Grapheme to phoneme conversion for Dutch

Brian Hicks
Stanford University
bhicks2@stanford.edu

Enze Chen
Stanford University
enze@stanford.edu

Minjia Zhong
Stanford University
mzhong2@stanford.edu

Abstract

Grapheme-to-phoneme conversion (G2P) refers to the conversion of a written word to its pronunciation, which plays a crucial role in text-to-speech software and speech-to-speech machine translation. We present a G2P model for the Dutch language based on a long short-term memory recurrent neural network (LSTM). Specifically, we use a deep bi-directional LSTM (biLSTM) that takes into consideration the context of the graphemes before predicting the phoneme sequence. By using contextual clues, our model also eliminates the need for explicit alignment of the dataset. We achieve an accuracy of 66.2% completely correct phoneme predictions using a single biLSTM-based model, and 75.5% accuracy when averaging the predictions from an ensemble of such models. Our results demonstrate the power of LSTMs in the G2P task and facilitate further G2P research using unaligned datasets.

1. Introduction

One of the common problems in natural language processing (NLP) is determining the correct pronunciation of a written word. This process, known as grapheme-to-phoneme conversion (G2P), is most commonly used in technologies for speech synthesis, speech recognition, and sounds-like queries in textual databases. Due to the relevance of G2P conversion to everyday technologies, there exists a rich literature on automating the process. Initially, researchers attempted to create a program that searched for the corresponding phoneme pronunciation in a dictionary, but with more than hundreds of thousands of entries, this method proved to be inefficient and slow. Researchers then investigated rule-based conversion, but natural languages frequently exhibit exceptions or irregularities, so such efforts were fruitless [1].

Given these deficiencies, machine learning has become a popular application for automating G2P conversion. One of the most common techniques applied to G2P conversion is applying alignment-based neural networks [2]. The data is preprocessed so that each training input grapheme character aligns with a phoneme character before input into a neural network. Other algorithms like Naive Bayes create a probability distribution to predict the phoneme output [3].

Although the aforementioned techniques have reported relatively high accuracy rates, they are difficult to implement if one does not already have specialized linguistics knowledge. Alignment-based algorithms, for example, require an understanding of the G2P conversion to properly align the characters during preprocessing, which is complicated by the lack of a one-to-one correspondence of the graphemes and phonemes in a word. Consider, for example, the grapheme ⟨x⟩, which (in English) represents a two-phoneme sequence /ks/, or the phoneme /ʃ/ which is represented by the two-grapheme sequence ⟨sh⟩.

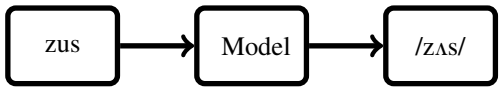

Figure 1. Our model takes Dutch graphemes (e.g. "zus") as inputs and it outputs a predicted phoneme string.

Given the domain expertise required for alignment-based methods, as well as the labor-intensive process of aligning grapheme and phoneme strings, we chose to construct a model that does not require explicit grapheme-phoneme alignment of the data. As shown in Figure 1, the inputs to our model are whole words written in Dutch, and our model seeks to predict the entire pronunciation string without any specified alignment between graphemes and phonemes. Such alignment-free models have been proposed for related machine translation tasks using sequence to sequence learning [4] and connectionist temporal classification [5], both of which leverage the power of recurrent neural networks to process sequential data. For the G2P task in particular, long short-term memory recurrent neural networks (LSTMs) have shown great promise because they can use contextual clues to perform alignment-free G2P conversion [6, 7], and this is the approach we take in this work.

2. Methodology

2.1. Data Preprocessing

Out dataset is derived from *ABN-Uitspraakgids* [8], a Dutch pronunciation dictionary available online from *Digitale bibliotek voor de Nederlandse letteren*.

Because the data extracted from the source was not well-formatted for machine learning, we performed a variety of preprocessing procedures on the data. This included the extraction of the text from the PDF pronunciation dictionary and subsequent cleaning of the data, including removal of incorrect data points and extraneous information.

