A Neural Network GUI Tested on Text-To-Phoneme Mapping

MAARTEN TROMPPER

Universiteit Utrecht
m.f.a.trompper@students.uu.nl

Abstract

Text-to-phoneme (T2P) mapping is a necessary step in any speech synthesis system. For some languages, like English, it can be hard to derive a good set of rules to cover all letter-phoneme correspondences. This paper presents an open-source GUI for training artificial neural networks that is tested on the problem of text-to-phoneme mapping. The results are just below 80% accuracy with a standard feed forward neural network.

I. INTRODUCTION

1.1 Problem description

Text-to-phoneme mapping is an important step in any text-to-speech system. Some languages, like Finnish or Japanese, have regular phonological mapping rules. For other languages, like English or French, it can be very hard to find a complete set of rules [1, p. 49]. Here, we consider the problem of converting English text to phonemes by training artificial neural networks. For this purpose, an environment to train and test the networks is needed, as well as a means of coding textual information into activation values that the networks can use.

II. BACKGROUND ON ANNS

2.1 Artificial neural networks

An artificial neural networks (ANN; for brevity also referred to as 'neural networks' or simply 'networks') is a mathematical graph that is inspired by natural neural networks; brains. They consist of inter-connected neurons that propagate or inhibit a certain flow of activation through neural pathways.

There are some distinct differences between the functioning of artificial and biological neural networks. ANNs usually have a controlled signal flow (from the input neurons to the output neurons, layer-to-layer) and don't use neurotransmitters. They also usually contain way less neurons than the brains of high functioning organisms do: up to a thousand in a typical ANN compared to billions of neurons in a human brain.

ANNs are frequently used for tasks that are associated with human functioning, like recognizing patterns for which a rule-based system or an efficient algorithm is hard or impossible to find. Examples include face recognition [9] and e-mail spam filtering [8].

An ANN can learn by adjusting the weights of the connections between neurons to values which correspond closer to the desired input-output mapping. This adjustment is most commonly done by a supervised learning algorithm, although any learning paradigm can theoretically be used. Other learning paradigms include unsupervised or reinforcement learning.

Most neural networks work in the following way, using supervised learning:

1. An input pattern is presented to the net-

work, represented as an array of numbers. The numbers are commonly real-valued numbers between 0 and 1. This is done by copying these numbers to the firing values of their corresponding input nodes.

2. The input pattern is propagated through the network by setting the firing values of all subsequent nodes. The activation value of the subsequent nodes depends on some activation function, most frequently the delta function 2. The activation function ensures that the firing value of any node is a real number between 0 and 1. Eventually the signal reaches the output nodes, and so the network presents its output.

3. If the network is in the training phase, the activation values of the nodes are compared to the desired activation values. The network weights are then updated to better respresent the desired mapping

2.1.1 Feed Forward Neural Network

The feed forward neural network (FFNN; also called perceptron, see Figure 1) was the first type of neural network to be developed. It is one of the most commonly used and studied network architectures, due to its simplicity and all-round performance.

An FFNN is built up in layers, which each consist of an array of nodes. A node in a layer is only allowed to have connections with nodes in the next layer. The final layer has no outgoing connections, and is the output layer. The output is computed by propagating the input layer-by-layer, hence 'feed forward'.

An FFNN has at least two layers: an input layer and an output layer. Because it has only been proven than FFNNs with one or more hidden layers are universal approximators, a hidden layer is usually added. [2] Perceptrons with at least one hidden layer are referred to as 'multi-layer perceptrons'.

Also, it is common for an FFNN to start out fully connected. That is, every node in a layer is connected to every node of the following layer. After the network is trained, it is common practice to delete the weights that play a neglectable role in the calculation of the output layer. This speeds up the propagation of the input signal.

2.1.2 Recurrent Neural Network

Because the input layer in an FFNN gets overwritten when a new pattern is presented, the network won't be able to show any sort of memory from the previous input. Having this kind of memory is desirable in certain situations, though. For example, it is desirable when the input size is not bound to a specific size, such as in the task of recognizing handwriting. The recurrent neural network (RNN) is an adaptation of the FFNN to allow it to model sequential data. This is done by making a connection be-

