will have a target
%T - the targets
%Pi = con2seq(Z(:,1:(GRAM-1)));
%P = con2seq(Z(:,GRAM:(si-1)));
%T = con2seq(Z(:,(GRAM+1):si));

net = newfftd(PR,ID,[HiddenNeurons OutputNeurons],{'purelin'
'logsig'}, ...
    'trainrp');

net.trainparam.epochs=100;
net = train(net,con2seq(Z(:,GRAM:(si-1))),con2seq(Z(:,(GRAM+1):si)),...
```

```
con2seq(Z(:,1:(GRAM-1))), [], [], []);

%Y=sim(net,con2seq(Z(:,GRAM:(si-1))),con2seq(Z(:,1:(GRAM-1))));
%T = Z(:,(GRAM+1):si);

%if use Test sample
Y=sim(net,TV.P);
si = Tsi;
T = Test(:,(GRAM+1):Tsi);

% Strike any key to continue...
%pause;

%Post-processing...
SymbolFreq = zeros(1, OutputNeurons);
for i = 1: OutputNeurons
    SymbolFreq(i) = 0.00;
end

Ybackup = Y;
%---------Hit-Rate-1----------
for i = 1: (si - GRAM)
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    for j = 1:OutputNeurons
        SymbolFreq(j) = SymbolFreq(j) + Y{i}(j);
        Y{i}(j) = 0;
    end
    Y{i}(Idx) = 1;
end
%Convert Cell to Numeric Array
Y = [Y{:}];
%Comparison between Y and T
Result = Y - T;
%Number of right Prediction
Num1 = 0;

%alphabetY1,2,3 will receive discrete results.
alphabetY1 = zeros(1, 53);
alphabetY2 = zeros(1, 53);
alphabetY3 = zeros(1, 53);
alphabetT = zeros(1, 53);
for i  = 1:(si-GRAM)
    if(any(Result(:, i)) == 0)
        Num1 = Num1 + 1;
    end
    Idx = find(Y(:,i)==1);
    alphabetY1(Idx) = alphabetY1(Idx) + 1;
    Idx = find(T(:,i)==1);
    alphabetT(Idx) = alphabetT(Idx) + 1;
end

%---------Hit-Rate-2----------
Y = Ybackup;
for i = 1: (si-GRAM)
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
```

```
    Y{i}(Idx) = 0;
    %find second maximum
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    for j = 1:OutputNeurons
        Y{i}(j) = 0;
    end
    Y{i}(Idx) = 1;
end
%Convert Cell to Numeric Array
Y = [Y{:}];
%Comparison between Y and T
Result = Y - T;
%Number of right Prediction
Num2 = 0;
for i  = 1:(si-GRAM)
    if(any(Result(:, i)) == 0)
        Num2 = Num2 + 1;
    end
    Idx = find(Y(:,i)==1);
    alphabetY2(Idx) = alphabetY2(Idx) + 1;
end

%----------Hit-Rate-3----------
Y = Ybackup;
for i = 1: (si-GRAM)
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    Y{i}(Idx) = 0;
    %find second maximum
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    Y{i}(Idx) = 0;
    %find third maximum
    Ma = max(Y{i});
    Idx = find(Y{i}==Ma);
    for j = 1:OutputNeurons
        Y{i}(j) = 0;
    end
    Y{i}(Idx) = 1;
end
%Convert Cell to Numeric Array
Y = [Y{:}];
%Comparison between Y and T
Result = Y - T;
%Number of right Prediction
Num3 = 0;
for i  = 1:(si-GRAM)
    if(any(Result(:, i)) == 0)
        Num3 = Num3 + 1;
    end
    Idx = find(Y(:,i)==1);
    alphabetY3(Idx) = alphabetY3(Idx) + 1;
end

%Write file...
fid = fopen('B2Helpline.txt', 'at');
```

```
fprintf(fid, '\n------%d-GRAM------%d-HiddenNeurons-------\n', ...
            GRAM, HiddenNeurons);
%Hit Rate level-1,2,3
fprintf(fid, 'Num1=[%d]\nNum2=[%d]\nNum3=[%d]\n', Num1, Num2, Num3);
fprintf(fid, 'Num =[%d]\n', (si-GRAM));
fprintf(fid, 'Num1/Num=[%f]\nNum2/Num=[%f]\nNum3/Num=[%f]\n',...
            Num1/(si-GRAM), Num2/(si-GRAM), Num3/(si-GRAM));
%Alphabet Hit Rate level-1,2,3
for i  = 1:53
    fprintf(fid, 'alphabetY1-[%d]=      %d\n', i, alphabetY1(i));
end
for i  = 1:53
    fprintf(fid, 'alphabetY2-[%d]=      %d\n', i, alphabetY2(i));
end
for i  = 1:53
    fprintf(fid, 'alphabetY3-[%d]=      %d\n', i, alphabetY3(i));
end
Entropy1 = 0;
for i  = 1:53
    if alphabetT(i)/(si-GRAM) ~= 0
        Entropy1 = Entropy1 + (alphabetT(i)/(si-GRAM)) * ...
                    log2(alphabetT(i)/(si-GRAM));
    end
    fprintf(fid, 'alphabetT-[%d]=       %d\n', i, alphabetT(i));
end
SFaddup = 0;
for i = 1: OutputNeurons
    SFaddup = SFaddup + SymbolFreq(i);
    fprintf(fid, 'SymbolFreq(%d)=       %f\n', i, SymbolFreq(i));
end
fprintf(fid, 'SFaddup=[%f]\n', SFaddup);
Entropy2 = 0;
for i = 1: OutputNeurons
    if SymbolFreq(i)/SFaddup ~= 0
        Entropy2 = Entropy2 + (SymbolFreq(i)/SFaddup) * ...
                log2(SymbolFreq(i)/SFaddup);
    end
    fprintf(fid, 'SymbolFreq(%d)/SFaddup=       %f\n', i,
SymbolFreq(i)/SFaddup);
    fprintf(fid, 'alphabetT(%d)/(si-GRAM)=      %f\n', i,
alphabetT(i)/(si-GRAM));
end
Entropy1 = abs(Entropy1);
Entropy2 = abs(Entropy2);
fprintf(fid, 'Entropy1=[%f]\nEntropy2=[%f]\n', Entropy1, Entropy2);

fclose(fid);
```

MATLAB – CallB2Helpline.m

```
% ------------------------------------------------------------------ %
% Function: Call B2Helpline                                          %
% Author:   Jun Li                                                   %
% Create:   03/12/2008                                               %
% Comment:                                                           %
% ------------------------------------------------------------------ %

% Turn on echoing of commands inside the script-file.
echo off;
% Clear command window.
%clc
clear all;
close all;

%MaxNumberGRAM = 10;
%GRAMarray = [1, 2, 3, 5, 7, 9, 13, 17, 25, 35];

%MaxNumberHiddenNeurons = 10;
%HiddenNeuronsArray = [1, 2, 3, 5, 7, 9, 15, 25, 50, 100];

%start GRAM-3, HN-15
%MaxNumberGRAM = 5;
%GRAMarray = [1, 3, 5, 7, 9];

%for GRAM-2
MaxNumberGRAM = 1;
GRAMarray = [2];

MaxNumberHiddenNeurons = 8;
HiddenNeuronsArray = [3, 5, 7, 9, 15, 25, 50, 100];

% Define GRAM and HiddenNeurons
for i = 1: MaxNumberGRAM
    GRAM = GRAMarray(i);
    for j = 1: MaxNumberHiddenNeurons
        HiddenNeurons = HiddenNeuronsArray(j);
        disp(sprintf('B2Helpline...GRAM-[%d]...HiddenNeurons-
[%d]......',...
            GRAM, HiddenNeurons));
        B2Helpline(GRAM, HiddenNeurons);
    end
end

%Backpropagation2(2, 5);
```

MATLAB – IK_1.m

```
% ------------------------------------------------------------------ %
% Function: Miss-stroke right letter prediction neural network      %
%           by Historical Networks.                                  %
% Author:   Jun Li                                                   %
% Create:   16/07/2008                                               %
% ------------------------------------------------------------------ %

clear all
close all

%Load PR, TR
load ./dataElmPR.txt
PR = dataElmPR;
clear dataElmPR;
% Create a Elman Network. PR=input range; TR=output range.
% trainlm is the default, fast but memory consuming.
% Out of memory solution:
% 1. setting net.trainParam.mem_reduc = 2 or more;
% 2. trainrp
net = newelm(PR, [3, 27], {'tansig', 'logsig'});

%load P, T
load ./dataElmP.txt

Ptmp = con2seq(dataElmP');
si = size(dataElmP, 1);
P = Ptmp(:, 1:(si-1));
T = Ptmp(:, 2:si);
clear dataElmP;
clear Ptmp;
% Train the network
%net.trainParam.mem_reduc = 5;
net.trainFcn = 'trainrp';

% There are other timers: etime, cputime, clock;
tic; % starts a stopwatch timer
net = train(net, P, T);
toc; % reads a stopwatch timer

% Simulation of Neural Network
%Y = sim(net, P);
```

MATLAB – LPonRH.m

```
% ------------------------------------------------------------- %
% Name: Letter Prediction on Right History                      %
% Function: Predict current letter based on the right history,  %
%           which means correcting the current input.           %
%           Also we add an additional Signal-'NULL', for letter %
%           absence and addition etc.                           %
% Author:   Jun Li                                              %
% Create:   06/08/2008                                          %
% ------------------------------------------------------------- %

%///////////////////////////////////////////////////////////////%
% Data Conversion: NULL <=> @   e.g.                            %
%   hthe quick brrooownn   fgow  jummppefd  iobverethe lwqazy   %
%   =>                                                          %
%   @the quick br@o@@wn@ @@f@ox@ jum@p@e@d @@o@ver the l@@azy   %
%///////////////////////////////////////////////////////////////%
function Num1 = LPonRH()

clear all
close all

% InputNeurons = Ngram * OutputNeurons
%OutputNeurons = 52; % 53-{Backspace, Delete}+{NULL}

%///////////////////////////////////////////////////////////////%
% Temporarily, actually we can design an very small Symbol, e.g. %
% {q, w, a, s; 9, 0, o, p}, each four are near each other.      %
% here, we consider 27 + addtional Signal '@' + time            %
% We distinguish time into three levels,                        %
%       over-fast => 001                                        %
%       user-Speed => 010                                       %
%       over-slow => 100                                        %
% Could further give an degree, e.g. 001 => 0 0 0.75            %
%///////////////////////////////////////////////////////////////%
SymbolNeurons = 27; %26 + SPACE
OutputNeurons = SymbolNeurons+1; % + NULL
TimeNeurons = 3;
Ngram = 2 - 1;
InputNeurons = SymbolNeurons*Ngram + TimeNeurons;

%Input Elements Range
IER = zeros(InputNeurons, 2);
for i = 1: InputNeurons
    IER(i, :) = [0, 1];
end
% Create a BPNN
% For the speed reason, choose trainrp rather than trainlm.
% About trainlm, can use mem_reduc to a little improvement.
net = newff(IER, [7, OutputNeurons], {'purelin', 'logsig'}, 'trainrp');
net.trainParam.show = 5;
net.trainParam.epochs = 100;
net.trainParam.goal = 1e-3;

%load data for key stroke analysis
```

```
load ./LPonRHP.txt
p1 = LPonRHP';
clear LPonRHP;
load ./LPonRHT.txt
t1 = LPonRHT';
clear LPonRHT;
si = size(p1, 2);

tic; % starts a stopwatch timer
%Ngram predict current letter, which is the last letter of current
column
%net = train(net, p1(:, 1:si), p1((OutputNeurons*(Ngram-1)+1)...
%    :(OutputNeurons*Ngram), 1:si));

net = train(net,p1, t1);
toc; % reads a stopwatch timer

%simulation
%Y = sim(net, p1(:, 1:si));

Y = sim(net, p1);
%t1 - Y

%---------Hit-Rate----------
for i = 1: si
    Ma = max(Y(:, i));
    Idx = find(Y(:, i)==Ma);
    for j = 1:OutputNeurons
        Y(j, i) = 0;
    end
    Y(Idx, i) = 1;
end

%Comparison between Y and T
Result = Y - t1;
Num1 = 0;
for i  = 1:si
    if(any(Result(:, i)) == 0)
        Num1 = Num1 + 1;
    end
end

%echo on;
%Num1
%echo off;
```

MATLAB – CallLPonRH.m

```
% ------------------------------------------------------------- %
% Name: Call LPonRH.m                                           %
% Function: LPonRH.m - Letter Predicton on Right History        %
%           CallLPonRH.m to get distribution of LPonRH's results %
% Author:   Jun Li                                              %
% Create:   08/12/2008                                          %
% ------------------------------------------------------------- %
clear all;
close all;

Results = [];
ResultsNum = 100;

for i = 1:ResultsNum
    result = LPonRH;
    Results = [Results, result];
end


%%Get by hand from matlab workspace where variable 'Results' is, 100
%%elements
%Results = [58    57    55    53    57    57    41    58    55    58
55    55    45    51    58    54    50    52    57    56    56    56
57    52    57    54    56    58    55    56    57    55    58    58
51    57    51    57    51    51    58    58    57    57    50    49
51    57    58    53    58    57    57    53    57    54    54    54
54    58    56    47    56    51    58    57    58    55    58    58
58    53    53    57    51    58    58    56    58    57    56    58
51    51    56    58    52    56    58    56    57    58    58    57
51    57    55    58    49    58];
```

MATLAB – pnn.m

```
%%------------------------------------------------------------ %
% Function: Miss-stroke right letter prediction neural network  %
%           Based on Radial Basis Network (--newpnn)           %
% Origin:   IK3 - but newbr couldnt give precise result        %
% Author:   Jun Li                                             %
% Create:   23/12/2008                                         %
% Comment:  Only consider the motor movement without predecessor.%
%           Only consider those mistaken letters.              %
%                                                              %
%               Input => horizontal distance(between pre and cur)%
%                        vertical distance                     %
%                        time gap                              %
%                  Output => horizontal distance               %
%                            vertical distance                 %
%              This is led to the possability of discovering the %
%              the relationship with mobility of hand.         %
% ------------------------------------------------------------ %
%%
%clear all
%close all
function Results = pnn()
load ./dataPNNTrain.txt;

n = size(dataPNNTrain, 1);
m = round(n/3);
r = iRand(n, m);
for i = 1:m
    if(dataPNNTrain(r(i), 4) > 0)
        dataPNNTrain(r(i), 4) = -dataPNNTrain(r(i), 4);
    end
end
inputP1 = [];
dataPNNTest = [];
for i = 1: n
    if(dataPNNTrain(i, 4) > 0)
        inputP1 = [inputP1; dataPNNTrain(i, :)];
    else
        dataPNNTrain(i, 4) = -dataPNNTrain(i, 4);
        dataPNNTest = [dataPNNTest; dataPNNTrain(i, :)];
    end
end
dataPNNTrain = inputP1;
clear inputP1;

inputP = dataPNNTrain(:, 1:3)';
targetT = dataPNNTrain(:, 4)';

%load ./dataPNNTest.txt
inputPTest = dataPNNTest(:, 1:3)';
targetTTest = dataPNNTest(:, 4)';

%%
%Normalization
%Horizontal
```

```
%inputP(1, :) = inputP(1, :) / 10; %Assumely
%inputPTest(1, :) = inputPTest(1, :) / 10;
%Vertical
%inputP(2, :) = inputP(2, :) / 5;
%inputPTest(2, :) = inputPTest(2, :) / 5;

%Time gap: gauss->netlab
%m = mean(inputP(3, :));
%c = cov(inputP(3, :));
%inputP(3, :) = (inputP(3, :) - m)/c;
%inputPTest(3, :) = (inputPTest(3, :) - m)/c;
%[inputP(1, :), ps1] = mapminmax(inputP(1, :));
%[inputP(2, :), ps2] = mapminmax(inputP(2, :));
%[inputP(3, :), ps3] = mapminmax(inputP(3, :));
%[inputPTest(3, :), psTest] = mapminmax(inputPTest(3, :));
%inputPTest(1, :) = mapminmax('apply',inputPTest(1, :),ps1);
%inputPTest(2, :) = mapminmax('apply',inputPTest(2, :),ps2);
%inputPTest(3, :) = mapminmax('apply',inputPTest(3, :),ps3);

%%
spread = 0.1;
T = ind2vec(targetT);
net = newpnn(inputP,T,spread);
%%
% Show results - 1
%plot(inputP,targetT,'+');
%title('Miss-stroke Distribution');
%xlabel('Keystroke Motor Distance');
%ylabel('Stroke Away Distince');

% Simulation
Y = sim(net,inputP);
YTest = sim(net, inputPTest);

Y = vec2ind(Y);
YTest = vec2ind(YTest);
%compare YTest with targetTTest, Y with targetT
Z = Y - targetT;
Ztest = YTest - targetTTest;

Num1 = 0;
for i  = 1:size(Z, 2)
    if(Z(1, i) == 0)
        Num1 = Num1 + 1;
    end
end

Num2 = 0;
for i  = 1:size(Ztest, 2)
    if(Ztest(i) == 0)
        Num2 = Num2 + 1;
    end
end

Results = Num2/size(Ztest, 2);

%%
```

```
fid = fopen('ppn.txt', 'at');
fprintf(fid, 'r=[ ');
for i = 1:m
    fprintf(fid, '%d ', r(1, i));
end
fprintf(fid, ']\n');
fprintf(fid, 'Results = %d/%d = [%.2f]\n\n', Num2, size(Ztest, 2),
Num2/size(Ztest, 2));

%RandomResults = 1/6
%fprintf(fid, 'RandomResults = [%.2f]', 1/6);
fclose(fid);
```

MATLAB – suggestTimeGap.m

```
% ------------------------------------------------------------- %
% Function: Suggested time gap between two consecutive letters by%
%           using backpropagation neural network                %
% Author:   Jun Li                                              %
% Create:   12/08/2008                                          %
% ------------------------------------------------------------- %
clear all
close all

%Initialization
IutputNeurons = 54; % letter coding
OutputNeurons = 1;  % timeGap parameter
Ngram = 2;

%Load
load ./suggestTimeGap.txt;
lines = size(suggestTimeGap, 1);
Z = suggestTimeGap(1:(lines-3), :)';
P = Z(1:IutputNeurons*Ngram, :);
T = Z((IutputNeurons*Ngram+1): (IutputNeurons*Ngram +1), :);
[T, ps] = mapminmax(T);

%MaxTG = suggestTimeGap((lines-2), 1); % maximum time gap
Mean = suggestTimeGap((lines-1), (IutputNeurons*Ngram+1));
Deviation = suggestTimeGap(lines, (IutputNeurons*Ngram+1));
%MinTG = 5; % minimun time gap
clear suggestTimeGap;
clear Z;

IER = zeros(IutputNeurons*Ngram, 2);
for i = 1: IutputNeurons*Ngram
    IER(i, :) = [0, 1];
end

% Create a BP-NN
net = newff(IER, [7, OutputNeurons], {'tansig', 'purelin'}, 'trainlm');
net.trainParam.show = 10;
net.trainParam.epochs = 100;
net.trainParam.goal = 0; % can we make it better?

%train without validation set
net = train(net, P, T);

%simulation
Y = sim(net, P);
%Convert Y to time gap
Y = mapminmax('reverse', Y, ps);

% if the program - Enstatistics has used mean=0 and deviation=1
%TG = Y * Deviation + Mean;
% clear Y;

%when testing, use below to pre-processing first.
%y2 = mapminmax('apply',x2,ps)
```

MATLAB – RankAlgorithms.m

```
% ------------------------------------------------------------------- %
% Name: BackPropagation three word correct algorithms                 %
% Funciton: Ranking => output the most probable word                  %
% Auther:   Jun Li                                                    %
% Create:   30/09/2008                                                %
% ------------------------------------------------------------------- %
function Num1 = RankAlgorithms(RankAlgorithmsP1, RankAlgorithmsTest1)
%clear all
%close all

%Initialization
NumOfAlgorithms = 3;
%number of words of each algorithm
NumoOfWords = 2;
%output of each word: distance and frequency
NumOfFeature = 2;

IutputNeurons = NumOfAlgorithms*NumoOfWords*NumOfFeature;
HiddenNeurons = 3;
OutputNeurons = NumOfAlgorithms*NumoOfWords;

%Load
%load ./data/RankAlgorithmsP1.txt;
%RankAlgorithmsP1 = RankAlgorithmsP1';

%load ./data/RankAlgorithmsTest1.txt;
%RankAlgorithmsTest1 = RankAlgorithmsTest1';

%Normalization
x = [];
for i = 1:2:IutputNeurons
    x = [x RankAlgorithmsP1(i, :)];
end
xmean = mean(x);
xstd = std(x);
for i = 1:2:IutputNeurons
    RankAlgorithmsP1(i, :) = normpdf(RankAlgorithmsP1(i, :), xmean,
xstd);
    RankAlgorithmsTest1(i, :) = normpdf(RankAlgorithmsTest1(i, :), xmean,
xstd);
end

P = RankAlgorithmsP1(1:IutputNeurons, :);
T = RankAlgorithmsP1((IutputNeurons+1):(IutputNeurons+OutputNeurons), :);

testP = RankAlgorithmsTest1(1:IutputNeurons, :);
testT =
RankAlgorithmsTest1((IutputNeurons+1):(IutputNeurons+OutputNeurons), :);
%clear RankAlgorithmsP1;

%value scope of each input
IER = zeros(IutputNeurons, 2);
for i = 1: IutputNeurons
    IER(i, :) = [0, 1];
```

```
end

% Create a BP-NN
%net.trainParam.epochs =1000; (Max no. of epochs to train) [100]
%net.trainParam.goal =0.01;        (stop training if the error goal hit)
[0]
%net.trainParam.lr =0.001;          (learning rate, not default trainlm)
[0.01]
%net.trainParam.show =1;            (no. epochs between showing error)
[25]
%net.trainParam.time =1000;   (Max time to train in sec) [inf]

net = newff(IER, [HiddenNeurons, OutputNeurons], {'purelin', 'logsig'},
'trainlm');
net.trainParam.show = 10;
net.trainParam.epochs = 100;
net.trainParam.goal = 0;
net.trainParam.lr =0.001;

%train without validation set
net = train(net, P, T);

%simulation
Y = sim(net, testP);

%compete to generate result
for i = 1: size(testT, 2)
    Ma = max(Y(:, i));
    Idx = find(Y(:, i)==Ma);
    for j = 1:OutputNeurons
        Y(j, i) = 0;
    end
    Y(Idx, i) = 1;
end

Result = Y - testT;

%Number of right Prediction
Num1 = 0;
for i  = 1:size(testT, 2)
    if(any(Result(:, i)) == 0)
        Num1 = Num1 + 1;
    end
end
```

MATLAB – Hmm.m

```
% ------------------------------------------------------------- %
% Funciton: Using hmmestimate & hmmviterbi to have a learnable  %
%           program for language modelling                      %
% Auther:   Jun Li                                              %
% Create:   28/06/2008                                          %
% ------------------------------------------------------------- %

% First load a heuristic Transition and Emission Matrices based on bi-
gram
% P(aj|ai)=P(ai,aj)/P(ai)
%
%                <successor>
%                    a(A)    ...      z(Z)   NULL
% <predecessor>     ---------------------------------
%                a(A)|P(a|a) ...      P(z|a) P(Null|a)
%                .   |
%                .   |
%                z(Z)|
%                NULL|
%

% Calculated based on MICHAEL's bigram and book <<Penguin 1978>>
% Format: uu -> ul -> ll -> lu
load BigramMatrix.txt;

% Estimating Transition and Emission Matrices, where Transition Matrix
% is acquired from the statistics to particular text reference, where
% Emission Matrix from the action to particular user's typing record.
% the probability of transition from ai to aj of user's typing record
% as emission sequence.
load seq.txt;

% the probability of transition from ai to aj of right text
% as states sequence.
load states.txt;

% The following takes the emission and state sequences and returns
% estimates of the transition and emission matrices.
[TRANS_EST, EMIS_EST] = hmmestimate(seq, states);

% Generate Ti, Tj Based on Multi-Parameters <=> STEP FUNCTION
% (discontinuous). Using Fuzzy Logic method to blur the boundary of step
% function.
%Ti = t(Xi);
%Tj = t(Xj);

% Generate TRANS based on adjusted weights Ti, Tj.
% Ti for BigramMatrix.txt; Tj for TRANS_EST.
TRANS = Ti * BigramMatrix + Tj * TRANS_EST;

% Generate EMIS based on adjusted weights Ei, Ej.
% Ei for seq; Ej for EMIS_EST;
EMIS = Ei * seq + Ej * EMIS_EST;
```

```
% Given the transition and emission matrices TRANS and EMIS, the
function
% hmmviterbi uses the Viterbi algorithm to compute the most likely
sequence
% of states the model would go through to generate a given sequence seq
of
% emissions.
% likelystates is a sequence the same length as seq.
likelystates = hmmviterbi(seq, TRANS, EMIS);
```

MATLAB – iRand.m

```
% ------------------------------------------------------------------- %
% Funciton: iRand => integer random for 1-n                           %
% Author:   Jun Li                                                    %
% Create:   28/11/2008                                                %
% Parameter:    n => the range of number                              %
%               m => the number of rand-number                        %
% ------------------------------------------------------------------- %

function r = iRand(n, m)
%Initialize RAND to a different state each time.
%suggest only do this when start a session.
rand('twister',sum(100*clock));

%Generate integers uniform on the set 1:n.
r = ceil(n.*rand(1, m));

%Generate uniform values from the interval [a, b].
%r = a + (b-a).*rand(100,1);
```

MATLAB – NgramFrequency.m

```
% ------------------------------------------------------------- %
% Funciton: To Get the N-gram frequency of English              %
% Author:   Jun Li                                              %
% Create:   12/08/2008                                          %
% Comment:                                                      %
% ------------------------------------------------------------- %

% Turn on echoing of commands inside the script-file.
%echo on
% Clear command window.
%clc

clear all
close all

load ./data/u.txt;
load ./data/l.txt;

load ./data/uu.txt;
load ./data/ul.txt;
load ./data/ll.txt;
load ./data/lu.txt;

[lines, columns] = size(uu);
uuf = zeros(lines, columns); uut = 0;
ulf = zeros(lines, columns); ult = 0;
llf = zeros(lines, columns); llt = 0;
luf = zeros(lines, columns); lut = 0;
for i = 1: lines
    for j = 1: columns
        if uu(i, j)== 0.00
            uuf(i, j) = 1;
        elseif uu(i, j) == -1
            uuf(i, j) = 0;
        else
            uuf(i, j) = round(exp(uu(i, j)));
        end
        uut = uut + uuf(i,j);
        %----------------------------
        if ul(i, j)== 0.00
            ulf(i, j) = 1;
        elseif ul(i, j) == -1
            ulf(i, j) = 0;
        else
            ulf(i, j) = round(exp(ul(i, j)));
        end
        ult = ult + ulf(i, j);
        %----------------------------
        if ll(i, j)== 0.00
            llf(i, j) = 1;
        elseif ll(i, j) == -1
            llf(i, j) = 0;
        else
            llf(i, j) = round(exp(ll(i, j)));
```

```
            end
            llt = llt + llf(i, j);
            %----------------------------
            if lu(i, j)== 0.00
                luf(i, j) = 1;
            elseif lu(i, j) == -1
                luf(i, j) = 0;
            else
                luf(i, j) = round(exp(lu(i, j)));
            end
            lut = lut + luf(i, j);
        end
    end
    %uuf = uuf/uut*100;
    %ulf = ulf/ult*100;
    %llf = llf/llt*100;
    %luf = luf/lut*100;

    % Caculate the 1-gram probability
    ut = 0;
    lt = 0;
    for i = 1: 26
        ut = ut + u(i);
        lt = lt + l(i);
    end
    uf = u/(ut+lt)*100;
    lf = l/(ut+lt)*100;

    % Caculate the 2-gram probability
    uuf = uuf/(uut+ult+llt+lut)*100;
    ulf = ulf/(uut+ult+llt+lut)*100;
    llf = llf/(uut+ult+llt+lut)*100;
    luf = luf/(uut+ult+llt+lut)*100;

    % Calculate p(aj|ai)=p(ai, aj)/p(ai)
    for i = 1:26
        uuf(i, :) = uuf(i, :)/uf(i);
        ulf(i, :) = ulf(i, :)/uf(i);
        llf(i, :) = llf(i, :)/lf(i);
        luf(i, :) = luf(i, :)/lf(i);
    end

    % Write file-BigramMatrix.txt for Hmm.m
    % But have added the Frequency between NULL and other letters
    Fbm = fopen('BigramMatrix.txt','w');
    if Fbm < 0
        error(['Could not open file: ', 'BigramMatrix.txt']);
    end
    fprintf(Fbm, '%% ATTENTION: need double check with the equation and
    number before publish\n\n');
    for i = 1:26
        for j = 1:26
            fprintf(Fbm, '%6.4f ', uuf(i, j));
        end
        fprintf(Fbm, '\n');
    end
```

```
fprintf(Fbm, '\n');
for i = 1:26
    for j = 1:26
        fprintf(Fbm, '%6.4f ', ulf(i, j));
    end
    fprintf(Fbm, '\n');
end
fprintf(Fbm, '\n');
for i = 1:26
    for j = 1:26
        fprintf(Fbm, '%6.4f ', llf(i, j));
    end
    fprintf(Fbm, '\n');
end
fprintf(Fbm, '\n');
for i = 1:26
    for j = 1:26
        fprintf(Fbm, '%6.4f ', luf(i, j));
    end
    fprintf(Fbm, '\n');
end
fprintf(Fbm, '\n');
fclose(Fbm);

%using weighting to further calculation.

%echo off;
```

Appendix B

MAIN ENSTATISTICS SOURCE CODE

CEnStatisticsDlg::OnBnClickedTextanalysis

```
this->UpdateData(); // initialize data in a dialog box

LPCTSTR strFilter = _T("Txt Files (*.txt)|*.txt|All Files (*.*)|*.*||");

CFile f, fDict;

CFileDialog FileDlg(TRUE, _T(".txt"), NULL, 0, strFilter);

if( FileDlg.DoModal() == IDOK )
{
        if( f.Open(FileDlg.GetFileName(), CFile::modeRead) == FALSE )
        return;
CArchive ar(&f, CArchive::load);

TCHAR lpBuf[80], wCurr[20], cLast2, cLast1, cCurr;
int wCurrSeq, wLast1Seq, wLast2Seq;
struct wordChain *lastUnit, *currUnit;
struct EnDictionary *dictCurrPtr, *dictLastPtr;

cLast2 = cLast1 = cCurr = 0;
wCurrSeq = wLast1Seq = wLast2Seq = -1;

headerUnit = (struct wordChainHeader *)malloc(sizeof(struct wordChainHeader));
headerUnit->numUnit = 0; //not sure if will use.
memset(headerUnit->idxChain, NULL, 256); //not sure how to use.
headerUnit->next = NULL;

memset(lpBuf, 0, 80);
                memset(wCurr, 0, 20);

                //allocate memory 100*1000 elements for dictionary
                dictBasePtr = (struct EnDictionary *)calloc(120*1000, sizeof(struct
EnDictionary));

/*dictCurrPtr = dictBasePtr;
for(int i=0; i<120*1000; i++)
{
        memset(dictCurrPtr->word, 0, 20);
        //dictCurrPtr->seq = 0;
        dictCurrPtr->frq = 0;
        dictCurrPtr++;
} */
                dictCurrPtr = dictBasePtr;
                dictLastPtr = dictCurrPtr;

dictOffset[_T('a')] = 0;
//load dictionary
TCHAR tmpStr[MAX_PATH+100];
memset(tmpStr, 0, MAX_PATH+100);
_tcscat_s(tmpStr, currDirectory);
_tcscat_s(tmpStr, _T("\\es.txt"));

if(fDict.Open(tmpStr, CFile::modeRead) == FALSE)
```

```
		{
			MessageBox(_T("Couldnt find the dictionary es.txt"));
			MessageBox(currDirectory);
			return;
		}
		CArchive arDict(&fDict, CArchive::load);
		CString dictLine = _T("");
		int i = 0;

// a simulation of ReadSring to ASCII code text
/*char aux [2];
CString m_sLine;
while (m_bMoreChars = m_rArchive.Read (aux, 1))
{
if ((aux [0] == '\n') || (aux [0] == '\r'))
break;
m_sLine += aux [0];
}*/
		while(arDict.ReadString(dictLine))
		{
			CString word, freq;
			//word
			AfxExtractSubString(word, dictLine, 0, ' ');
			//frequency
			AfxExtractSubString(freq, dictLine, 1, ' ');
			memset(dictCurrPtr->word, 0, 20);
			_tcscpy_s(dictCurrPtr->word, (word.GetLength()+1), word); //caution:
must be exact size
			dictCurrPtr->frq = _ttoi(freq);
			dictCurrPtr->seq = i++;
			if(dictCurrPtr->word[0] != dictLastPtr->word[0])
			{
				dictOffset[dictCurrPtr->word[0]] = dictCurrPtr - dictBasePtr;
				dictLastPtr = dictCurrPtr;
			}
			dictCurrPtr++;
			if(i >= 100000) break;
		}
		//if(i = 100000) or < 100000 are all same, give an endpoint
		dictCurrPtr->frq=0;
		dictCurrPtr->seq = i;
		memset(dictCurrPtr->word, 0 ,20);

		arDict.Close();
		fDict.Close();

		while(ar.Read(lpBuf, 1)) //text must be saved as ASCII?
		{
			cCurr = lpBuf[0];
			alphabetFrequency[cCurr]++;
			bigramFrequency[cLast1][cCurr]++;
			trigramFrequency[cLast2][cLast1][cCurr]++;

			if(!((cCurr>='a') && (cCurr<='z')) && !((cCurr>='A') && (cCurr<='Z')) &&
(cCurr != '_'))
```

```
                        {
                                //then we think a word has been met
                                // using an ordered chain
                                wCurrSeq = checkDict(wCurr, 1); //if word is new, add to space
which starts from 100*1000+1
                                if(wCurrSeq == -1) continue;
/*TCHAR str1[100];
memset(str1, 0, 100);
_stprintf_s(str1, _T("wCurr = [%s] wCurrSeq = [%d]"), wCurr, wCurrSeq);
AfxMessageBox(str1);*/
                                //check chain to get the right place for wCurrSeq
                                //if((wCurrSeq != -1) && (wLast1Seq != -1) && (wLast2Seq != -1))
                                //because I we calculate word bi-gram as well, so the first two
words give problem. Have to ignore && (wLast2Seq != -1)
                                if((wCurrSeq != -1) && (wLast1Seq != -1))
                                {
                                        /*TCHAR str[100];
                                        _stprintf_s(str, _T("start checkChain [%d][%d][%d]"),
wCurrSeq, wLast1Seq, wLast2Seq);
                                        AfxMessageBox(str);*/
                                        currUnit = checkChain(headerUnit, wCurrSeq, wLast1Seq,
wLast2Seq);
                                }

                                memset(wCurr, 0, 20);
                                wLast2Seq = wLast1Seq;
                                wLast1Seq = wCurrSeq;

                        }
                        else
                        {
                                _tcscat_s(wCurr, lpBuf);
                        }
                        cLast2 = cLast1;
                        cLast1 = cCurr;
                } // end of while
                ar.Close();
                f.Close();

//AfxMessageBox(_T("start print"));
                //print statistics;
                CFile fResult;
                CArchive arResult(&fResult, CArchive::store);

                memset(tmpStr, 0, MAX_PATH+100);
                _tcscat_s(tmpStr, currDirectory);
                _tcscat_s(tmpStr, _T("\\statistics_result.txt"));
                if(fResult.Open(tmpStr, CFile::modeCreate|CFile::modeWrite) == FALSE)
                {
                        MessageBox(_T("Couldnt create the statistics_result.txt file"));
                        MessageBox(tmpStr);
                        return;
                }

                //printf 1-gram
```

```
arResult.WriteString(_T("==================LETTER GRAM-1==================\r\n"));
                CString str;
                int letterTotal = 0;
                for(int i=0; i<16; i++)
                {
                        for(int j=0; j<16; j++)
                        {
                                str.Format(_T("%8d"), i*16+j);
                                arResult.WriteString(str);
                        }
                        arResult.WriteString(_T("\r\n"));
                        for(int j=0; j<16; j++)
                        {
                                arResult.WriteString(_T("--------"));
                        }
                        arResult.WriteString(_T("\r\n"));
                        for(int j=0; j<16; j++)
                        {
                                letterTotal += alphabetFrequency[i*16+j];
                                str.Format(_T("%8d"), alphabetFrequency[i*16+j]);
                                arResult.WriteString(str);
                        }
                        arResult.WriteString(_T("\r\n\r\n"));
                }
                str.Format(_T("letterTotal = [%d]\r\n\r\n"), letterTotal);
                arResult.WriteString(str);

                //below is added temporally for Matlab hmm.m
                arResult.WriteString("%below is added temporally for Matlab hmm.m\r\n");
                arResult.WriteString("32      45      97      98      99      100     101     102
                103     104     105     106     107     108     109     110     111     112     113     114
                115     116     117     118     119     120     121     122\r\n");
                                arResult.WriteString("----------------------------------------------------------------------------------------------------
                ----------------------------------------------------------------\r\n");
                                str.Format(_T("%-8.4f"), (double)alphabetFrequency[32]/(double)letterTotal);
                arResult.WriteString(str);
                                str.Format(_T("%-8.4f"), (double)alphabetFrequency[45]/(double)letterTotal);
                arResult.WriteString(str);
                                for(i='a'; i<='z'; i++)
                                {
                                        str.Format(_T("%-8.4f"),
                (double)alphabetFrequency[i]/(double)letterTotal); arResult.WriteString(str);
                                }
                                arResult.WriteString("\r\n\r\n");

                arResult.WriteString(_T("\r\n\r\n==================LETTER GRAM-2==================\r\n"));
                //printf 2-gram
                arResult.WriteString(_T("        "));
                for(int i=0; i<256; i++)
                {
                        str.Format(_T("%-8d"), i);
                        arResult.WriteString(str);
                }
                arResult.WriteString(_T("\r\n\r\n"));
```

```
for(int i=0; i<256; i++)
{
        str.Format(_T("%-8d"), i);
        arResult.WriteString(str);
        for(int j=0; j<256; j++)
        {
                str.Format(_T("%-8d"), bigramFrequency[i][j]);
                arResult.WriteString(str);
        }
        arResult.WriteString(_T("\r\n"));
}