After this preprocessing was completed, our final dataset consisted of 24,404 orthography-pronunciation pairs.

2.2. Model Construction

2.2.1 Baseline Model

Our initial baseline model is only slightly more complicated than basic one-to-one character mappings. Given a set of rules provided by [9], we designed a simple G2P program which maps characters deterministically to a pronunciation, using only a superficial knowledge of Dutch. In general, the only complexities of pronunciation considered in this baseline were vowel combinations and word-final consonant devoicing. Because this baseline approach mimics a brute-force approach that could be attempted by anyone with a basic pronunciation table, it provides a solid benchmark for our machine learning algorithm. We anticipate low prediction accuracy due to the difficulty of codifying G2P conversion, as well as the fact that it makes little attempt to take into account context when making predictions.

2.2.2 Long Short-term Memory Recurrent Neural Network

In developing machine learning models for this application, the lack of alignment between graphemes and phonemes was highly problematic. Many of the models that we could have considered, including many of the most commonly used machine learning algorithms such as support vector machines or logistic classification, would have required aligned data in order to accurately train the models. Given that our dataset was not pre-aligned, and the process of aligning it would have been incredibly labor-intensive (not to mention, theoretically disingenuous from a linguistic standpoint), this type of model was immediately disqualified as an option.

Common sense tells us that the pronunciation of any given grapheme is highly context dependent, since in different scenarios, the same letter can be pronounced in a variety of ways. Consider, for example, the English grapheme ⟨g⟩. Its pronunciation changes between /g/ and /ʤ/ depending on what letter follows it. Thus, any model that we use needs to be context-aware, in that any successful model must be able to consider the surrounding graphemes when making a prediction. This makes long-short-term-memory recurrent neural networks a natural choice as the foundation of our model, since they excel in processing context-sensitive data.

As such our own G2P model is based on a bidirection LSTM that was constructed using Google's TensorFlow library [10]. Our model draws from a recently demonstrated application of deep learning for alignment-free English G2P conversion from Rao *et al.* at Google [6].

Figure 2 below shows the overall architecture of the neural network (NN). The input to our NN is an $N \times D$ matrix, where N is a ***fixed*** number of characters, corresponding to the length of the longest word in the dataset, and D is the total number of graphemes that our model can process, including a null-character, which we used to pad words where the number of characters in the word is less than N. In our case $D = 41$, and $N = 27 + \delta$, where δ is the output-delay (described below). Each row i of the matrix is a one hot vector corresponding to the character at index i in the word.

The first layer of our NN is a bi-directional LSTM (biLSTM), with the forward cell containing 2048 units, and the backward cell containing 1024 units. This choice of first layer allows us to create contextual representations of each character of the input, taking into account both previous and upcoming characters. This step is key to the success of the model, as it allows the predictions to be made with full awareness of how each grapheme fits into the overall context of the word.

After the input has been processed by the biLSTM, we concatenate the forward and backward hidden states to form an $N \times 3072$ matrix. Then, depending on the output delay δ, we remove the first δ rows of the matrix, which forces the prediction model to have considered at least the first δ characters before it makes a prediction. Ultimately, this layer returns an $(N - \delta) \times 3072$ matrix that represent a context-aware and output-delayed encoding of our input sequence.

Each row of this new encoding is then used as the input to a simple feed-forward NN with one hidden layer of 1024 neurons. The activation function for the hidden layer is a simple ReLU function, and softmax is the evaluated on the output layer. For each row of the context-aware matrix passed into the network, a single phoneme (or null-phoneme) is returned, which can then be used to construct a string representation of the original input.