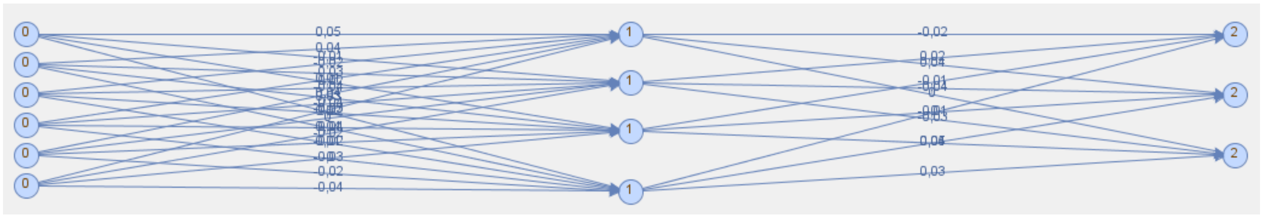


Figure 1: *A feed forward neural network. The nodes marked '0' make up the input layer. These nodes propgatate to nodes marked '1'; nodes on the hidden layer. Finally, these nodes propagate to nodes marked '3', which is the output layer.*

tween some or all output nodes to some input nodes and propagating the output values when a new input pattern is presented.

This means that for each new pattern, in addition to the input pattern some information about the previous pattern(s) is also presented to the network. In the words of Ilya: [4]

> "The RNN's high dimensional hidden state and nonlinear evolution endow it with great expressive power, enabling the hidden state of the RNN to integrate information over many timesteps and use it to make accurate predictions. Even if the non-linearity used by each unit is quite simple, iterating it over time leads to very rich dynamics."

III. Methods

To test the performance of T2P mapping with neural networks, two tools have been developed and made available on GitHub.[1] The tools consist of an encoding GUI and a training GUI.

The encoding GUI allows the user to convert input-output maps of text characters to maps of activation values that can be read by neural networks. The tool also allows the user to randomly split the resulting map into a training set and a testing set.

The training GUI can load these maps and construct a new neural network. Using one of multiple training algorithms, the network can then be trained and tested.

The tools were designed to be easily extensible so that they can be made to work with different data formats, encodings and network (training) types.

3.0.3 Learning rate and momentum

Learning rate and momentum are two important variables in training the network. The both numbers determine the rate with which neuron weights are nudged to their desired values. The learning rule for updating these is the following formula, called the delta rule [1]:

$$\Delta w_{j,i} = a(t_j - y_j)g'(h_j)x_i \qquad (1)$$

where
$\Delta w_{j,i}$ is the addition made to the weight from neuron i to neuron j
a is the learning rate
$g(x)$ is the neuron's activation function
t_j is the desired output for neuron j
h_j is the weighted sum of neuron j's input
y_j is the actual output of neuron j
x_i is the ith firing value from neuron i

A momentum is added so that the networks don't fall into local minima. The momentum adds to the delta rule as a fraction of the previous weight increment.

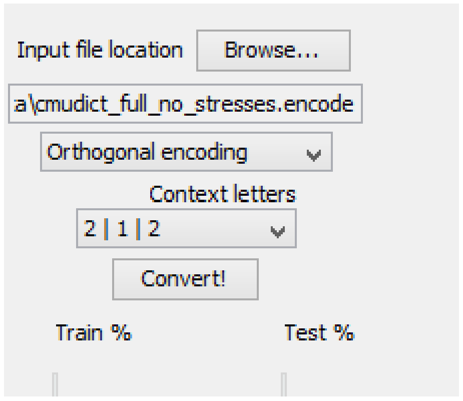
$$\Delta w_{j,i}(t) = \Delta w_{j,i} + m(\Delta w_{j,i}(t-1)) \qquad (2)$$

where
m is the momentum
t is the current training iteration
$m(\Delta w_{j,i}(t-1)) = 0$ for $t < 1$ (there is no momentum if there has been no previous training iteration)

3.1 Database

Testing and training was done using the CMU Pronouncing Dictionary[2] , an ASCII-encoded (Arpabet) phonetic pronunciation dictionary for American English which contains some 125.000 words (about 110.000 words after pruning). The dictionary was pre-processed and pruned for the purpose of this experiment.

[1] https://github.com/digitalheir/Neural-Network-GUI
[2] http://www.speech.cs.cmu.edu/cgi-bin/cmudict



a\cmudict_full_no_stresses.encode

Orthogonal encoding

Context letters

2 | 1 | 2

Figure 2: *Detail of the encoding tool*

3.1.1 Database pre-processing

Because the words-to-phonemes mapping of the CMU dictionary is not aligned (e.g., it does not show which letter maps to which phoneme), some pre-processing of the dictionary was necessary. Word alignment is still an active area of research, but an approximation was made with a state-of-the-art open-source program called M2M aligner[3].