//below is added temporally for Matlab hmm.m
arResult.WriteString("\r\n%below is added temporally for Matlab hmm.m\r\n");
arResult.WriteString(_T("       "));
for(int i='a'; i<='z'; i++)
{
        str.Format(_T("%-8d"), i);
        arResult.WriteString(str);
}
str.Format(_T("%-8d"), 32);
arResult.WriteString(str);
str.Format(_T("%-8d"), 45);
arResult.WriteString(str);

arResult.WriteString(_T("\r\n"));
for(int i='a'; i<='z'; i++)
{
        str.Format(_T("%-8d"), i);
        arResult.WriteString(str);
        for(int j='a'; j<='z'; j++)
        {
                str.Format(_T("%-8d"), bigramFrequency[i][j]);
                arResult.WriteString(str);
        }
        str.Format(_T("%-8d"), bigramFrequency[i][32]);
        arResult.WriteString(str);
        str.Format(_T("%-8d"), bigramFrequency[i][45]);
        arResult.WriteString(str);
        arResult.WriteString(_T("\r\n"));
}
//print 32 and 45
i = 32;
str.Format(_T("%-8d"), i);
arResult.WriteString(str);
for(int j='a'; j<='z'; j++)
{
        str.Format(_T("%-8d"), bigramFrequency[i][j]);
        arResult.WriteString(str);
}
str.Format(_T("%-8d"), bigramFrequency[i][32]);
arResult.WriteString(str);
str.Format(_T("%-8d"), bigramFrequency[i][45]);
arResult.WriteString(str);
arResult.WriteString(_T("\r\n"));
```

```
i = 45;
str.Format(_T("%-8d"), i);
arResult.WriteString(str);
for(int j='a'; j<='z'; j++)
{
        str.Format(_T("%-8d"), bigramFrequency[i][j]);
        arResult.WriteString(str);
}
str.Format(_T("%-8d"), bigramFrequency[i][32]);
arResult.WriteString(str);
str.Format(_T("%-8d"), bigramFrequency[i][45]);
arResult.WriteString(str);
arResult.WriteString(_T("\r\n"));

arResult.WriteString(_T("\r\n\r\n==================LETTER GRAM-3==================\r\n"));
                //printf 3-gram
                for(int i=0; i<256; i++)
                {
                        for(int j=0; j<256; j++)
                        {
                                for(int k=0; k<256; k++)
                                {
                                        if(trigramFrequency[i][j][k] != 0)
                                        {
                                                str.Format(_T("[%d %d %d]=[%d]\r\n"), i, j, k,
trigramFrequency[i][j][k]);
                                                arResult.WriteString(str);
                                        }
                                }
                        }
                }
arResult.WriteString(_T("\r\n\r\n==================WORD GRAM-1==================\r\n"));
                //printf word frequency 1
                dictCurrPtr = dictBasePtr;
                while(dictCurrPtr->word[0] != 0)
                {
                        if(dictCurrPtr->frq != 0)
                        {
                                str.Format(_T("[%s] = [%d]\r\n"), dictCurrPtr->word,
dictCurrPtr->frq);
                                arResult.WriteString(str);
                        }
                        dictCurrPtr++;
                }
                dictCurrPtr = dictBasePtr + 100*1000 + 1;
                while(dictCurrPtr->word[0] != 0)
                {
                        if(dictCurrPtr->frq != 0)
                        {
                                str.Format(_T("[%s] = [%d]\r\n"), dictCurrPtr->word,
dictCurrPtr->frq);
                                arResult.WriteString(str);
                        }
                        dictCurrPtr++;
                }
```

```
arResult.WriteString(_T("\r\n\r\n==================WORD GRAM-2&3==================\r\n"));
                //printf word frequency 2&3
                currUnit = headerUnit->next;
                lastUnit = currUnit;
                BOOL isBiBegin = 1;
                while(currUnit != NULL)
                {
                        struct EnDictionary *dictPtr1, *dictPtr2, *dictPtr3, tmpdict;

                        memset(tmpdict.word, 0, 20);
                        if(currUnit->wordSeq[0] >= 0)
                                dictPtr1 = dictBasePtr + currUnit->wordSeq[0];
                        else
                                dictPtr1 = &tmpdict;
                        if(currUnit->wordSeq[1] >= 0)
                                dictPtr2 = dictBasePtr + currUnit->wordSeq[1];
                        else
                                dictPtr2 = &tmpdict;
                        if(currUnit->wordSeq[2] >= 0)
                                dictPtr3 = dictBasePtr + currUnit->wordSeq[2];
                        else
                                dictPtr3 = &tmpdict;
                        if(isBiBegin)
                        {
//if(inMyArray(dictPtr2->word))
//{
                                str.Format(_T("%s|%s = %d\r\n"), dictPtr2->word, dictPtr1->word,
currUnit->bifrq);
                                arResult.WriteString(str);
//}
                        }
//AfxMessageBox(_T("after isBiBegin"));
                        /*if(currUnit->frq != 0) // never
                        {
                                str.Format(_T("%s|%s|%s = %d\r\n"), dictPtr3->word, dictPtr2-
>word, dictPtr1->word, currUnit->frq);
                                arResult.WriteString(str);
                        }*/
                        lastUnit = currUnit;
                        currUnit = currUnit->next;
                        if(currUnit == NULL) break;
                        if((lastUnit->wordSeq[0] == currUnit->wordSeq[0]) && (lastUnit-
>wordSeq[1] == currUnit->wordSeq[1])) isBiBegin = 0;
                        else isBiBegin = 1;
                }
                arResult.Close();
                fResult.Close();
        }
        else
                return;

        //Flag that indicates whether dialog box is being initialized (FALSE) or data is being
retrieved (TRUE).
        this->UpdateData(FALSE);
```

CEnStatisticsDlg::OnBnClickedStrokestat

```
	this->UpdateData(); // initialize data in a dialog box

	LPCTSTR strFilter = _T("Txt Files (*.txt)|*.txt|All Files (*.*)|*.*||");

	CFile f;

	CFileDialog FileDlg(TRUE, _T(".txt"), NULL, 0, strFilter);

	int vkc_nn[] = { 0x08, 0x0D, 0x10, 0x14, 0x20,
                                        0x2E, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
0x38, 0x39,
                                        0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49,
0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F, 0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59,
0x5A,
                                        0xBA, 0xBB, 0xBC, 0xBD, 0xBE, 0xBF, 0xC0,
                                        0xDB, 0xDC, 0xDD, 0xDE
	};

	int lvq[] = { 0, 0, 0, 0, 0,
                                 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0,
                                 0, 0, 0, 0, 0, 0, 0,
                                 0, 0, 0, 0
	};

	int a[2][256] = {0};
while(1) {
	if( FileDlg.DoModal() == IDOK )
	{
                        if( f.Open(FileDlg.GetFileName(), CFile::modeRead) == FALSE )
                                return;
                        CArchive ar(&f, CArchive::load);

                        CString ptr = _T("");
                        int upORdown = 0, stepup = 0, stepdown = 0;
                        while(ar.ReadString(ptr))
                        {
                                if(ptr.GetLength() == 0) continue;

                                CString word, strnum;

                                if(!AfxExtractSubString(word, ptr, 0, ' '))
                                {
                                        continue;
                                }
//AfxMessageBox(word);
                                if(word == "UP") upORdown = 0;
                                else if(word == "DOWN") upORdown = 1;
                                else continue;

                                int i = 0, ta = 0, getNum = 0;
                                while(ptr[i] != 0)
```

```
{
        if((ptr[i] >= _T('0')) && (ptr[i]<=_T('9')))
        {
                strnum = strnum + ptr[i];
                getNum = 1;
        }
        else
        {
                if(getNum)
                {
                        if(upORdown)
                        {
                                a[1][stepdown++] += atoi(strnum);
                        }
                        else
                        {
                                //AfxMessageBox(strnum);
                                a[0][stepup++] += atoi(strnum);
                        }
                }
                strnum = _T("");
                ta = 0;
                getNum = 0;
        }
        i++;
}
if(strnum != _T(""))
{
        if(upORdown)
        {
                a[1][stepdown++] += atoi(strnum);
        }
        else
        {
                a[0][stepup++] += atoi(strnum);
        }
                }
        }
        ar.Close();
        f.Close();
}
else
{
        break;
}
}
CFile fResult;
CArchive arResult(&fResult, CArchive::store);

TCHAR tmpStr[MAX_PATH+100];
memset(tmpStr, 0, MAX_PATH+100);
_tcscat_s(tmpStr, currDirectory);
_tcscat_s(tmpStr, _T("\\Strokestat.txt"));
if(fResult.Open(tmpStr, CFile::modeCreate|CFile::modeWrite) == FALSE)
{
```

```
                MessageBox(_T("Couldnt create the Strokestat.txt file"));
                MessageBox(tmpStr);
                return;
        }
//AfxMessageBox(_T("step...1"));
        CString strResult;
        double totup=0, totdown=0, totUPothers=0;
        for(int i=0; i<16; i++ )
        {
                arResult.WriteString("        ");
                for(int j=0; j<16; j++)
                {
                        strResult.Format("%8x", i*16+j);
                        arResult.WriteString(strResult);
}
arResult.WriteString("\r\n--------");
for(int j=0; j<16; j++)
{
        strResult.Format("--------");
        arResult.WriteString(strResult);
}
arResult.WriteString("\r\nUP      ");
for(int j=0; j<16; j++)
{
        totup += a[0][i*16+j];
        strResult.Format("%8d", a[0][i*16+j]);
        arResult.WriteString(strResult);
        int rtn = kcINvkc_nn(i*16+j, vkc_nn, sizeof(lvq)/sizeof(int));
        if(rtn != -1)
        {
                lvq[rtn] = a[0][i*16+j];
        }
        else
        {
                totUPothers += a[0][i*16+j];
        }
}
arResult.WriteString("\r\nDOWN    ");
for(int j=0; j<16; j++)
{
        totdown += a[1][i*16+j];
        strResult.Format("%8d", a[1][i*16+j]);
        arResult.WriteString(strResult);
}
arResult.WriteString("\r\n\r\n\r\n");
}
        double all = 0;
        for(int i=0; i<(sizeof(lvq)/sizeof(int)); i++)
        {
                //strResult.Format("freq[%x]=[%d]                [%0.4f percentage]\r\n\r\n",
vkc_nn[i], lvq[i], double(lvq[i])/totup);
                strResult.Format("%0.4f      ", double(lvq[i])/totup);
                arResult.WriteString(strResult);
                all += atof(strResult);
```

```
        }

        double ddd = totUPothers/totup;
        //strResult.Format("freq[others]=[%d]    [%0.4f percentage]\r\n\r\n", int(totUPothers),
ddd);
        //here we use 1-all rather than ddd
        //strResult.Format("\r\n%0.4f = %f\r\n\r\n", ddd, (1.00 - all));
        strResult.Format("%0.4f\r\n\r\n", (1.00 - all));
        arResult.WriteString(strResult);

        strResult.Format("TOTAL_UP = [%4.0f]\r\nTOTAL_DOWN = [%4.0f]\r\nTOTAL_OTHERS =
[%4.0f]\r\n", totup, totdown, totUPothers);
        arResult.WriteString(strResult);

        arResult.Close();
        fResult.Close();

