# Optimizing Neural Networks for Usage on English Word Pronunciations

**Pablo Alvarez** Akamai Technologies
**William Ang** Boston University
**Randall Schwager** Boston University
**Tommy Yang** Boston University

## Abstract

We use a neural network to convert graphemes to phonemes, and replicate the performance of the well known NETTALK network. We then attempt to improve generalization performance with a variety of changes to the network structure, model, and parameters. The main significant improvements were seen with the choice of a larger and more representative training set, and by changing the output to be phonemes rather than articulatory features. Our findings suggest there are limits to the ability of neural nets to generalize English pronunciation.

## 1 Introduction

Pronouncing written English (grapheme to phoneme conversion) is a non-trivial problem in machine learning. An early success in this field was achieved by Sejnowski and Rosenberg in 1988[10], who showed that a two-layer feedforward neural network trained by backpropagation of errors could learn to pronounce phonemes. The network was constructed so that each input pattern was a binary-coded vector representing the seven letters around the letter of interest (that is, seven binary vectors of length 27 with only one element set to 1). The output was two binary-coded patterns representing (i) the articulatory features of the output phoneme (e.g Voiced=1, Palatal=1, Glide=1 for y as in yet, with the other 26 features set to 0); and (ii) the stress/syllabic features of the output phoneme (e.g. Secondary=1, Stress=1, Strong=1, with others set to 0). The network was trained by backpropagation of errors and was shown to perform very well (97%) on the corpus of 1000 words it was trained on, and generalize usefully (77% correct) to a larger corpus of 22000 words.

This work was one of the founding pieces of the interest in neural networks, and several variants or improvements on it have been proposed (e.g. Wolpert, 1990; Damper, 2001)[12][6]. In addition, the Nettalk dataset is now publicly available and has been used as a benchmark or training set for a number of different machine learning paradigms, including non-linear backpropagation (Hertz et al., 1997), tree-structure approaches (Andersen et al, 1996)[2] and parallel or concurrent computing (Andonie et al., 2006; Abbas, 2004)[3][1].

We will use this dataset to explore some of the properties of backpropagation and neural networks. Our goal is to first replicate the original results, and then alter the properties of the neural network and measure changes in performance (both on the training set and generalization performance) and training time needed. We will try the following changes to the original network:

- Evaluate the efficacy of different learning rates
- Implement weight decay, and investigate whether generalization performance improves
- Enforce a sparsity constraint on the hidden layer, with varying parameters
- Evaluate the extent to which ignoring small errors, as in the paper, improves generalization
- Compare the effect of different training sets, and training set sizes
- Choosing the correct phoneme as the one with the lowest mean squared error, instead of the one with the smallest angle to the feature vector

- Removal of homographs from the data set to see to what extent they add noise to the data
- Addition of a third layer, combined with changes in the size of the second layer
- Combine the outputs of several implementations of the network
- Evaluate the ability of a version of this network to directly predict the output phonemes instead of their component articulatory features

## 2 Methods

### 2.1 Initial replication

To replicate the findings of Sejnowski and Rosenberg and to evaluate the effect of changes to network structure, we simulated a neural network as specified in [10]. Neural networks were implemented in Python using the PyBrain library [11].

Our implementation was trained using the dictionary and list of 1000 most common words given in the UCI machine learning repository [4]. As specified in [10], the network was trained to output articulatory features. The articulatory features were then translated to phonemes to measure the network's correctness. Since neither [10] nor [4] provided articulatory features corresponding to the French 'oi' phoneme, 21 words containing such a phoneme were discarded from the training and testing datasets.

### 2.2 Measuring performance

We measured the ability of our network to generalize its activity to previously unobserved words by testing it on the subset of the 20,000-word dictionary that had not been used for training. Initially, we used the same training set described in the paper (most common 1000 words). For later experiments, the 20000-word dictionary was randomly split into two sets of approximately equal size; the network was trained on the first set and tested on the second set. We defined accuracy as the ratio of exact matches between the network's pronunciation and the testing set's pronunciation to the total number of words in the testing set. For each of the techniques used to improve generalization performance, we plotted the network's accuracy against the number of words presented from the training set to the network. Ideal performance of the network would be an accuracy of 1.00 at every point of evaluation. To measure our network's overall performance, we defined a second measure of performance: the sum of the accuracy ratios across every point of evaluation divided by the number of performance evaluations. This pseudo-area-under-the-curve measure captures absolute performance as the network gets more exposure to the training set. Visualizations were made with Matplotlib [9].