This program is designed to work with the CMU dictionary syntax, and consequently our encoding tool was designed to work with the output syntax of the M2M aligner.

However, the M2M aligner ended up with 575 symbol types, while the CMU dictionary uses only about 100 (consisting of alphabetic letters, numbers, and phonemes).

The reason why the M2M aligner outputs so many different types is because the M2M aligner will merge two letters into a single type if they account for one phoneme. Conversely, if a single letter accounts for two phonemes, the phoneme types will be merged.

A word that demonstrates this last case is 'nixes', where 'x' accounts for both a 'K' and an 'S' phoneme. This word maps to the phonetic translation:

N IH K S IH Z

The M2M aligner aligns the pair in the following manner:

N|I|X|E|S| N|IH|K:S|IH|Z|

Because it is desirable to have the least amount of types possible in our neural network, only one letter was allowed to map to one phoneme. To fill the resulting gaps, null-phonemes and null-letters are inserted. The example above would generate:

N|I|X|_|E|S| N|IH|K|S|IH|Z|

Because we employ a number of context letters when training and testing the neural network, inserting null-letters should not pose a big problem, although there is some loss of information.

Note that the insertion of null letters is a choice made to keep the experiment simple. The challenge of mapping one input letter to multiple phonemes persists for real-world text, where information about the number of phonemes a letter maps to is not known a priori.

After this, lexical stress information was removed from the phonemes. For example, 'OW', 'OW0', 'OW1' and 'OW2' were all converted to

[3] http://code.google.com/p/m2m-aligner/

'OW'. This was done to keep the experiments as simple as possible, as well as to reduce the number of symbol types in the phoneme dictionary.

To further reduce the number of types, all characters were converted to uppercase. Because the input only consists of letters characters, and the output consists only of phoneme characters, there is no risk of mis-interpreting a character.

After pre-processing, the dictionary was pruned from very rare letters letter-to-phoneme pairs. 'Very rare' was rather arbitrarily defined as occurring less than 40 times in the entire dictionary. Rare letters include the hyphen and the character 2 in the word 'C-2'.

3.2 Encoding

For encoding dictionaries of aligned input-output strings into neural patterns, a GUI was made. (See Figure 2.)

The method of encoding text characters to neural patterns is of consiberable importance. Because most networks require a fixed number of input values and return a fixed number of output values, it is impossible or impractical to map entire words to their complete phonetic translations.

Furthermore, a holistic approach might hinder the network to abstract the phonological rules.

Because of these reasons, the strings were encoded as single letter-to-phoneme pairs. Letters were accompanied by a fixed number of context letters to the left and to the right.

Also important is the method of encoding character types into activation patterns. Two algorithms have been implemented to do this: orthogonal and non-orthogonal encoding. Both encodings translate a token into a binary number, corresponding to an activation pattern, but the orthogonal encoding allows only one '1' in the binary number. For an alphabet $\{aa, bb, cc\}$, an example encoding could be as follows:

Orthogonal: $\{aa \rightarrow 001, bb \rightarrow 010, cc \rightarrow 100\}$

Non-orthogonal: $\{aa \rightarrow 01, bb \rightarrow 10, cc \rightarrow 11\}$

Orthogonal encoding generally produces better results than non-orthogonal encoding, except when the number of weights is unrealistically low. [1, p. 74] Because of this, in the experiment only orthogonal encodings are used.