        this->UpdateData(FALSE);
```

Appendix C

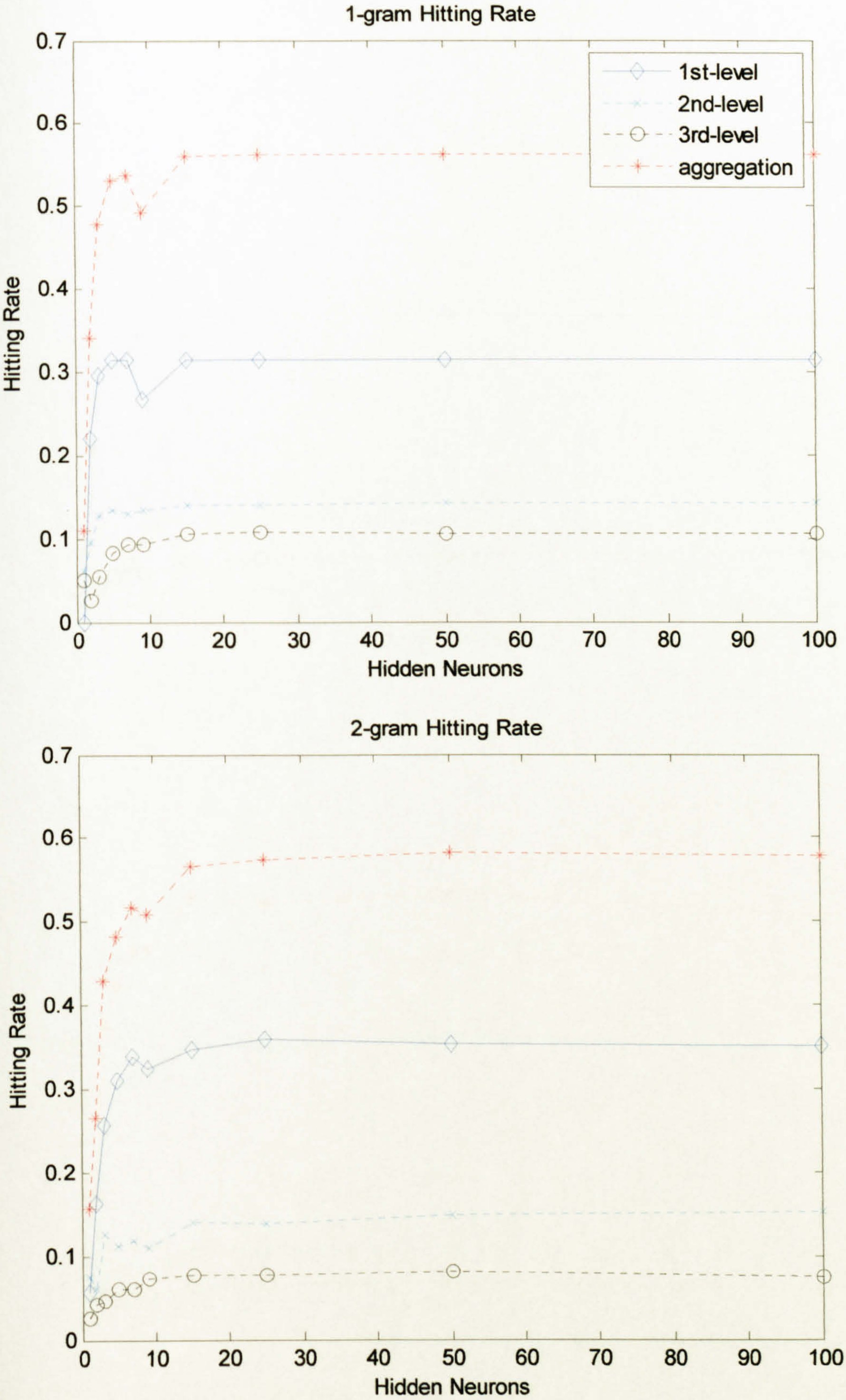
MAIN INTELLIGENT KEYBOARD – ENGLISH INPUT METHOD PROGRAMS

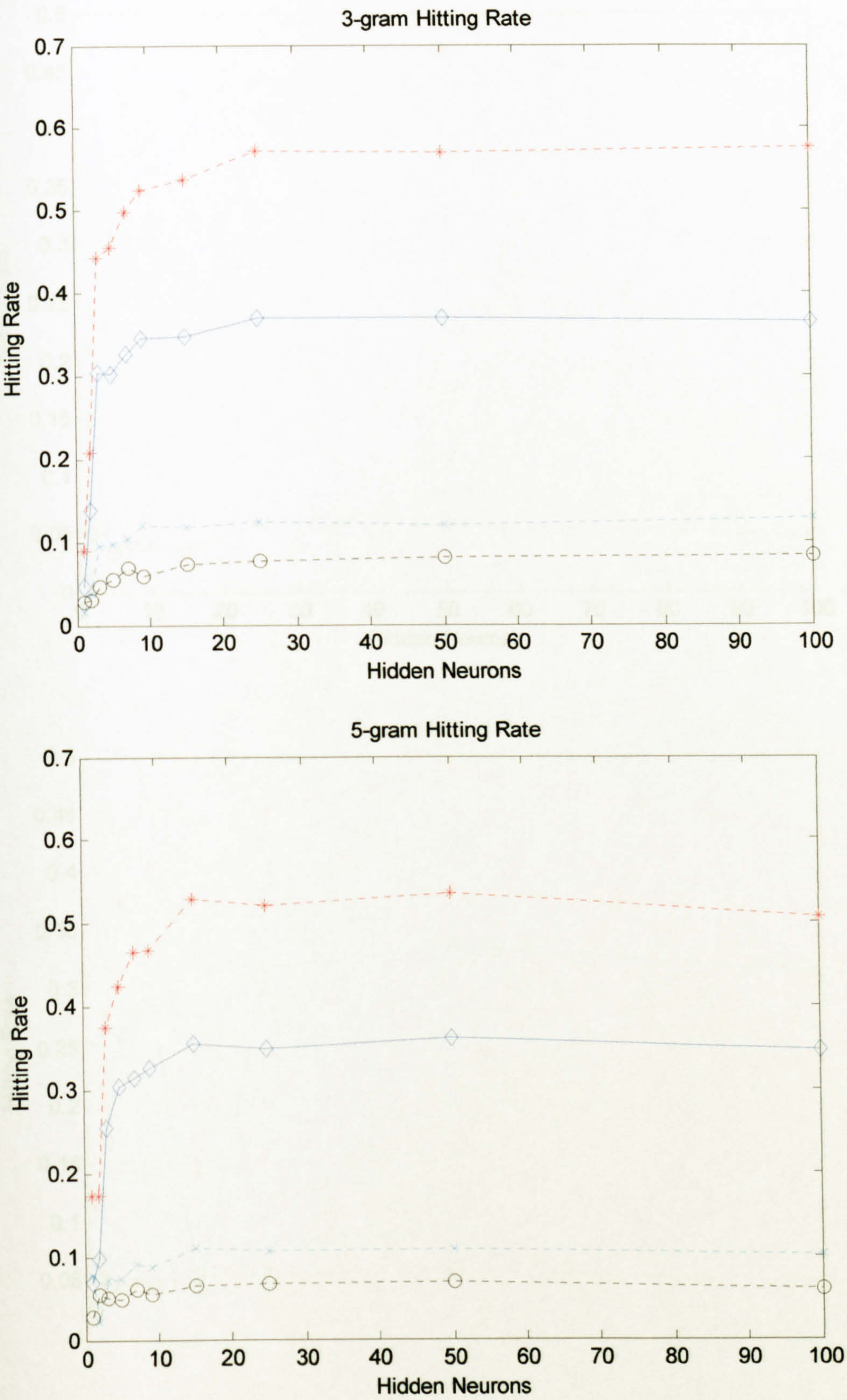
NAME	FUNCTION
backpropagation.c backpropagation.h	Neural Network BackPoropagation algorithm implementation which has been used to rank the word-list based on words frequency and distance which is between the typed word and the table words
backspace.c backspace.h	Record the relationship between wrong string and right string into table 'arules' as soon as backspace is typed. That is a many-to-many relationship (lefts -> right) and will trigger when a typing complete
clipboard.c clipboard.h	Get string from Clipboard to analyze (count the gram of letters and words etc)
DateTime.c DateTime.h	Date and time processing (e.g. +, -)
edpa.c	Main .dll file (converted to .ime after compilation) used as an English IME
edpa.h	Main header file shared by all files
edpalib.c handle.c	Two main libraries of edpa.dll
edpa.def	Defines the functions of input method editor api and shared memory – 'edpasm'
global.c global.h	Define and initialize all global variables shared by files
gram.c gram.h	One & two letter gram, two words gram operation
imm.c imm.h	IME Entry functions which are corresponding to edpa.def
jaro.c jaro.h	Calculate distance between two words based on Jaro algorithm
KeysAround.c KeysAround.h	Adjacent key press processing (only consider three consecutive letters)
metaphone.c metaphone.h	Calculate the similarity of two words based on metaphone algorithm.
parameter.c parameter.h	Configuration parameters of edpa.ime
preprocess.c preprocess.h	Pre-process before the typed string is delivered to prediction and correction functions (prolong key press is processed at this stage) Correspond to the architecture's pre-processing unit.
rank.c rank.h	Uses neural network and heuristic method to rank the word-list
scheck.c scheck.h	A simple spell checking based on Diego C. Barrientos's function using Levenshtein Word Distance.
shortm.c shortm.h	Using short memory to record the particular words (e.g. used recently) and calculate the ranks
tcheck.c tcheck.h	Tables Operations (used by edpa.ime)

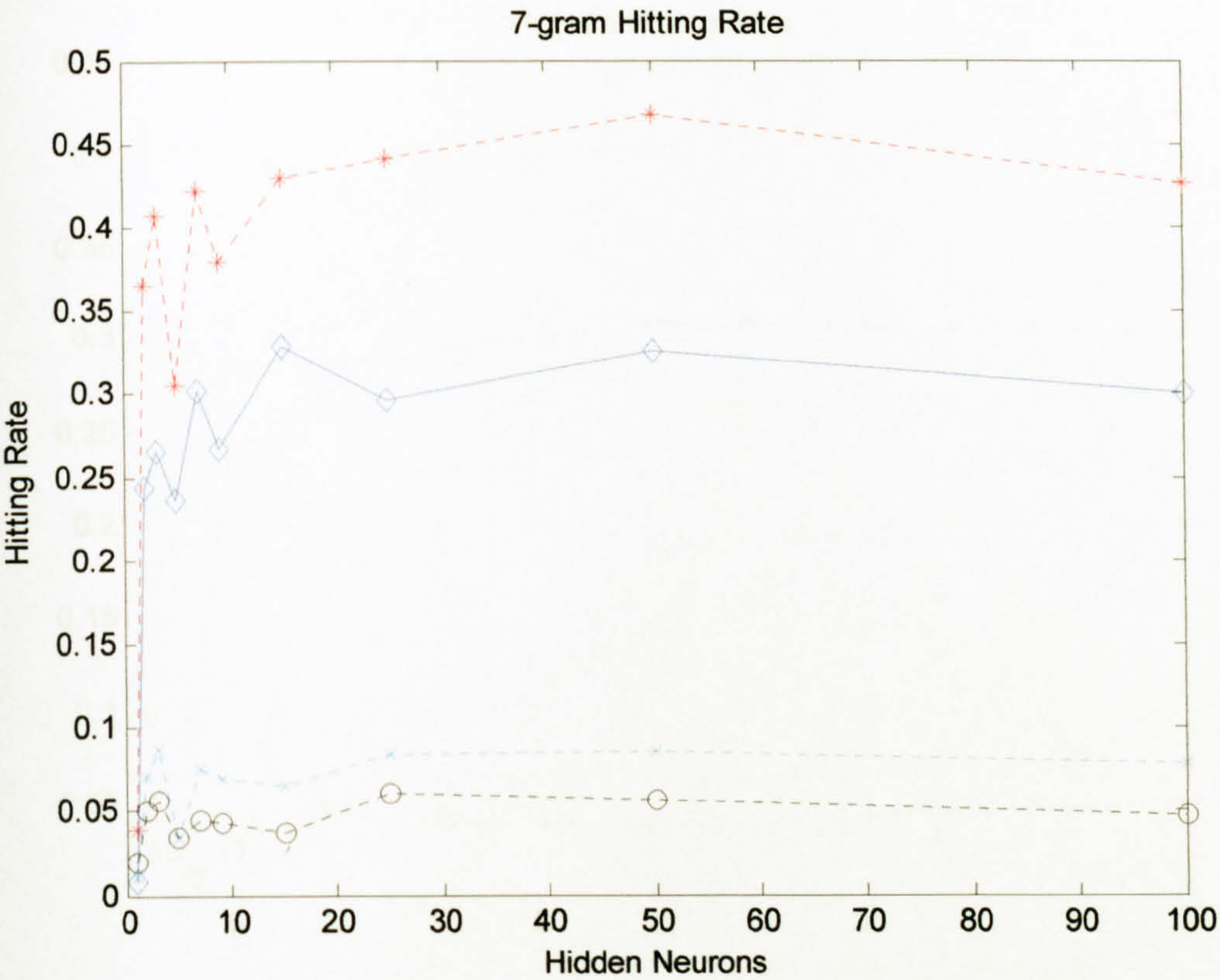
Appendix D

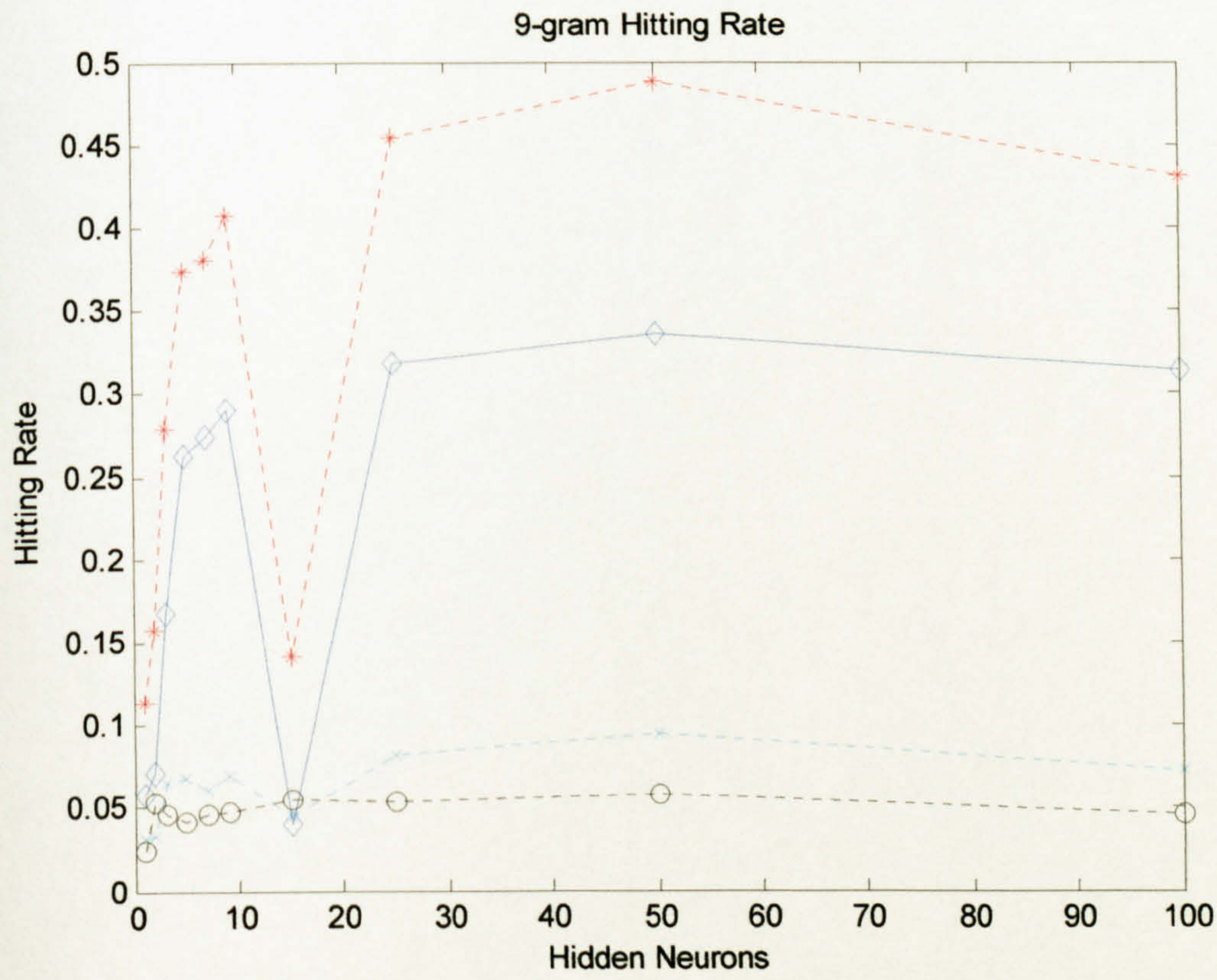
NEURAL NETWORKS MODELLING RESULTS DIAGRAMS

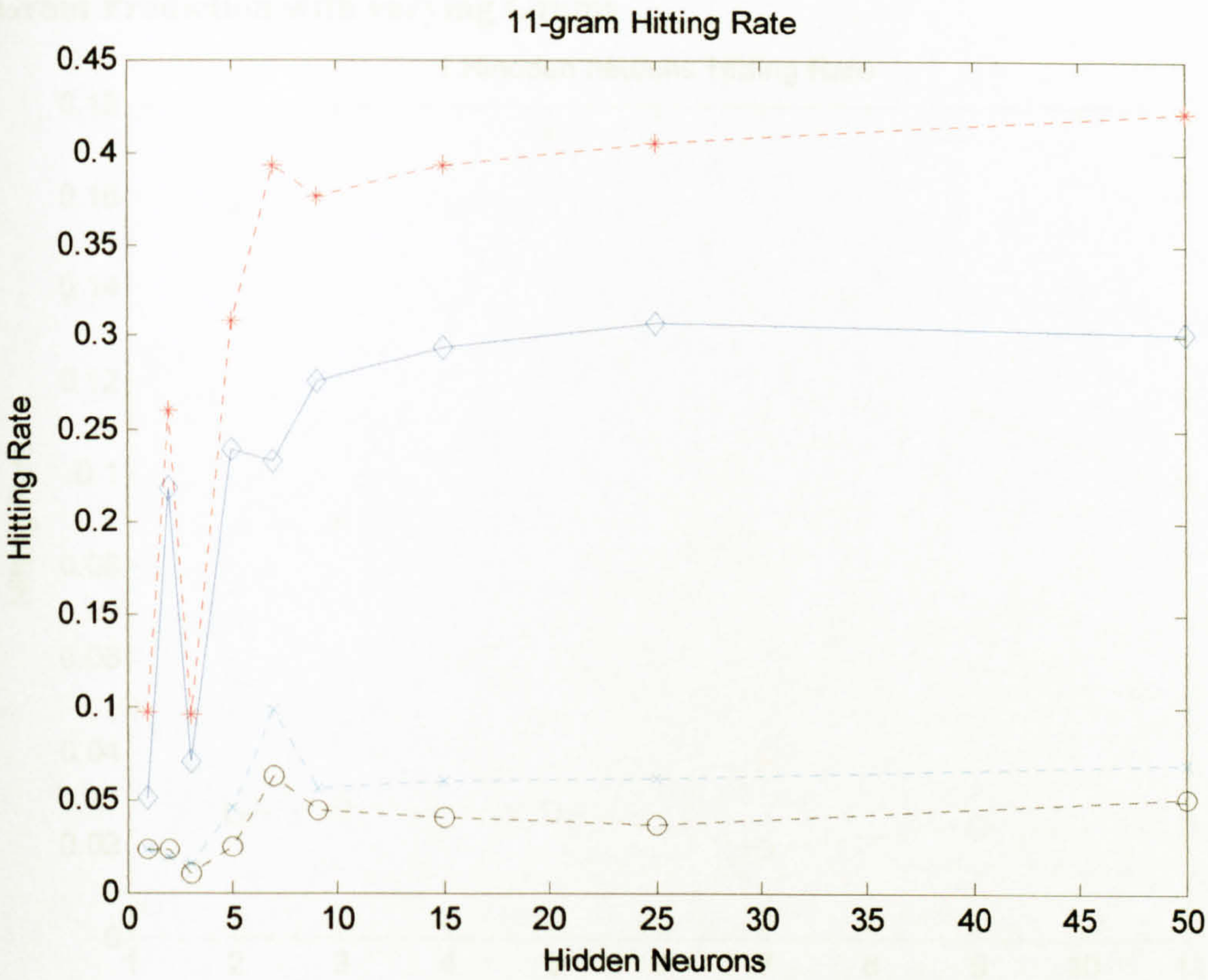
N-Gram Prediction – with varying hidden neurons



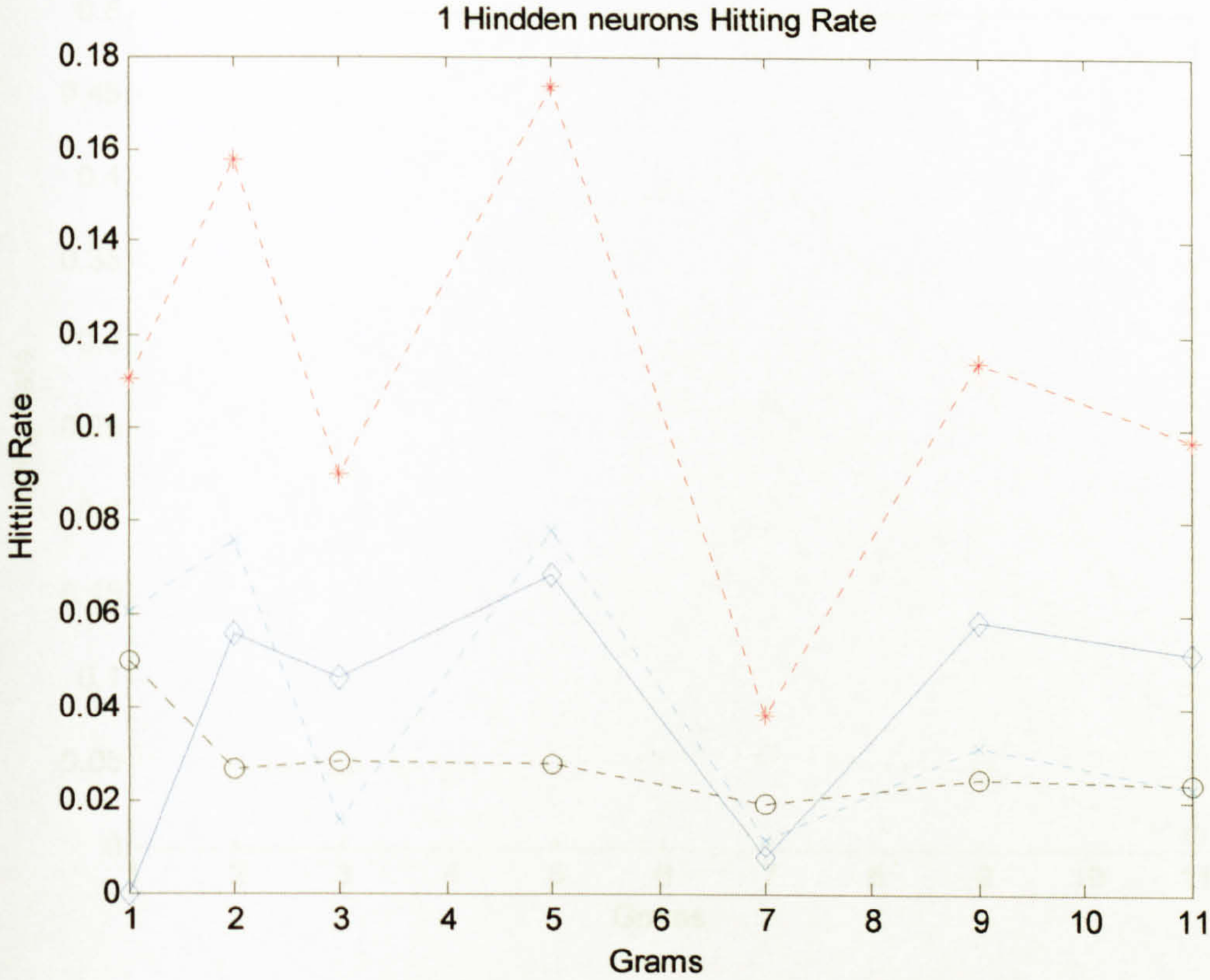

3-gram Hitting Rate
Hitting Rate
Hidden Neurons
5-gram Hitting Rate
Hitting Rate
Hidden Neurons

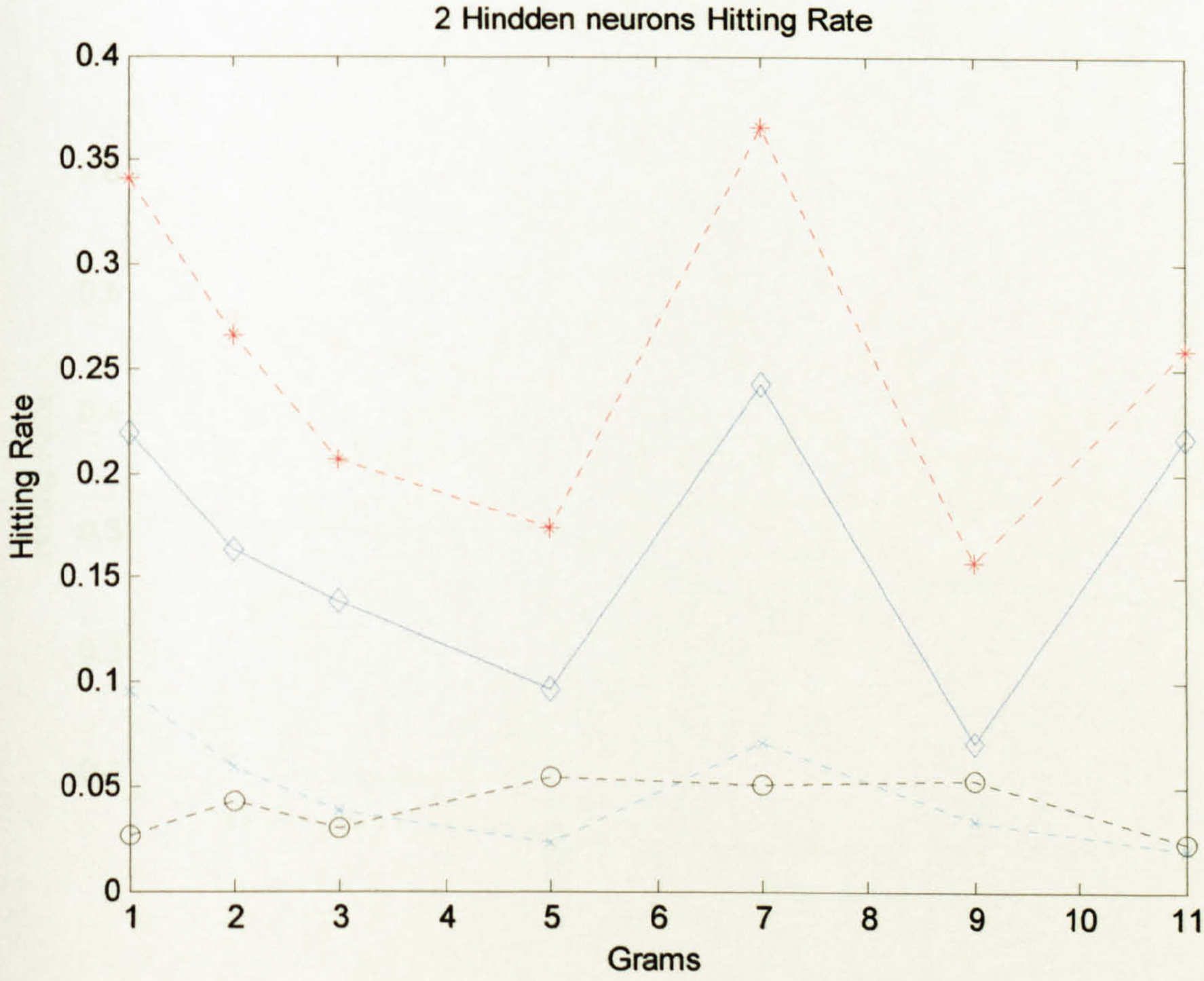

7-gram Hitting Rate
Hitting Rate
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons


9-gram Hitting Rate
Hitting Rate
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons

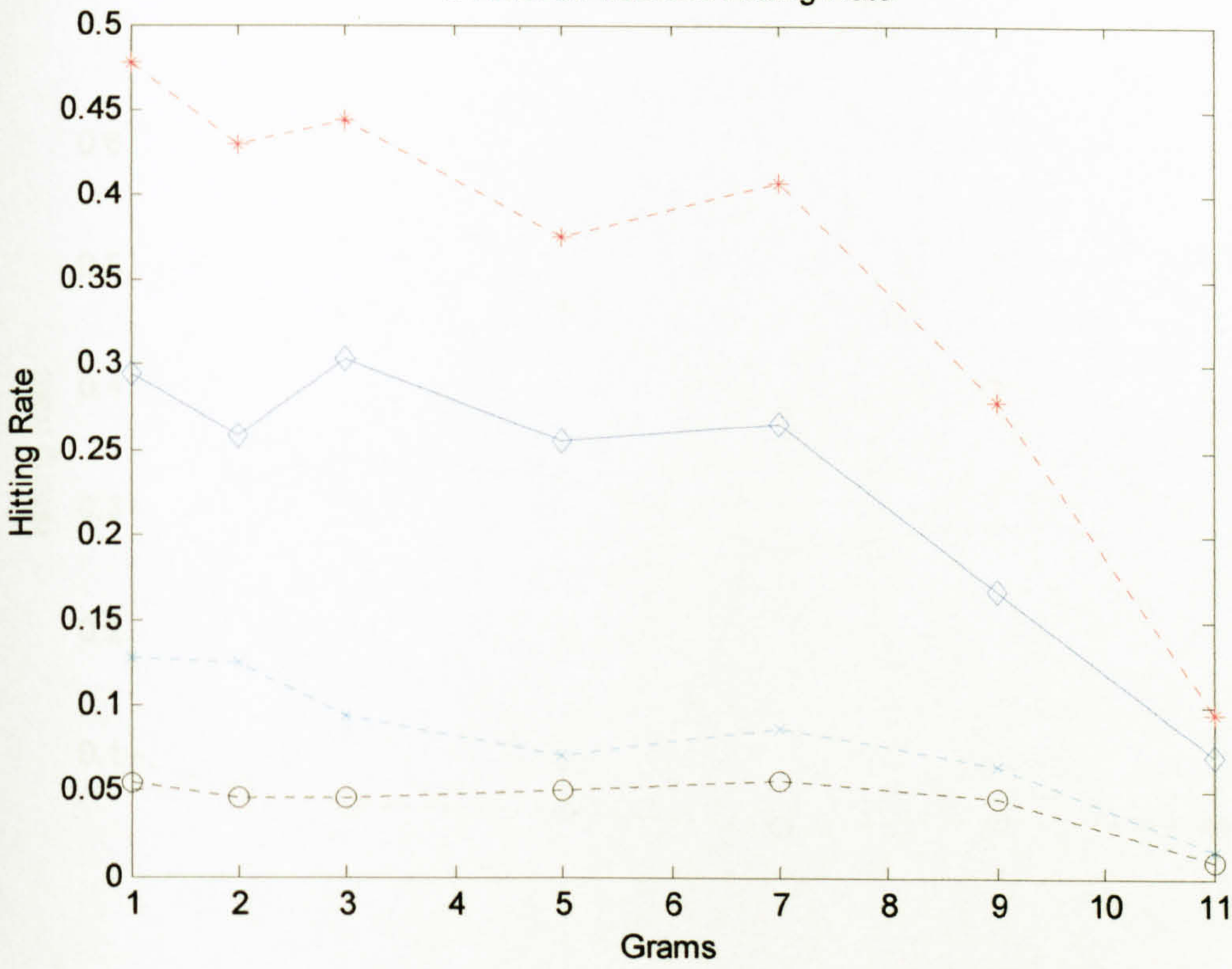

11-gram Hitting Rate
Hitting Rate
Hidden Neurons
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40
45
50

N-Gram Prediction with varying Grams

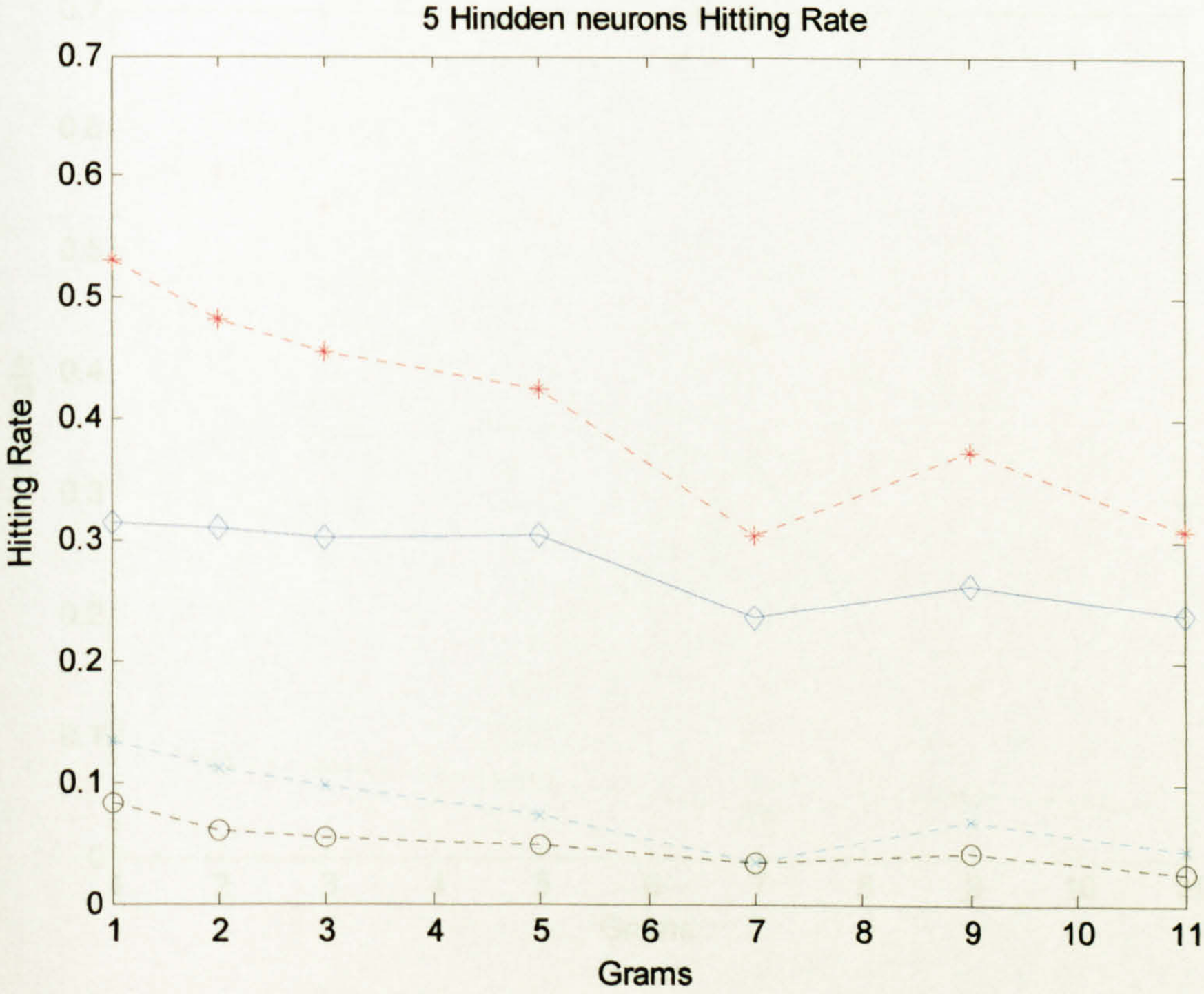


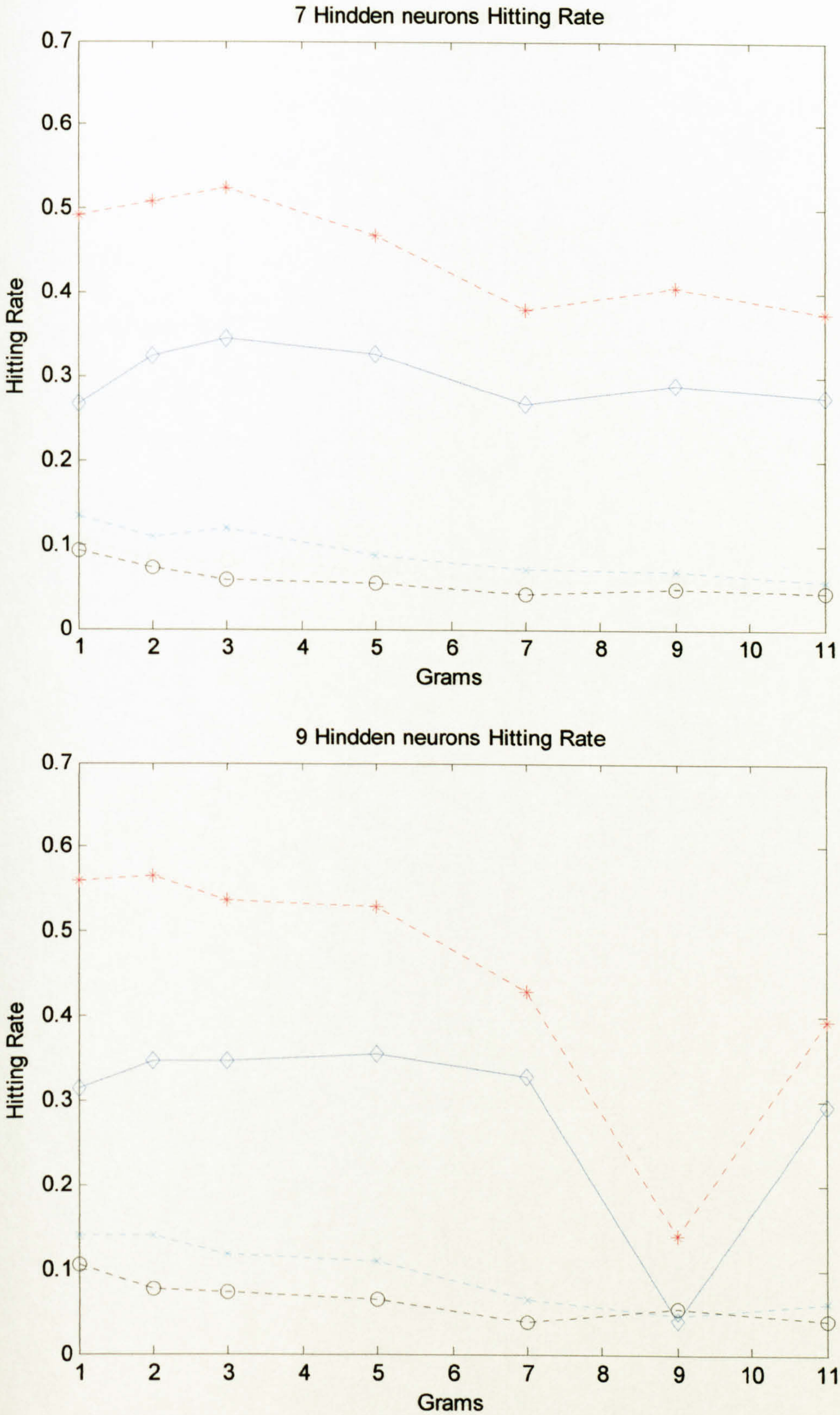


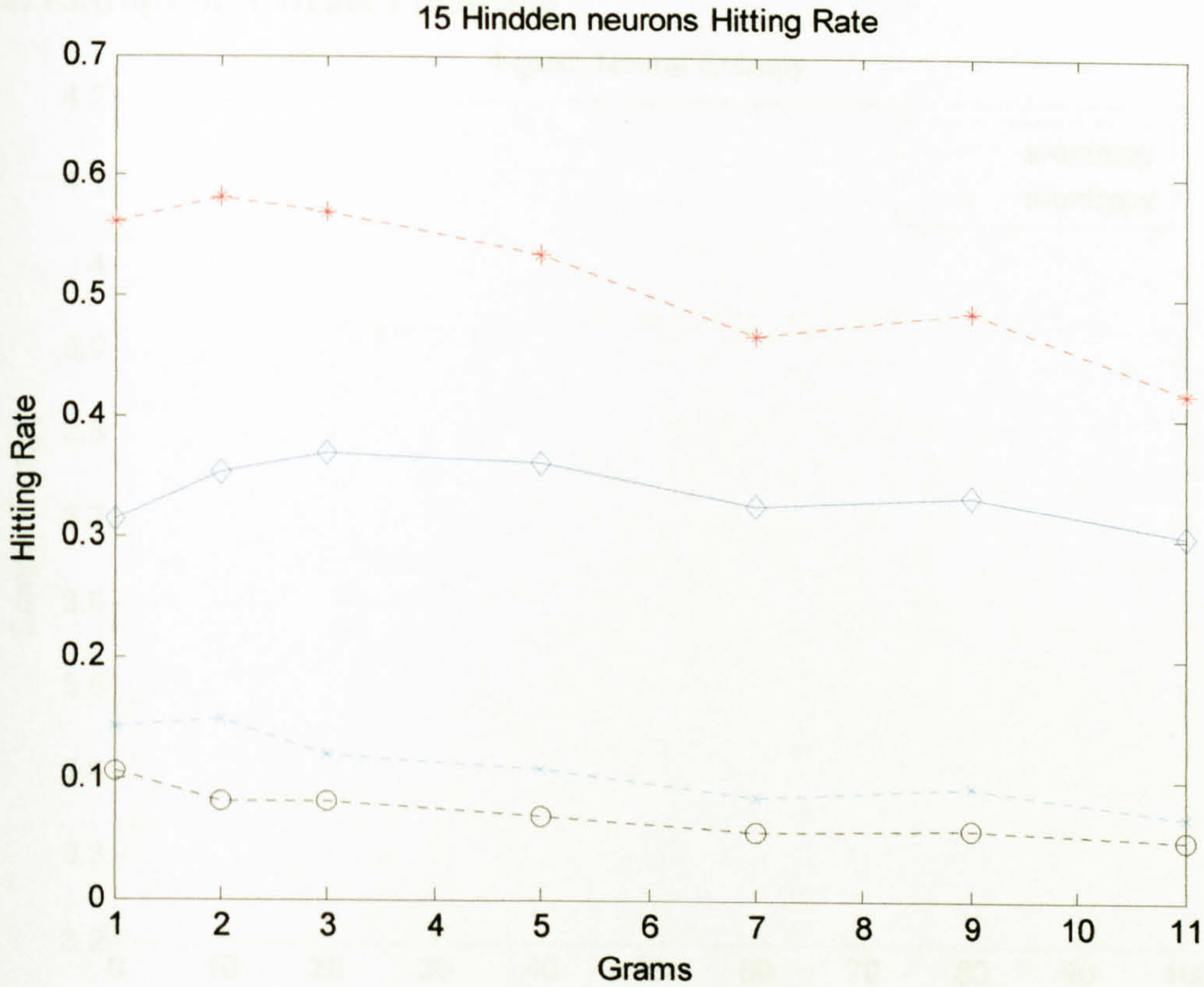
3 Hindden neurons Hitting Rate



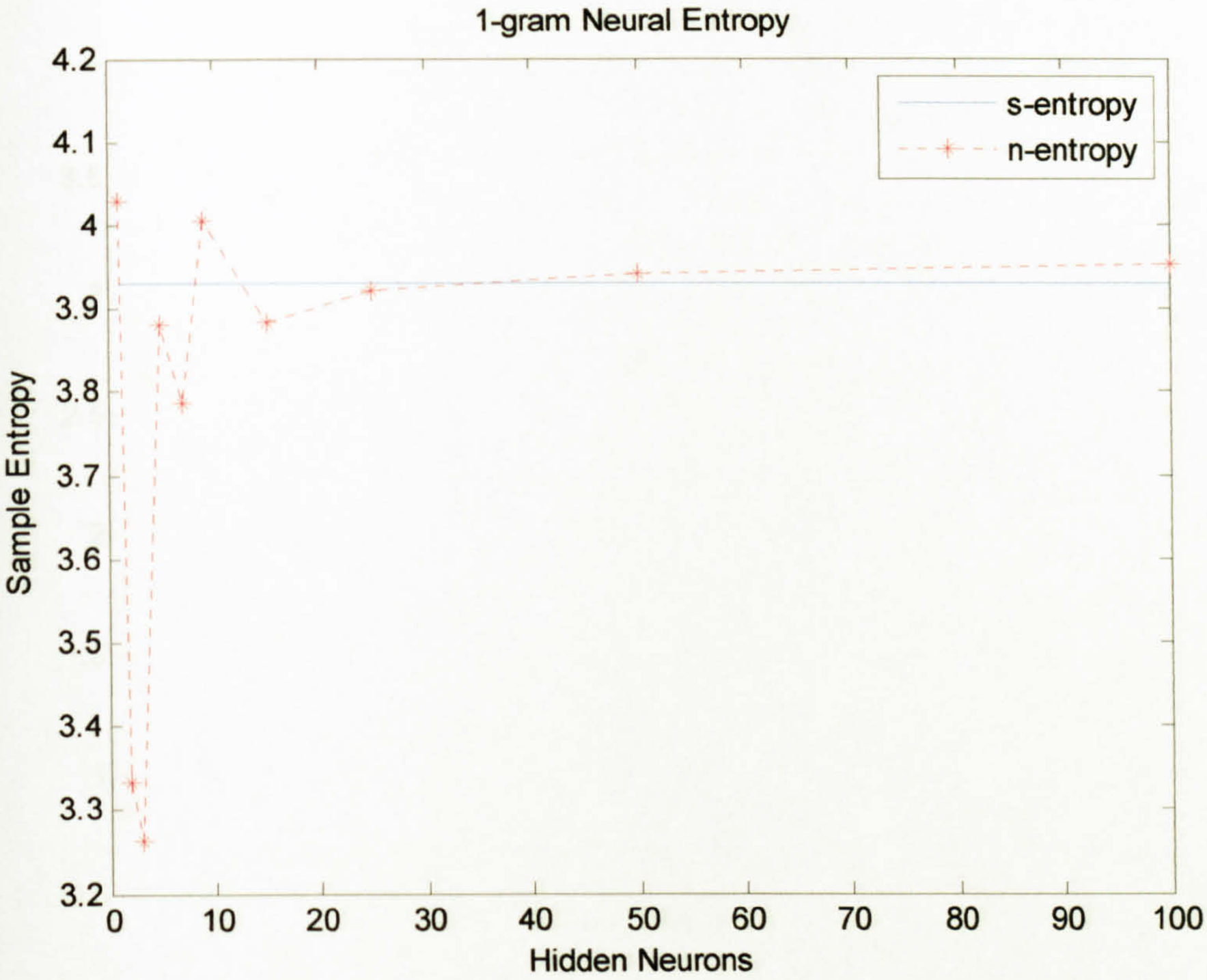
5 Hindden neurons Hitting Rate

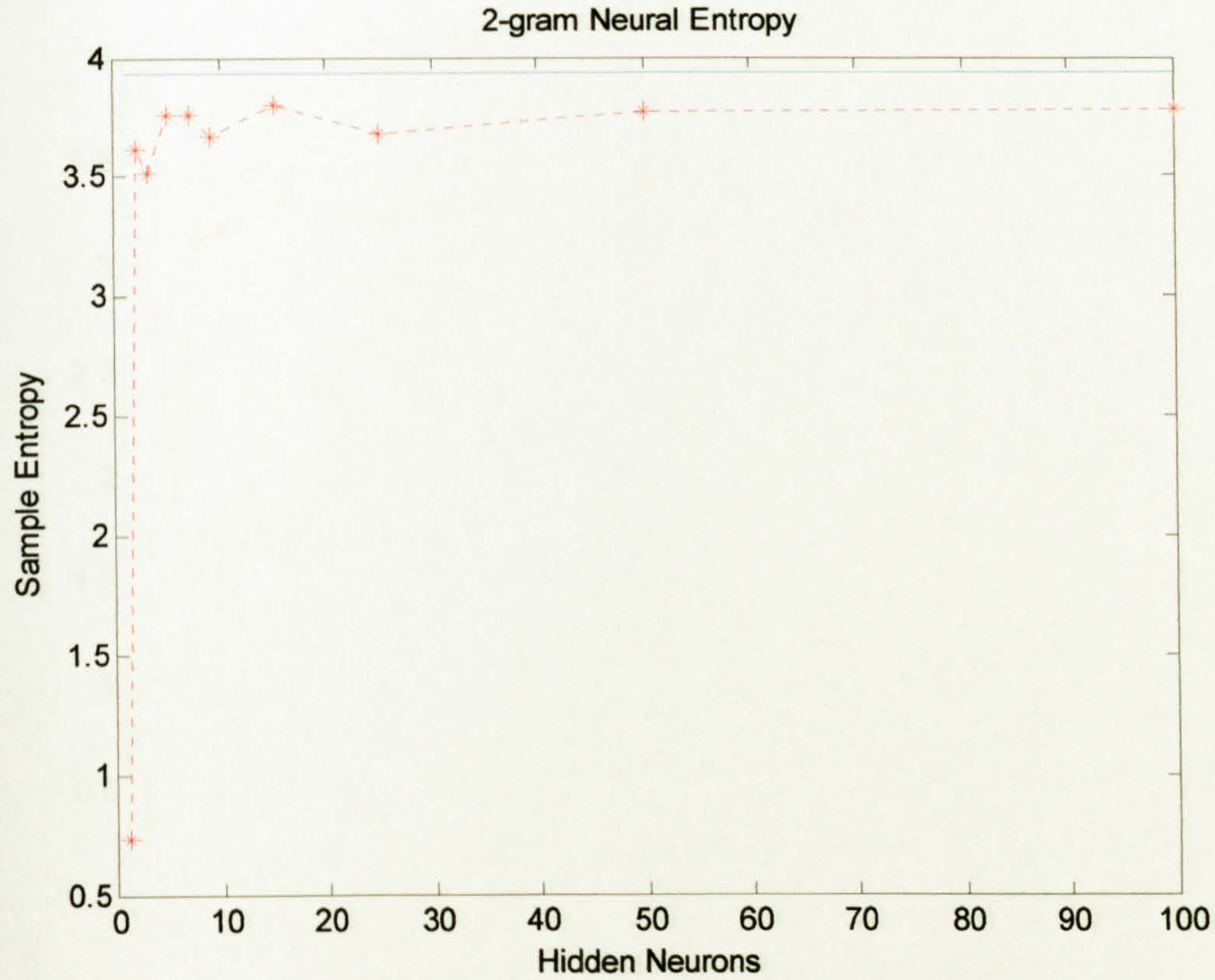


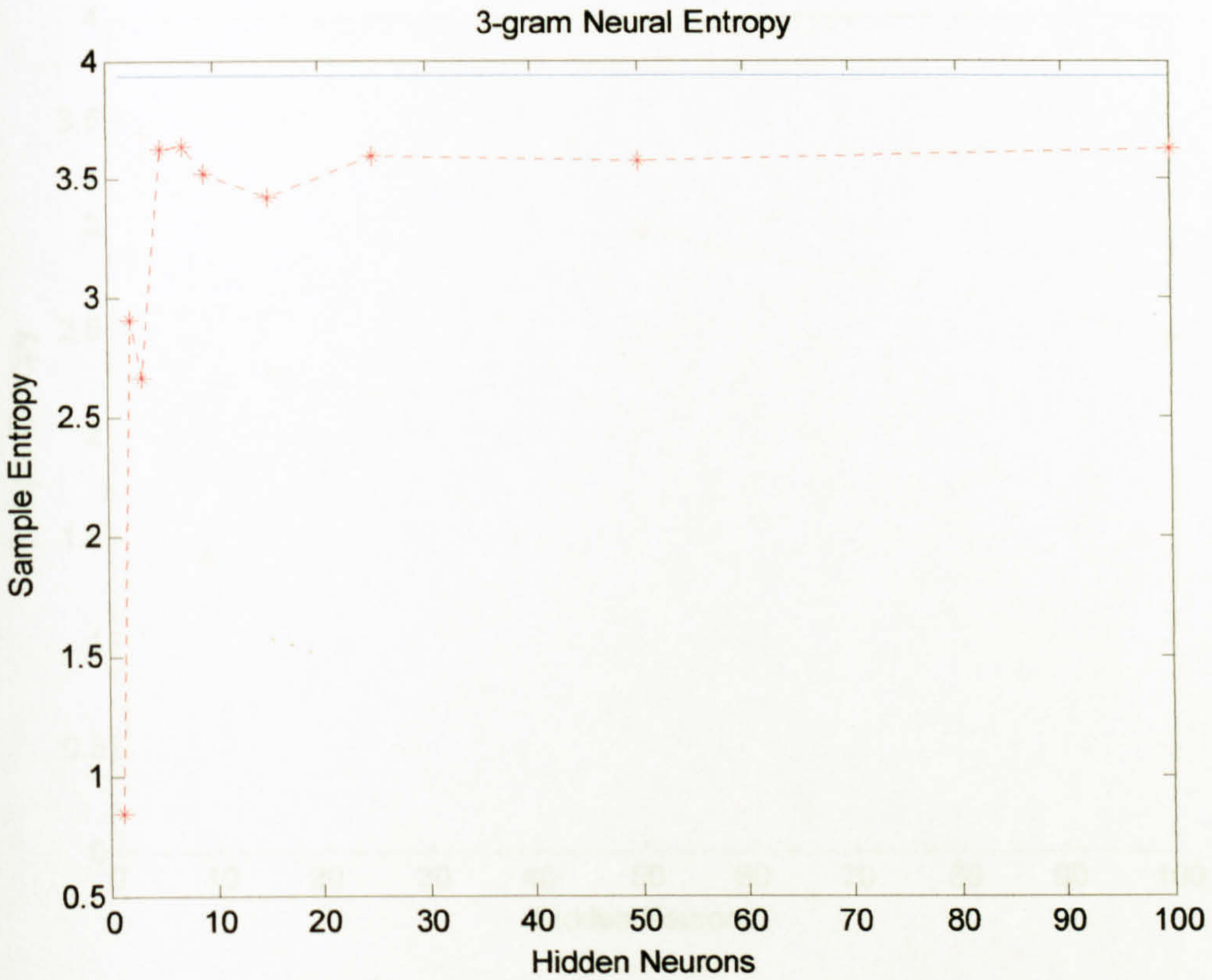

7 Hindden neurons Hitting Rate
Hitting Rate
Grams
9 Hindden neurons Hitting Rate
Hitting Rate
Grams

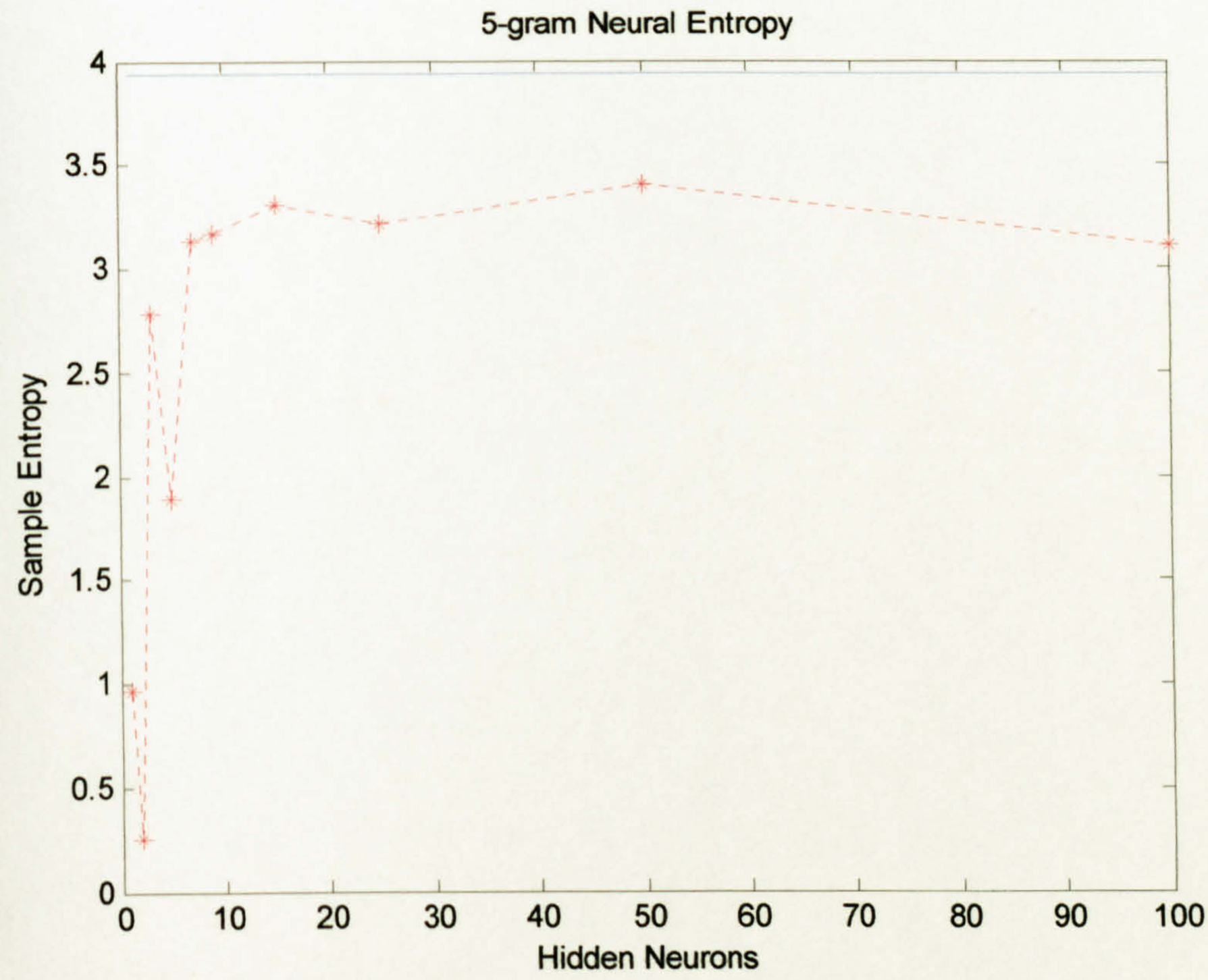

15 Hindden neurons Hitting Rate
Hitting Rate
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6
7
8
9
10
11
Grams

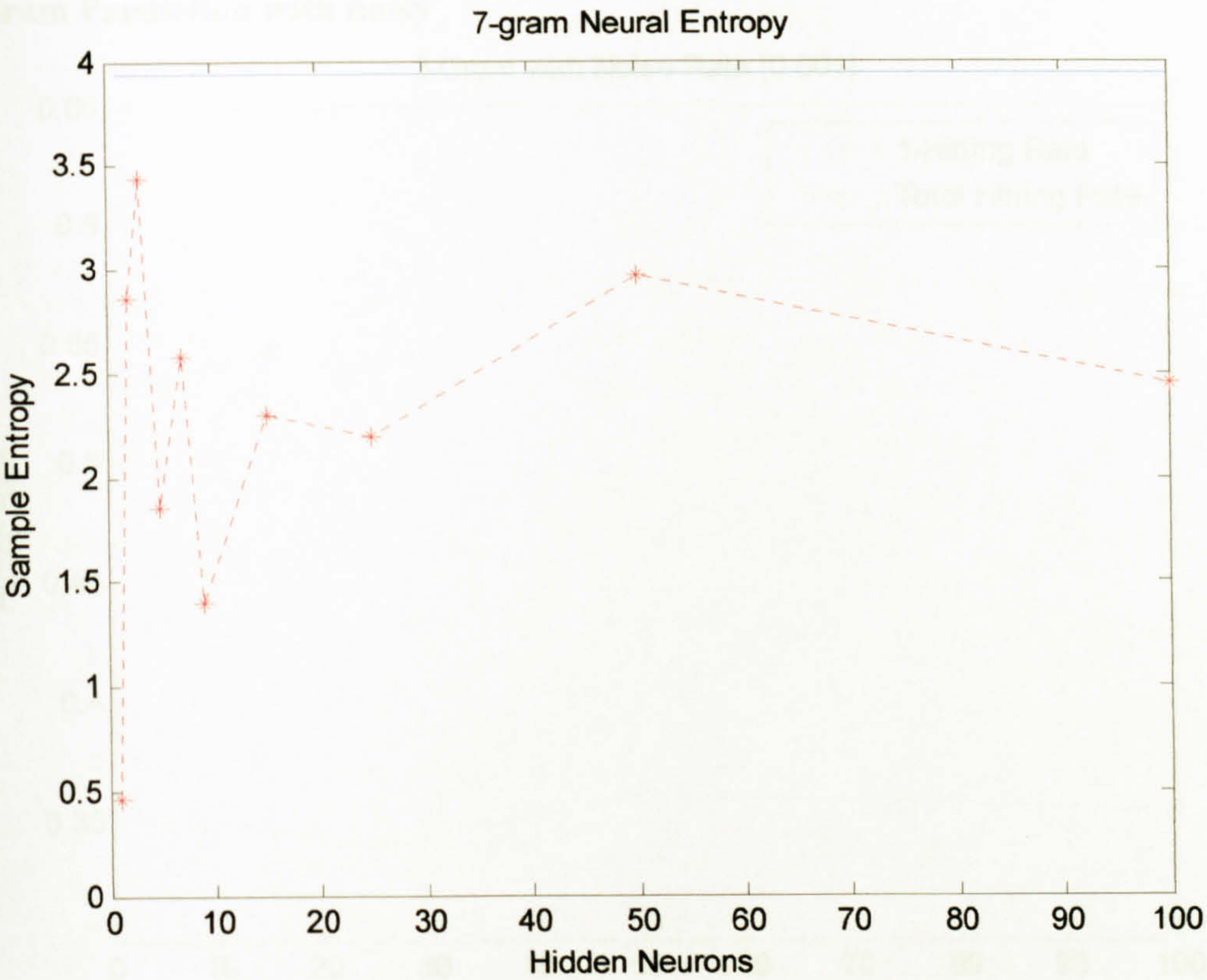
Local Entropy of N-Gram Prediction

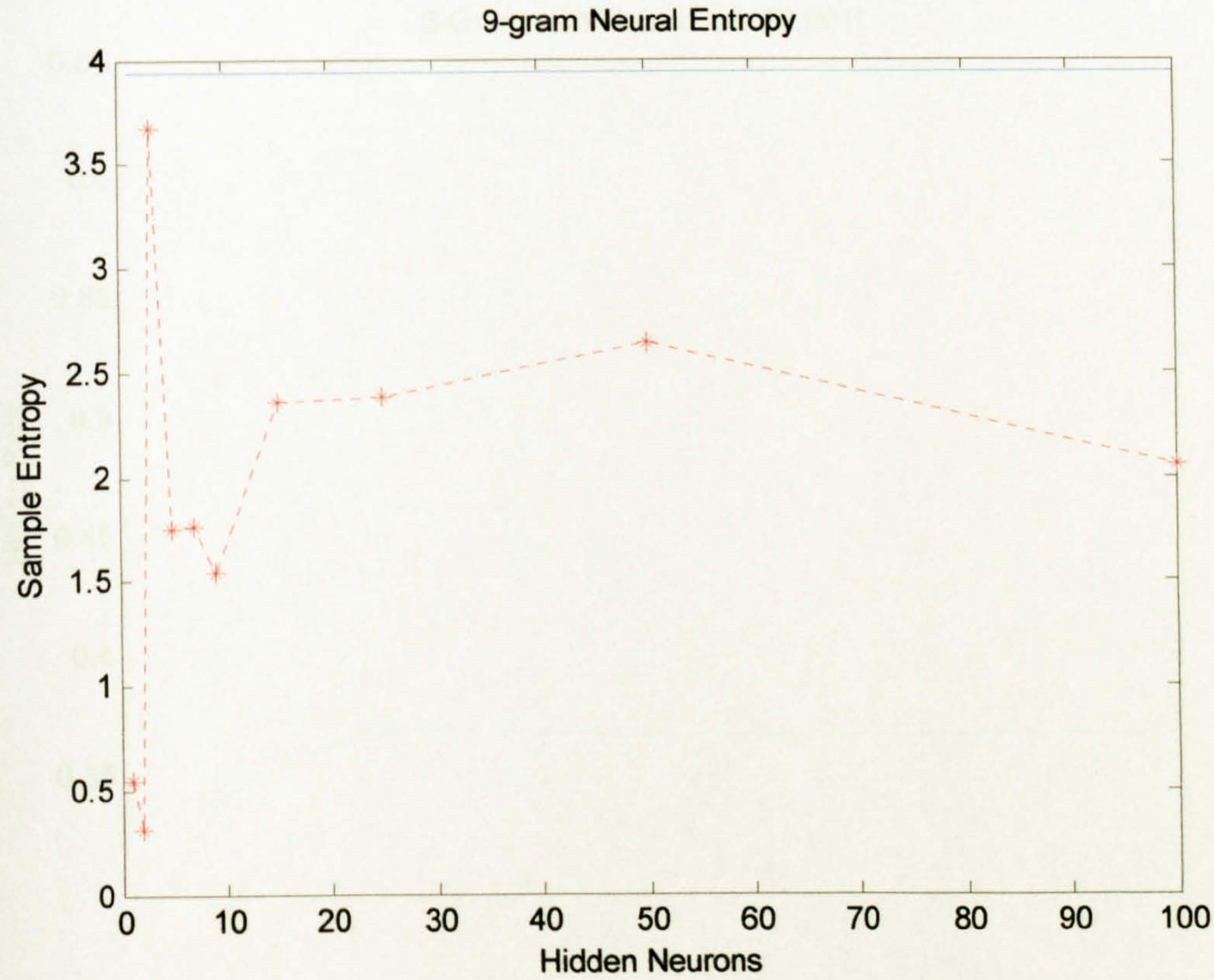




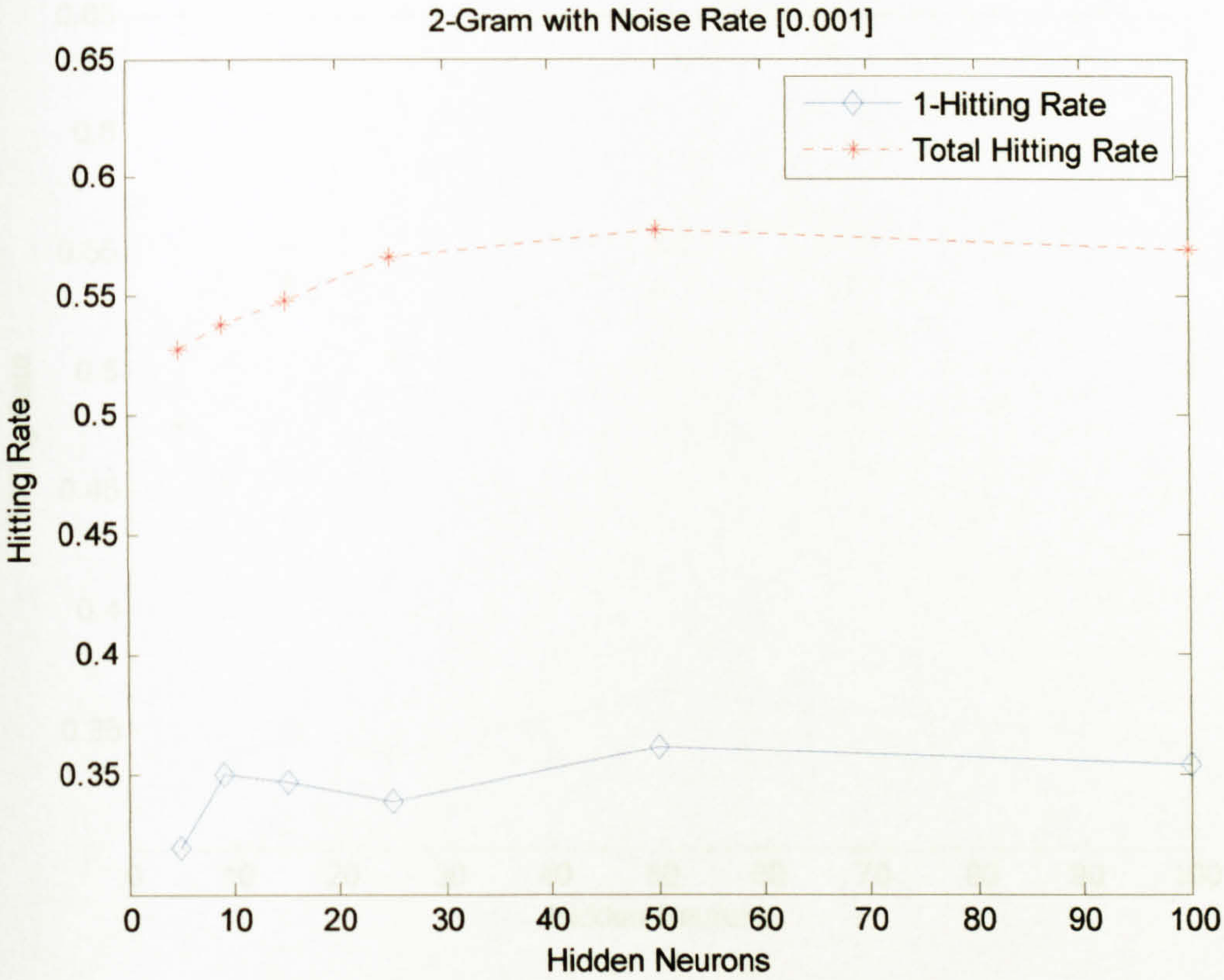

3-gram Neural Entropy
Sample Entropy
Hidden Neurons
4
3.5
3
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60
70
80
90
100

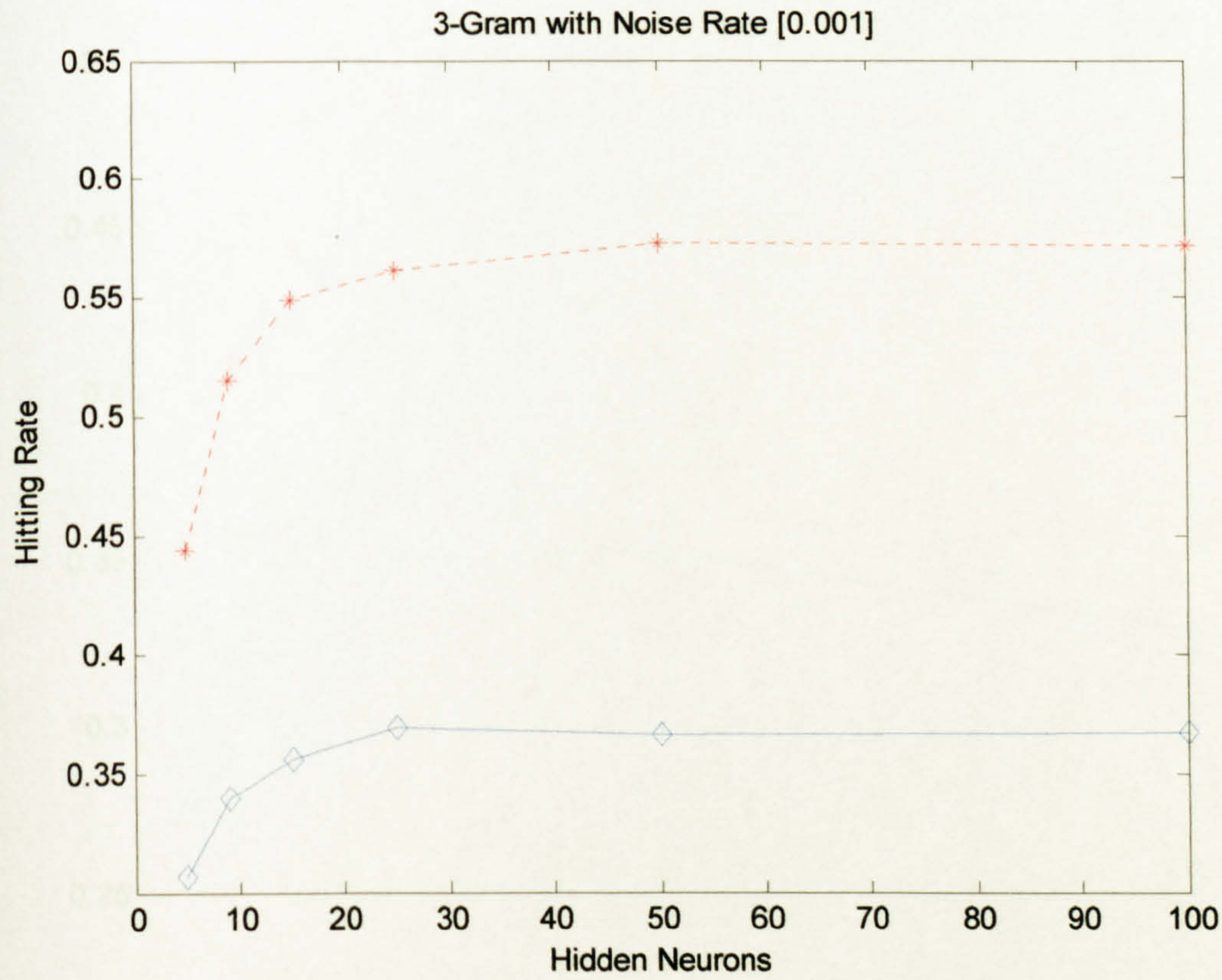

5-gram Neural Entropy
Sample Entropy
Hidden Neurons
4
3.5
3
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60
70
80
90
100

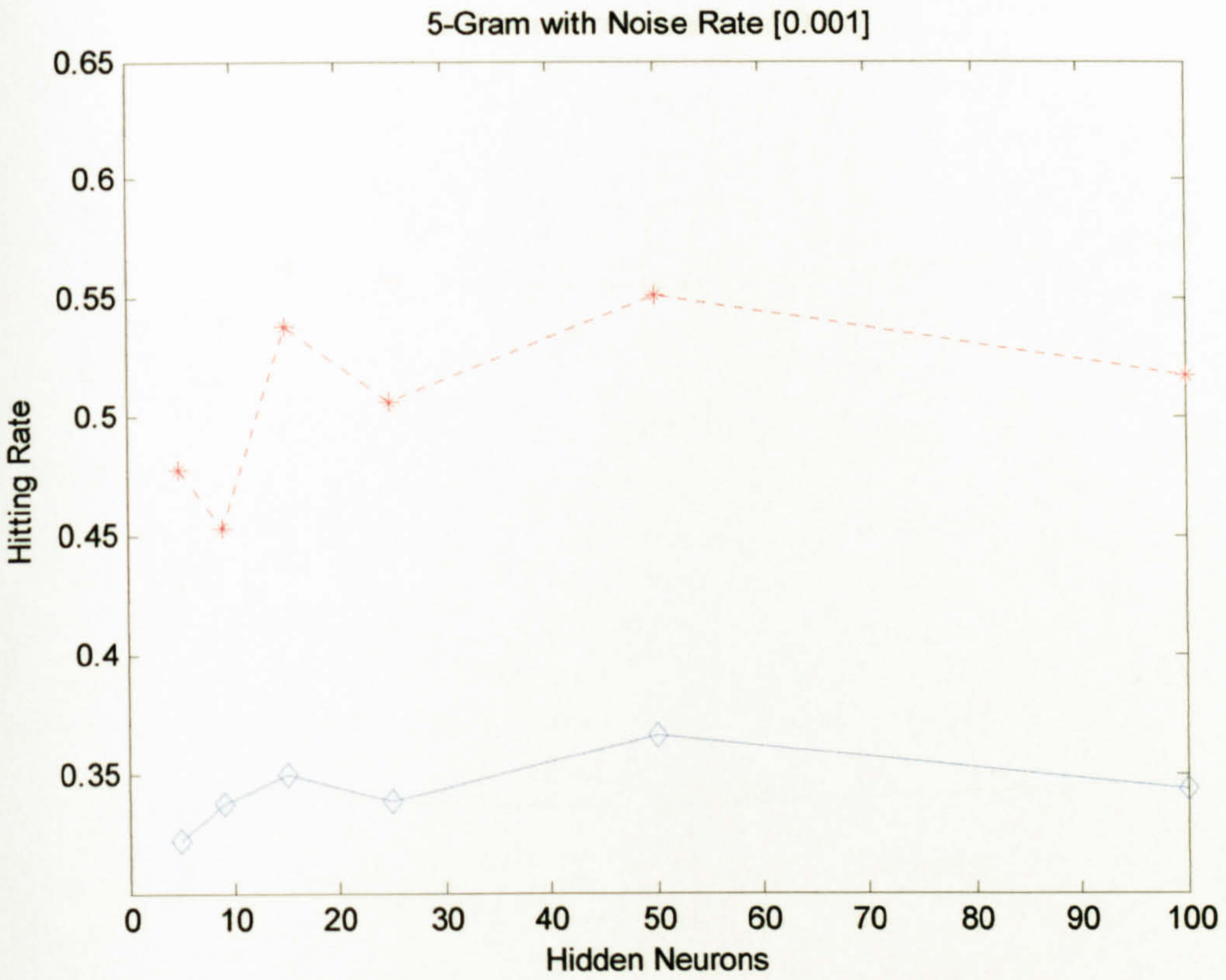

7-gram Neural Entropy
Sample Entropy
Hidden Neurons
0
0.5
1
1.5
2
2.5
3
3.5
4
0
10
20
30
40
50
60
70
80
90
100


9-gram Neural Entropy
Sample Entropy
Hidden Neurons
0
0.5
1
1.5
2
2.5
3
3.5
4
0
10
20
30
40
50
60
70
80
90
100

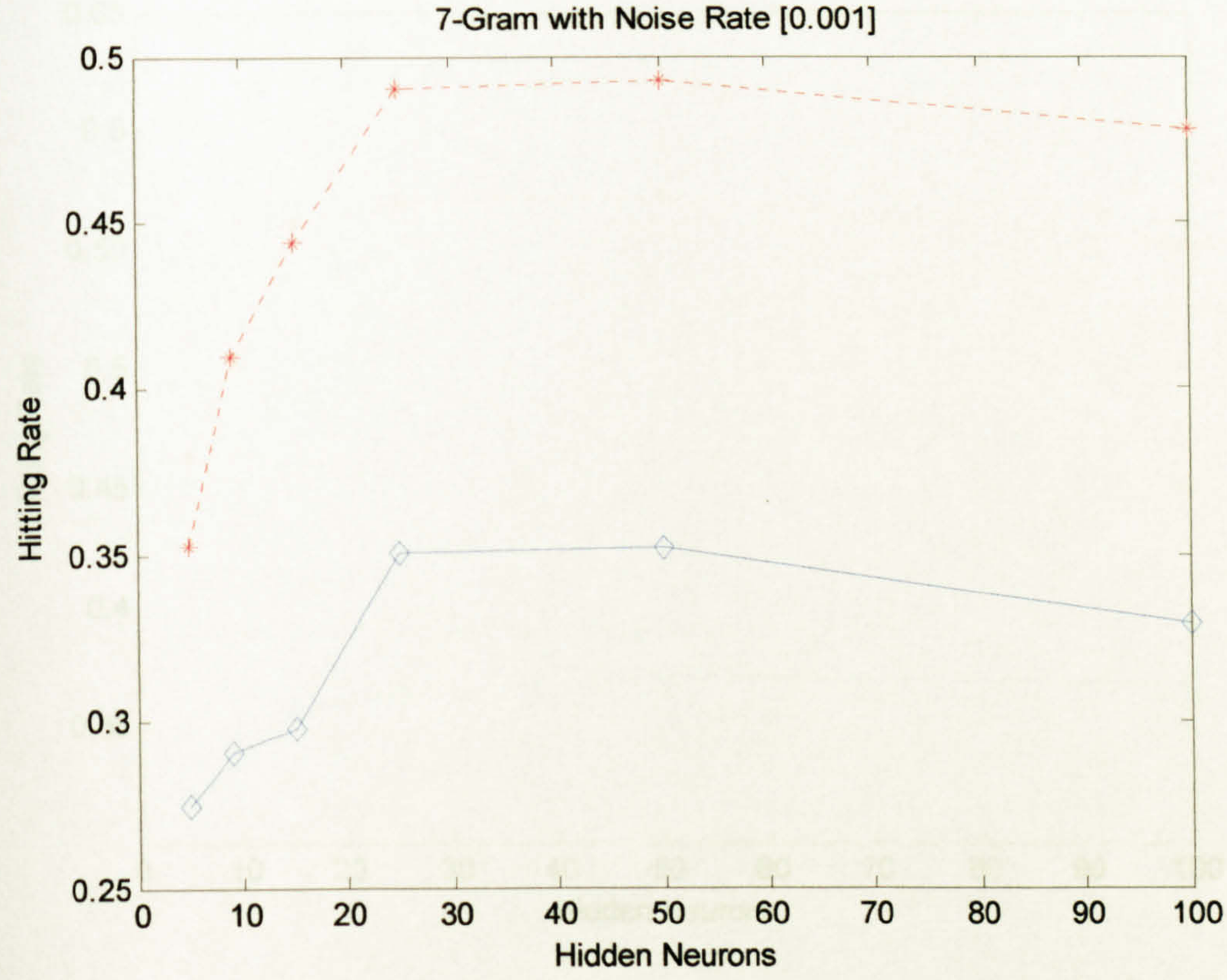
N-Gram Prediction with noisy

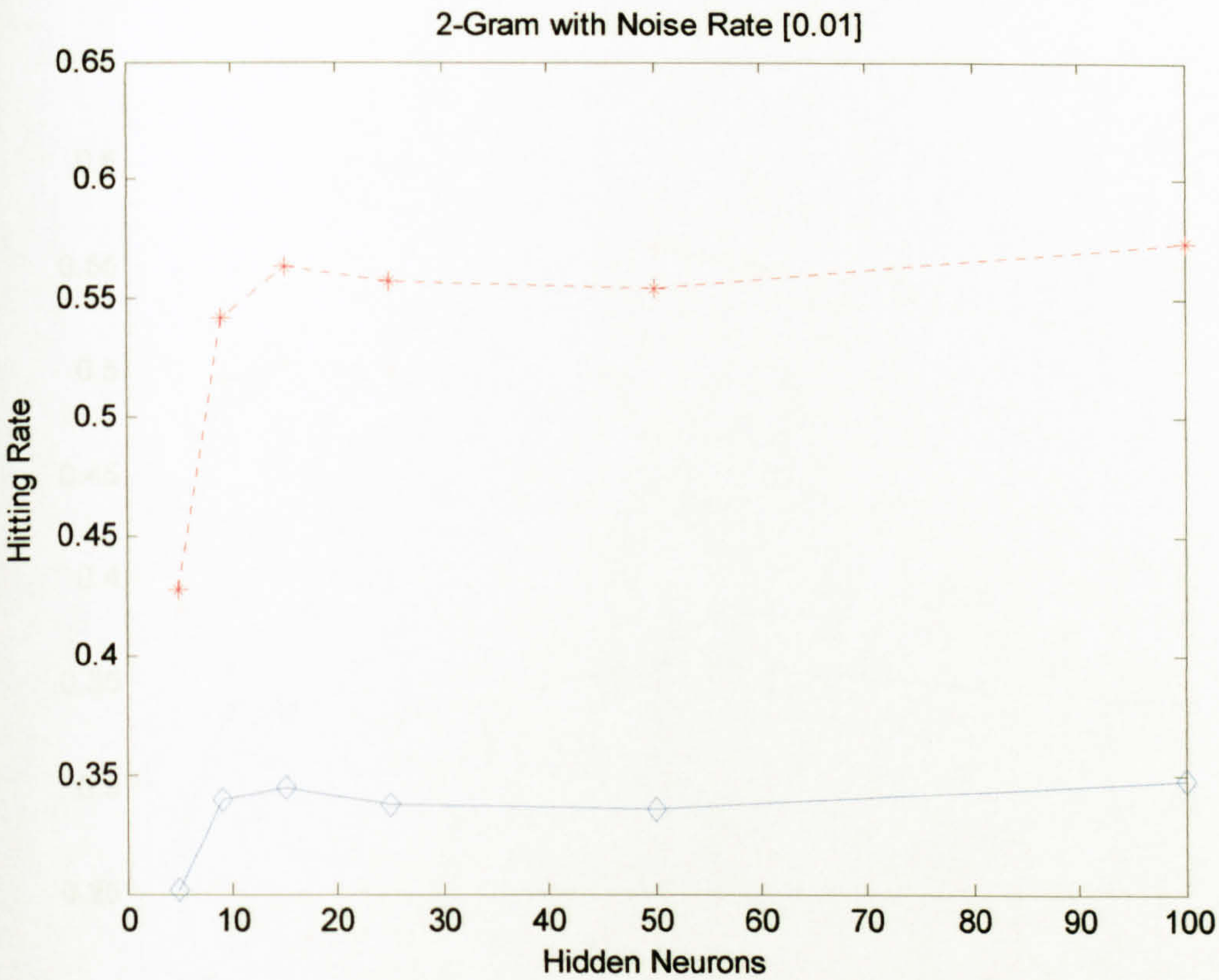


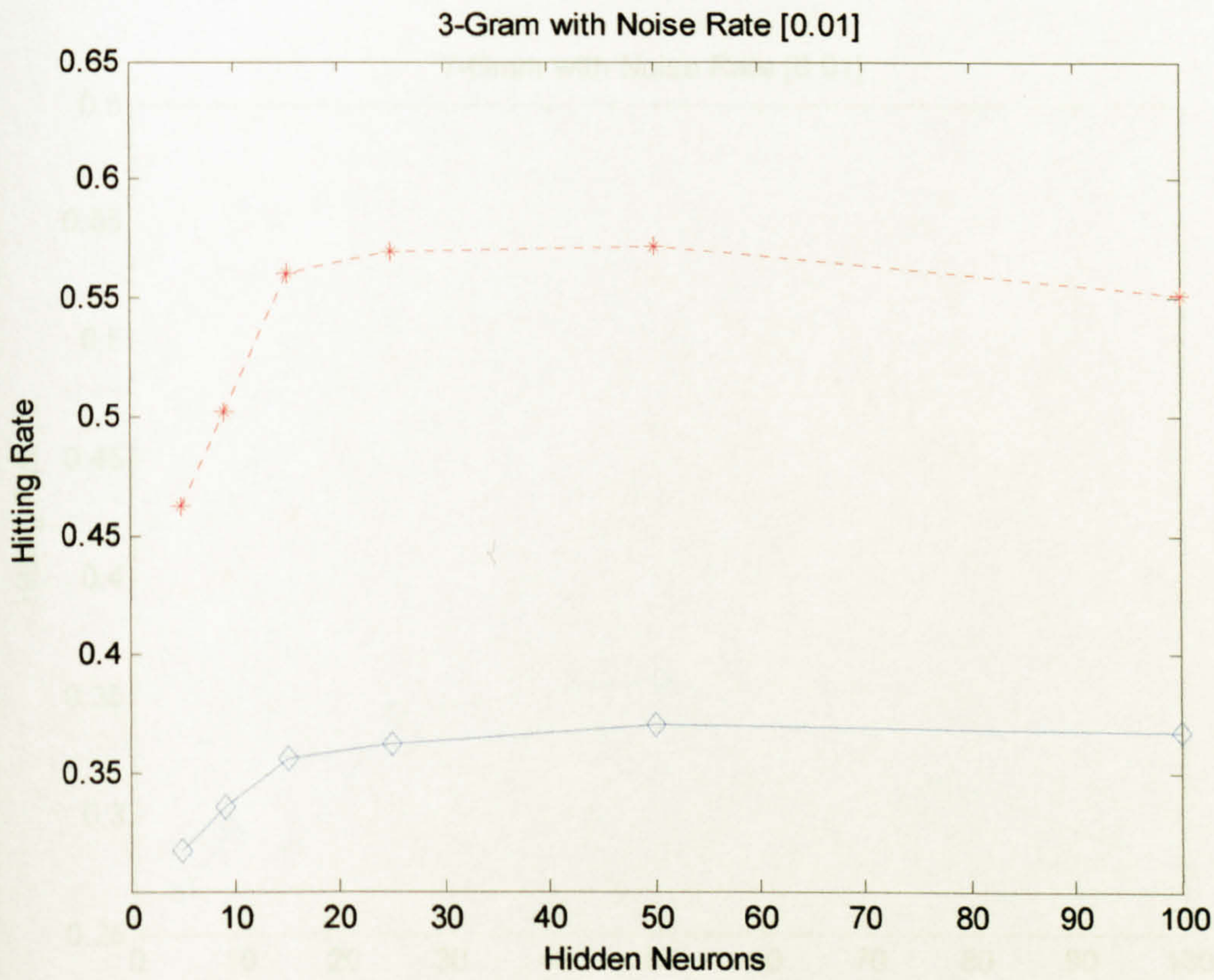


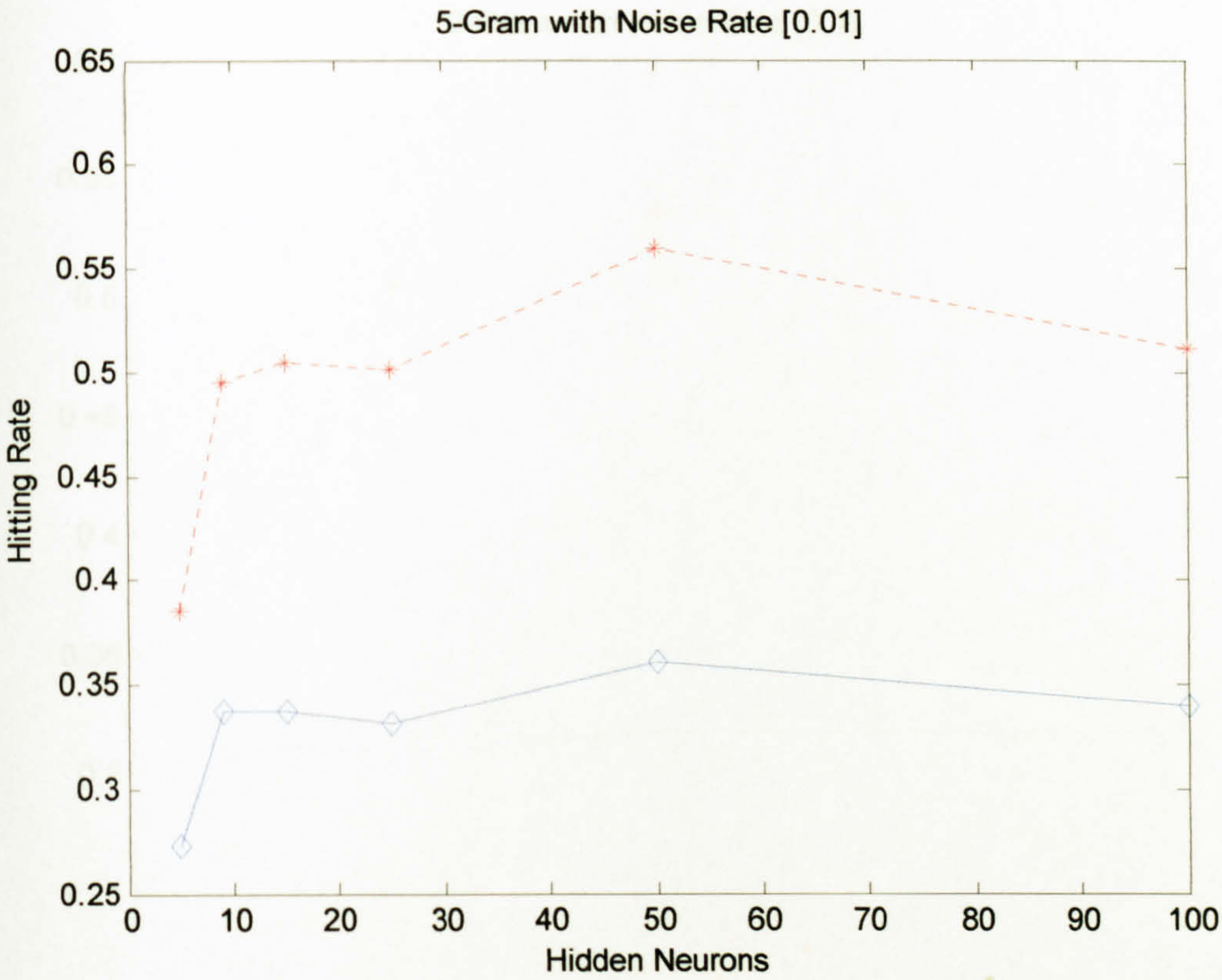


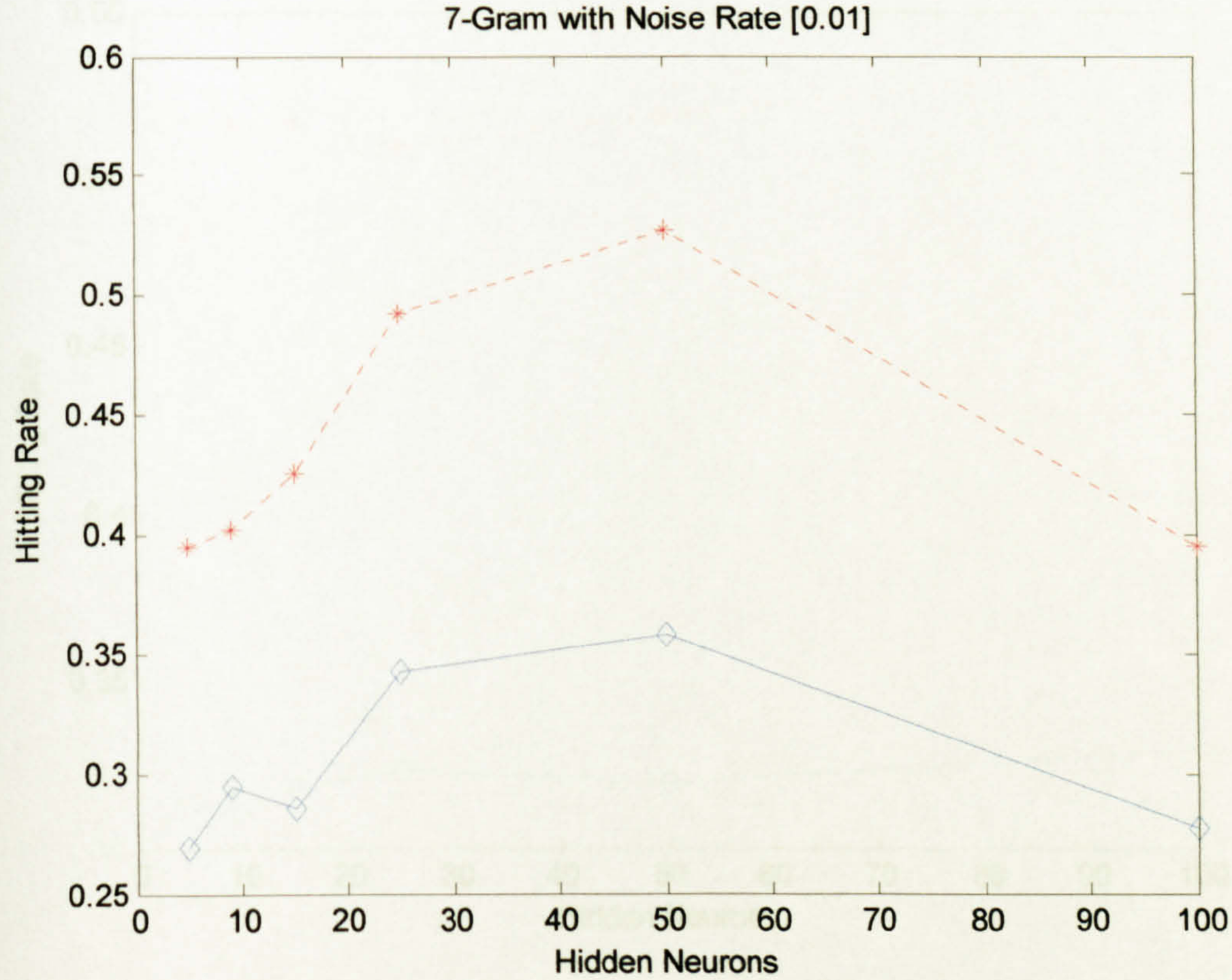
7-Gram with Noise Rate [0.001]

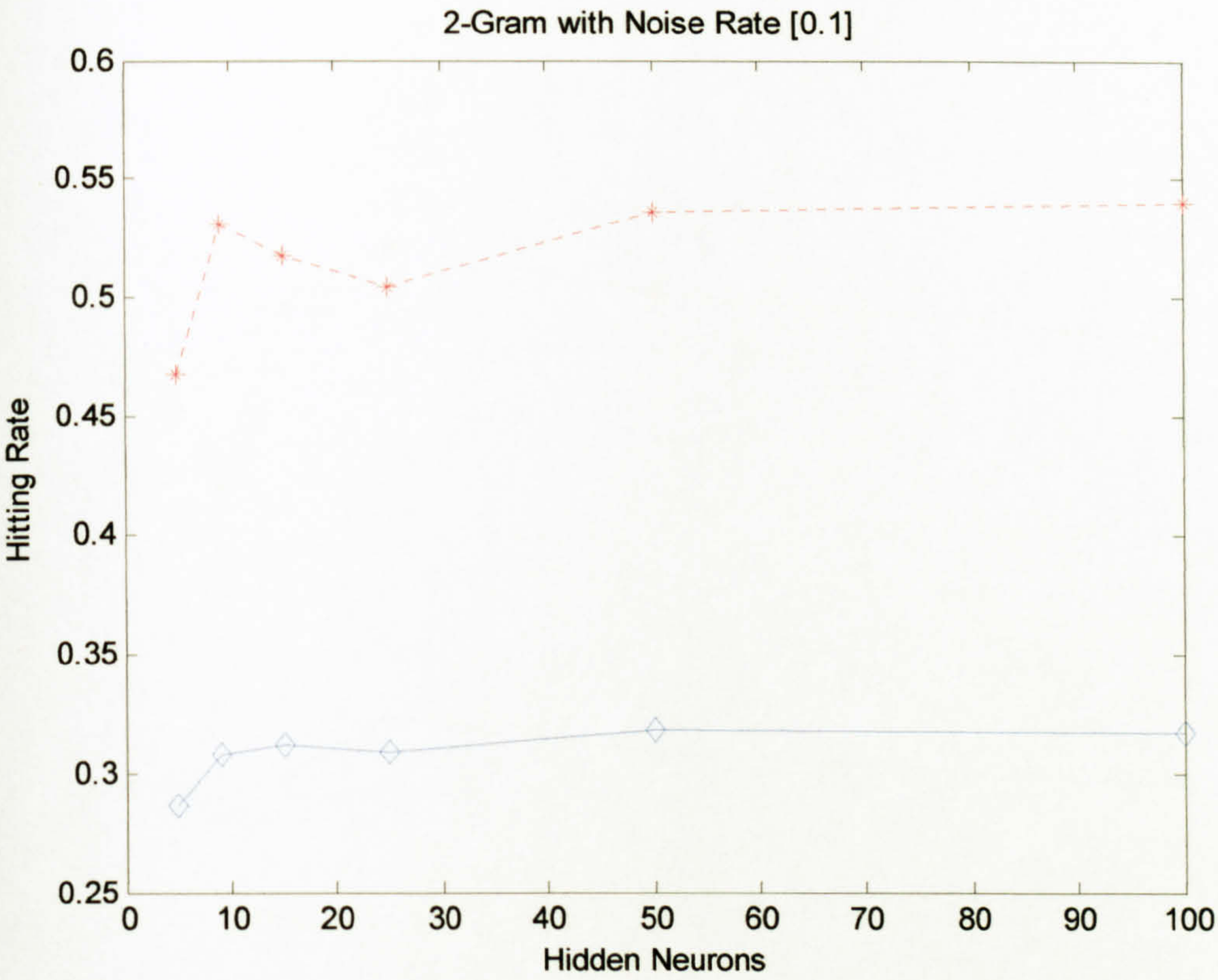


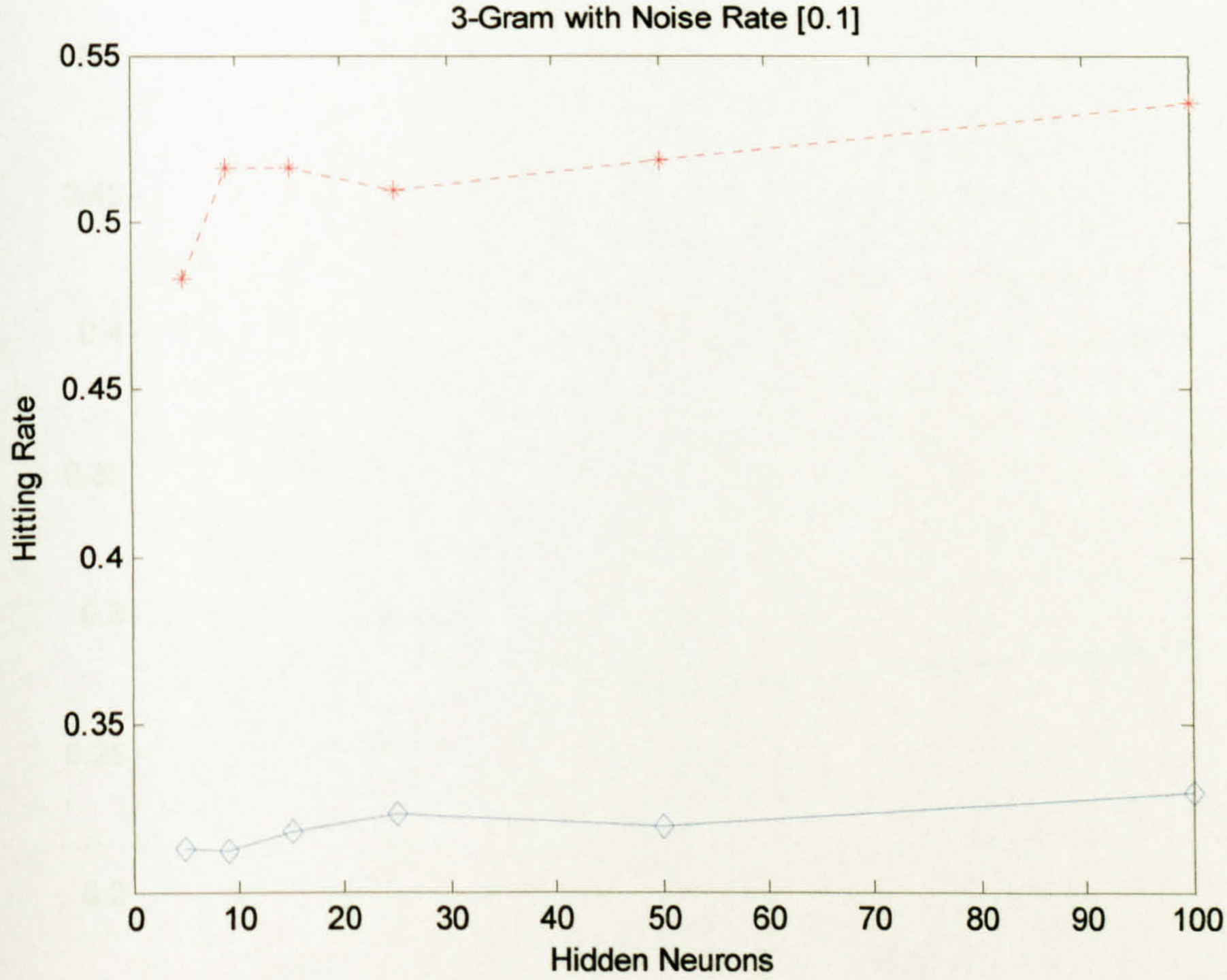

2-Gram with Noise Rate [0.01]
Hitting Rate
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons

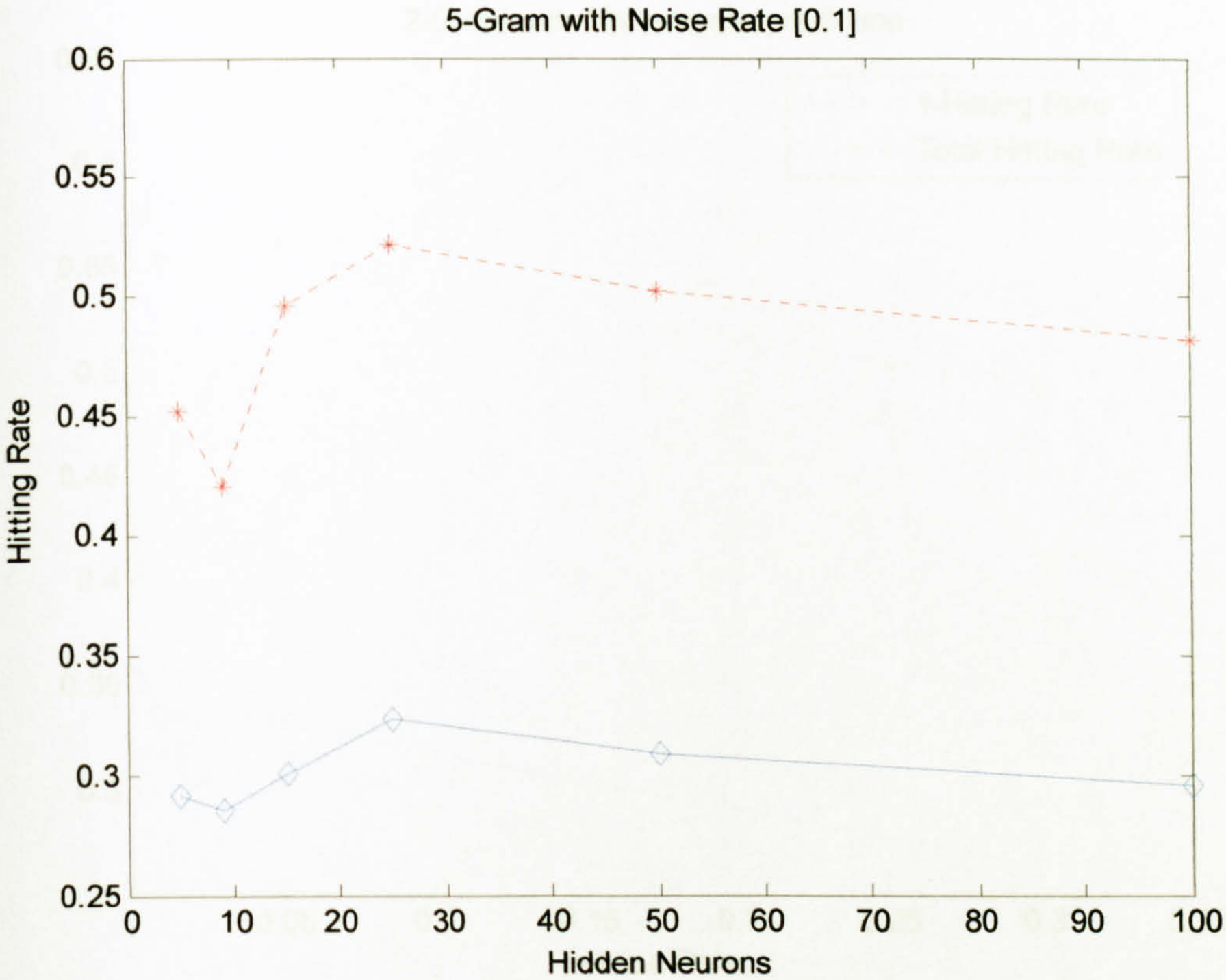

3-Gram with Noise Rate [0.01]
Hitting Rate
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons

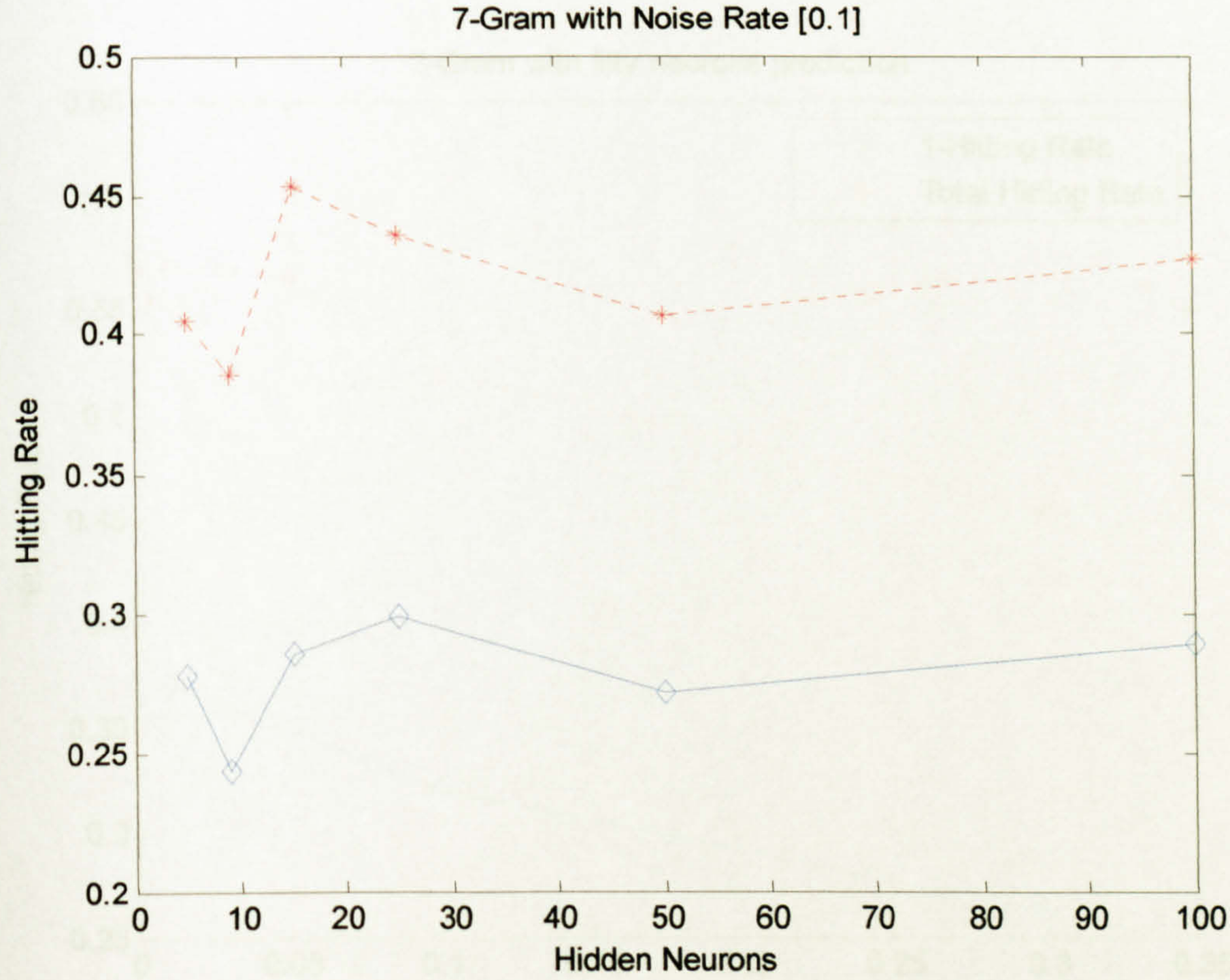

5-Gram with Noise Rate [0.01]
Hitting Rate
Hidden Neurons
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
10
20
30
40
50
60
70
80
90
100

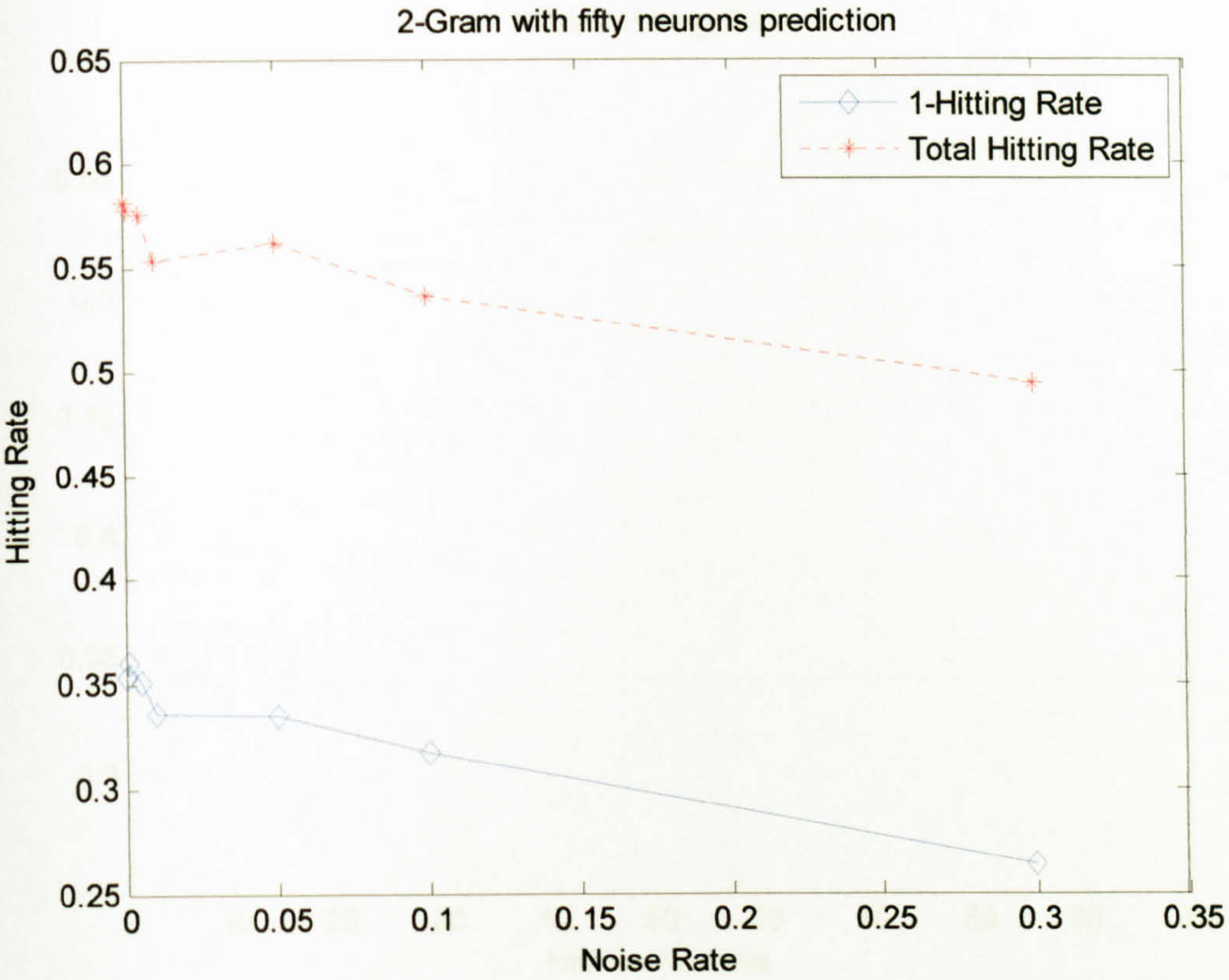

7-Gram with Noise Rate [0.01]
Hitting Rate
Hidden Neurons
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
10
20
30
40
50
60
70
80
90
100

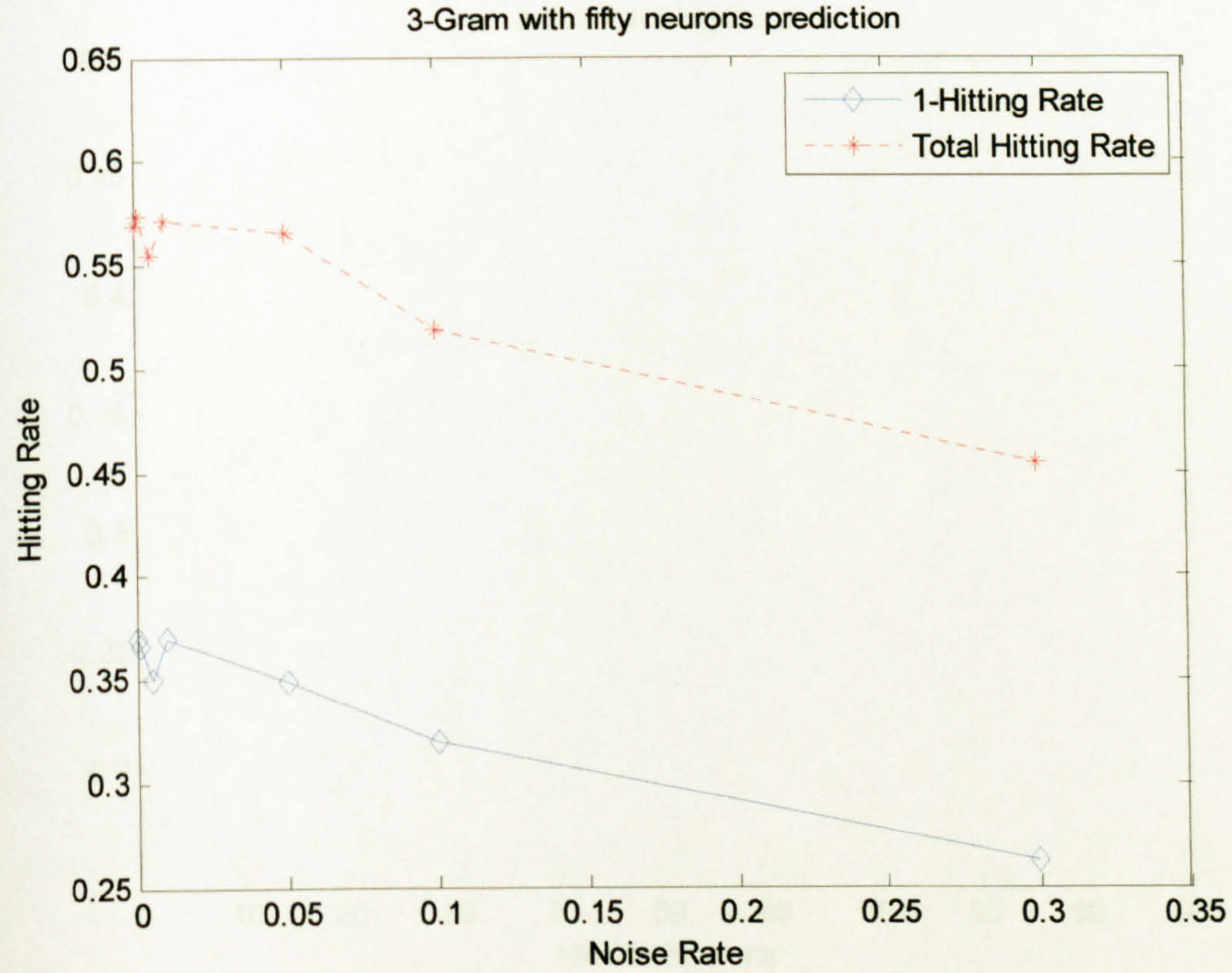

2-Gram with Noise Rate [0.1]
Hitting Rate
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons

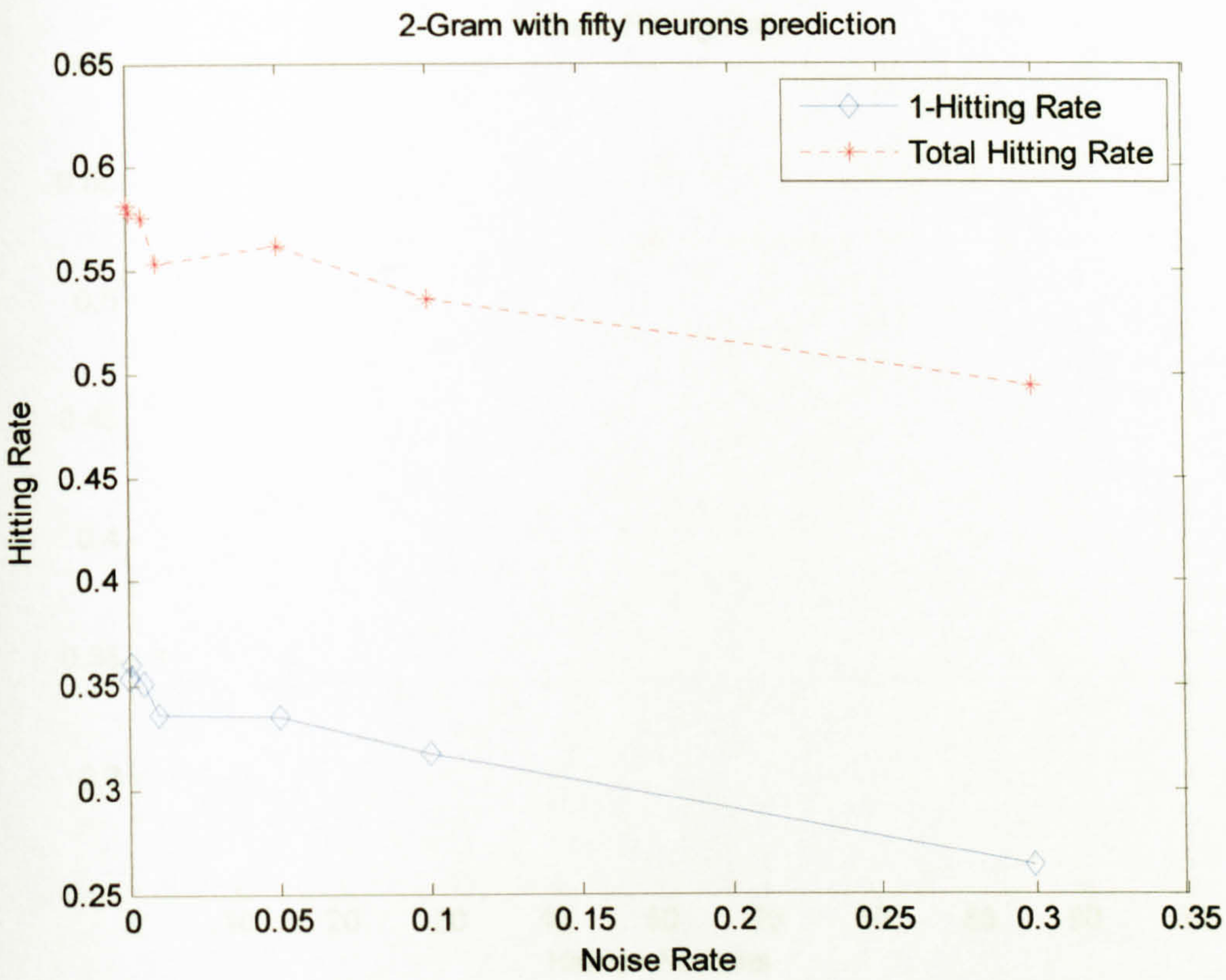

3-Gram with Noise Rate [0.1]
Hitting Rate
0.55
0.5
0.45
0.4
0.35
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons

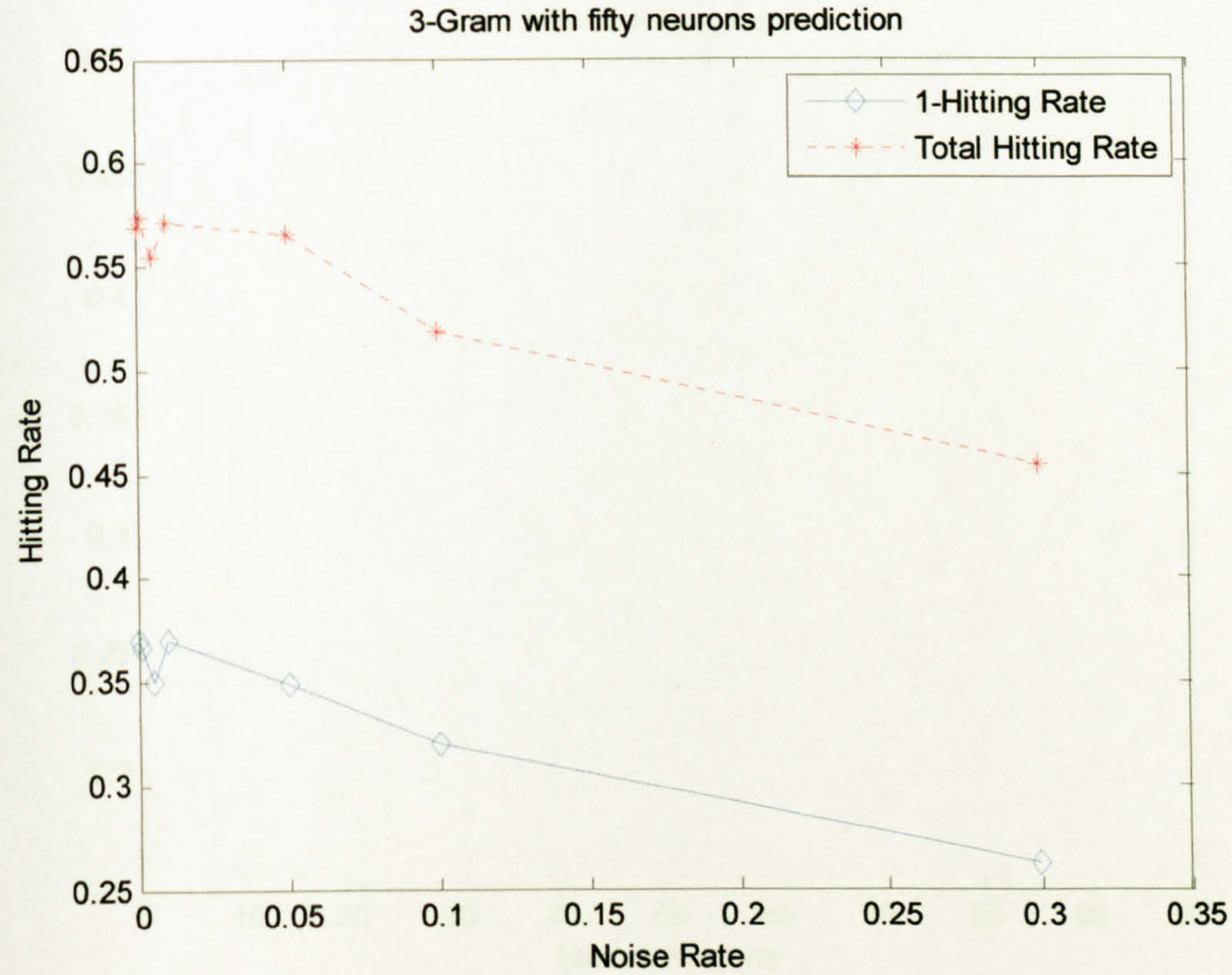

5-Gram with Noise Rate [0.1]
Hitting Rate
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons

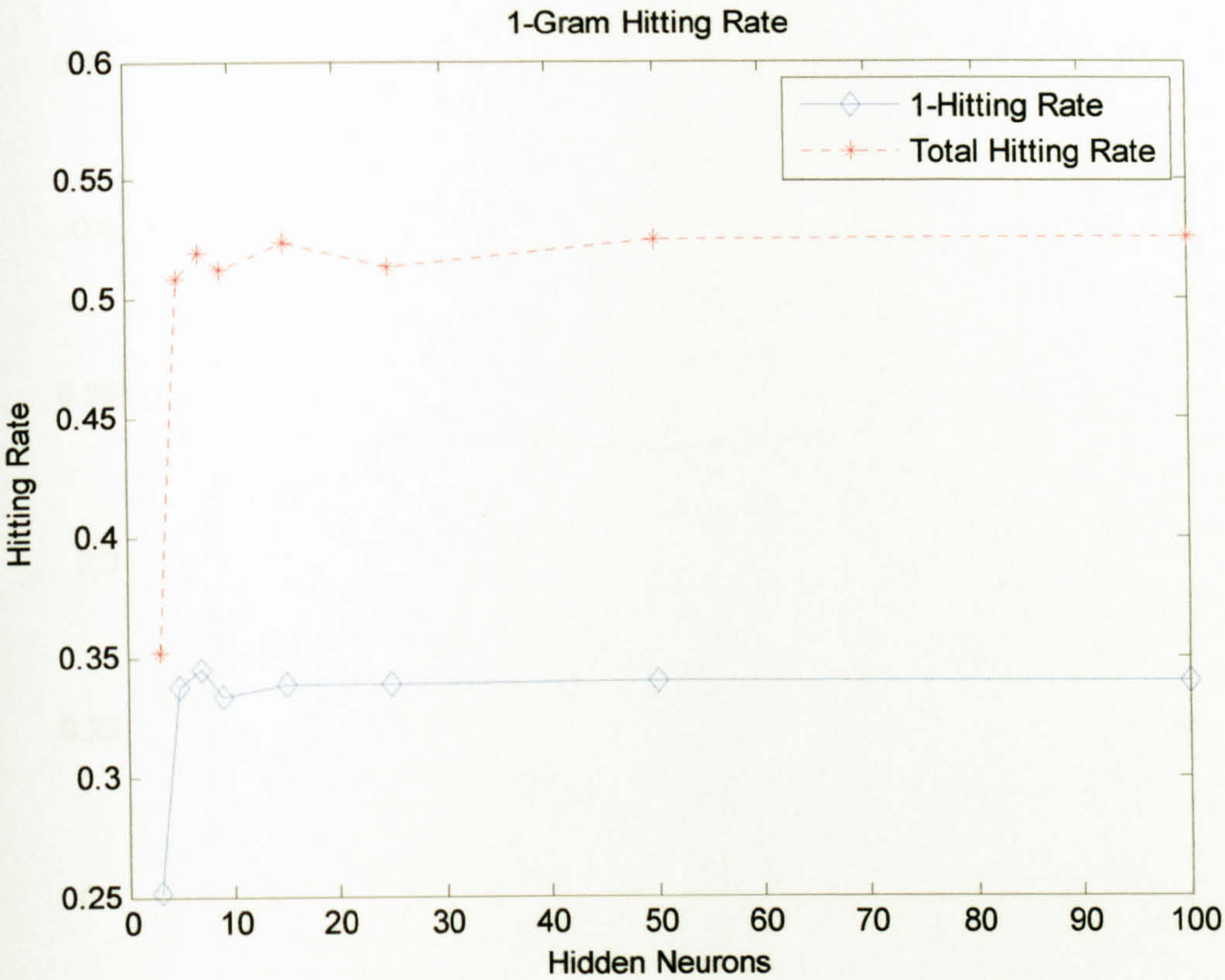

7-Gram with Noise Rate [0.1]
Hitting Rate
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons

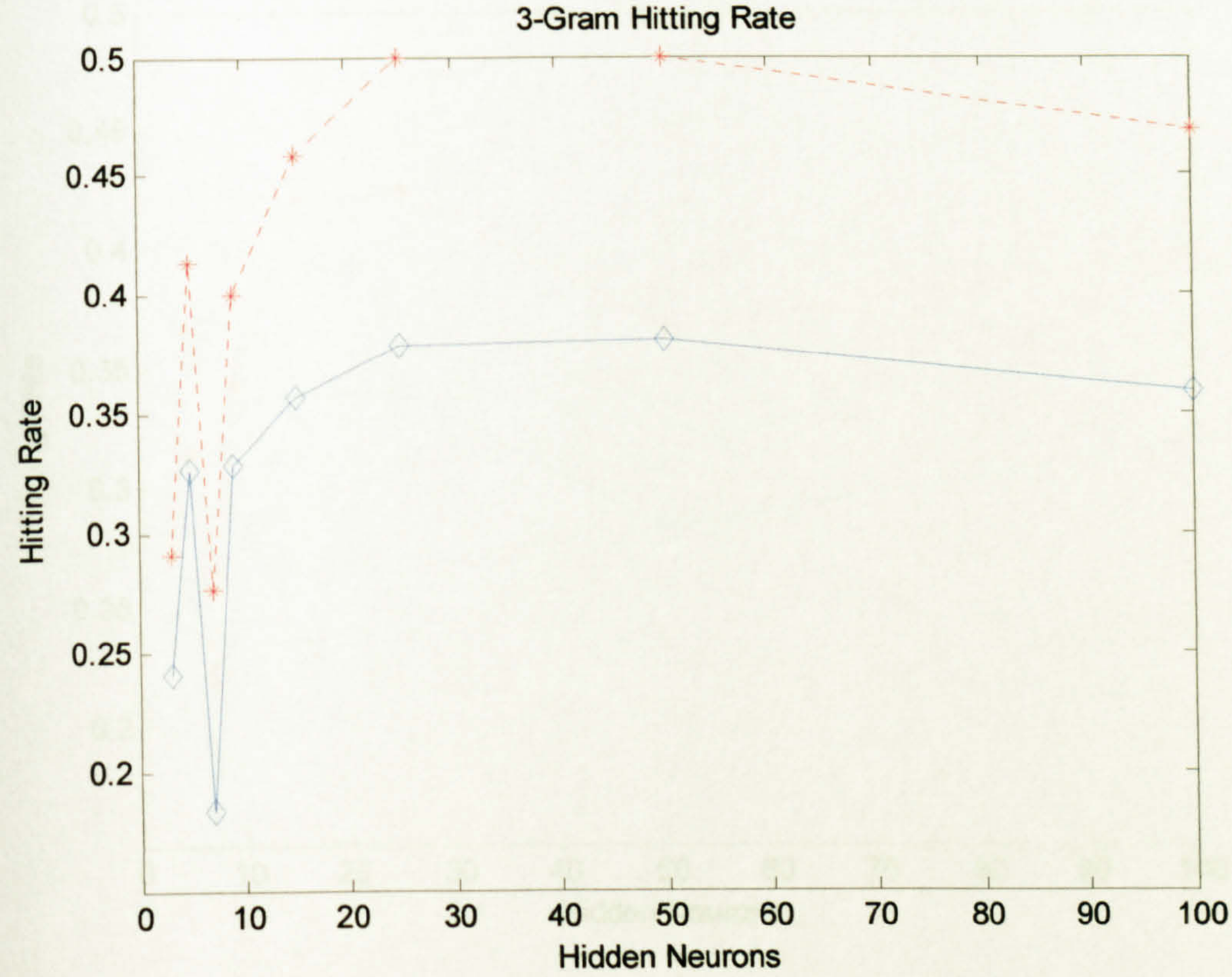

2-Gram with fifty neurons prediction
1-Hitting Rate
Total Hitting Rate
Hitting Rate
Noise Rate
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35

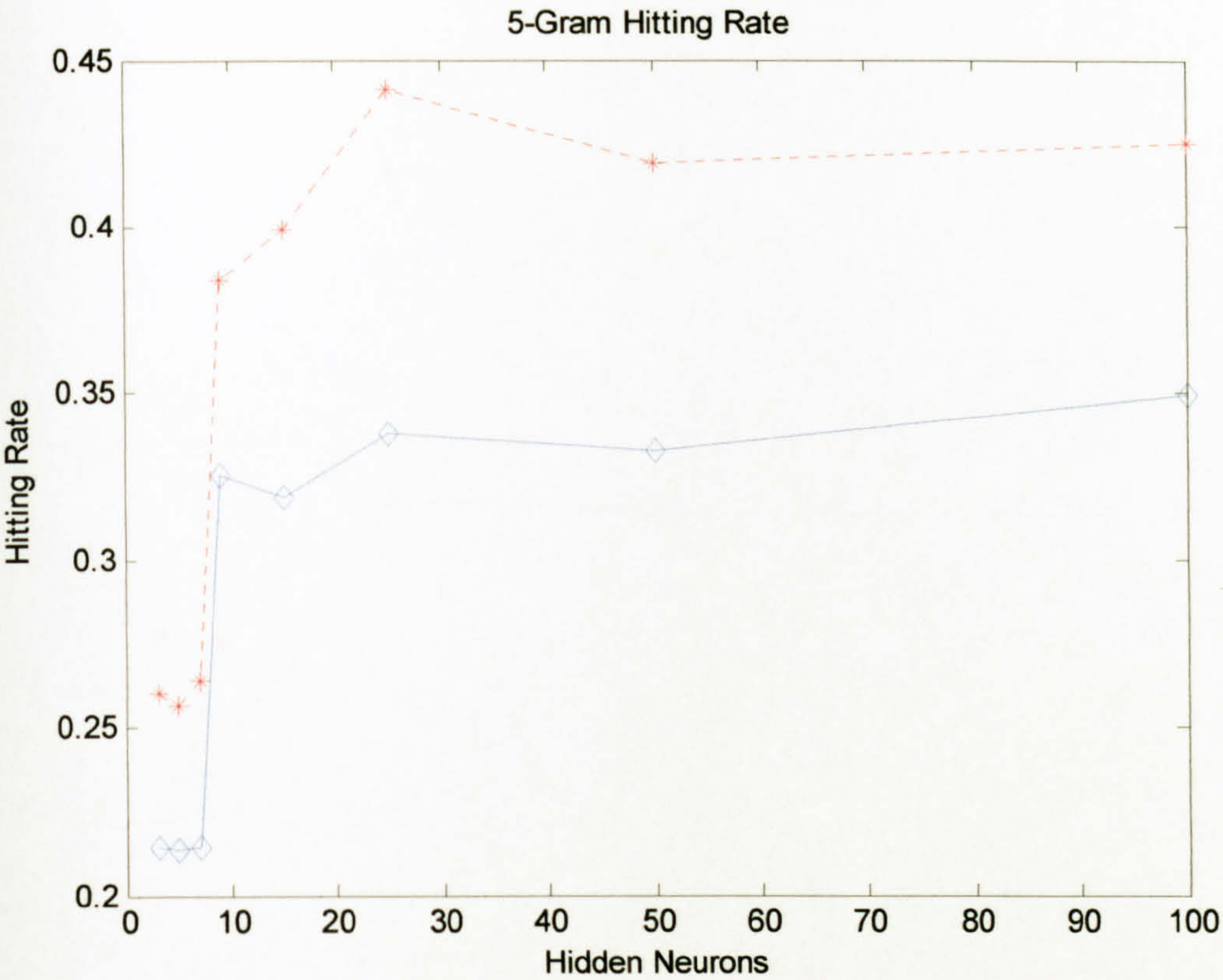

3-Gram with fifty neurons prediction
1-Hitting Rate
Total Hitting Rate
Hitting Rate
Noise Rate
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35

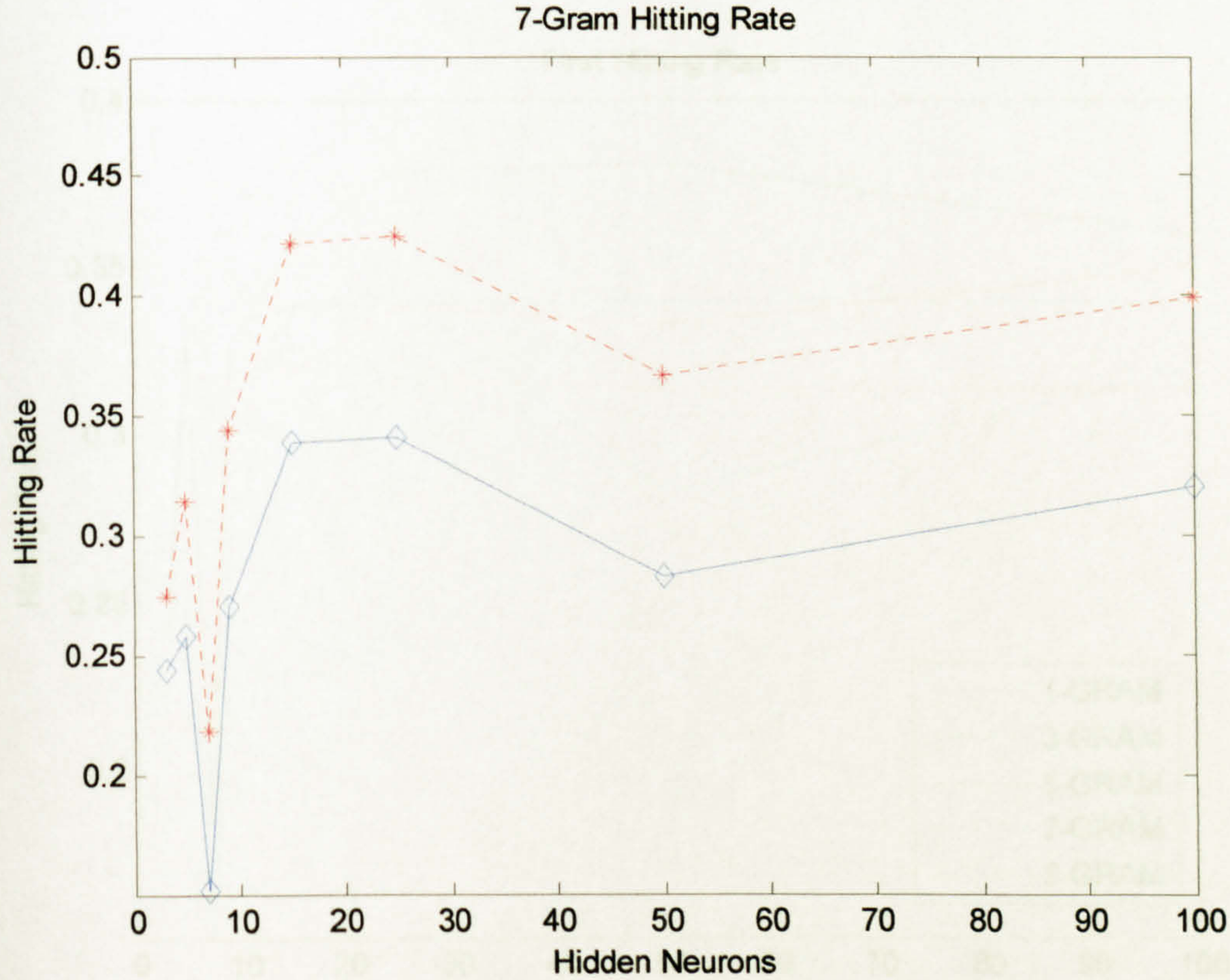

2-Gram with fifty neurons prediction
1-Hitting Rate
Total Hitting Rate
Hitting Rate
Noise Rate
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35

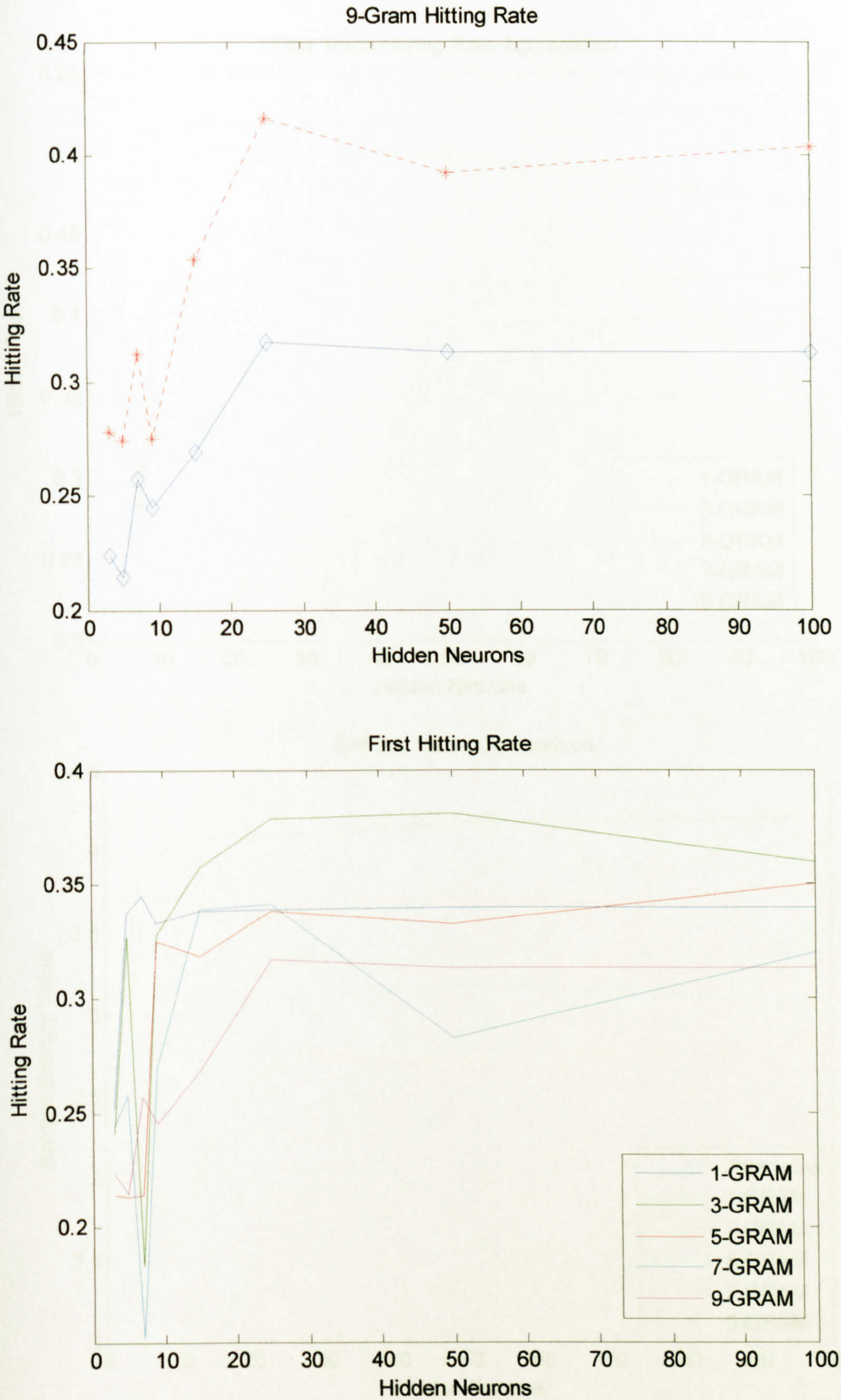

3-Gram with fifty neurons prediction
1-Hitting Rate
Total Hitting Rate
Hitting Rate
Noise Rate
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35

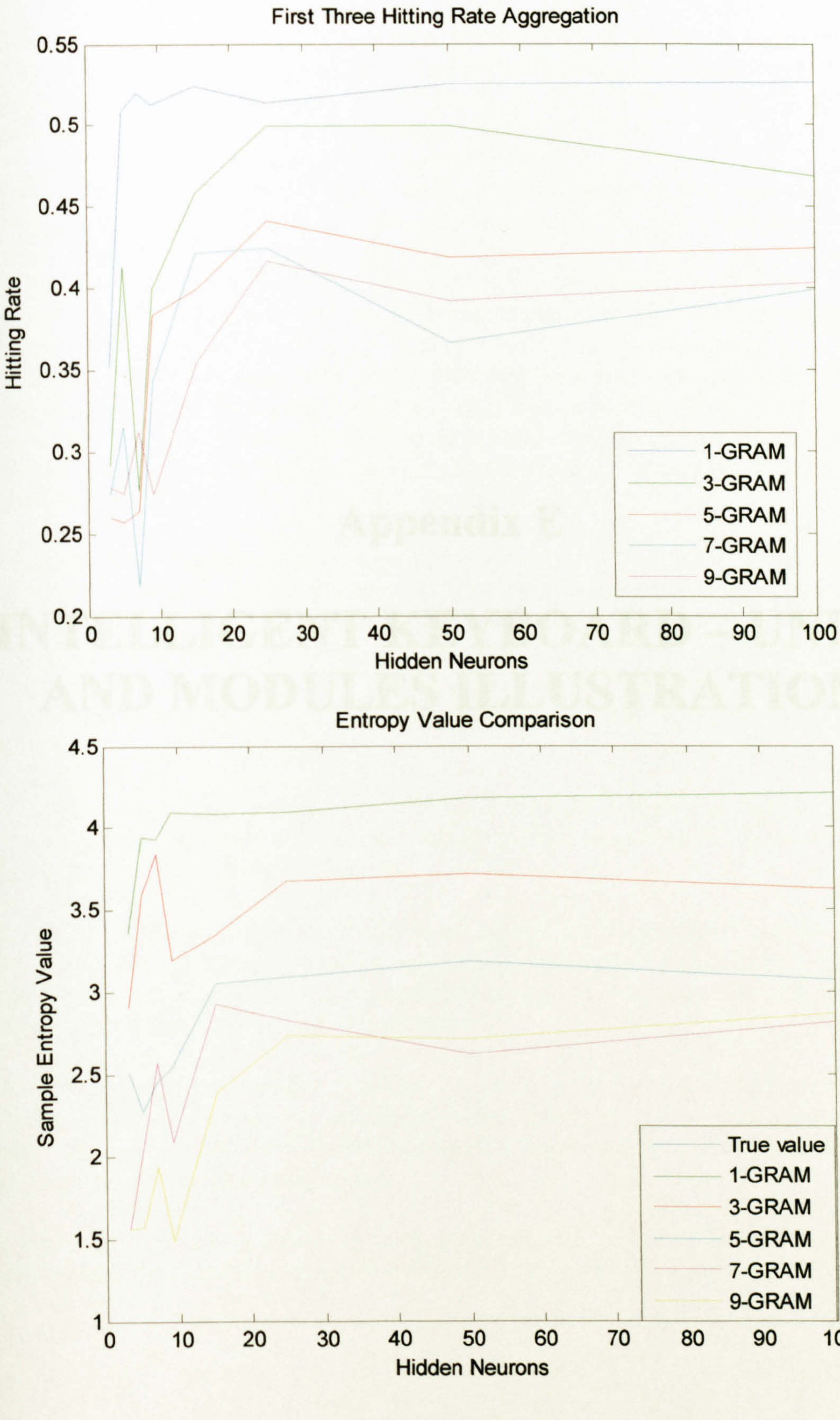

1-Gram Hitting Rate
1-Hitting Rate
Total Hitting Rate
Hitting Rate
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons


3-Gram Hitting Rate
Hitting Rate
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons


5-Gram Hitting Rate
Hitting Rate
0.45
0.4
0.35
0.3
0.25
0.2
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons


7-Gram Hitting Rate
Hitting Rate
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons


9-Gram Hitting Rate
0.45
0.4
0.35
0.3
0.25
0.2
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons
First Hitting Rate
Hitting Rate
0.4
0.35
0.3
0.25
0.2
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons
1-GRAM
3-GRAM
5-GRAM
7-GRAM
9-GRAM


First Three Hitting Rate Aggregation
Hitting Rate
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons
1-GRAM
3-GRAM
5-GRAM
7-GRAM
9-GRAM
Entropy Value Comparison
Sample Entropy Value