### 2.3 Network improvements

A common method for regularizing error functions in neural networks is through addition of a weight decay term [7]. Beyond the other standard parameters of a neural network which are easily editable (i.e. momentum, learning rate, setting initial weights), we implemented several improvements in hope of better generalization performance.

#### 2.3.1 Sparsity

In an effort to improve generalization performance, we enforced sparsity among network connections. Encouraging sparsity in neural networks has significantly improved network generalization performance for many applications [5]. In our implementation, sparsity was encouraged by adding a regularization term to our error function. The sparsity term is calculated as the Kullback-Leibler divergence between the means two Bernoulli random variables: the average activation of a hidden unit and a parameter $\rho$, the desired average activation of any given hidden unit. The regularization term is calculated as

$$\beta \sum_{k=1}^{K} \rho log \frac{\rho}{\hat{\rho}_k} \; + \; (1-\rho) log \frac{(1-\rho)}{(1-\hat{\rho}_j)}, \tag{1}$$

where the sum operates over all units in hidden layer $K$ and $\hat{\rho}_k$ denotes the average activation of units in hidden layer $K$. $\beta$ is a parameter used to govern the relative importance of sparsity in the cost function. During training, as errors are backpropogated through the network, all errors back-propogating through sparse layers are added to the derivative of the above sparsity penalty, calculated as

$$\beta\,(-\frac{\rho}{\hat{\rho}_k} + \frac{1-\rho}{1-\hat{\rho}_k}). \tag{2}$$

#### 2.3.2 Combining multiple networks

In another attempt to increase performance, we combined the results of several varying neural networks into another master network using a multistage approach [13]. We trained neural networks of hidden unit sizes 80, 90, 100, and 120 (similar to our original networks) on several equal randomized subsets of our training data. After each network had been trained on one word, we took the outputs of each of them and combined them into one large vector. This became the input for a larger master network with input size several times larger, and hidden unit count several times larger as well to account for the increased input data. We then trained the master network on all the words that the sub-networks had trained on by taking their output and input it into the master network. Our hope in doing this was that by combining the results of varying neural networks, trained on different subsets of the data, the combined neural network might be able to detect deeper patterns that one network alone could not detect.

### 2.4 Choosing the best matching phoneme

The output of the network is a real-valued vector with values in the [0,1] range that represent a set of articulatory features and need to be mapped to a phoneme. The method used by Sejnowsky and Rosenberg involved finding the phoneme whose vector of articulatory features was closest in angle to the network output. Another option is to find the phoneme with the smallest squared-error measure when compared to the network output. Most experiments were run with the first method which is less computationally intensive, but we also tested the second method.

### 2.5 Direct prediction of phonemes

We also adapted the network to predict phonemes directly, skipping the step of mapping articulatory features to phonemes. In that case, it can be shown that both methods (smallest angle and smallest mean-squared error) are mathematically equivalent, since only one element of the phoneme output vectors we are comparing against is non-zero.

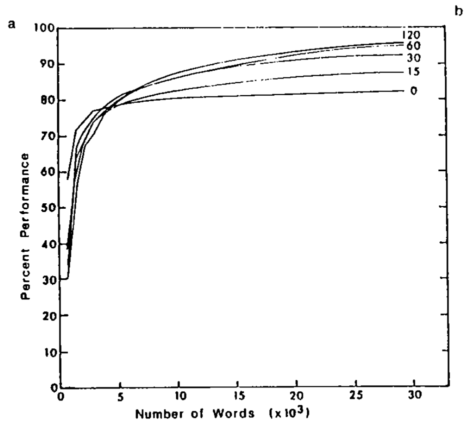
### 2.6 Data and code

All code, data, and instructions for replicating our findings can be found on Github at https://github.com/gnrhxni/CS542.

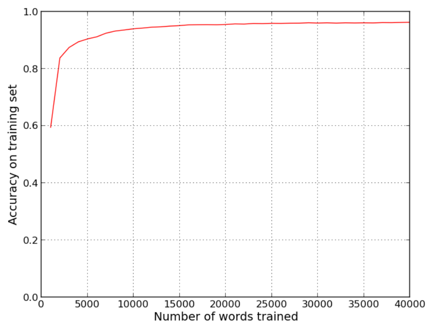
## 3 Results

### 3.1 Initial results

Our initial implementation of the network imitated the network in the Sejnowski and Rosenberg paper. At first, we were unable to replicate the performance on the training set. It turns out that this discrepancy was due to a difference in testing methods. In the original paper a word would be tested immediately after the network had been trained on that word. By using this technique we were able to replicate the paper's performance (Figure 1).