3.3 Training and Testing

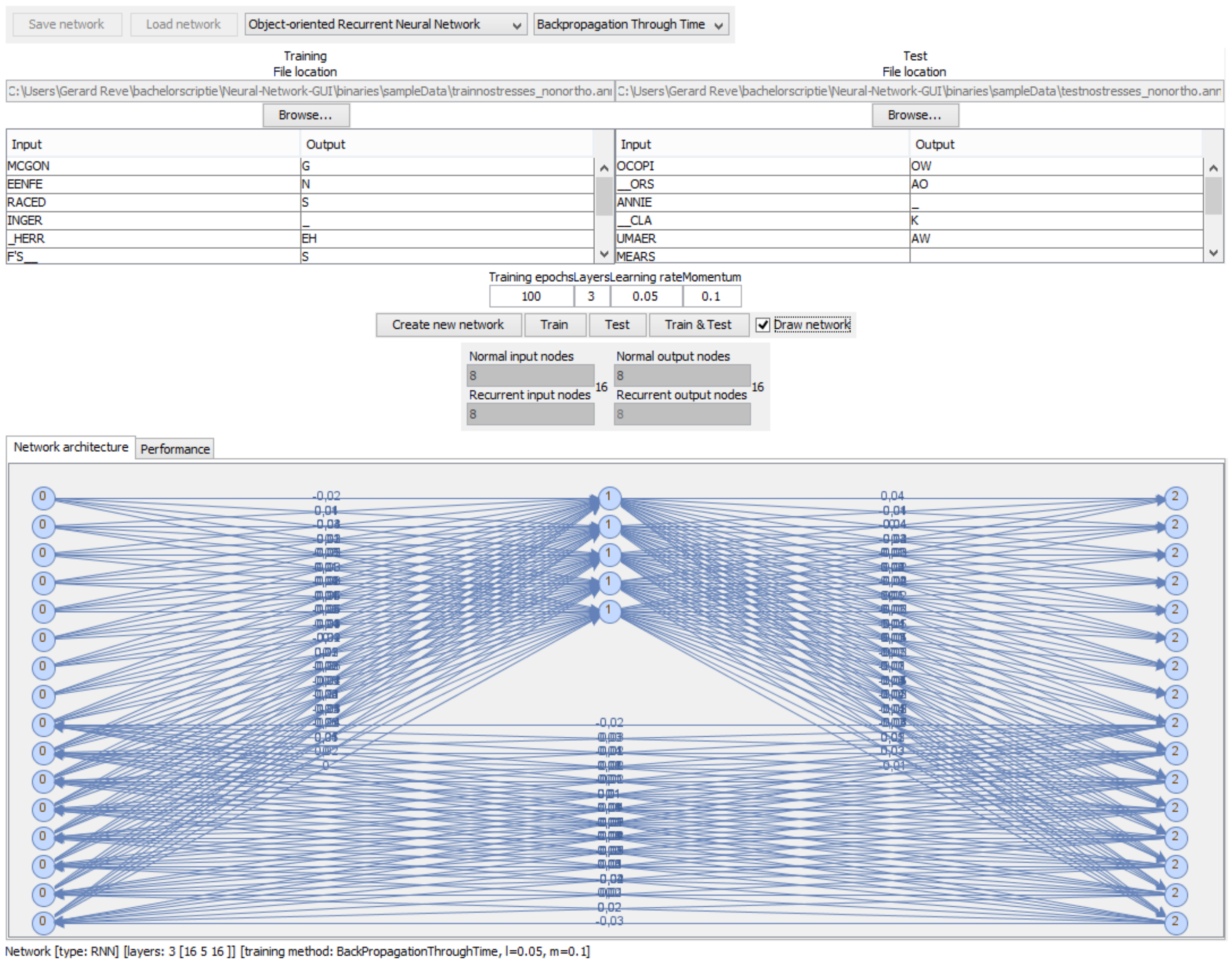
To train and test the neural network, a GUI was made with two implementations of the feed forward neural network. (See Figure 3.) One of the implementations was object-oriented (for clarity) and the other primitives-based (for performance). The implemented training function is backpropagation. Networks nodes were instantiated with the sigmoid activation function.

Interfaces for RNNs and a rudimentary implementation were also made, together with the backpropagation-through-time algorithm, but these were not tested well enough to produce reliable results for the experiment.

The network GUI allows the user to load data sets for training and testing that are generated with the encoding tool. It allows the user to instantiate a new network and training function along with the ability to change important parameters such as learning rate. The GUI can render a graph representation of the network and keeps track of network performance over the test set. The test results can be exported to a CSV file.

For the experiment, a number of networks were tested against a number of training and testing sets. The testing conditions varied in:

- learning rate
- momentum



Object-oriented Recurrent Neural Network | Backpropagation Through Time

Training File location: C:\Users\Gerard Reve\bachelorscriptie\Neural-Network-GUI\binaries\sampleData\trainnostresses_nonortho.anı

Test File location: C:\Users\Gerard Reve\bachelorscriptie\Neural-Network-GUI\binaries\sampleData\testnostresses_nonortho.anr

Input	Output
MCGON	G
EENFE	N
RACED	S
INGER	_
_HERR	EH
F'S__	S

Input	Output
OCOPI	OW
__ORS	AO
ANNIE	_
__CLA	K
UMAER	AW
MEARS	

Training epochs	Layers	Learning rate	Momentum
100	3	0.05	0.1

[✓ Draw network]

Normal input nodes: 8
Recurrent input nodes: 8
16
Normal output nodes: 8
Recurrent output nodes: 8
16