4.5
4
3.5
3
2.5
2
1.5
1
0
10
20
30
40
50
60
70
80
90
100
Hidden Neurons
True value
1-GRAM
3-GRAM
5-GRAM
7-GRAM
9-GRAM

Appendix E

INTELLIGENT KEYBOARD – UNITS AND MODULES ILLUSTRATION

***No. 1-5*, Blue boxes and their connections:** represent the system's input and output process. A sentence's input should be a process passing through different structure status from letter, word to sentence, during which distinct UNITS will be evoked up corresponding to the structure status' change.

The processing units from left to right, which has been marked as light yellow, are named as Text prediction unit (No. 6), Inference engine unit (No. 7), Error correction unit (No. 8), and Natural Language Processing (NLP) unit (No. 9). There are two additional modules: Noise process module (No. 10) and User interface module (No. 11), which are responsible for the interaction with out environment, here are keyboard and users.

***No. 6*, Text Prediction unit:** help user reduce the number of keystrokes necessary for a typing words. It monitors the input letter-by-letter, and produces a list of words beginning with the letter sequence recorded. Each time a letter is added, the list is updated. When the target word appears in the list, it can be chosen and inserted into the ongoing text with a single keystroke.

No.7, Inference Engine unit: as a central unit of the system, it is responsible for analyzing user behavior of input, summing up related association rules and maintaining user profile. These will run in real-time by applying Neural Network model and statistics; database can be further analyzed in background by using other technologies such as data mining, which is both time and resource consuming. Inference Engine unit also communicate with other units to give user information, system configuration information etc.

***No. 8*, Error correction unit:** deal with the known behaviors of users, such as spell error, miss-stroke error, phoneme mistakes with dyslexic. The more behavior is generalized, the more modules could be added to this unit.

***No. 9*, NLP (Natural Language Processing) unit:** This is a reinforce processing unit to those errors that can't be analyzed by other units, but could be studied in term of input context based on the syntax and semantics analysis. At this stage the unit will not be considered so it has been marked in dash line.

***No.10*, Noise process module:** provides pre-process for just input data stream, which includes analyzing and wiping off illegal letters, composing keys and sending data to Text prediction unit and Error correction unit for further process. It also works with user interface unit waiting for typing signal.

***No.11*, User interface module:** In order to provide a user-friendly interface, this unit would require frequent communication with learning unit. This unit's theoretical basis is HCI.

***No.12*, Data storage:** the memory of IK is divided into short-term memory and long-term memory. The long-term memory has included knowledge base which is represented as a set of rules, and facts which are used to match the rules. The rules inferred from Inference Engine and some other facts such as user profile and frequently used text dictionary are saved in long-term memory database. Other facts such as recently used new words are stored in short-term memory which will be turned to long-term memory if a certain threshold is reached. All other units are able to communicate with long-term and short-term memory directly.

Appendix F

VIRTUAL KEY CODES

Symbolic constant name	Value (hex)	Mouse or keyboard equivalents
VK_LBUTTON	01	Left mouse button
VK_RBUTTON	02	Right mouse button
VK_CANCEL	03	Control-break processing
VK_MBUTTON	04	Middle mouse button (three-button mouse)
VK_XBUTTON1	05	**Windows 2000/XP/2003/Vista/2008:** X1 mouse button
VK_XBUTTON2	06	**Windows 2000/XP/2003/Vista/2008:** X2 mouse button
-	07	Undefined
VK_BACK	08	BACKSPACE key
VK_TAB	09	TAB key
-	0A-0B	Reserved
VK_CLEAR	0C	CLEAR key
VK_RETURN	0D	ENTER key
-	0E-0F	Undefined
VK_SHIFT	10	SHIFT key
VK_CONTROL	11	CTRL key
VK_MENU	12	ALT key
VK_PAUSE	13	PAUSE key
VK_CAPITAL	14	CAPS LOCK key
VK_KANA	15	Input Method Editor (IME) Kana mode
VK_HANGUEL	15	IME Hanguel mode (maintained for compatibility; use **VK_HANGUL**)
VK_HANGUL	15	IME Hangul mode
-	16	Undefined
VK_JUNJA	17	IME Junja mode
VK_FINAL	18	IME final mode
VK_HANJA	19	IME Hanja mode
VK_KANJI	19	IME Kanji mode
-	1A	Undefined
VK_ESCAPE	1B	ESC key
VK_CONVERT	1C	IME convert (Reserved for Kanji systems)
VK_NONCONVERT	1D	IME nonconvert (Reserved for Kanji systems)
VK_ACCEPT	1E	IME accept (Reserved for Kanji systems)
VK_MODECHANGE	1F	IME mode change request (Reserved for Kanji systems)
VK_SPACE	20	SPACEBAR
VK_PRIOR	21	PAGE UP key
VK_NEXT	22	PAGE DOWN key
VK_END	23	END key
VK_HOME	24	HOME key
VK_LEFT	25	LEFT ARROW key
VK_UP	26	UP ARROW key
VK_RIGHT	27	RIGHT ARROW key
VK_DOWN	28	DOWN ARROW key
VK_SELECT	29	SELECT key
VK_PRINT	2A	PRINT key
VK_EXECUTE	2B	EXECUTE key

VK_SNAPSHOT	2C	PRINT SCREEN key for Windows 3.0 and later
VK_INSERT	2D	INS key
VK_DELETE	2E	DEL key
VK_HELP	2F	HELP key
VK_0	30	0 key
VK_1	31	1 key
VK_2	32	2 key
VK_3	33	3 key
VK_4	34	4 key
VK_5	35	5 key
VK_6	36	6 key
VK_7	37	7 key
VK_8	38	8 key
VK_9	39	9 key
-	3A-40	Undefined
VK_A	41	A key
VK_B	42	B key
VK_C	43	C key
VK_D	44	D key
VK_E	45	E key
VK_F	46	F key
VK_G	47	G key
VK_H	48	H key
VK_I	49	I key
VK_J	4A	J key
VK_K	4B	K key
VK_L	4C	L key
VK_M	4D	M key
VK_N	4E	N key
VK_O	4F	O key
VK_P	50	P key
VK_Q	51	Q key
VK_R	52	R key
VK_S	53	S key
VK_T	54	T key
VK_U	55	U key
VK_V	56	V key
VK_W	57	W key
VK_X	58	X key
VK_Y	59	Y key
VK_Z	5A	Z key
VK_LWIN	5B	Left Windows key (Microsoft Natural Keyboard)
VK_RWIN	5C	Right Windows key (Microsoft Natural Keyboard)
VK_APPS	5D	Applications key (Microsoft Natural Keyboard)
-	5E	Reserved
VK_SLEEP	5F	Computer Sleep key
VK_NUMPAD0	60	Numeric keypad 0 key

VK_NUMPAD1	61	Numeric keypad 1 key
VK_NUMPAD2	62	Numeric keypad 2 key
VK_NUMPAD3	63	Numeric keypad 3 key
VK_NUMPAD4	64	Numeric keypad 4 key
VK_NUMPAD5	65	Numeric keypad 5 key
VK_NUMPAD6	66	Numeric keypad 6 key
VK_NUMPAD7	67	Numeric keypad 7 key
VK_NUMPAD8	68	Numeric keypad 8 key
VK_NUMPAD9	69	Numeric keypad 9 key
VK_MULTIPLY	6A	Multiply key
VK_ADD	6B	Add key
VK_SEPARATOR	6C	Separator key
VK_SUBTRACT	6D	Subtract key
VK_DECIMAL	6E	Decimal key
VK_DIVIDE	6F	Divide key
VK_F1	70	F1 key
VK_F2	71	F2 key
VK_F3	72	F3 key
VK_F4	73	F4 key
VK_F5	74	F5 key
VK_F6	75	F6 key
VK_F7	76	F7 key
VK_F8	77	F8 key
VK_F9	78	F9 key
VK_F10	79	F10 key
VK_F11	7A	F11 key
VK_F12	7B	F12 key
VK_F13	7C	F13 key
VK_F14	7D	F14 key
VK_F15	7E	F15 key
VK_F16	7F	F16 key
VK_F17	80H	F17 key
VK_F18	81H	F18 key
VK_F19	82H	F19 key
VK_F20	83H	F20 key
VK_F21	84H	F21 key
VK_F22	85H	F22 key
VK_F23	86H	F23 key
VK_F24	87H	F24 key
-	88-8F	Unassigned
VK_NUMLOCK	90	NUM LOCK key
VK_SCROLL	91	SCROLL LOCK key
VK_OEM_NEC_EQUAL	92	NEC PC-9800 kbd definitions: '=' key on numpad
VK_OEM_FJ_JISHO	92	Fujitsu/OASYS kbd definitions: 'Dictionary' key
VK_OEM_FJ_MASSHOU	93	Fujitsu/OASYS kbd definitions: 'Unregister word' key
VK_OEM_FJ_TOUROKU	94	Fujitsu/OASYS kbd definitions: 'Register word' key
VK_OEM_FJ_LOYA	95	Fujitsu/OASYS kbd definitions: 'Left OYAYUBI' key

VK_OEM_FJ_ROYA	96	Fujitsu/OASYS kbd definitions: 'Right OYAYUBI' key
-	97-9F	Unassigned
VK_LSHIFT	A0	Left SHIFT key
VK_RSHIFT	A1	Right SHIFT key
VK_LCONTROL	A2	Left CONTROL key
VK_RCONTROL	A3	Right CONTROL key
VK_LMENU	A4	Left MENU key
VK_RMENU	A5	Right MENU key
VK_BROWSER_BACK	A6	**Windows 2000/XP/2003/Vista/2008:** Browser Back key