(a) Performance of original network from Sejnowski and Rosenberg paper



(b) Our neural network's performance

Figure 1: Comparison of standard neural networks between our implementation and Sejnowski's, with the testing after training modification added

## 3.2 Learning rate

We varied learning rates between 0.2 and 2.8. A learning rate of 2.0 had the greatest performance among all the learning rates. As learning rates deviated farther away from 2.0 the performance appears to decrease (Figure 2).

## 3.3 Initial weights

Changing the range in which initial weights were randomized resulted in a final performance similar or worse than the original neural network (Figure 3).

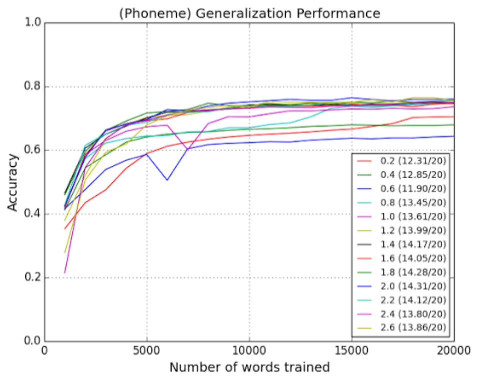


Figure 2: Performance of network when learning rate varied

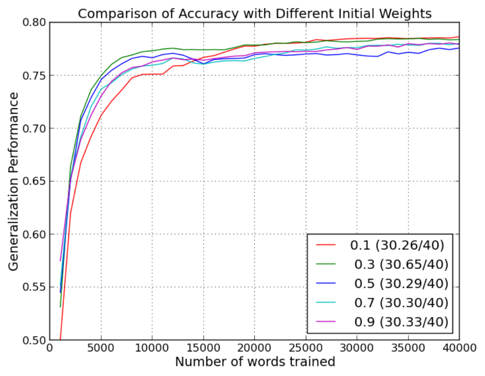


Figure 3: Performance of network when initial weight range varied

## 3.4 Sparsity

Enforcing sparsity on network connections seemed to have a negative effect on the training of the network. While having a very small sparsity parameter did not appear to have a negative effect on the network. Large sparsity parameters, a beta of .0010 or higher, hurt the performance of the network quite significantly (Figure 4).

## 3.5 Training set size

The most successful method of improving performance was to change the training data. Instead of training on the 1000 most common words, training on 10,000 random words improved the learning of the network significantly (Figure 5). Note that all results above asymptote around 78%, whereas all results after this have a subsequent jump in performance to about 85%, due to our change in the training data.

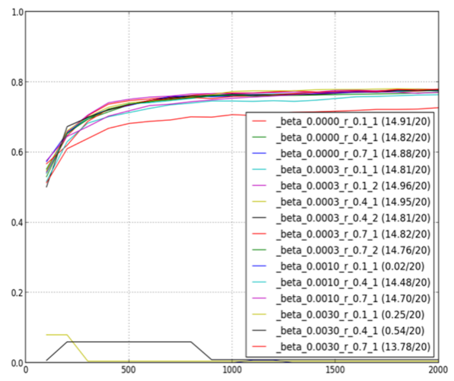


Figure 4: Performance of network when sparsity varied

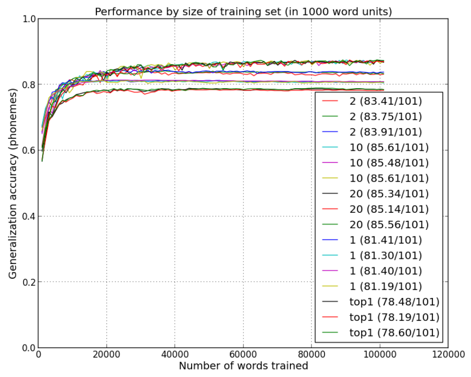


Figure 5: Performance of varying training sets

## 3.6 Momentum

Changing momentum seemed to result in similar or worse performance than the original neural network (Figure 6).

## 3.7 Weight decay

We varied the weight decay between 0 and 5. Figure 3 shows that the best performance is achieved when the weight decay is zero. The larger the weight decay the worse the learning curve becomes (Figure 7).