Network [type: RNN] [layers: 3 [16 5 16]] [training method: BackPropagationThroughTime, l=0.05, m=0.1]

Figure 3: *Training & testing tool showing a recurrent neural network*

- number of context letters

Another important variable is the number of weights. This variable was held constant at 6000 weights for the purpose of this experiment. Dealing with different numbers of context letters means dealing with differing numbers of input nodes, which would alter the total number of weights in the network. Because of this, some care had to be taken to instantiate the network with a correct number of hidden nodes. Because the FFNNs that are used are fully connected, the number of nodes in the hidden layer can be calculated with the following equation:
$n * x + x * m = 6000$
Where n is the number of input neurons and m the number of output neurons. To get a feeling of how the number of weights influences the performance of a network, consult [1].

IV. Results and discussion

(Consult the appendix for more detailed results.)

The top performing networks score just under 80% accuracy (see Figure A.1). As would be expected, raising the number of context letters raises accuracy, up to a point of about 5 input letters. Both of these observations are corroborated by [1, p. 62]. Within the testing bounds, increasing the learning rate had a positive effect on overall network accuracy. Varying the momentum has had minimal effect, which suggests that there is little risk of local minima on the error minimization surface for this problem.

It is observed that most difficulty lies in converting vowels to their phonemes, as can be seen in Figure A.3. The vowel groups monophthongs and diphthongs respectively score 56% and 46% accuracy over the entire testing set. Compare this to nasals (97%), liquids (93%) and fricatives (86%). It is hypothesized that this is due to the ambiguous nature of vowels (e.g., the non-trivial distribution of possible vowels among input patterns), but such a statement merits more research.

So it would make sense, if one were to try and improve the overall network accuracy, to focus on improving accuracy on vowels.

It is also noted that lowering the learning rate tends to produces a more erratic-looking performance graph, the shape of which is consistent when varying momentum. See Figure A.2 for the four testing extremes (minimum and maximum for learning rate and momentum).

References

[1] Enikö Beatrice Bilcu, *Text-To-Phoneme-Mapping Using Neural Networks*. Tampere University of Technology, 2008. 1.1, 3.0.3, 3.2, 3.3, 4

[2] Balázs Csanád Csáji, *Approximation with Artificial Neural Networks*. Faculty of Sciences; Eötvös Loránd University. 2.1.1

[3] Stuart J. Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009.

[4] Ilya Sutskever, e.a., *Generating Text with Recurrent Neural Networks*. University of Toronto. 2.1.2

[5] Paul J. Werbos, *Backpropagation Through Time: What It Does and How to Do It*. Proceedings of the IEEE, Vol. 78, No. 10, 1990.

[6] Simon Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1998.

[7] Orhan Karaali, e.a., *Speech Synthesis with Neural Networks*. Motorola, Inc., 1996.

[8] Yue Yang, *Anti-Spam Filtering Using Neural Networks and Baysian Classifiers*. North Florida Univ., 2007. 2.1

[9] Tom Mitchell, *Machine Learning*. McGraw Hill, 1997. 2.1

I. Appendix A: Figures

For the raw result data, which is a more detailed list of results, consult the generated result files in the GitHub repository[4]. This data contains the scores for all individual letter-to-phoneme mappings, for each tested network condition.