VK_BROWSER_FORWARD	A7	**Windows 2000/XP/2003/Vista/2008:** Browser Forward key
VK_BROWSER_REFRESH	A8	**Windows 2000/XP/2003/Vista/2008:** Browser Refresh key
VK_BROWSER_STOP	A9	**Windows 2000/XP/2003/Vista/2008:** Browser Stop key
VK_BROWSER_SEARCH	AA	**Windows 2000/XP/2003/Vista/2008:** Browser Search key
VK_BROWSER_FAVORITES	AB	**Windows 2000/XP/2003/Vista/2008:** Browser Favorites key
VK_BROWSER_HOME	AC	**Windows 2000/XP/2003/Vista/2008:** Browser Start and Home key
VK_VOLUME_MUTE	AD	**Windows 2000/XP/2003/Vista/2008:** Volume Mute key
VK_VOLUME_DOWN	AE	**Windows 2000/XP/2003/Vista/2008:** Volume Down key
VK_VOLUME_UP	AF	**Windows 2000/XP/2003/Vista/2008:** Volume Up key
VK_MEDIA_NEXT_TRACK	B0	**Windows 2000/XP/2003/Vista/2008:** Next Track key
VK_MEDIA_PREV_TRACK	B1	**Windows 2000/XP/2003/Vista/2008:** Previous Track key
VK_MEDIA_STOP	B2	**Windows 2000/XP/2003/Vista/2008:** Stop Media key
VK_MEDIA_PLAY_PAUSE	B3	**Windows 2000/XP/2003/Vista/2008:** Play/Pause Media key
VK_LAUNCH_MAIL	B4	**Windows 2000/XP/2003/Vista/2008:** Start Mail key
VK_LAUNCH_MEDIA_SELECT	B5	**Windows 2000/XP/2003/Vista/2008:** Select Media key
VK_LAUNCH_APP1	B6	**Windows 2000/XP/2003/Vista/2008:** Start Application 1 key
VK_LAUNCH_APP2	B7	**Windows 2000/XP/2003/Vista/2008:** Start Application 2 key
-	B8-B9	Reserved
VK_OEM_1	BA	**Windows 2000/XP/2003/Vista/2008:** For the US standard keyboard, the ';:' key
VK_OEM_PLUS	BB	**Windows 2000/XP/2003/Vista/2008:** For any country/region, the '+' key
VK_OEM_COMMA	BC	**Windows 2000/XP/2003/Vista/2008:** For any country/region, the ',' key
VK_OEM_MINUS	BD	**Windows 2000/XP/2003/Vista/2008:** For any country/region, the '-' key
VK_OEM_PERIOD	BE	**Windows 2000/XP/2003/Vista/2008:** For any country/region, the '.' key
VK_OEM_2	BF	**Windows 2000/XP/2003/Vista/2008:** For the US standard keyboard, the '/?' key
VK_OEM_3	C0	**Windows 2000/XP/2003/Vista/2008:** For the US standard keyboard, the '` ~' key
-	C1-D7	Reserved
-	D8-DA	Unassigned
VK_OEM_4	DB	**Windows 2000/XP/2003/Vista/2008:** For the US standard keyboard, the '[{' key
VK_OEM_5	DC	**Windows 2000/XP/2003/Vista/2008:** For the US standard keyboard, the '|' key
VK_OEM_6	DD	**Windows 2000/XP/2003/Vista/2008:** For the US standard keyboard, the ']}' key
VK_OEM_7	DE	**Windows 2000/XP/2003/Vista/2008:** For the US standard keyboard, the 'single-quote/double-quote' key

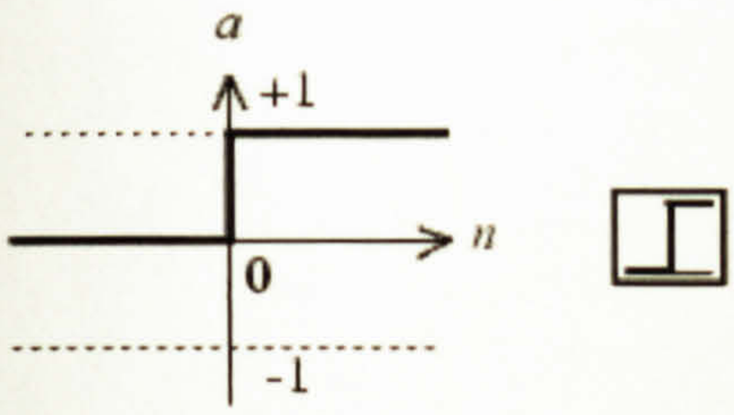
VK_OEM_8	DF	Used for miscellaneous characters; it can vary by keyboard.
-	E0	Reserved
	E1	OEM specific
VK_OEM_102	E2	**Windows 2000/XP/2003/Vista/2008:** Either the angle bracket key or the backslash key on the RT 102-key keyboard
-	E3-E4	OEM specific
VK_PROCESSKEY	E5	**Windows 95/98/Me, Windows NT/2000/XP/2003/Vista/2008:** IME PROCESS key
-	E6	OEM specific
VK_PACKET	E7	**Windows 2000/XP/2003/Vista/2008:** Used to pass Unicode characters as if they were keystrokes. The VK_PACKET key is the low word of a 32-bit Virtual Key value used for non-keyboard input methods. For more information, see Remark in KEYBDINPUT , SendInput , WM_KEYDOWN , and WM_KEYUP
-	E8	Unassigned
VK_OEM_RESET	E9	Only used by Nokia.
VK_OEM_JUMP	EA	Only used by Nokia.
VK_OEM_PA1	EB	Only used by Nokia.
VK_OEM_PA2	EC	Only used by Nokia.
VK_OEM_PA3	ED	Only used by Nokia.
VK_OEM_WSCTRL	EE	Only used by Nokia.
VK_OEM_CUSEL	EF	Only used by Nokia.
VK_OEM_ATTN	F0	Only used by Nokia.
VK_OEM_FINNISH	F1	Only used by Nokia.
VK_OEM_COPY	F2	Only used by Nokia.
VK_OEM_AUTO	F3	Only used by Nokia.
VK_OEM_ENLW	F4	Only used by Nokia.
VK_OEM_BACKTAB	F5	Only used by Nokia.
VK_ATTN	F6	Attn key
VK_CRSEL	F7	CrSel key
VK_EXSEL	F8	ExSel key
VK_EREOF	F9	Erase EOF key
VK_PLAY	FA	Play key
VK_ZOOM	FB	Zoom key
VK_NONAME	FC	Reserved for future use.
VK_PA1	FD	PA1 key
VK_OEM_CLEAR	FE	Clear key
	FF	Multimedia keys. See ScanCode keys.

* Colored in orange (in all 53 VKCs) are specially considered by some neural network models in the research.

Appendix G

USUAL NEUAL NETWORK ACTIVATION FUNCTION SIN MATLAB

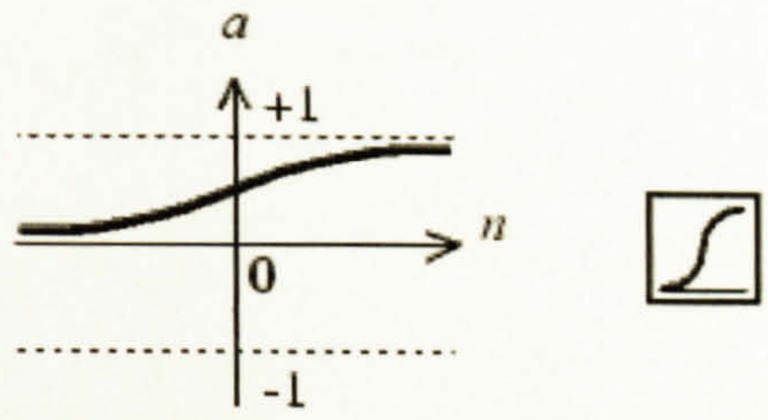
hardlim



$a = hardlim(n)$

Hard-Limit Transfer Function

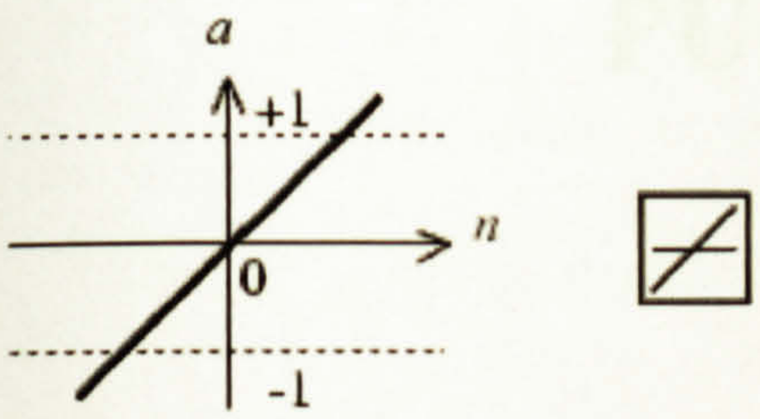
logsig



$a = logsig(n)$

Log-Sigmoid Transfer Function

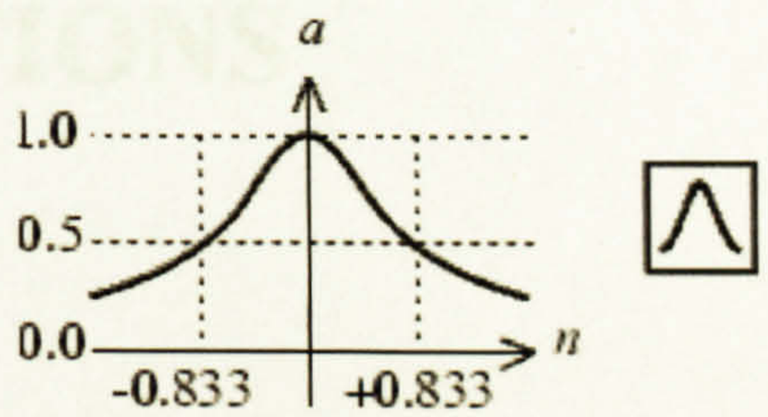
pureline



$a = purelin(n)$

Linear Transfer Function

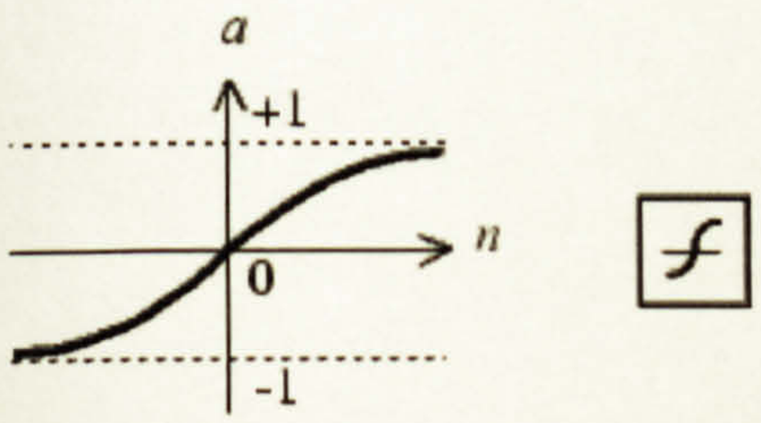
radbas



$a = radbas(n)$

Radial Basis Function

tansig



$a = tansig(n)$

Tan-Sigmoid Transfer Function

Appendix H

PUBLICATIONS

Evolutionary Ranking on Multiple Word Correction Algorithms Using Neural Network Approach

Jun Li, Karim Ouazzane, Yanguo Jing,
Hassan Kazemian,Richard Boyd

Faculty of Computing, London Metropolitan University,
166-220 Holloway Road, London. N7 8DB, UK.
{Jul029, k.ouazzane, y.jing, h.kazemian}@londonmet ac.uk
richard boyd@disabilityessex.org

Abstract. Multiple algorithms have been developed to correct user's typing mistakes. However, an optimum solution is hardly identified among them. Moreover, these solutions rarely produce a single answer or share common results, and the answers may change with time and context. These have led this research to combine some distinct word correction algorithms to produce an optimal prediction based on database updates and neural network learning. In this paper, three distinct typing correction algorithms are integrated as a pilot research. Key factors including Time Change, Context Change and User Feedback are considered. Experimental results show that 57.50% Ranking First Hitting Rate (HR) with the samples of category one and a best Ranking First Hitting Rate of 74.69% within category four are achieved.

Keywords: Neural Network, Metaphone, Levenshtein distance, word *2*-gram, Jaro distance, Jaro-Winkler distance, ranking First Hitting Rate.

1 Introduction

Computer users inevitably make typing mistakes. These may be seen as spelling errors, prolong key press and adjacent key press errors etc [1]. Multiple solutions such as Metaphone [2] and *n*-grams [3] have been developed to correct user's typing mistakes, and each of them may have its unique features. However, an optimum solution is hardly identified among them. Therefore, it is desired to develop a hybrid solution based on combining these technologies, which can put all merits of those distinct solutions together.

Moreover, each function may rarely generate a single answer, let alone multiple functions which may produce a larger list of suggestions. This requires developing an evolutionary and adjustable approach to prioritize the suggestions in this list. Also, the answers may change within different context; and the solutions are also required to evolve based on user's feedbacks. Therefore, this research is motivated by the requirement of combining distinct word correction algorithms and subsequently producing an optimal prediction based on dataset updates and neural network learning process.

2 Typing Correction Functions

There are many types of errors caused by users, for example, spelling errors, hitting adjacent key and cognitive difficulties. Some efforts have been made based on different technologies such as spell checking, natural language processing and control signals filter. In this paper, a pilot research is carried out and three distinct algorithms (referred to as L.M.T), namely, Levenshtein word distance algorithm [4], Metaphone algorithm, and *2*-gram word algorithm are used.

Metaphone is a phonetic algorithm indexing words by their sound, which can be adjusted to correct typing errors. These are two examples,