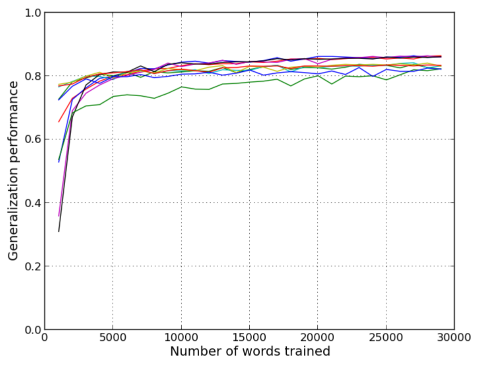


Figure 6: Performance of network when momentum varied

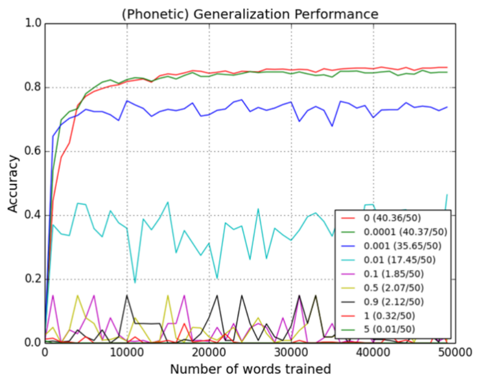
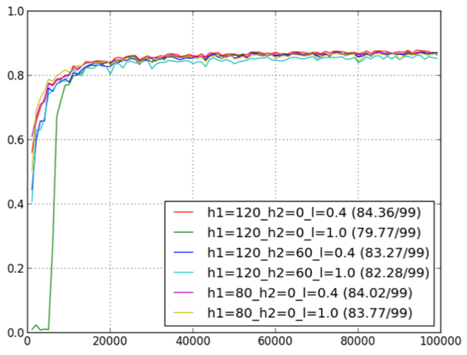


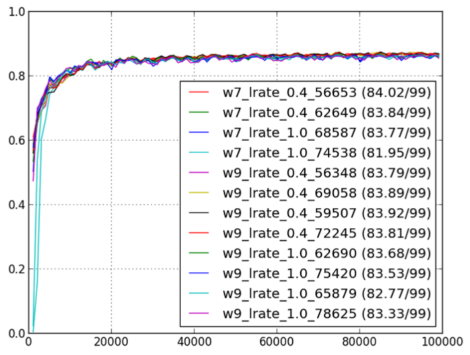
Figure 7: Performance of network when weight decay varied

### 3.8 Network Size

Changing the configuration of the network seemed to have little effect on the results. Increasing the size of the hidden layer to 120 nodes and/or adding an additional hidden layer of 60 nodes did not appear to affect the network in any significant way. Increasing the window size around the letter to nine instead of 7 also seemed to have no effect on the network's performance (Figure 8).



(a) Performance of network when hidden layer size varied

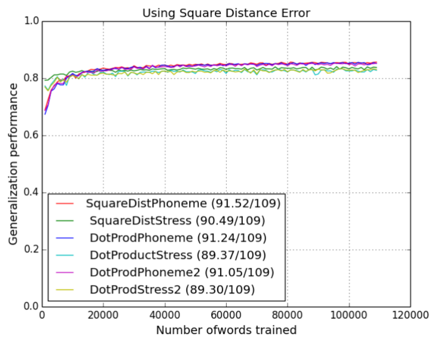


(b) Performance of network when window size varied

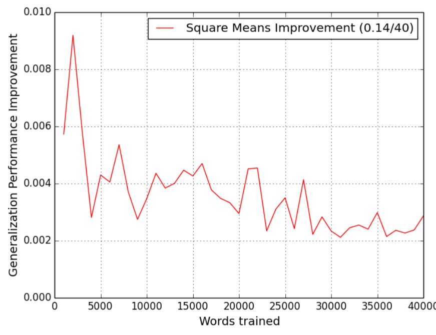
Figure 8: Performance when network size varied

### 3.9 Using Mean Squared Error

Using the mean squared method of determining output of the network instead of the minimum angle method showed a small but consistent improvement in the performance of the network. When compared using the same network, the mean squared method performed better at every instance (Figure 9).



(a) Performance of dot product calculated network vs a square means calculated one



(b) Improvement of networks using square means

Figure 9: Performance of mean squared error

### 3.10 Combining networks

Combining multiple networks resulted in similar performance to our standard single neural network (Figure 10).