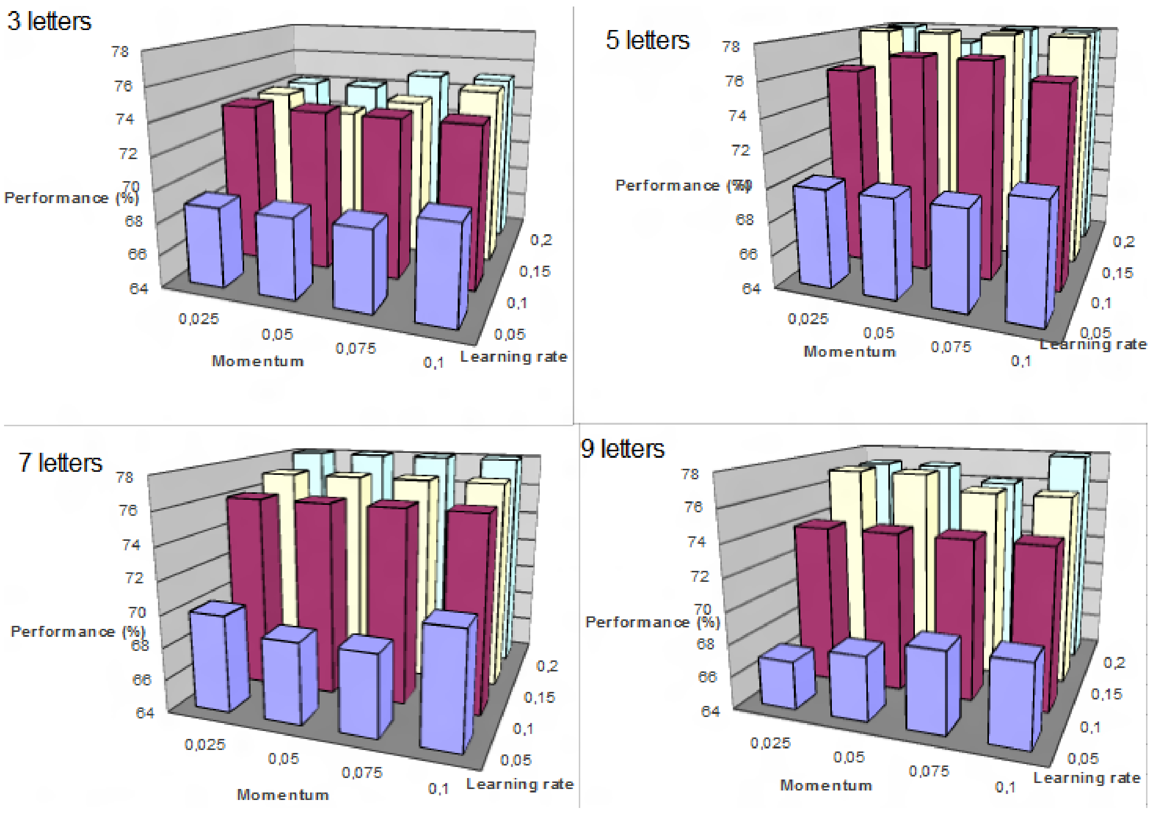


Figure A.1: *Bar charts showing performance as a function of momentum and learning rate, for various amounts of input letters*

