Phonemer: Grapheme-to-Phoneme Translation using a Neural Network

Cecilia Alm

Alexander Dean **David Stalnaker**

Abstract:
In this report we look at the effectiveness of a Neural network for Grapheme to Phoneme translation through the introduction of our application, Phonemer. We look at several methods of network tuning and feature generation we used, and their additions to the application. We ignore the complexity of character alignment and focus on the task of accurately prescribing the correct Phoneme per Grapheme. We hope to provide support for further research in the area of neural network classification techniques for natural language processing.

I. Introduction:
Phonemer, our application, is a tool for classifying input graphemes with output phonemes. That is, it produces the pronunciation of input words. For example the input “close” (as each letter is a grapheme) should produce the the output phonemes “k l O s #” (where “#” is the null or “silent” phoneme). Phonemer provides facilities for training its classifier, testing the same against ground truth, and providing the output phonemes for a given word. Notice phoneme stresses can be learned as well (as demonstrated through the capitalization of ‘O’).

The most visible use of grapheme-to-phoneme translation is in speech synthesis - a simplistic design for such a system is to simply translate to phonemes and then play recorded audio clips for each phoneme. More generally, it is also useful to compare words phonetically. For example, a system to compare if two differently-spelled names are pronounced the same would use grapheme-to-phoneme translation.

I.I. Description of System:
Phonemer can be locally split into two parts: preprocessing / feature selection and classification.

The feature set we used was fairly simple. Our features for each grapheme of a word were: a) the character before, b) the character afterward, c) the current character, d) the character 2 steps afterward, and e) what the class of the character is according to the Soundex algorithm. For the last feature, several characters such as ‘h’ and ‘y’ do not belong to any of the classes, so they got their own class. However, all vowels were grouped into one class.

Phonemer can be expanded and more features could be added with ease for later trial. Some examples for future review could be, the number of vowels before and after the current character, or even whether this character is a duplicate of the one previous such as with the character ‘t’ in the word ‘bottle’. Pronunciation classes apart from the Soundex Class can be used here as well, but we ignored these and instead focused on simple and widely used features to put emphasis on the abilities of neural networks.

The feature set generated for each character is then turned into a binary input vector for the neural network. Each entry represents a possibility for each feature class. So for feature c, the possible characters could be any character of the alphabet. For features a, b, and d there are more than 26 possibilities as we also have the front of word class (regex style ‘^’) and end of word class (also regex style ‘$’). So overall, we had an input vector of length 113.

Because of the large number of binary features involved, we experimented with Principal Component Analysis (PCA) to reduce the feature space. PCA finds mutually orthogonal vectors in the n-dimensional feature space that have the highest variance, so that a subset (those with the highest variance) of the vectors in the new feature space still contain most of the information.

The system used for classification is a neural network we implemented in Python using the Numpy mathematical computing library. The network is trained using the standard backpropagation algorithm with cross-validation. More specifically, each epoch of training runs backpropagation on a random subset of the training samples, and validates on a (different) random subset of the samples. When this validation error has not increased in 50 epochs, the network is considered sufficiently trained.

I.II. Dataset:

The data set we used was provided by Sproat with his work on the pmtools toolkit for word pronunciation modeling [1]. The data set has almost 50,000 entries which we split up into a dev set and a final set at an 80 and 20 percent randomized split.

With neural nets, the amount of training data severely affects its accuracy. So to avoid this testing error, we couldn’t test the validity of our application by just training on a percentage of the final set and then testing with the rest. Instead, for final review, we trained on all of dev and then tested using final. For incremental tests however, we trained using 80% of dev and then tested using the latter 20%.

The dataset was already aligned by character, so we could ignore the problem of character alignment, however the phoneme classes were fairly complex. They included stresses, as well as silent and added pronunciation. As mentioned above, silent pauses were indicated via ‘#’, but there was also special pronunciations like ‘_k_s_’ for the ‘x’ in ‘six’.

I.III. Usage:

Phonemer can save its trained neural network into a file to be used for later classifications. This aids in unit testing as training typically takes a substantial amount of time on larger data-sets. These trained networks can be saved and reused, so that comparisons can be made accurately.

Phonemer is also tunable, as the number of hidden nodes/layers and the number of PCA vectors used in the training are adjustable. The size of the training and testing set is also adjustable by using the data-set splitting functionality provided. As a neural network, Phonemer will train and test on all data provided, so splitting up testing and training data is necessary before passing it to Phonemer to train on.

II. Results:

As outlined in section I.II, we trained on 80% of the full dataset. Our goal was to keep track of the time it took to train as well as the accuracy of the classifications. Although since we were able to save the network for later use, the time it takes to train was of less concern and only useful for comparing tuning methods.

We found that training a network with 1 hidden layer of 100 nodes yielded a correct classification rate of 87.11% after 175 epochs.