In order to train the model, we use batched Adam optimization, as provided by TensorFlow [10], of a modified cross-entropy loss-function. If b is the number of data points per batch and D is the number of possible classes (i.e.,

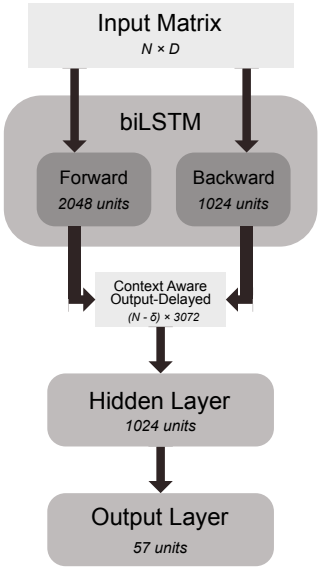

Figure 2. Our neural network architecture. We featurize a grapheme sequence of N characters using one-hot encodings of length D, and this is input into our biLSTM. We concatenate the outputs of the forward and backward memory units to form inputs of size $(N - \delta) \times 3072$ to another hidden layer before making the final prediction. We use a ReLU activation function for the hidden layer and the softmax function for the output layer.

phonemes), then our cross-entropy loss CE is given as

$$CE\left(y, \hat{y}\right) = -\frac{1}{b}\sum_{i=1}^{b}\sum_{j=1}^{N-\delta}\sum_{k=1}^{D} y_{ijk}\log \hat{y}_{ijk}$$

given that $y, \hat{y} \in \mathbb{R}^{b\times N\times D}$ and every y_{ij} is a one-hot vector.

2.3. Ensembles

During our training, we discovered that even for a fixed δ, the results of each model were highly variable. In order to account for this, we elected to construct an ensemble that incorporated several models. Ultimately, we ended up constructing four different varieties of ensemble, each of which combined the predictions of the constituent models in a different way.

Our first ensemble was a simple averaging ensemble. Formally, this was designed such that the final prediction $\hat{y}$ of the ensemble comprising p individual models was given by

$$\hat{y} = \frac{1}{p}\sum_{i=1}^{p}\hat{y}^{(i)}$$

where $\hat{y}^{(i)}$ is the prediction of the ith constituent model. We also created a slightly modified version of this ensemble that relied on a weighted average instead. In order to determine the weights, we evaluated each model in the ensemble on a previously unseen development set. If $\omega^{(i)}$ was the accuracy of model i, then the final prediction of the weighted average model was given by

$$\hat{y} = \frac{1}{\sum_{i=1}^{p}\omega^{(i)}}\sum_{j=1}^{p}\omega^{(i)}\hat{y}^{(i)}$$

In addition to the two averaging models, we also implemented two voting models. In these ensembles, rather than average the probabilities output by each model, each model in our ensemble makes a prediction about what each output character will be, which we record as that model's "vote". Once each model has had a chance to vote, we identify which character had the most votes, and return that. One of our voting models was an egalitarian voting model, where each constituent model receives a single vote. The other was a weighted voting model, where, similarly to the weighted averaging model, we determined the accuracy $\omega^{(i)}$ for each model i in the ensemble, and then gave that model a number of votes proportional to its accuracy.

2.4. Evaluation Metrics

In order to evaluate the performance of our models, we use two primary evaluation metrics. First, and most intuitively, is simply the ratio of the number of correctly prediction pronunciations to the total number of pronunciations generated. We refer to this hereafter as our prediction accuracy.

Although the prediction accuracy is an intuitive metric of the correctness of our model, it fails to explain how incorrect a prediction will be if it is not correct. Therefore, we also use the ratio of the average edit-distance of the incorrect predictions to the average length of the correct pronunciations. Edit-distance calculates the total number of insertions, deletions, and substitutions needed to change one string into another.

The rationale behind edit-distance is that this metric gives information about how close our prediction is to the correct pronunciation. However, even small edit-distances are significant in short words, so we take the ratio of the edit distance to the average length in order to account for this, and we refer to this simply as the "ED ratio."

3. Results and Discussion

3.1. Baseline

Our baseline model performed better than anticipated on our development data, although its prediction accuracy was still very low at 18.2%. The average edit distance between the incorrect predictions and correct pronunciations was 2.310, while the average length of incorrectly predicted words was 8.166. This gives us an ED ratio of 0.282.

3.2. biLSTM

Overall, we consider our biLSTM model a success due to the high accuracy rates and low ED ratios, especially when compared with the baseline. Accuracy scores ranged from 59.9% to 66.2%, which are significantly higher than the baseline accuracy of 18.2%. These results show that alignment-free approaches are effective for G2P conversion and far more successful than our deterministic baseline. ED ratios are slightly less than 0.2, meaning that the output phoneme contains one incorrect character for every five characters.