[4]https://github.com/digitalheir/Neural-Network-GUI/tree/master/report/results

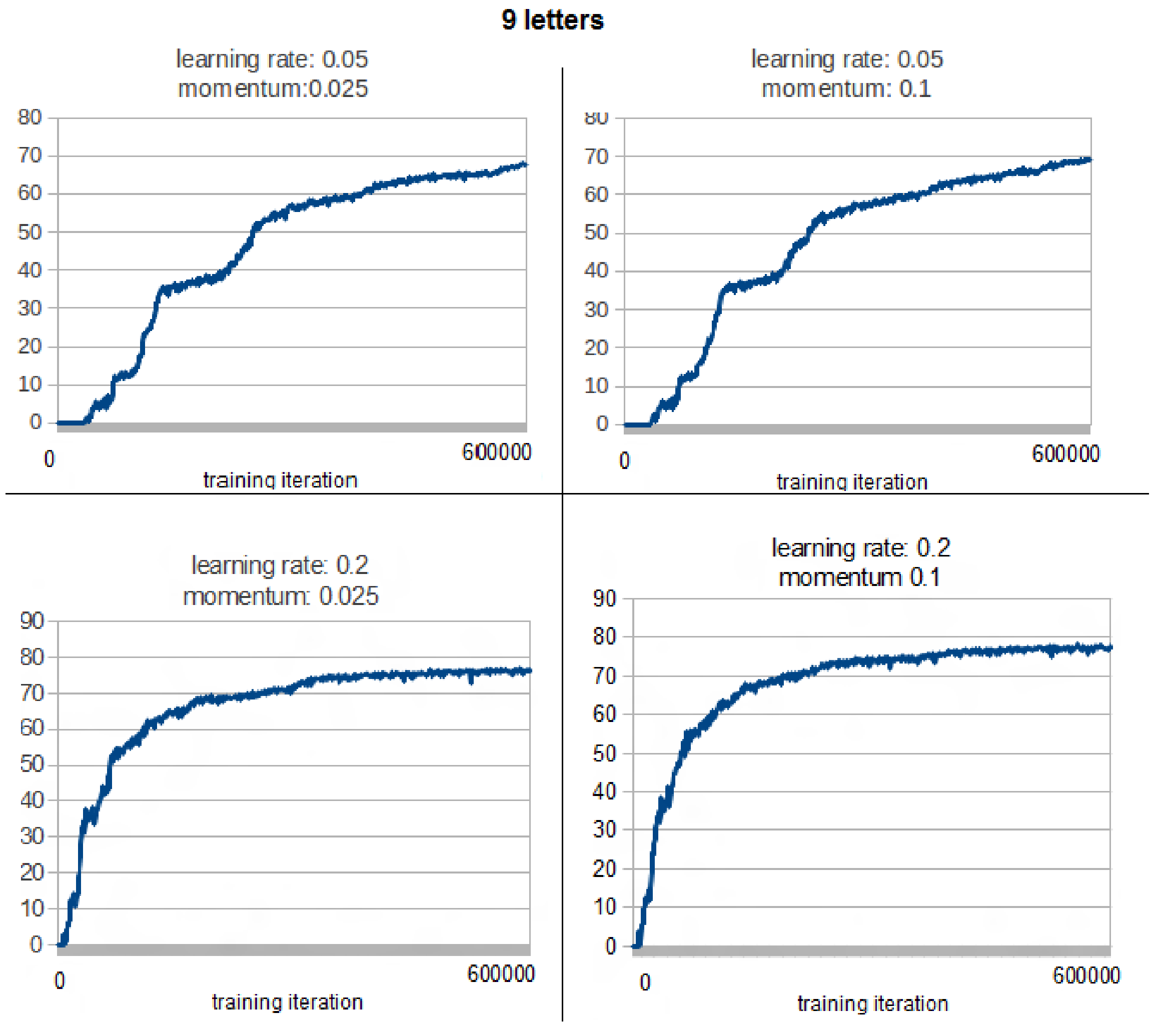
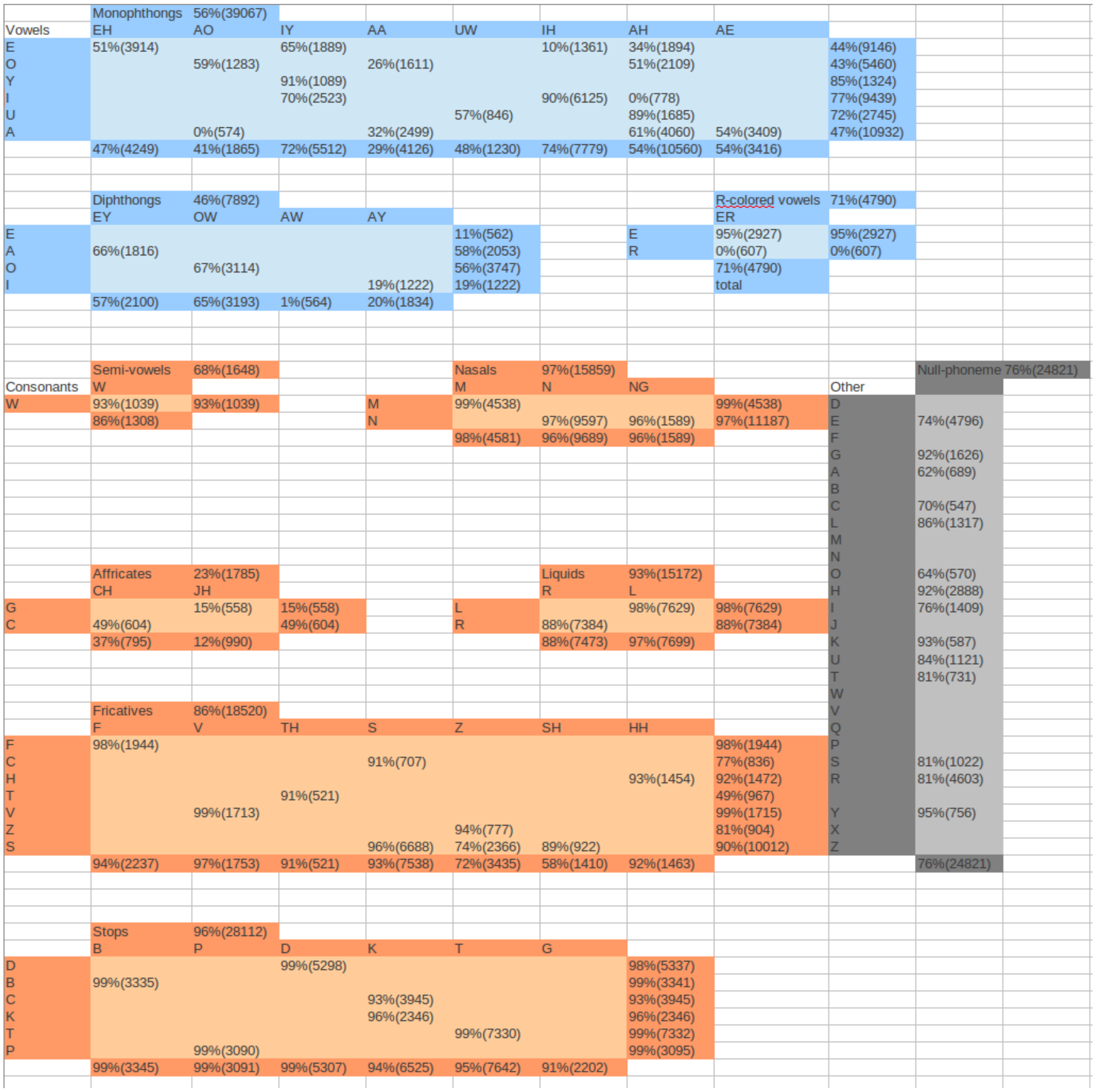