```
able -> APL
hello-> HL
```

The right side of the arrow is words' phonetic keys. Let's assume that a user intends to type a word 'hello' but mistakenly typed 'hallo' instead, whose phonetic keys are identical. Subsequently, the system is able to index and retrieve possible words from the database based on the phonetic key and present them to a user for selection.

Levenshtein distance is another function that needs to be explored. It is designed based on the calculation of minimum number of operations required to transform one string into another, where an operation is an insertion, deletion, or substitution of a single character, for instance,

```
hello <-> hallo //the string  distance is one
hello <-> all   //the string  distance is three
```

After a comparison with each string stored in the memory, the pair with the least distance can be considered as having the highest similarity, and then the one or the group with the least distance can be presented through the user interface module.

Word *2*-gram (i.e. word digram or word bigram) is groups of two consecutive words, and is very commonly used as the basis for simple statistical analysis of text. For instance, given a sentence, 'I am a student', some word *2*-gram samples are,

```
I am
am a
```

For example, under an ideal condition, 'am' can be predicted if its predecessor 'I' is typed. Then the predicted word 'am' can be used to make sure that user types it correctly.

Another similar method to Levenshtein distance is Jaro metric [5]. The Jaro distance metric states that, given two strings s_1 and s_2, two characters a_i, b_j from s_1 and s_2 respectively are considered matching only if,

$$i - \frac{\min(|s_1|,|s_2|)}{2} \le j \le i + \frac{\min(|s_1|,|s_2|)}{2} \quad . \qquad (1)$$

then their distance d is calculated as,

$$d = \frac{1}{3}\left(\frac{|s_1^{'}|}{|s_1|} + \frac{|s_2^{'}|}{|s_2|} + \frac{|s_1^{'}| - t}{|s_1^{'}|}\right) \quad . \tag{2}$$

Where $|s_1^{'}|, |s_2^{'}|$ are the numbers of s_1 matching s_2 and s_2 matching s_1 characters respectively, and t is the number of transpositions.

A variant of Jaro metric uses a prefix scale p, which is the longest common prefix of string s_1 and s_2. Let's define Jaro distance as d, then Jaro-Winkler [6] distance can be defined as,

$$Jaro - Winkler(d + \frac{\max(p, 4)}{10} * (1 - d)) \quad . \tag{3}$$

The result of the Jaro-Winkler distance metric is normalized into the range of [*0, 1*]. It is designed and best suited for short strings.

3 Word List Neural Network Ranking and Definitions

As the above solutions rarely produce a single answer or share common results, this implies that a combination will definitely be a more accurate solution. However, it requires a word-list with words priority rather than a single word to be generated. For instance, a user intends to type a word 'hello' but mistakenly typed 'hallo' instead. Let's assume that two functions, namely, Metaphone method and Levenshtein distance are integrated together and the correction results are produced as follows,

```
Metaphone:                'hello', 'hall'
Levenshtein distance: 'hello', 'all', 'allow'
```

Then a words list with 'hello', 'hall', 'all' and 'allow' is made available to the user. It is evident that a ranking algorithm computing each individual's priority is necessary before a word list is presented to a user.

In a real-time interaction, it requires that the word-list priority computation is able to adapt itself timely based on the user behavior and some other factors. In practice, this can be simplified by considering the word-list priority computation as a function of three variables: Time Change, Context Change and User Feedback. Therefore, a ranking algorithm, which is able to learn from user's selection and context changing over time, and subsequently adjust its weights, can be developed. In this research, these three variables are further quantified and represented by frequency increase, word *2*-gram statistic and a supervised learning algorithm respectively, and subsequently a novel Word List Ranking neural network model associated with the variables is developed. The definitions introduced below are useful as they are part of the rules which dictate the whole process.

First Rank Conversion Values and First Hitting Rate definition: In a neural network post processing, if its output follows a 'winner takes all' strategy, that is, the maximum value in output is converted into one and the rest values are converted into zeros, then the converted elements are named as First Rank Conversion Values. Given testing metrics P, target metrics T and testing result metrics R where their numbers of lines and columns are equal and expressed as n, m respectively, then the Hitting Rate is $HR = \{hr_i \mid hr_i = zeros(T - R_i)/n, i \in m\}$, where R_i is the i^{th} Rank Conversion Values of R, *zeros()* is the function to compute the number of zero vector included in metrics, and the First Hitting Rate is hr_1.

Word-List n-formula Prediction definition: Let's assume that one has distinct algorithms set $A = \{a_1 ... a_i ... a_n\}$, where $1 \le i \le n$ and i, n are positive integers. To process a sequence s, if there exists a one-to-many mapping $\{s \to O_i\}$ associated with algorithm a_i between input and output, where $O_i = \{o_i^j \mid 1 \le j \le m_i\}$, o_i^j is a generated sequence from the algorithm, and j, m_i are positive integers, then one has $\sum_{i=1}^{n} m_i$ sequence generated and the sequence set is defined as Word-List. The process based on the use of n algorithms to generate a word-list is called n-formula Prediction.

Word-List Success Prediction Rate Definition: Given a word list generated by several algorithms to correct a wrong typing, if the intended word is in the word list, then it is a Success Prediction. If there is a set of wrong typing, the proportion between the number of Success Prediction and wrong typing is called Word-List Success Prediction Rate (*SP* Rate). Let's define the number of Success Prediction as o_1 and the number of wrong typing as o_2, then one has $o_1 \le o_2$ and *SP Rate* = o_1/o_2.

Simulation Rate Definition: Given natural numbers i, m, n, where $i \le n$ and $m \le n$, let's simulate a testing dataset $p_1 ... p_i ... p_n$ with a trained neural network, and its target dataset $t_1 ... t_i ... t_n$, if output $r_1 ... r_i ... r_n$ has m elements which are $r_i = t_i$, then the Simulation Rate (*SM* Rate) is m/n. Given Word-List Success Prediction Rate SP and Simulation Rate SM, then the *First Hitting Rate* = *SP* * *SM*.

As illustrated above, a word correction function can combine multiple algorithms and all of them produce their self-interpreted results independently, which is the so-called Word-List n-formula Prediction. The results could be rarely similar while a

user may require only one of them if Success Prediction is fulfilled. So a functional ranking model will play a major role to present an efficient word list with priority. If one considers the learning factor required by a word list and variability of its related dataset, a neural network model is a good choice with the dataset being constantly updated.

In L.M.T combination, Levenshtein word distance algorithm calculates the similarity between each two words, where all the most similar ones are presented; Metaphone algorithm retrieves words based on phonetic index while word *2*-gram algorithm retrieves them based on last typed word index. From the definition of Word-list *n*-formula prediction, L.M.T correction can be referred to as Word-List *3*-formula Prediction. Let's use the example shown below, where the word 'shall' is wrongly typed as '*sahll*'.