### 3.11 Removal of homographs

Homographs are words with identical spelling but different pronunciation. It is obviously impossible for the same network to predict both homographs correctly, and moreover their presence in the training set could confuse the network and disrupt performance. When homographs (about 0.4% of the sample) were removed from the data, the network was found to perform better by about 1% (Figure 5).

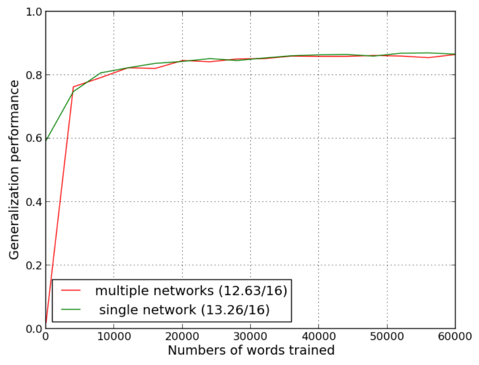


Figure 10: Performance of combined multiple networks vs a single neural network

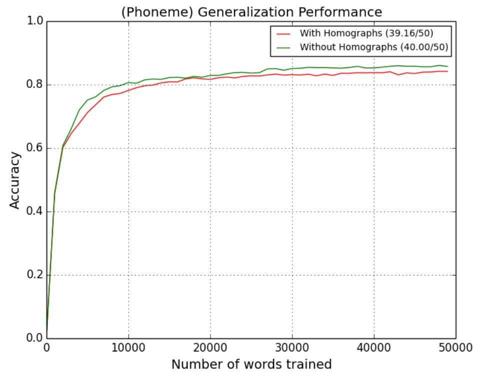
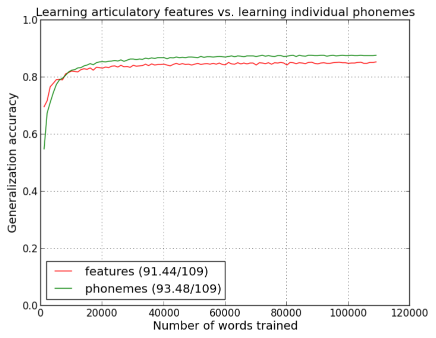


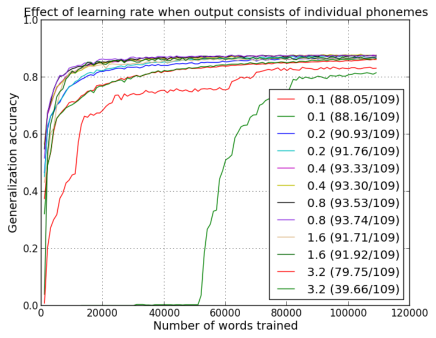
Figure 11: Performance of network without homographs vs with.

### 3.12 Directly predicting phonemes vs features

Changing the output layer and training outputs to use individual phonemes instead of combinations of articulatory features induced a 2-3% increase in accuracy (Figure 12).



(a) Performance phoneme-only network vs the standard network



(b) Varying learning rate when directly predicting phonemes

Figure 12: Performance of directly predicting phonemes vs articulation features

## 4 Discussion

We used the Pybrain library and custom code to construct a neural network with one hidden layer that was capable of predicting phonemes based on single words from a dictionary. We were able to replicate the results in the original Rosenberg and Sejnowski paper (1988), with training set performance reaching 95%. However, we found that the way the training performance was measured was somewhat misleading: each individual word was tested directly after it had been trained. This means performance on that word had just been optimized and contained less noise from other words in the training set than would be expected if we picked a random training set word to be tested. Accordingly, for the rest of our experiments we adopted a split train-test model, where after a certain number of words were trained we tested the network on a separate set of words (usually the complement of the training set). With this more unbiased model, we found that the generalization performance of the network was somewhat disappointing, on the order of 78%.

Manipulation of learning rates, weight decay, momentum, or the addition of a sparsity constraint did not improve performance. Many different learning rates produced good results, but any significant level of weight decay or sparsity enforcement degraded network performance. In fact, we had to modify the sparsity calculations to avoid getting too close to 0 and incurring approximation errors. Consideration of the nature of the inputs and outputs helps to understand this: they are in fact very sparse to begin with, so the network naturally leans towards sparsity, and forcing it in that direction is not helpful.

One interesting phenomenon was the importance of initial conditions: it can be seen clearly in the experiment where weights were initially normalized to different ranges, as well as in one of the large network cases in Figure 7, where one of the networks hovers around 5% correct for thousands of training words until it finally starts to learn something. This finding is a useful reminder that the energy surfaces of neural nets can be very complex, and that the network can find itself in a part of the space where the training has very little effect.