Our LSTM appears to most accurately execute G2P conversion when the input delay δ is 2. This differs from results found in Rao et al., which found that maximal input delay (i.e. the entire word) produced the highest accuracy. Note, however, that the edit ratio is fairly consistent across the δ values we tested.

3.3. Ensemble

As we described above, we found significant variability in our how well any given model performed, even if we assume a fixed δ, depending on the order that the training data was presented. The variation was also significant across different δ values. In order to counteract this variation, we constructed four different types of ensembles, which process the output of 20 different NN models (five fully trained models for each value of $\delta \in \{0, 1, 2, 4\}$.

When we evaluated the performance with these four ensembles, we found that we could attain up to an 11.6 percentage-point increase in accuracy. The basic (unweighted) average was the just barely the best performing of all the ensembles, but ultimately the performance of all four ensembles was comparable.

Table 1 shows the complete table of our results, including the performance of each the individual models for each value of δ considered, the baseline score, as well as the performance of each of the ensembles.

3.4. Discussion

Given that our model (using the highest performing ensemble) was able to achieve a 57.3 percentage-point improvement over the baseline's accuracy (for a total of 75.5%), we feel that our model is a useful tool for G2P conversions. Not only does our model beat our baseline by a significant margin, we also found that our model attained a final accuracy comparable to those achieved by the models considered by [6] (it must be noted, however, that Rao *et al.*'s models were trained on English, which means a direct comparison is impossible). As a result, it is apparent that our model is producing accuracy rates for alignment-free Dutch G2P that are on par with the alignment-free performance found for English.

INDIVIDUAL AND ENSEMBLE PERFORMANCE

Model	Accuracy (%)	ED	ED ratio
Baseline	18.2	2.310	0.282
$\delta = 0$	59.9	1.734	0.194
$\delta = 1$	64.7	1.658	0.188
$\delta = 2$	66.2	1.656	0.188
$\delta = 4$	64.7	1.639	0.185
Individual Model Average	63.9	1.672	0.189
Averaging Ensemble	**75.5**	1.535	0.174
Avg. Ensemble (Weighted)	75.4	1.527	**0.173**
Voting Ensemble	74.9	1.583	0.179
Voting Ensemble (Weighted)	74.9	1.573	0.179

Table 1. Our results for the baseline, individual LSTM models with varying delay, and ensembles of models (5 of each δ value). The best performance of 75.5% accuracy was achieved by a simple average of the outputs from 20 separately-trained LSTM models, while the best ED ratio was achieved by taking a weighted average.

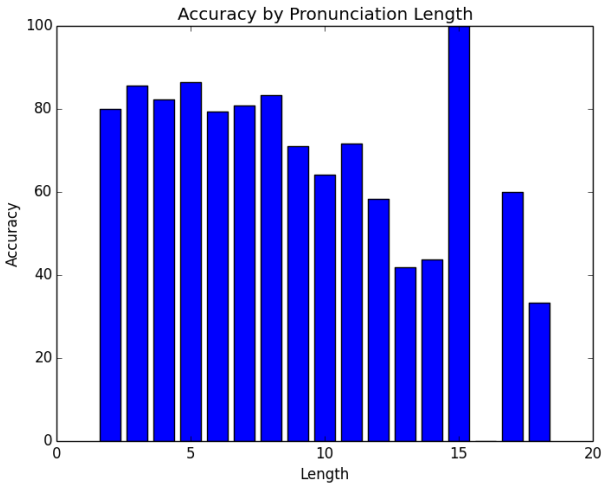

Figure 3. The accuracy attained by our averaging ensemble on our test set graphed against the number of characters in the correct pronunciation string. In general, we see that there is a slight tendency that the longer the word, the harder it is to accurately predict.