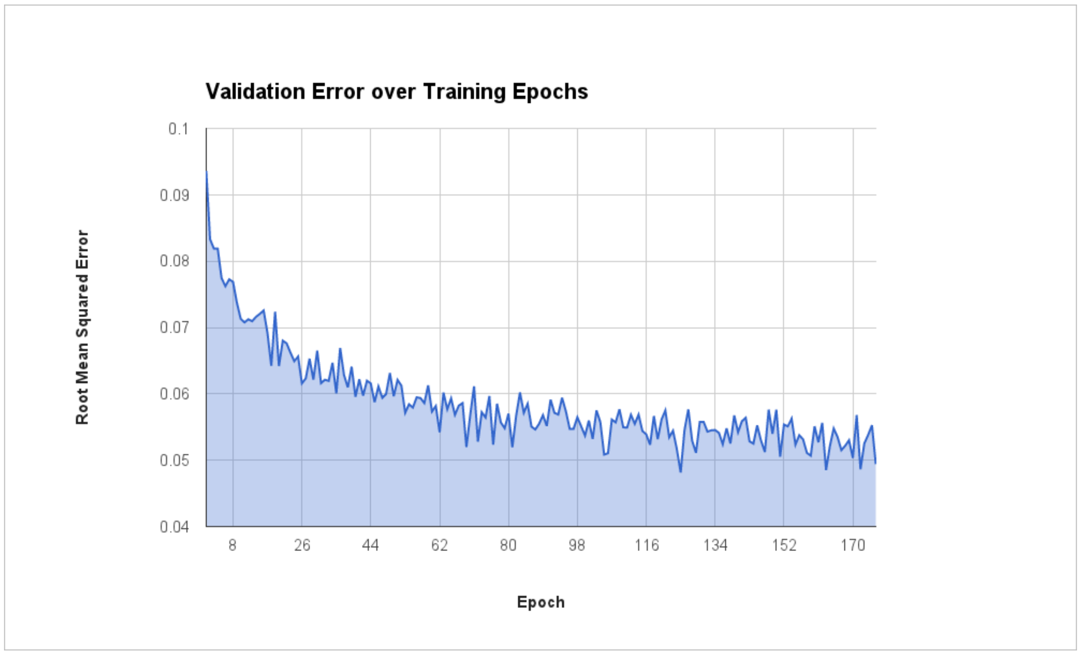

Figure 1. Validation Error

One negative result was that PCA was not particularly effective. With 100 hidden nodes on 1 layer and selecting only the top 55 pca vectors, we acquired a 82.83% accuracy after 189 epochs. We suspect this is because, while the feature space is large, there is little correlation between each feature. Most of the features for any given sample are 0, so most feature classes are all equally deterministic of the output phoneme. Typically PCA would need a scenario where only a few features are the primary determiners, phoneme classification doesn't seem to lend itself to this (at least for our feature classes).

Regrettably adding more layers didn't seem to improve the accuracy as one would think either. In fact due to our training thresholds we noticed a decrease in accuracy. Training of a multi-layered neural net would see smaller incremental improvements and we have the neural net cut off if there has been limited gains after so many epochs. We ended up meeting that threshold very quickly and ending up with near 12% accuracies after around 20 epochs with greater than 2 layers.

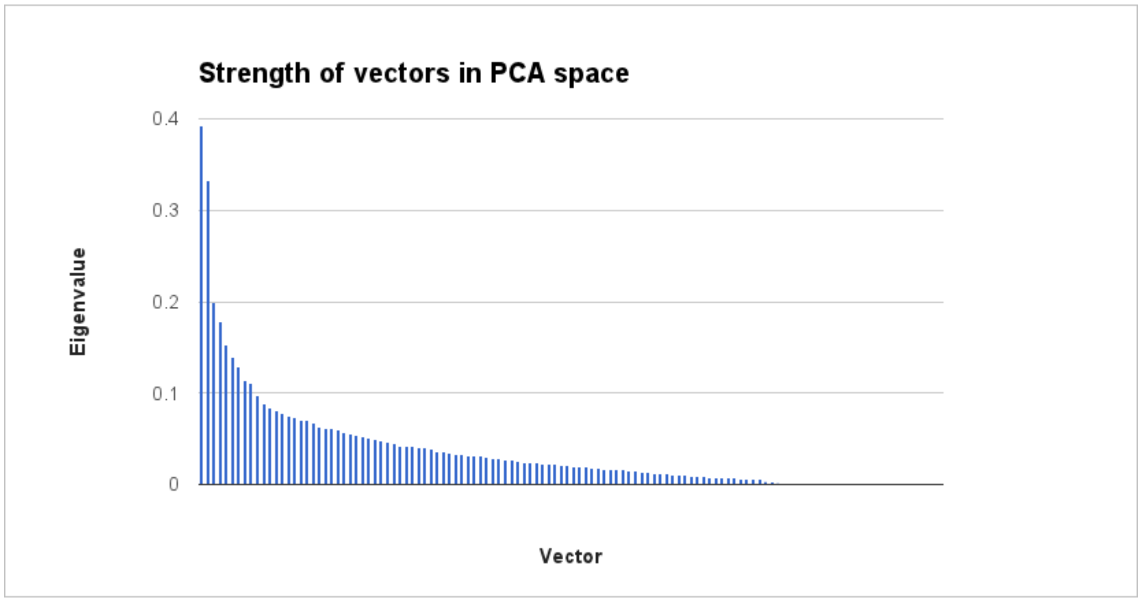

Figure 2. PCA Vector Strength

The only real improvements made were seen when adding more hidden nodes to a single layer. For 200 nodes on 1 hidden layer, after 182 epochs it was able to achieve 88.21% accuracy.

III. Conclusion:

From the data we presented above, we believe we are safe to say that Neural networks are valid and affordable models for NLP classifications. We believe there is more room for improvement and other avenues for further research. One possibility is in leveraging network training techniques for classifying phoneme alignments as well. If our neural network were trained to sufficient accuracy one could argue it could do grapheme-phoneme alignment on unknown input. Another possibility would be to more explicitly involve previous inputs and outputs, for example using a Time-Lagged Neural Network or a Markov Model.

IV. Development:

Much of the development in this project was done in tandem. David wrote the Neural Network that we used, while Alex wrote more of the preprocessing / feature selection code.

Bibliography:

1. Richard Sproat, "Pmtools: A Pronunciation Modeling Toolkit", Proceedings of the Fourth ISCA Tutorial and Research Workshop on Speech Synthesis, Blair Atholl, Scotland, 2001.