```
Tomorrow sahll we go to the park?
```

and assume that a database, which includes a *1*-gram & *2*-gram table, has been initialized by a sentence,

```
Out of your shell! Tomorrow all of us shall
start a new training.
```

Then, L.M.T correction result of word '*sahll*' based on *2*-gram word algorithm is '*all*', the correction results based on Metaphone algorithm are '*shall*' and '*shell*' and the correction results based on Levenshtein word distance algorithm are '*all*' and '*shall*'.

4 Word List Neural Network Ranking Modelling

Let's suppose that, corresponding to every wrong typing, each algorithm generates a maximum of two words in a descending frequency order. Each word is represented by its two features: word frequency and word similarity values. In a real-time database, the word frequency is updated along with user typing. Both, word frequency and word similarity datasets are normalized before the neural network training and testing.

In this paper, a neural network model with *12-3-6* three layer structure is developed as shown in Figure 1, where the number of the input layer neurons is determined by the expression: *Number of Algorithms* (=3) * *Number of Words predicted* (=2) * *Number of Features of each word* (=2). The model is named as word list neural network ranking (WLR) model and BackPropagation algorithm is adopted as its learning algorithm. Each algorithm generates two predictions based on the input, which is a wrongly typed word. Each prediction is represented by its two features, namely, Jaro-Winkler distance and word frequency.

Generally speaking, WLR model is designed to predict a highest ranked word amongst every six recommendations. Then, a ranking issue is converted to a neural network classification question solving issue. At the output layer of WLR model, there is only one neuron fired once at a time. To normalize the difference between the

typed word and a predicted word, Jaro-Winkler metric is applied. It normalizes words difference also called words similarity value, into a range of [*0, 1*]. Another parameter: word frequency, is normalized by Normal Probability Density function based on frequencies' mean value and standard deviation.

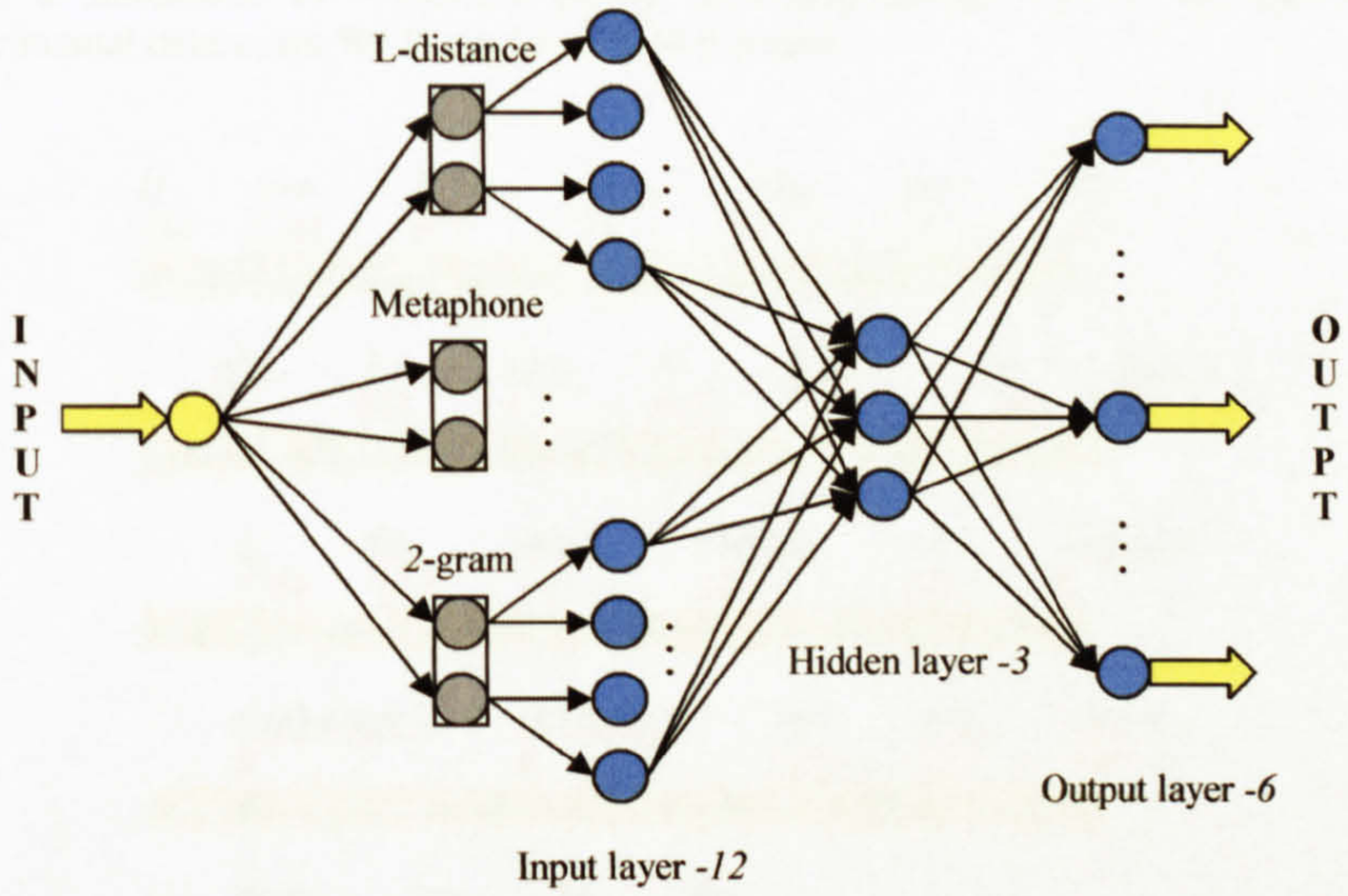


Fig. 1. The circles in blue are neurons of WLR model; the circles in grey are predicted words; the three rectangles represent the three algorithms, namely, Levenshtein distance, Metaphone and *2*-gram; the shapes in yellow show the input and the output of WLR model.

An application and its related Access database are developed to generate an experimental dataset for WLR model. The related database has been initialized by words' *1 & 2*-gram frequency statistics of a novel - 'Far from the Madding Crowd' [7][8] before the experimental dataset is generated. The database initialization has followed these rules,

- *A word is defined as a sequence of alphabets between two separators.*
- *Any symbols are considered as a separator except alphabets.*
- *ALL uppercase are converted into lowercase, e.g. 'If' → 'if', then 'If' is counted as 'if'*
- *Other special cases are not considered. For example, 'read' and 'reading' are considered as two independent words.*

Based on these rules, the word dictionary table and *2*-gram dictionary table including their words occurrences are initialized in the Access database. Moreover, for database efficiency purpose, all the *2*-gram records whose occurrences are less than two are eliminated. Overall, about *79.10%* of all *2*-gram records are eliminated. This will only produce a very limited influence on the performance of WLR model if

one considers thousands of repetitive trials in a neural network training and testing. The occurrences of the words' *1* & *2*-gram are kept updated along with user's typing progress (if there is a new *2*-gram generated, the *2*-gram and its occurrence will be inserted into the database). Therefore, these updated frequencies can well represent a user's temporal typing state captured and stored in a database.

As a simulation to dyslexic's typing, a testing sample [9] is used as the experimental dataset for WLR model as shown below,

Fig. 2. The numbers in red and black indicate words *1* and *2*-gram frequencies respectively

As shown in Figure 2, some words within sentences are wrongly typed, such as 'hvae' (should be 'have') and 'raed' (should be 'read'). The numbers which are right under each word (in red) indicate the frequency of the word after the database initialization. For example, the frequency of the word 'If' is *414* and the frequency of the word 'you' is *1501* in the database. The numbers in black indicate the 2-gram frequency between two consecutive words. For example, the frequency between the first two words 'If' and 'you' is eighty-five, shown as '|---*85*---|'

Let's assume the frequencies of the words shown above gradually increases in the database while other words are rarely typed. Consequently, the change of other words' frequencies will not have a big effect on the algorithms. Therefore, a simulation can be performed by using the testing dataset which has ignored the influence brought by other words' frequency changes. In this research, *5505* trials of test samples are inserted into the database gradually without considering other words' frequency changes.

Let's define a sampling point as a starting point of sampling in these *5505* trials, and define a sampling step as a gap between two consecutive sampling actions.

Twenty five sampling points are set up to collect the three algorithms' prediction results. Only those wrongly typed and completed words are considered at every sampling point. For example, the prediction results for words such as 'hvae' and 'raed' are collected; while the prediction results for right words such as 'if', 'you' and uncompleted words such as 'hva' of 'hvae' are ignored. At each sampling point, the whole dataset are gathered and called a sample. Then, twenty five samples are gathered. The determination of sampling points and sampling step is based on a heuristics method, which shows that the influence of initial frequency updating is essential while further updating influence is waning.

Figure 3 illustrates the sampling procedure, which are classified in four categories [*0→5, 10→50, 55→505, 1505→5505*]. As illustrated, the influence of frequency updating is waning from one category to another although the sampling steps are actually increasing. The four categories are shown in red lines of Figure 3. For instance, five samples have been collected with the frequency being changed from zero to five (i.e. the sampling step is one), and ten samples are collected when the frequency changed from *55* to *505* (i.e. the sampling step is *50*).

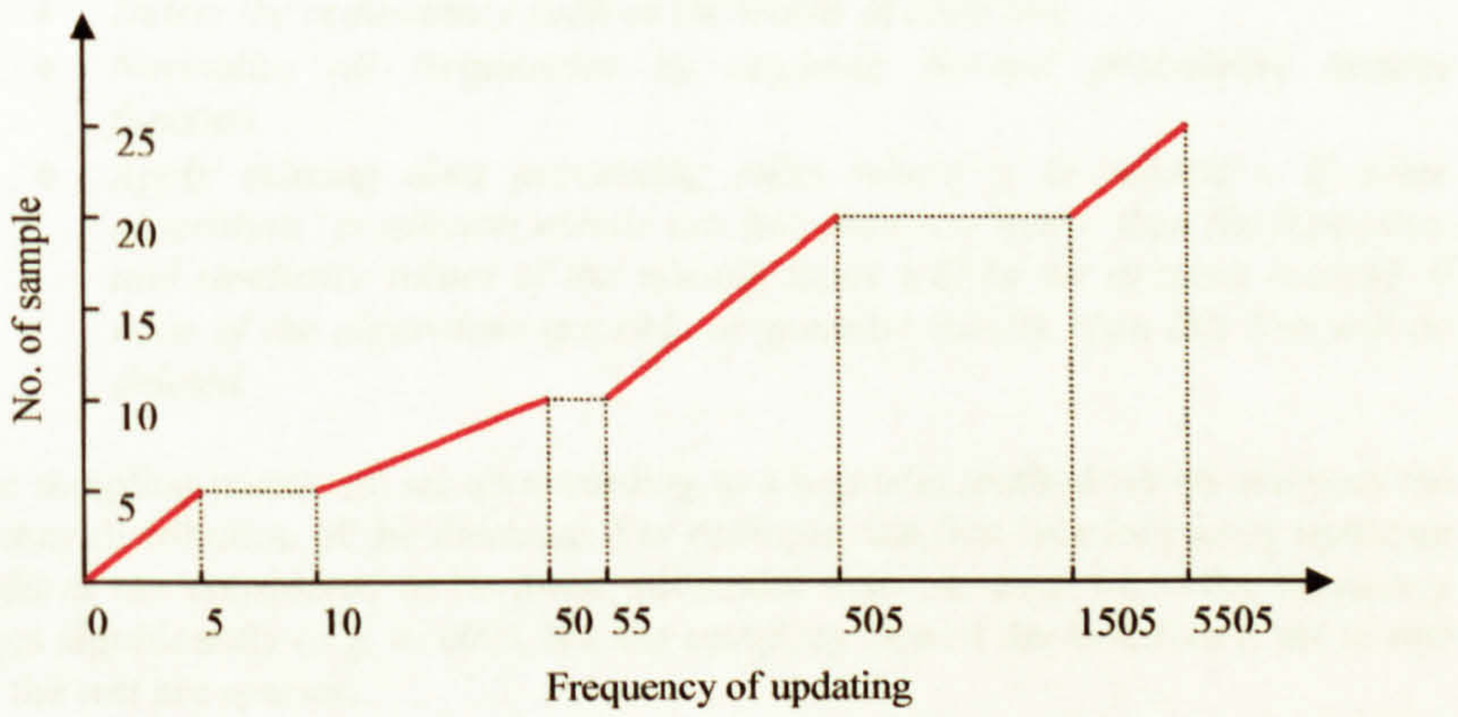


Fig. 3. X-axis refers to the frequency of the whole sample; y-axis refers to the numbers of sampling.

The first two subsets of sample one are shown in Figure 4, which lists the predicted results of two mistakenly typed words, which are 'hvae' and 'raed'.

```
#Levenshtein word distance          Metaphone                          Two-Gram word                          %output
have 679 0.925000 |hae   3 0.925000 |ve    93 0.583333 |fee1  1 0.500000 |are   86 0.527778 |know 65 0.000000 |1 0 0 0 0 0
red   43 0.925000 |read 35 0.925000 |right 65 0.483333 |road 64 0.850000 |been 112 0.500000 |a    27 0.750000 |0 1 0 0 0 0
```

Fig. 4. First line is a comment which marks the three algorithms names and 'output'. The rest are two prediction results based on the three algorithms.

As shown in the columns of Figure 4, each of the three algorithms has generated two predicted words. For instance, Levenshtein word distance algorithm gives two suggestions to the word - 'hvae', which are 'have' and 'hae'. Next to each word, the word's frequency and the similarity values to the target word are displayed. For example, the frequency of the word 'have' is *679* and its similarity to 'hvae' is *0.925*.

The last six columns of Figure 4 clearly show the required output for WLR neural network model. Each of those columns corresponds to one of the words that those three algorithms could generate. If the prediction is true, the corresponding column is set to one, otherwise it is set to zero. For example, the first line of Figure 4 is a prediction for mistakenly typed word 'hvae' while among the six predictions only the first result of Levenshtein word distance algorithm is a correct prediction, therefore the first column of the output is set to one while others are set to zeros. By default, the processing stops at the first '*1*', and the others will be set to zeros. So the output will have a maximum of one '*1*'.

The data shown in Figure 4 still can not be used by WLR model directly, as further pre-processing is required. Therefore the following procedures are applied.

- *Delete the redundancy such as the words of each line.*
- *Normalize all frequencies by applying Normal probability density function*
- *Apply missing data processing rules where it is needed – If some algorithms' prediction results are less than two items, then the frequency and similarity values of the missing items will be set to zeros instead; if none of the algorithms are able to generate results, then this line will be deleted.*

The sampling points are set up according to a heuristic method which analyzes the frequency distribution of the database. For example, the first five frequency updating procedures are considered to be more influential than the case when the frequency changes significantly (e.g. *>1000*). So, the sampling step of the first five is set to one while the rest are sparser.

In this experiment, a vector [*5, 5, 10, 5*] of samples are collected from the four categories and their sampling steps are set to [*1, 10, 50, 1000*]. For example, the first five samples are collected in a step distance of one, the third ten samples are collected in a step distance of fifty.

The dataset is further separated into training dataset [*4, 4, 7, 3*], and testing dataset [*1, 1, 3, 2*]. The post-processing of WLR model follows a 'winner takes all' rule – the neuron which has the biggest value among the six outputs are set to one while others are set to zeros.

After the training process, the Hitting Rates of the testing dataset associated with each category are shown in Figure 5.

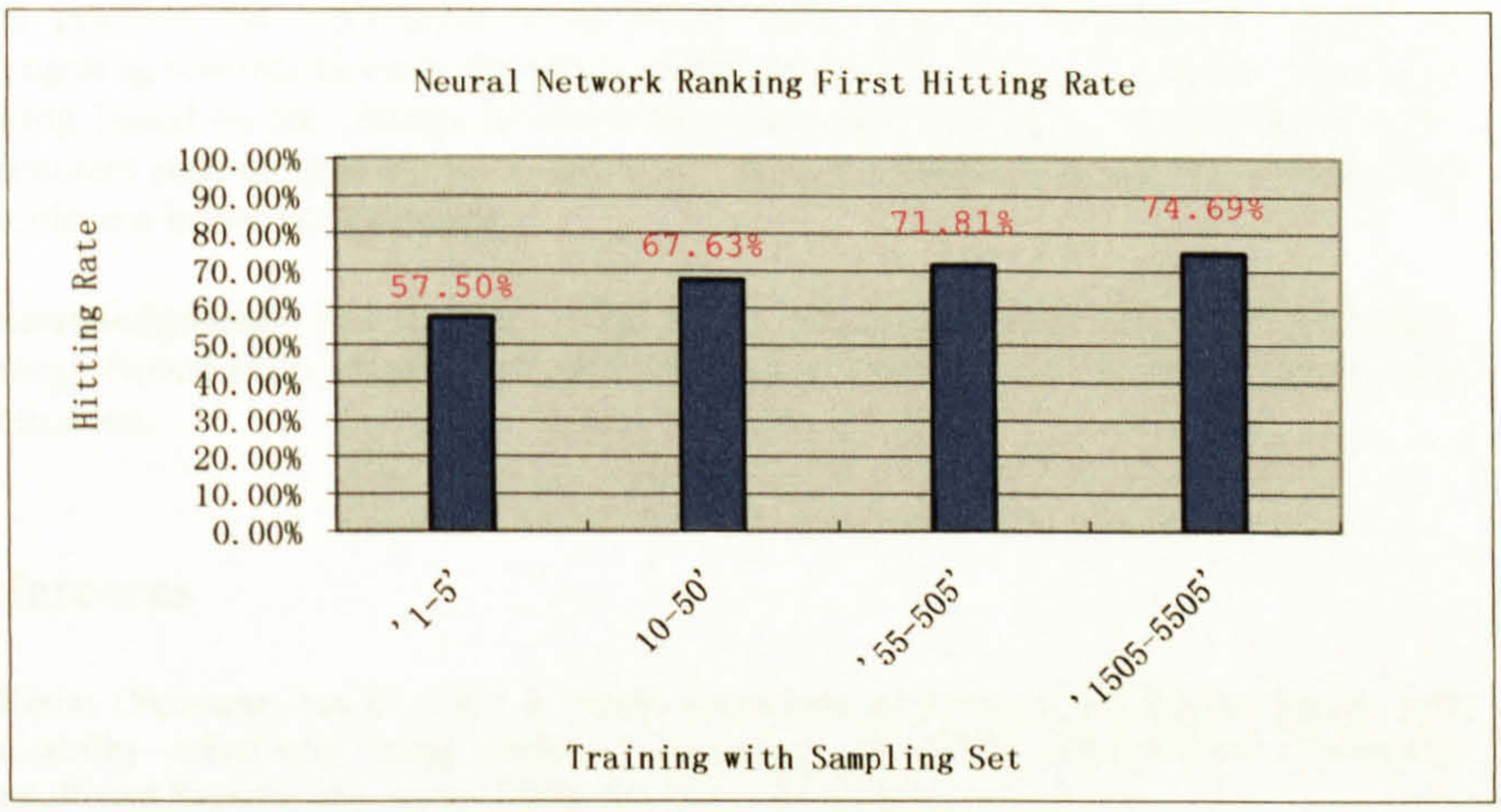


Fig. 5. X-axis refers to the increase of words frequency difference; y-axis refers to the Hitting Rate of WLR model ranking.

Figure 5 shows that the samples are separated into four categories based on the step distance of [*1, 10, 50, 1000*]. For example, the first histogram shows a *57.50%* Ranking First Hitting Rate with the samples of category one; the fourth histogram shows a best achievement of *74.69%* Ranking First Hitting Rate with more samples collected between frequency *1505* and *5505* in five separated sampling points. Figure 5 shows an increase of ranking Hitting Rate as words frequency difference and the amount of testing samples increase. This is also partly influenced by the three algorithms previously introduced with learning factors. All the algorithms are adjusting gradually toward a better prediction rate as trials increase.

5 Conclusion

In this paper a hybrid solution based on multiple typing correction algorithms and a Word List Neural Network Ranking model to produce an optimal prediction are presented. Three distinct algorithms, namely, Metaphone, Levenshtein distance and word *2*-gram are used in a pilot study. Several key factors including Time Change, Context Change and User Feedback are considered. Experimental results show that 57.50% ranking First Hitting Rate with initial samples is achieved. Further testing with updated samples indicates a best ranking First Hitting Rate of 74.69%. The findings demonstrate that neural network as a learning tool, can provide an optimum solution through combining distinct algorithms to learn and subsequently to adapt to reach a high ranking Hitting Rate performance. This may inspire more researchers to use a similar approach in some other applications.

In practice, an application using WLR theory can be implemented based on propagating rewards to each algorithm and/or word. Currently WLR model adjusts its ranking based on the change of word frequency and similarity. In the future, more parameters such as time element and more typing correction algorithms can be added to achieve a better performance.

Acknowledgments. The research is funded by Disability Essex [10] and Technology Strategy Board [11]. Thanks to Pete Collings and Ray Mckee for helpful advice and discussions.

References

1. Karim Ouazzane, Jun Li et al.: A hybrid framework towards the solution for people with disability effectively using computer keyboard. In: IADIS International Conference Intelligent Systems and Agents 2008, pp. 209-212 (2008)
2. Metaphone, http://en.wikipedia.org/wiki/Metaphone [accessed 23 January 2009]
3. N-gram, http://en.wikipedia.org/wiki/N-gram [accessed 18 January 2009]
4. Levenshtein algorithm, http://www.levenshtein.net/ [accessed 23 January 2009]
5. Cohen, William W., Ravikumar, Pradeep and Fienberg, Steve.: A Comparison of String Distance Metrics for Name-Matching Tasks, *IIWeb 2003*: 73—78 (2003)
6. Jaro-Winkler distance, http://en.wikipedia.org/wiki/Jaro-Winkler [accessed 23 January 2009]
7. Far from the Madding Crowd, http://en.wikipedia.org/wiki/Far_from_the_Madding_Crowd [accessed 15 April 2009]
8. Calgary Corpus, ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/text.compression.corpus.tar.Z [accessed 18 January 2009]
9. Matt Davis, reading jumbled texts, http://www.mrc-cbu.cam.ac.uk/~mattd/Cmabrigde/ [accessed 26 January 2009]
10. Disability Essex, http://www.disabilityessex.org [accessed 18 January 2009]
11. Knowledge Transfer Partnership, http://www.ktponline.org.uk/ [accessed 18 January 2009]

FOCUSED TIME-DELAY NEURAL NETWORK MODELING TOWARDS TYPING STREAM PREDICTION

Jun Li, Karim Ouazzane, Hassan Kazemian, Yanguo Jing
London Metropolitan University
Department of Computing and Communication Technology
{Jul029, k.ouazzane, h.kazemian, y.jing}@londonmet.ac.uk

ABSTRACT

User's typing stream contains all the information of user's interaction with computer by using QWERTY keyboard, which may include user's vocabulary, typing habit and typing performance. This paper suggests a Focused Time-Delay Neural Network model to analyze plain text and user's historical typing data. The experimental results demonstrate about 50% First Three (FT) Hitting Rate, which can be explored to both typing prediction and correction.

KEYWORDS

Focused Time-Delay Neural Network, Unary Coding, First Rank Conversion Values, Hitting Rate, FT Hitting Rate.

4. INTRODUCTION

User's typing stream generated from using computer QWERTY keyboard is a reflection of user's typing behavior that includes user's particular vocabulary, typing habit and typing performance. For example, research shows disabled keyboard users have more various performance and make more various mistakes (e.g. prolong key press and adjacent key press [1]) than others. Computer users inevitably make errors [2] and their typing stream implies all users' self-rectification actions.

N-gram prediction model is a type of probabilistic model for predicting the next item in a sequence [3]. It is widely adopted in natural language processing. But most current language modeling research have been using samples collected from some large corpus. Soukoreff and MacKenzie [4] argued that the corpus text is not a representative of user language, and it ignores the editing process and does not capture input modalities.

This research explores a Focused Time-Delay Neural Network (FTDNN) model to predict user's typing intention within a Virtual Key Code character set based on the historical typing data from Windows users. N-gram prediction can be achieved by using adjusted time delay neural network model and correction can be achieved in the same way by considering the correction as a type of predictions, which produces the right symbol based on the inaccurate historical data.

5. FOUCUSED TIME-DELAY NUERAL NETWORK MODELLING

2.1 N-gram prediction and FTDNN

N-gram prediction definition: let's assume existing string $S = \{s_1 \ldots s_i \ldots s_j \ldots s_k \ldots s_m \mid i \le j \le k \le m\}$ and $(j-i) = n, (k-j) = l$, where s_i, s_j, s_k are symbols and *i, j, k, l, m, n* are natural numbers, if one build a relation $R_n = \{x, y \mid x = (s_i \ldots s_j)_n \rightarrow y = (s_{j+1} -> s_k)_l\}$, then we call the relation as *n*-gram's l-prediction; if one consider $l = 1$ the special case, then it's called *n*-gram's one-prediction, in brief, *n*-gram prediction. For example, given string *S= {student}*, some *2*-gram prediction cases are,

'st' → 'u'
'tu' → 'd'
'en' → 't'

The Focused Time-Delay Neural Network [5] [6] consists of a feed-forward network with a tapped delay line at the input. It is part of a class of dynamic networks called focused networks, in which the dynamics appear only at the input layer. This network is well suited to time-series prediction.

Studying user's typing behavior would require the network to study user's history and trace back to certain length of context (*n*-gram) to predict the next probable occurrence. Adding one more gram requires one more time delay. Experiment has shown the FTDNN is more reliable than some other networks in response to time and memory requirement [5].

2.2 Network design and data processing

Two datasets have been used in this research: dataset one – novel 'Far from the Madding Crowd (1874)' and dataset two – Disability Essex [7] helpline keystroke log. The novel was written by Thomas Hardy [8]. It has been used as a testing sample by some compression algorithm researchers. The version used here is from Calgary Corpus [9] with a size of 751kb. The computer of Disability Essex helpline has been used as a question recording, database querying and email tool by a disabled volunteer. As discovered in the log, the typing mistakes are predominately about adjacent key press and prolong key press errors. KeyCapture software [10] is modified to record user's typing log. It runs under Windows background and records keystrokes without interfering with user's work. A typical sample of the log is demonstrated below.