After running the averaging ensemble and seeing the predictions that are made compared to the actual pronunciation, it is possible to piece together an understanding of what kind of errors our model is making at the moment, so as to better understand how to improve it in future work. The first thing we looked at in this regard was how well the model does in predicting pronunciations of various lengths (Figure 3 shows the accuracy by length). In general, we see that the longer a word, the harder it is to accurately predict. From a purely probabilistic standpoint, this makes sense, as the more characters that need to be predicted, the more likely we are to make an error. Nonetheless, this is a useful insight into future avenues of research, as Dutch, like German, is a heavily compounding language. This means

that many Dutch words are formed by combining shorter words into single-word compounds (for example, *sinaasappel* (orange) combines with *sap* (juice) to form *sinaasappelsap* (orange juice)). Thus, in many cases we might be able to break longer words down into shorter words by developing a segmentation system to identifying the constituent morphemes of compound words, and then constructing the pronunciation of each segment individually.

Another type of error that is quite common in our data set is duplication of a phoneme in the output. As an example, if the pronunciation of a word is actually /brur/, our model might predict /brurr/. This is likely due to the LSTM not "forgetting" as quickly as it should, and as a result the last phoneme is duplicated. It is possible more training may be able to fix this more readily, but it is also feasible that penalizing this kind of duplicated character more heavily would encourage the model to improve its ability to "forget" past phonemes, though this was avenue of investigation was not ultimately pursued due to time pressures.

Another type of error that is very common is for intuitively related phonemes to be frequently mistaken for each other. For example, we found that the phonemes /a/ and /ɑ/ were frequently confused with one another. In many ways, this makes sense, as the grapheme ⟨a⟩ is often used to represent both sounds. In order to resolve this issue, it would be useful to collect more data, specifically data that contains many /a/s and /ɑ/s, so that the model can have more examples to learn from in order to better distinguish between them.

Additional avenues of inquiry include implementing sequence-sequence or connectionist temporal classification. While we considered these models as potential candidates of implementation, we chose to pursue biLSTM due to the simplicity of its design.

4. Conclusion

To build our G2P conversion model for the Dutch language, we relied on a biLSTM due to its ability to operate without aligned data sets and its inclusion of graphemes' contextual clues before outputting a phoneme sequence. By combining our biLSTM with an ensemble of predictions, we achieved an accuracy of 75.5%. Potential methods to improve this model include classifying graphemes according to their compound structure and implementing sequence-sequence or connectionist temporal classification. Ultimately, our G2P conversion model was successful at producing accurate predictions of G2P conversion, in that it achieved similar rates of accuracy as [6]'s models did in English.

5. Author Contributions

All authors discussed the design of the LSTM architecture. B.H. performed the data preprocessing, wrote the model baseline, and implemented the neural network models. All authors discussed the results, designed the poster, and co-wrote the manuscript.

6. Acknowledgements

We would like to express our sincerest gratitude to the professors and course staff of CS229, for dedicating their time, experience, and knowledge to the class.

References

[1] Maximilian Bisani and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech communication*, 50(5):434–451, 2008.

[2] Terrence J Sejnowski and Charles R Rosenberg. Parallel networks that learn to pronounce english text. *Complex systems*, 1(1):145–168, 1987.

[3] John Lucassen and Robert Mercer. An information theoretic approach to the automatic determination of phonemic baseforms. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'84.*, volume 9, pages 304–307. IEEE, 1984.

[4] Ilya Sutskever, Oriol Vinyals, and V. Le, Quoc. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[5] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *ICML*, 2006.

[6] Kanishka Rao, Fuchun Peng, Haşim Sak, and François Beaufays. Graphme to phoneme conversion using long short-term memory recurrent neural netowrks. In *IEEE ICASSP*, pages 4225–4229, 2015.

[7] Kaisheng Yao and Geoffrey Zweig. Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. In *ISCA - International Speech Communication Association*, 2015.

[8] P.C. Paardekooper. *ABN-uitspraakgids*. Digitale bibliotheek voor de Nederlandse letteren, 1978.

[9] Simon Ager. Omniglot: Dutch, 2015. Accessed: Nov. 7, 2017.

[10] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.