One manipulation that clearly did provide an improvement in performance was increasing the size of the training set up to about half the items, but not beyond that. This can be understood by considering the many possible 7-letter combinations that the network needs to learn, and the fact that 1000 words are clearly not enough to cover them. Interestingly, it turned out that the 1000 most common words initially chosen by the authors as a training set gave worse performance than other randomly chosen sets of 1000 words. Therefore, for most experiments we randomly split the data into two equal-sized training and test halves.

Another feature of the paper that appeared to not have the intended effect was ignoring small errors: any error $< 0.1$ was not backpropagated, in order to avoid overtraining. In our experiments we found that this change had no detectable effect on performance, or even on the distribution of weight sizes, suggesting that most of the weight changes are driven by relatively large errors.

But trying different hidden layer sizes and adding a hidden layer were also shown to have no effect on the asymptotic performance of the network.

Three other manipulations did improve network performance:

- The use of min(mean-squared error) instead of min(angle between vectors) to choose the best match, which gave a very small but consistent improvement of about 0.4%
- Removing all homonyms, which resulted in a performance improvement of approximately 1%
- Trying to directly predict phonemes rather than their component articulatory features, which improved performance by about 3%.

When considered together, the lack of improvement in most cases and the very small improvements observed in some of the experiments suggest that we are reaching the limit of a feedforward neural-net classifiers ability to correctly predict phonemes from graphemes when using only individual words. It is possible that we have reached the Bayes error, given the well-known unpredictability of English spelling, or simply that this particular type of classifier cannot find a sufficiently general solution because the energy surface has many similar local minima. To differentiate between those two hypotheses, it would be interesting to compare the errors produced by well-trained networks: if they are consistent it would support the hypothesis that we are at or near the Bayes limit and the training data does not contain enough information to do better, while if different networks show different patterns of errors it might mean that they are stuck in different local minima. While that analysis has not been performed, the fact that a combination of different networks does not perform any better than a single network supports the idea that the current solution is fairly close to the Bayes error.

## References

[1] H. M. Abbas et al., (2004) *Performance of the Alex AVX-2 MIMD architecture in learning rate NetTalk database* IEEE Trans. Neural Networks **15**:505-514.

[2] Andersen, 0., Kuhn, R., Lazarides, A., Dalsgaard, P., Haas, J., Noth, E. (1996) *Comparison of two tree-structured approaches for grapheme-to-phoneme conversion* , ICSLP 96. Proceedings, **3**Andersen, 0., Kuhn, R., Lazarides, A., Dalsgaard, P., Haas, J., Noth, E.,,"Comparison of two tree-structured approaches for grapheme-to-phoneme conversion", ICSLP 96. Proceedings, Volume 3, 3-6 Oct. 1996, pp.1700 - 1703.:1700 - 1703.

[3] (1, 2) Bache K, Lichman M. (2013) UCI Machine Learning Repository [http://archive.ics.uci.edu/ml].

[4] Bengio Y. (2009) *Learning Deep Architectures for AI.* Foundations and Trends in Machine Learning **2**(1): 1-127.

[5] Damper, Robert I. (2001) *Learning About Speech from Data: Beyond NETTalk*. Data Driven Techniques in Speech-Synthesis pp.1-25

[6] Hanson, S. J. & Pratt, L. (1990). *A comparison of different biases for minimal network construction with back-propagation.* In D. S. Touretzky (Ed.), Advances in Neural Information Processing Systems II (pp. 177185). San Mateo: Morgan Kaufmann.

[7] Hunter J. (2007) *Matplotlib: A 2D Graphics Environment*. Computer Science and Engineering **9**(3): 90-95.

[8] (1, 2, 3) Sejnowski T, Rosenberg C. (1987) *Parallel Networks that Learn to Pronounce English Text* Complex Systems **1**: 145-168.

[9] Shaul T, Bayer J, Wierstra D, Sun Y, Felder M, Sehnke F, Rckstie T, Schmidhuber J. (2010) *PyBrain*. Journal of Machine Learning Research **11**: 743-746.

[10] Wolpert, DH. (1990) *Constructing a generalizer superior to NETTalk via a mathematical theory of generalization.* Neural Networks **3**:445-452

[11] Yang, S., Browne, A. (2004), *Neural network ensembles: combining multiple models for enhanced performance using a multistage approach*, Expert Systems, Vol 21, Issue 5, 279-288