```
01929 KeyPress    20080605-132149-593 "T" Status=(down) Key(84) Extra(0x14) KeyDistance(3.500000) TimeGap(307)
01930 KeyPress    20080605-132149-655 "T" Status=(up) Key(84) Extra(0xc014) KeyDistance(0.000000) TimeGap(62)
01931 KeyPress    20080605-132149-658 "H" Status=(down) Key(72) Extra(0x23) KeyDistance(2.500000) TimeGap(3)
01932 KeyPress    20080605-132149-694 "H" Status=(up) Key(72) Extra(0xc023) KeyDistance(0.000000) TimeGap(36)
01933 KeyPress    20080605-132149-804 "A" Status=(down) Key(65) Extra(0x1e) KeyDistance(5.000000) TimeGap(110)
01934 KeyPress    20080605-132149-992 "A" Status=(up) Key(65) Extra(0xc01e) KeyDistance(0.000000) TimeGap(188)
```

As raw data, the gathered dataset need to be preprocessed before it can be used by FTDNN model to simulate the probability of each predicted symbol. Let's suppose to model a data sequence $C = \{s_1 ... s_i ... s_n\}$ on an alphabet basis of size $A = \{a, ..., z\}$, where $s_i \in A$. Two input coding methods based on unary code and ASCII code can be considered. A sample is shown in Table 2.2.1,

Table 2.2.1. Unary coding and ASCII coding sample

Alphabet	Unary Coding	ASCII Coding
a	1	97 = 01100001
b	01	98 = 01100010
c	001	99 = 01100011
d	0001	100=01100100
e	00001	101=01100101

Unary coding is an entropy encoding [11] that represents a symbol by using *n−1* zeros followed by a one. From Table 2.2.1, '*a*' is represented by a one while '*c*' is represented by two zeros followed by a one. The ASCII code uses a fixed length (here 8 bits) to stand for a symbol. As shown in the table, '*a*' is represented by *01100001* which consist of three ones and five zeros. The unary codes can be adapted to a fixed length to fit the requirement of neural network unchanging number of input neurons. Let's consider a data set with three symbols {*a, b, c*}, and then one can code it as fixed unary codes, {*100, 010, 001*}.

A twenty-seven symbols set {*a...z, space*} is applied to dataset one. A preprocessing logic followed by an example is illustrated below,

for each symbol $s_i \in$ *context C, where* $C = \{s_1 ... s_n\}$

{

if 'a' < s_i < 'z' then write unary code to file

else if 'A' < s_i < 'Z' then convert to {a,...,z} and write unary code to file

else convert to blank and write unary code to file

}

%a b c d e f g h i j k l m n o p q r s t u v w x y z ''

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 %'t'

0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 %'h'

0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 %'e'

At the output a post-processing function is used to generate unary code by ranking the twenty-seven outputs in a descending order. The output of the sample above is presented below:

0.02 0.03 0.01 0.01 0.01 0.01 0.01 0.88 0.01 0.01 0.01 0.01 0.01 0.01 - 0.01 0.01 0.01 0.01 0.01 0. 01 0.01 0.01 0.01 0.01 0.01 0.01 0.01

0.02 0.04 0.01 0.01 0.97 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 - 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01

0.03 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.02 0.01 - 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.79 0.01 0.01

Firstly the maximum value of each line is converted into one and the rest values are converted into zeros. This is named as First Rank Conversion Values as shown below. The unary code, which is converted based on second biggest value among output, is named as Second Rank Conversion Values and so on.

%a b c d e f g h i j k l m n o p q r s t u v w x y z ''

0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 %'h'

0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 %'e'

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 %'y'

Then, according to the 2-gram prediction model the relationship between input and output is shown as,

't' → *'h'*

'h' → *'e'*

'e' → *'y'*

A piece of data (100k) is extracted from dataset one. It is subsequently divided into training data, validation data and testing data. For this n-gram prediction model, a three-layer Focused Time-Delay Neural Network with twenty-seven input neurons, twenty-seven output neurons, extendible numbers of hidden layer neurons and extendible numbers of time delays is designed. Both input and output are encoded in unary code. A piece of software is designed to extract Virtual Key Codes [12] values according to the keystroke status from dataset two. In this research only the most commonly used fifty-three Virtual Key Codes are adopted, others such as arrows and functional keys are deleted from the log. Then the three-layer Focused Time-Delay Neural Network model is adjusted to have fifty-three input neurons and fifty-three output neurons.

2.3 Experimental results

In order to evaluate the experimental results, two concepts are introduced here namely, Hitting Rate (HR) and First Three (FT) Hitting Rate. Given testing metrics P, target metrics T and testing result metrics R where their numbers of lines and columns are equal and expressed as n, m respectively, then the Hitting

Rate is $HR = \{hr_i \mid hr_i = zeros(T - R_i)/n, i \in m\}$, where R_i is the i^{th} Rank Conversion Values of R, *zeros()* is the function to compute the number of zero vector included in metrics, obviously the sum of all Hitting Rates is $\sum HR = \sum_{i=1}^{m} hr_i = 100\%$. Then the First Hitting Rate and First Three Hitting Rate are hr_1 and $\sum_{i=1}^{3} hr_i$ respectively. Based on the previous example and above definition, then we have testing metrics *P= {'the'}*, testing result metrics *R= {'hey'}*, *n = 3* and *m = 27*. Assume the target dataset *T = {'hem'}*, then the first hitting rate is $hr_1 = zeros(T - R_1)/n = zeros('hey' - 'hem')/3 = 2/3 = 66.67\%$, whereas the 2nd and 3rd Rank Conversion Values are,

--2nd rank conversion values

%a b c d e f g h ij k l m n o p q r s t u v w x y z ''	
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	*%'b'*
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	*%'b'*
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	*%'a'*

--3rd rank conversion values

%a b c d e f g h ij k l m n o p q r s t u v w x y z ''	
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	*%'a'*
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	*%'a'*
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	*%'m'*

So, the First Three Hitting Rate is $2/3 + 0/3 + 1/3 = 100\%$, which is an ideal case. The average rate is just below 50%.

During the training and testing of the FTDNN model related to dataset one, the numbers of grams – [*1, 2, 3, 5, 7, 9, 11, 13*] which are represented by time delays, and the numbers of hidden neurons – [*1, 2, 3, 5, 7, 9, 15, 25, 50, 100*] are cross-designed and implemented. Thereinto as the gram reaches *11* and the number of hidden neurons reaches *100*, the gram reaches *13* and the number of hidden neurons reaches *15* onwards, the memory of current system is beyond its limit. So the experimental results are abandoned from *G-11 & H-100* onwards. As illustrated, the model uses a *27-n-27* three-layer structure. The experimental results related to dataset one plotted with First Hitting Rate and First Three Hitting Rate are shown below,

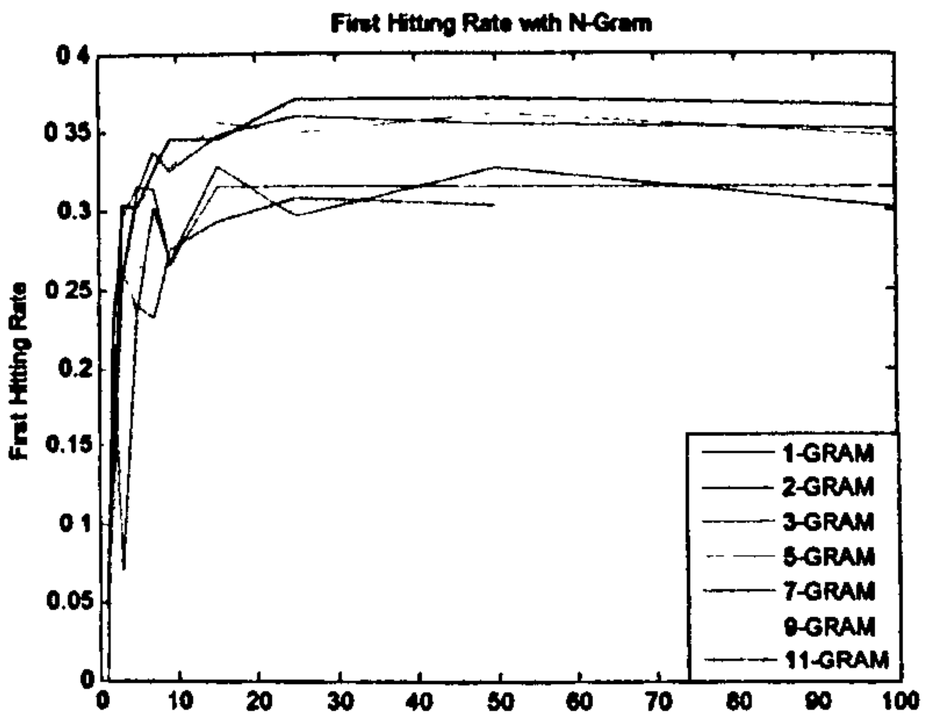


Figure 2.3.1. First Hitting Rate with N-gram

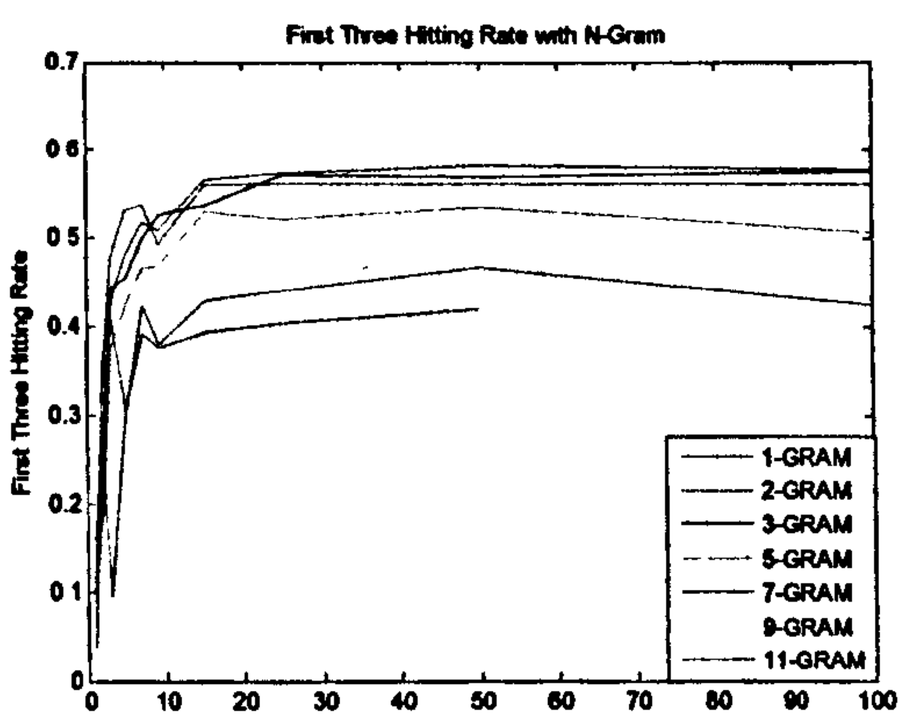


Figure 2.3.2. First Three Hitting Rate with N-gram

Fig 2.3.1 and Fig 2.3.2 show *1, 2 & 3*-gram give the three best Hitting Rate (by winning in a small margin, *3*-gram gives the best First hitting rate and *2*-gram gives the best FT hitting rates), all of which show a better convergence toward the maximum Hitting Rate (about 56% of FT Hitting Rate and 33% of First Hitting Rate). Both pictures illustrate the smaller Hitting Rates from *4*-gram onward. The results

Rate is $HR = \{hr_i \mid hr_i = zeros(T - R_i)/n, i \in m\}$, where R_i is the i^{th} Rank Conversion Values of R, *zeros()* is the function to compute the number of zero vector included in metrics, obviously the sum of all Hitting Rates is $\sum HR = \sum_{i=1}^{m} hr_i = 100\%$. Then the First Hitting Rate and First Three Hitting Rate are hr_1 and $\sum_{i=1}^{3} hr_i$ respectively. Based on the previous example and above definition, then we have testing metrics *P= {'the'}*, testing result metrics *R= {'hey'}*, *n = 3* and *m = 27*. Assume the target dataset *T = {'hem'}*, then the first hitting rate is $hr_1 = zeros(T - R_1)/n = zeros('hey' - 'hem')/3 = 2/3 = 66.67\%$, whereas the 2nd and 3rd Rank Conversion Values are,

--2nd rank conversion values

%a b c d e f g h ij k l m n o p q r s t u v w x y z ''	
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	*%'b'*
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	*%'b'*
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	*%'a'*

--3rd rank conversion values

%a b c d e f g h ij k l m n o p q r s t u v w x y z ''	
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	*%'a'*
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	*%'a'*
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	*%'m'*

So, the First Three Hitting Rate is $2/3 + 0/3 + 1/3 = 100\%$, which is an ideal case. The average rate is just below 50%.

During the training and testing of the FTDNN model related to dataset one, the numbers of grams – [*1, 2, 3, 5, 7, 9, 11, 13*] which are represented by time delays, and the numbers of hidden neurons – [*1, 2, 3, 5, 7, 9, 15, 25, 50, 100*] are cross-designed and implemented. Thereinto as the gram reaches *11* and the number of hidden neurons reaches *100*, the gram reaches *13* and the number of hidden neurons reaches *15* onwards, the memory of current system is beyond its limit. So the experimental results are abandoned from *G-11 & H-100* onwards. As illustrated, the model uses a *27-n-27* three-layer structure. The experimental results related to dataset one plotted with First Hitting Rate and First Three Hitting Rate are shown below,

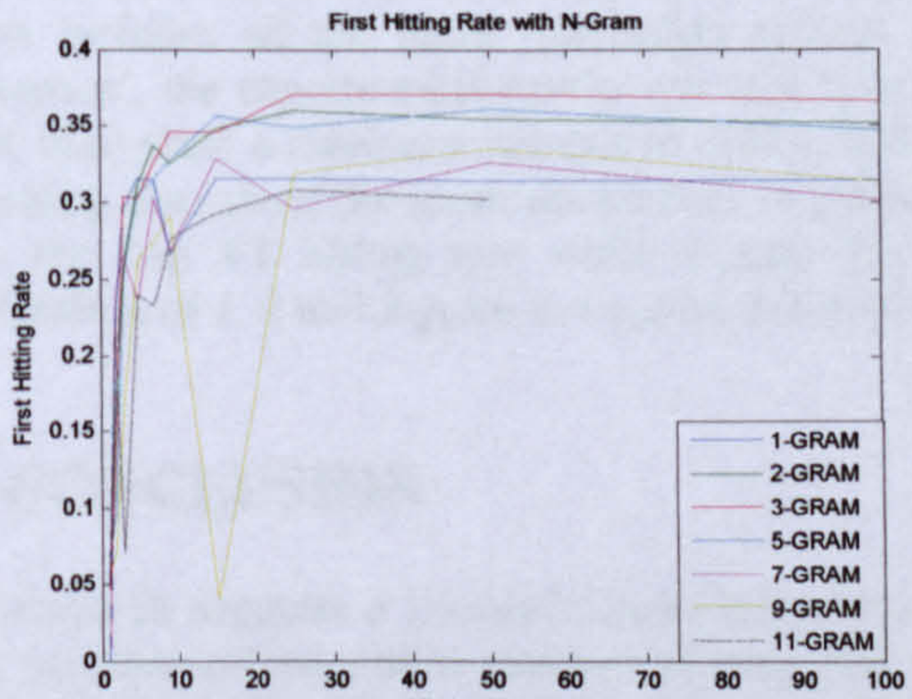


Figure 2.3.1. First Hitting Rate with N-gram

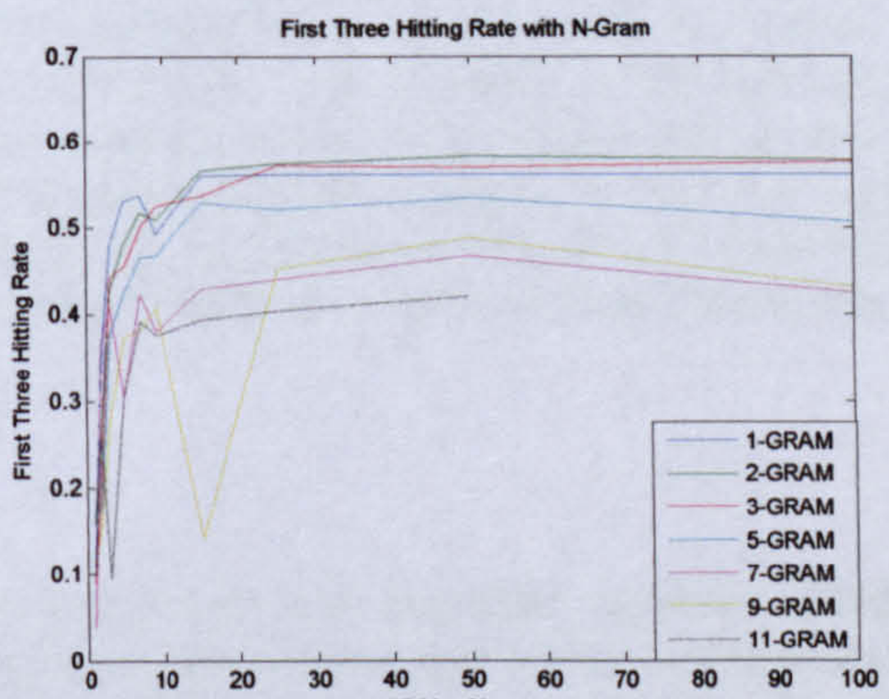


Figure 2.3.2. First Three Hitting Rate with N-gram

Fig 2.3.1 and Fig 2.3.2 show *1, 2 & 3*-gram give the three best Hitting Rate (by winning in a small margin, *3*-gram gives the best First hitting rate and *2*-gram gives the best FT hitting rates), all of which show a better convergence toward the maximum Hitting Rate (about 56% of FT Hitting Rate and 33% of First Hitting Rate). Both pictures illustrate the smaller Hitting Rates from *4*-gram onward. The results

suggest that under the training sample, there would have been a best gram with certain number of hidden unit to suit the prediction best. After a certain increase, further increase of gram or hidden unit doesn't help finding a good prediction. The figures also show that the number of neuron in hidden layer affects the model's learning ability and Hitting Rate. As suggested, the hitting rate in a hierarchy levels also can be used in prediction ranking.

Due to the limited learning ability of less number of hidden neurons as shown in the experimental results, the testing relating to dataset two with one and two hidden neuron are ignored. And due to the memory limitation, the testing of *11*-gram and *13*-gram are abandoned. So, for typing stream dataset two, the chosen grams set is [*1, 2, 3, 5, 7, 9*] and the hidden neurons set is [*3, 5, 7, 9, 15, 25, 50, 100*]. The model uses a *53-n-53* three-layer structure. The experimental results related to dataset two plotted in First Hitting Rate and First Three Hitting Rate are outlined below,

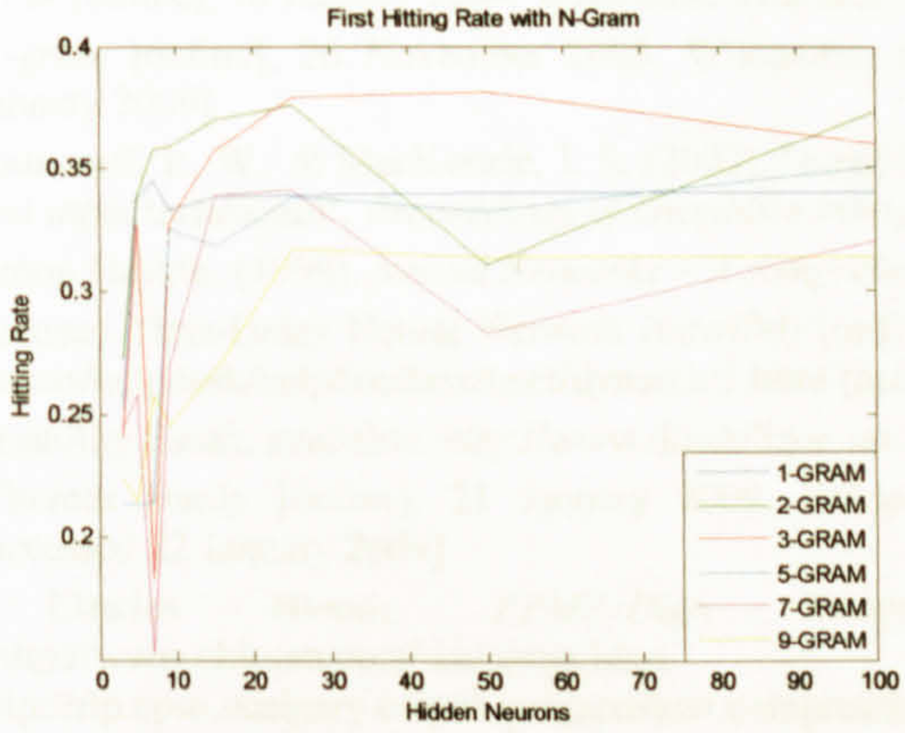


Figure 2.3.3. First Hitting Rate with N-gram

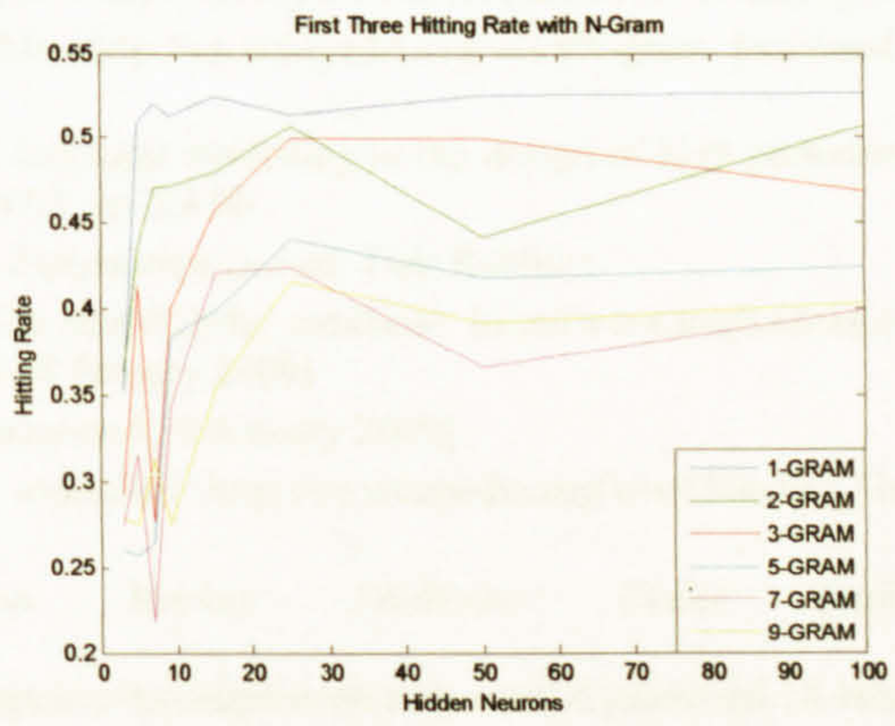


Figure 2.3.4. First Three Hitting Rate with N-gram

As shown in Fig 2.3.3 and Fig 2.3.4, *1*-gram has gained the maximum FT Hitting Rate – 53% and *3*-gram with fifty hidden neurons produces the maximum First Hitting Rate – 38.1%. Similar results have been obtained when testing with dataset one: the lower grams (*1, 2&3*-gram) show a better solution with the FTDNN model prediction under current circumstance. Both datasets demonstrated a highly accurate prediction rate (FT Hitting Rate around 50%) with FTDNN model.

The experimental results can be used to predict users' typing intention. In practice a higher prediction rate could be obtained by combining the FT Hitting Rate with an English word dictionary. As the typing stream includes all the users' correction actions and the predicted next symbol could be 'delete' or 'backspace', the experimental results can also be used to correct users' current typing. Both tests (dataset one & two) show a minimum number of hidden neurons are required in order to get a good hitting rate. But the testing also show the gram uncertainty in getting a best hitting rate, for example, in Fig2.3.2, 2-gram gives the best FT hitting rate while *3*-gram has the best FT hitting rate in Fig 2.3.4. Therefore, a combination of *1, 2* and *3*-gram is a optimum solution to keep a considerably high and stable hitting rate.

6. CONCLUSION

This research suggests a Focused Time-Delay Neural Network model with extendible numbers of hidden layer neurons and extendible numbers of time delays to analyze plain text and user's historical typing data. Approximately 50% FT Hitting Rate has been obtained from experimental results. In practice, the results can be applied to symbol prediction and correction.

Further research will include using a distributed representation method [13] to preprocess the typing symbols, where each symbol will be represented by several features such as key distance, time stamp and symbol itself. Then the prediction will not only be based on the symbols themselves but also the related *n*-gram features. Another near future work is to apply FTDNN model to predict *l*-length string based on *n*-gram's *l*-prediction. Therefore with the same *n*-gram input as presented in this research, more symbols can be predicted.

ACKNOWLEDGEMENT

The research is funded by Disability Essex and Technology Strategy Board [14]. Thanks to Richard Boyd and Pete Collings for helpful advice and discussions.

REFERENCES

[1] Karim Ouazzane, Jun Li and Marielle Brouwer (2008). 'A hybrid framework towards the solution for people with disability effectively using computer keyboard'. *IADIS International Conference Intelligent Systems and Agents 2008*, pp. 209-212

[2] *Error* [online], 18 January 2009, Wikipedia, available: http://en.wikipedia.org/wiki/Error [accessed 18 January 2009]

[3] *N-gram* [online], 26 November 2008, Wikipedia, available: http://en.wikipedia.org/wiki/N-gram [accessed 18 January 2009]

[4] Soukoreff, R. W., & MacKenzie, I. S. (2003). 'Input-based language modelling in the design of high performance text input techniques'. *Proceedings of Graphifcs Interface 2003*, pp. 89-96

[5] Simon Haykin, (1999). *Neural Networks – A comprehensive Foundation 2nd ed.* Tom Robbins

[6] Focused Time-Delay Neural Network (newfftd) [online], The MathWorks, available: http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/dynamic3.html [accessed 18 January 2009]

[7] Disability Essex, available: http://www.disabilityessex.org [accessed 18 January 2009]

[8] Thomas Hardy [online], 21 January 2009, Wikipedia, available: http://en.wikipedia.org/wiki/Thomas_Hardy [accessed 22 January 2009]

[9] Charles Bloom, *PPMZ–High Compression Markov Predictive Coder* [online], http://www.cbloom.com/src/ppmz.html and ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/text.compression.corpus.tar.Z [accessed 18 January 2009]

[10] William Soukoreff and Scott MacKenzie, n.d. *KeyCapture* [online], available: http://dynamicnetservices.com/~will/ academic/textinput/keycapture/ [accessed 18 January 2009]

[11] *Entropy encoding* [online], 7 September 2008, Wikipedia, available: http://en.wikipedia.org/wiki/Entropy_encoding [accessed 18 January 2009]

[12] Charles Petzold, (1998). *Programming Windows, 5th Edition.* Microsoft Press

[13] *Neural net language models* [online], 19 April 2008, Wikipedia, available: http://www.scholarpedia.org/article/Neural_net_language_models [accessed 18 January 2009]

[14] Knowledge Transfer Partnership, available: http://www.ktponline.org.uk/ [accessed 18 January 2009]

A HYBRID FRAMEWORK TOWARD THE SOLUTION FOR PEOPLE WITH DISABILITY EFFECTIVELY USING COMPUTER KEYBOARD

Karim Ouazzane
Jun Li
London Metropolitan University
Department of Computing and Communication Technology
{k.ouazzane, Jul029}@londonmet.ac.uk

Marielle Brouwer
Disability Essex, Rocheway, Rochford, Essex SS4 1DQ
marielle.brouwer@disabilityessex.org

ABSTRACT

In this paper a hybrid framework was presented based on machine learning model to offer an efficient solution for people with disability using QWERTY keyboard. It integrates neural network, language model and natural language processing etc technologies and provides user with two fundamental functions: word prediction and typing correction. A development of a pilot application as an English input method has been introduced.

KEYWORDS

Language model, machine learning, neural network.

1. INTRODUCTION

Computer users with motor disabilities or cognitive problems may have difficulties in accurately manipulating the QWERTY keyboard. As for motor disability this may be seen in a form of tremor owing to a certain disease such as Parkinson's or any other factor, for instance reduced range of hand motions due to Arthritis. Cognitive problems usually are caused by loss of the ability to process, learn and remember information [1]. For example, Dyslexia can cause significant problems in remembering even short sequences of numbers in the correct order.

Those types of disability frequently cause typing mistakes, which haven't been well solved by current solution [2] [3], not to mention problems people may have with several symptoms, which cannot be categorized as common types of symptoms. Although alternative input devices or software such as keyguard, Dasher [2] are available for use, none of them prove more efficient or comfortable than the current QWERTY keyboard. Some work associated with standard keyboard has been developed such as Windows' Accessibility Options, ProtoType [3], however the solution towards typing difficulties by disabled people hasn't been achieved yet.

This paper intends to give a comprehensive solution to help disabled people to use keyboard more efficiently. A novel architecture has been suggested based on machine learning model and neural network. Neural network and language model have been studied and used as central modules. User's input context can be checked in sequence by each module along with user's typing process.

2. HYBRID FRAMEWORK

2.1 User investigation

About 27 people have been interviewed. Both, old and disabled people were involved. Their performance can be classified as four categories as illustrated below.

- **Motor disability [1]**
    1. **Long key press.** This occurs when an alphanumeric key is unintentionally pressed for longer than the default key repeat delay.
    2. **Modifier keys.** For example, "Shift"+ "a". One-hand typists in particular may find it difficult to press two keys at once.
    3. **Additional keys.** Some users often press keys adjacent to the intended keys.
    4. **Bounce errors.** These occur when the user unintentionally presses a key more than once.
    5. **Prefer big keys.** They don't like laptop because of the smaller key. They prefer big keys, for example, "space bar"
    6. **Easily tired.** It's a very hard task for them to input more than hundred words.
- **Dyslexia**
    1. **Miss letters or add letters.** For instance, "student" -> "studnt"
    2. **Letters reverse.** For instance, "student"->"studnet"
    3. **Spelling errors.** For instance, leave vowel out of word, "magic"->"mjc"
    4. **Similar word errors.** For instance, "dose"->"does"
    5. **Phonetic form.** For instance, "shud"->"should"
- **Unfamiliar with computer**
    1. **Difficult to find keys.** Especially function and punctuation keys (e.g. F12).
    2. **'Enter' key puzzle.** Some computers are with no "enter" or "shift" printed on the key surface, so it's difficult for old people to find where those keys are.
    3. **Compound keys problem.** Due to different definitions in distinct software, compound keys' meanings are causing trouble to many people.
- **Others performance**
    1. **Miss words.** Leave out words in the typed sentences.
    2. **Mix words.** Reverse words in a sentence.
    3. **Mix lines.** If there are some similarities (for example, same words) between two or more lines, the user could mix lines.
    4. **Additional words.** User could add additional words to a structured and fully meaningful sentence.
    5. **Non-sense sentence.** From the context of paragraph, the sentence which user is typing is not what they want to type.
    6. **One-hand users' difficulties.** There are unclear different difficulties for left hand and right hand user in using the same kind of keyboard.

In order to provide a solution for practical use, this paper has aimed at "Motor Disability" and "Dyslexia". As a hybrid system, solutions for other performance can be integrated in the future.

2.2 The new solution

Current the research concentrates on how to correct the typing mistakes and foresee users' typing intention. Single technology such as Neural Network alone is difficult to produce a full solution, so we adopt a hybrid intelligent system architecture based on a machine learning model and neural network. It consists of three subsystems, namely, neural network, language model and natural language processing.

Recurrent Neural Network and Boltzmann Machine have been studied and used in this paper as a tool to learn users typing behaviors from the bytes stream of keyboard. Lossless data compression methods such as PPM and PAQ [4] were investigated as language models for the purpose of word prediction. Then, three combinations based on neural network and language model were proposed and researched respectively. Figure 1 is a logical picture of the proposed hybrid system.

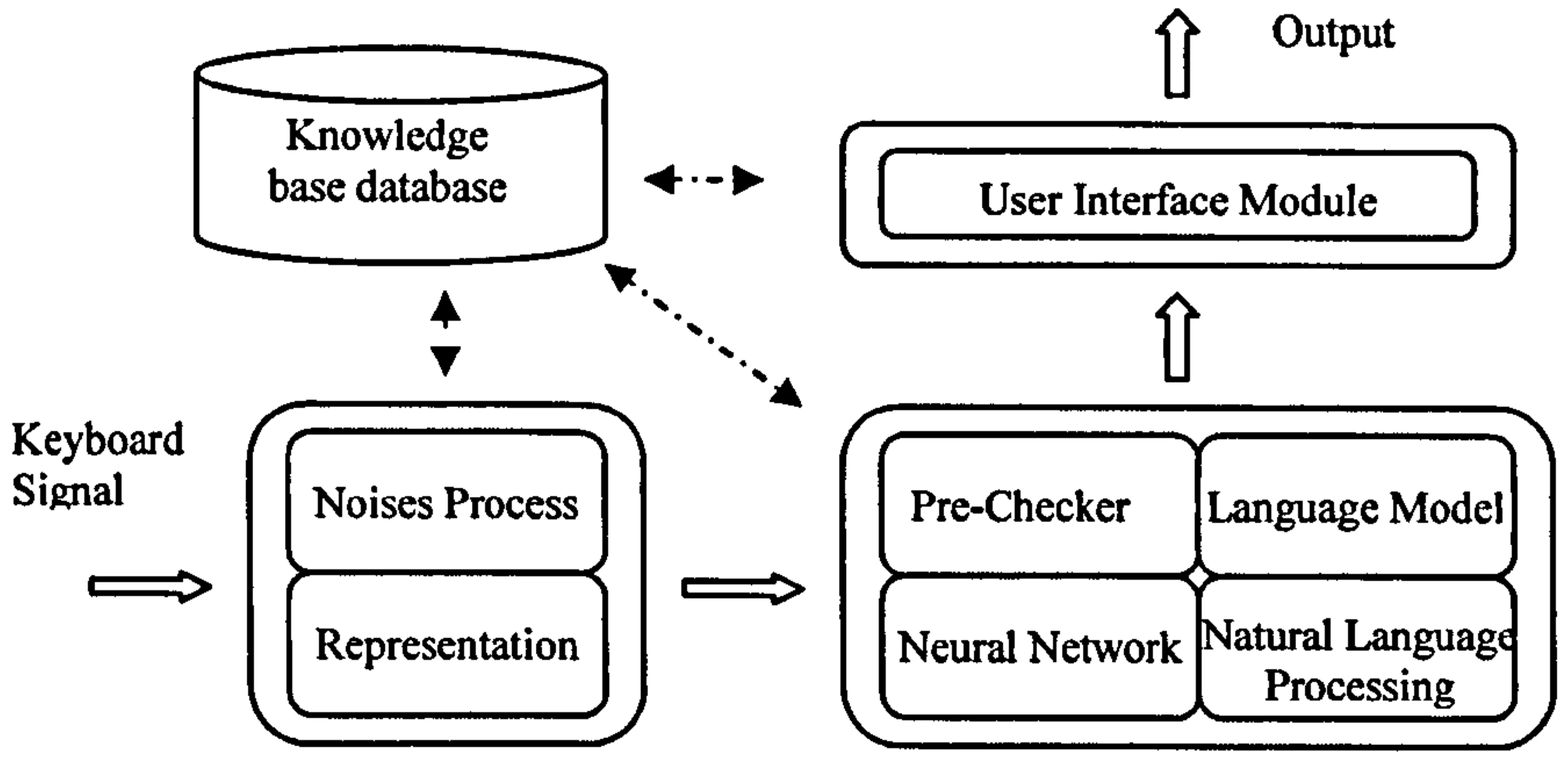


Figure 1. The architecture of the hybrid system

The developed hybrid system was inspired by this machine learning model. It includes pre-processing, central processing and user interface etc units.

Much of the typing data stream could be un-preprocessed, incomplete and noisy. For example, a long key press generates more than one Window's message. For further analysis, the data stream needs to undergo Noises Processing module. Subsequently, a representation format would be chosen to feed the central processing.

Pre-Checker is a customized module. It checks user configuration to decide whether to perform a unique processing or to further send a signal to other modules. Language Model and Neural Network module provided users with two fundamental functions: Word Prediction and Typing Correction. For the purpose of enhancing the efficiency, three novel combinations models based on neural network and language model were developed, namely, sequential model, parallel model and the-one model. The system will adopt the right model in terms of the user profile. The typing mistakes, which are being speculative, are further sent to Natural Language Processing module to be analyzed. The results then are refurbished and shown to the user through User Interface module.

Here, a topology sample of Neural Network is given. Suppose, one has to model the following typing data sequence $D = \{d_i\}_{i=1}^{*}$ within an alphabet range of $A = \{a_i \mid a_i \in [a,z]; i = 1,...,27\}$, then recurrent neural network to model the probability of the right symbol is used; two input coding method may be considered as the network input, namely, unary code and ASCII. For instance, if unary code shown in Figure 2 is used, the architecture of neural network with three letters {a, b, c} and two hidden layer is illustrated in Figure 3, where P_e is Error probability summarized based on the output.

Alphabet	Coding
a	1
b	01
c	001
d	0001
e	00001

Figure 2. Unary code sample

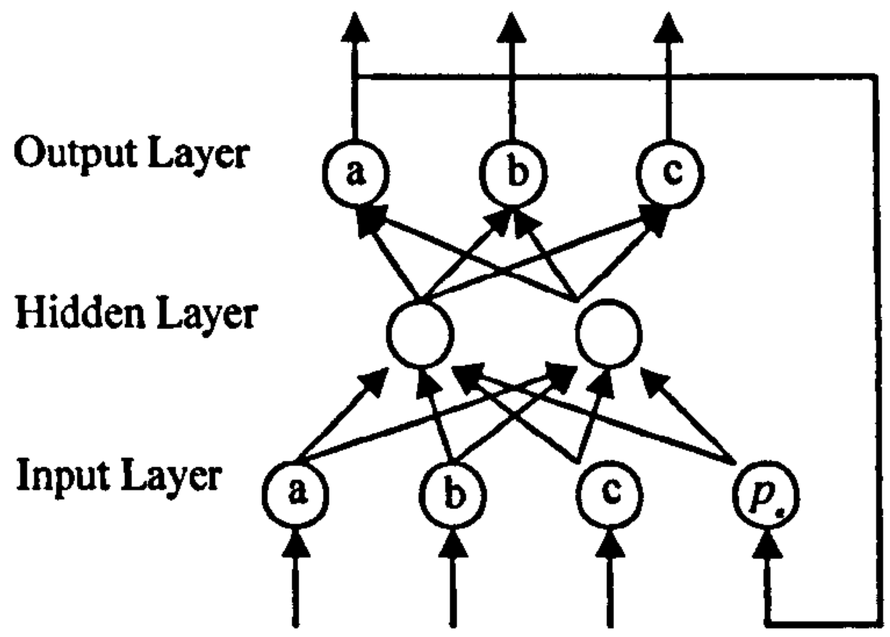


Figure 3. Topology sample of Neural Network

ASCII coding requires less neurons (between 8 and 20 at input layer), but need to summarize all the possible output (>8bits).

VC6 is used as a development tool within Windows XP based environment. IME (Input Method Editor) API is used to provide the hybrid system with a way to communicate with most of the editors. For efficiency reasons a unique database and its interface has been developed instead of using ODBC. An interface of pilot application using Notepad as an editor is shown in Figure 4.

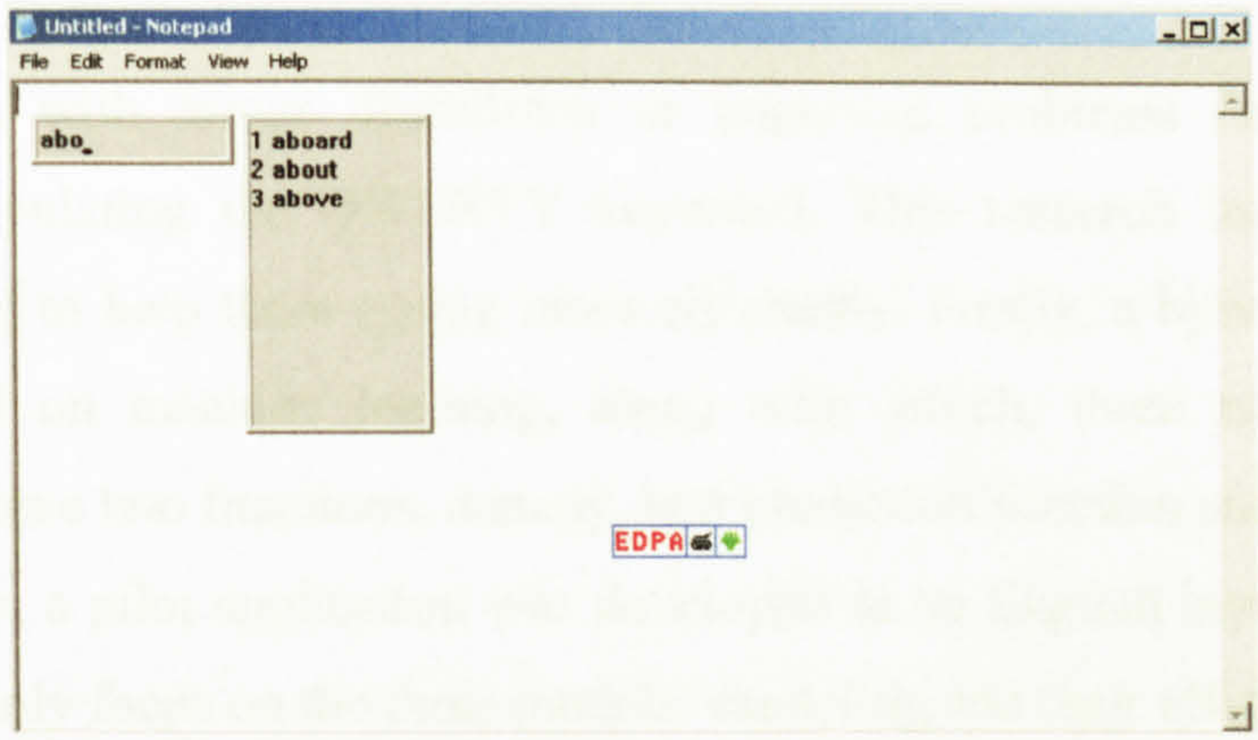


Figure 4. The user interface of the hybrid system

3. CONCLUSION

The research provides disabled people with a comprehensive solution using QWERTY keyboard. The hybrid system integrates multiple technologies and presents two major functions: Typing Correction and Word Prediction. The user typing intention is predicted based on the input history; and the typing errors in data streams are gradually corrected as the process goes through each module. A pilot application based on Windows XP and IME API has been developed to demonstrate the hybrid system architecture. Further work would explore the efficiency comparison among these three combined models. Bayesian Learning can be used as an alternative method to evaluate the research result.

ACKNOWLEDGEMENT

The research was funded by Disability Essex. Thanks to Richard Boyd, Pete Collings, Stuart Kirk and YanGuo Jing for helpful advice and discussions.

REFERENCES

[1] Shari Trewin & Helen Pain, 1998. A Model of Keyboard Configuration Requirements. *Proceedings of the third international ACM conference on Assistive technologies*, Marina del Rey, CA: pp. 173-181.

[2] Inference Group of Cambridge, *Dasher*, available at <URL: http://www.inference.phy.cam.ac.uk/dasher/ > [access 25 January 2008]

[3] Sensory Software International Ltd, *ProtoType,* available at <URL: http://www.sensorysoftware.com/prototype.html > [access 15 February 2008]

[4] Wikipedia, *Data compression,* available at <URL: http://en.wikipedia.org/wiki/Data_compression > [access 15 February 2008]

Poster – Intelligent Keyboard

Jun Li

Summery

Computer users with motor disabilities or cognitive problems have difficulties in accurately manipulating the QWERTY keyboard. This research intends to apply an intelligent model to help them typing more efficiently. Firstly, a hybrid framework was presented based on machine learning, along with which, three novel models were proposed to achieve two functions, namely, text prediction function and typing correction function. Finally, a pilot application was developed as an English input method. Further work would mainly focus on the three models' modeling and their efficiency evaluation.

Introduction

Computer users with motor disabilities or cognitive problems may have difficulties in accurately manipulating the QWERTY keyboard. Through user investigation, four categories of disability were concluded, which includes motor disability, dyslexia, unfamiliar with computer and other performance. In order to provide a solution for practical use, this research has aimed at "Motor Disability". As a hybrid system, solutions for other performance can be integrated in the future.

Hybrid Architecture and Three Models

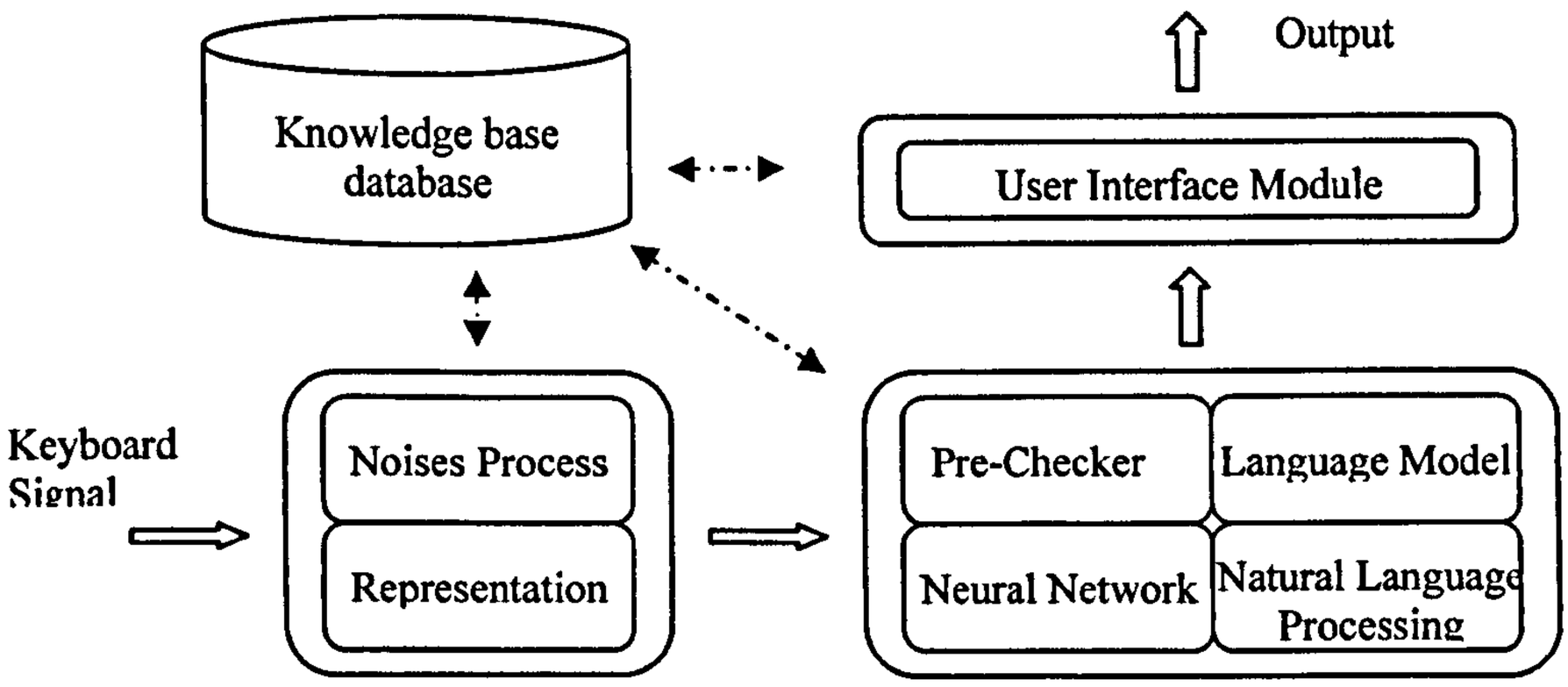


Figure1. The architecture of the hybrid system

Keyboard signals need to undergo Noises Processing module first. Subsequently, a representation format would be chosen to feed the central processing. Pre-Checker module checks user configuration to decide whether to perform a unique processing or further send a signal to other modules. Language Model and Neural Network module provided users with two fundamental functions: Word Prediction and Typing Correction. For the purpose of enhancing the efficiency, three novel combinations models based on neural network and language model were developed, namely, sequential model, parallel model and the-one model shown below,

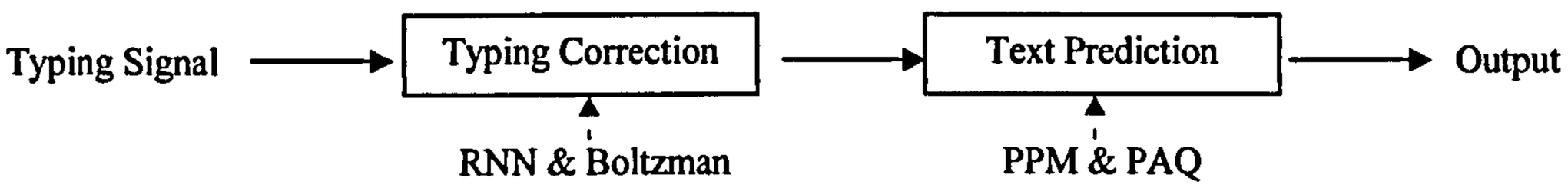


Figure2. Sequential model

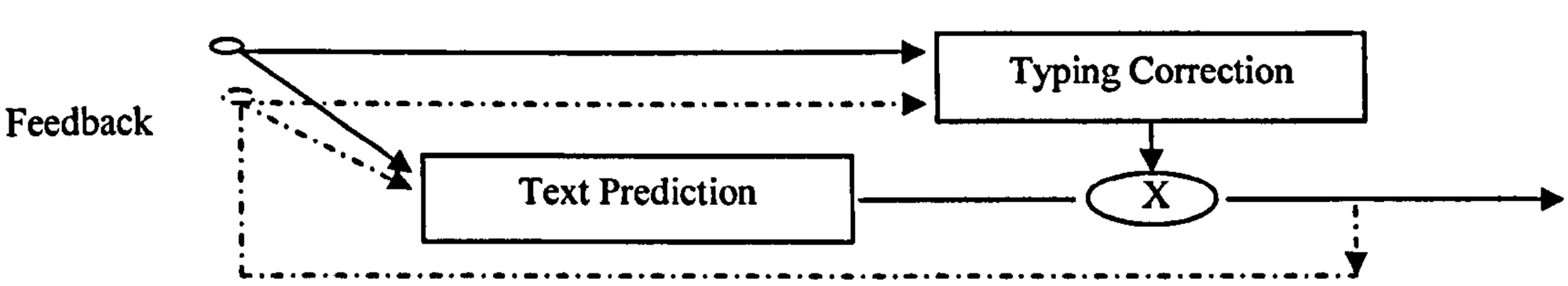


Figure3. Parallel model

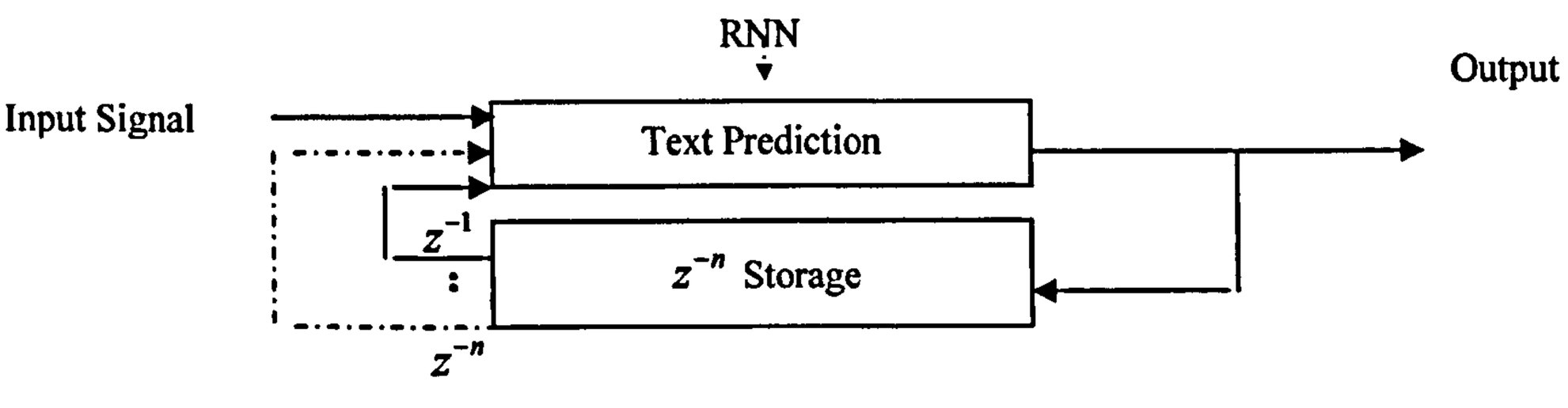


Figure4. The one model

At last a pilot application was developed for purpose of demonstration. VC is used as a development tool within Windows XP based environment. An interface by using Notepad as an editor is shown in Figure below,

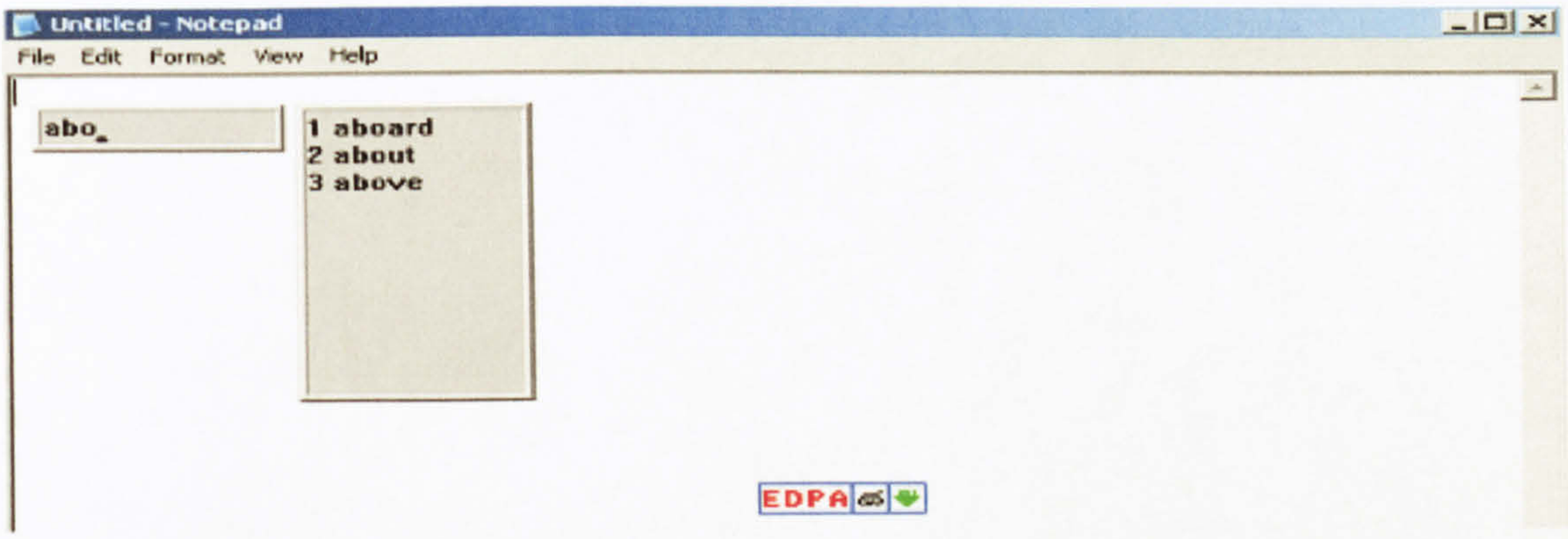


Figure5. The user interface of the hybrid system

Conclusion and Future work

The research provides disabled people with a comprehensive solution using QWERTY keyboard. The hybrid system integrates multiple technologies and presents two major functions: Typing Correction and Word Prediction. The user typing intention is predicted based on the input history; and the typing errors in data streams are gradually corrected as the process goes through each module. A pilot application based on Windows has been developed to demonstrate the hybrid system architecture. Further work would explore the efficiency comparison among these three combined models. Bayesian Learning can be used as an alternative method to evaluate the research result.