Figure A.2: *Performance graphs for various learning rates and momentums, with 9 input letters*



	Monophthongs	56%(39067)									
Vowels	EH	AO	IY	AA	UW	IH	AH	AE			
E	51%(3914)		65%(1889)			10%(1361)	34%(1894)		44%(9146)		
O		59%(1283)		26%(1611)			51%(2109)		43%(5460)		
Y			91%(1089)						85%(1324)		
I			70%(2523)			90%(6125)	0%(778)		77%(9439)		
U					57%(846)		89%(1685)		72%(2745)		
A		0%(574)		32%(2499)			61%(4060)	54%(3409)	47%(10932)		
	47%(4249)	41%(1865)	72%(5512)	29%(4126)	48%(1230)	74%(7779)	54%(10560)	54%(3416)			
	Diphthongs	46%(7892)						R-colored vowels	71%(4790)		
	EY	OW	AW	AY				ER			
E					11%(562)		E	95%(2927)	95%(2927)		
A	66%(1816)				58%(2053)		R	0%(607)	0%(607)		
O		67%(3114)			56%(3747)			71%(4790)			
I				19%(1222)	19%(1222)			total			
	57%(2100)	65%(3193)	1%(564)	20%(1834)							
	Semi-vowels	68%(1648)			Nasals	97%(15859)				Null-phoneme	76%(24821)
Consonants	W				M	N	NG		Other		
W	93%(1039)	93%(1039)		M	99%(4538)			99%(4538)	D		
	86%(1308)			N		97%(9597)	96%(1589)	97%(11187)	E	74%(4796)	
					98%(4581)	96%(9689)	96%(1589)		F		
									G	92%(1626)	
									A	62%(689)	
									B		
									C	70%(547)	
									L	86%(1317)	
									M		
									N		
	Affricates	23%(1785)				Liquids	93%(15172)		O	64%(570)	
	CH	JH				R	L		H	92%(2888)	
G		15%(558)	15%(558)		L		98%(7629)	98%(7629)	I	76%(1409)	
C	49%(604)		49%(604)		R	88%(7384)		88%(7384)	J		
	37%(795)	12%(990)				88%(7473)	97%(7699)		K	93%(587)	
									U	84%(1121)	
									T	81%(731)	
									W		
	Fricatives	86%(18520)							V		
	F	V	TH	S	Z	SH	HH		Q		
F	98%(1944)							98%(1944)	P		
C				91%(707)				77%(836)	S	81%(1022)	
H							93%(1454)	92%(1472)	R	81%(4603)	
T			91%(521)					49%(967)			
V		99%(1713)						99%(1715)	Y	95%(756)	
Z					94%(777)			81%(904)	X		
S				96%(6688)	74%(2366)	89%(922)		90%(10012)	Z		
	94%(2237)	97%(1753)	91%(521)	93%(7538)	72%(3435)	58%(1410)	92%(1463)			76%(24821)	
	Stops	96%(28112)									
	B	P	D	K	T	G					
D			99%(5298)				98%(5337)				
B	99%(3335)						99%(3341)				
C				93%(3945)			93%(3945)				
K				96%(2346)			96%(2346)				
T					99%(7330)		99%(7332)				
P		99%(3090)					99%(3095)				
	99%(3345)	99%(3091)	99%(5307)	94%(6525)	95%(7642)	91%(2202)					

Figure A.3: *Table showing accuracy on single letters and phonemes, divided into phoneme groups. Conversions that occurred less than 500 times in the test set were deleted from the table, so totals do not necessarily add up.*