Interspeech 2018
2-6 September 2018, Hyderabad

Dictionary Augmented Sequence-to-Sequence Neural Network for Grapheme to Phoneme prediction

Antoine Bruguier[1], *Anton Bakhtin*[2], *Dravyansh Sharma*[1]

[1]Google LLC, Mountain View, CA
[2]Facebook AI Research, New York, NY

tonybruguier@google.com, yolo@fb.com, drasha@google.com

Abstract

Both automatic speech recognition and text to speech systems need accurate pronunciations, typically obtained by using both a lexicon dictionary and a grapheme to phoneme (G2P) model. G2Ps typically struggle with predicting pronunciations for tail words, and we hypothesized that one reason is because they try to discover general pronunciation rules without using prior knowledge of the pronunciation of related words. Our new approach expands a sequence-to-sequence G2P model by injecting prior knowledge. In addition, our model can be updated without having to retrain a system. We show that our new model has significantly better performance for German, both on a tightly controlled task and on our real-world system. Finally, the simplification of the system allows for faster and easier scaling to other languages.

Index Terms: G2P, sequence-to-sequence, memory network, dictionary augmentation

1. Introduction

Knowing the correct pronunciation for words is a critical part of a text-to-speech (TTS) system. While some end-to-end automatic-speech-recognition (ASR) systems [1, 2] do not use pronunciations, most systems still require them to improve recognition of tail words because they appear infrequently in the training data.

The pronunciation system is usually a combination of a dictionary and a predictive model. For common words, knowing how to pronounce a word is simply a matter of looking up its pronunciation (represented as a string of phonemes) in a dictionary. However, given the size of the vocabulary in both TTS and ASR, it is infeasible to have a complete coverage of all the words. Thus, as a fallback, we typically resort to a predictive grapheme-to-phoneme (G2P) model [3, 4, 5, 6].

How much effort to put into building a pronunciation dictionary is usually dictated by a cost/quality tradeoff. Dictionaries can be built by hiring expert linguists to transcribe the words but this process is typically expensive and slow. Automatic pronunciation learning techniques [7] are useful to grow the dictionary but a G2P will always be needed and thus the option of improving its quality should be explored.

Typically, human languages follow some implicit rules to predict pronunciation rules using prior knowledge. For example, if you know that the word `back` is pronounced `b { " k` and the word `pack` is pronounced `p { " k` then it is not unexpected that the word `backpack` is pronounced `b { " k p { % k`, where we see that the primary stress markers in both components, `"` is changed to a secondary stress `%` for one of a component. The concatenation of pronunciation is not always correct (for example for the city name `Plymouth`), but we hypothethized that the more infrequent the word is, the more predictable its pronunciation is.

An accurate G2P should also be able to predict pronunciation of inflected form. For example, in American English, if the pronunciation of the word `cat` is `k { " t`, then native speakers naturally derive the pronunciation of the word `cat's` to be `k { " t s`. However, appending the phoneme `s` is not a strict rule to get the pronunciation of the possessive form, as evidenced by the word `dog's` pronounced `d O " g z`. For proper nouns (`Charles's`, `Robert's`) the set is impractical to cover in a lexicon. Thus, it would be useful to have a G2P that is able to infer pronunciations from prior knowledge which in this case is the pronunciation for the base form: `cat`.

One approach would be to hand-craft some rules, applied after dictionary lookup but before G2P prediction. For example, we could have some algorithm that strips the word to its base form, then applies the G2P, then modifies the output to predict the pronunciation of the full word. Even for American English, this is nontrivial to do manually, as we would need, for example, to handle possessive (e.g. `cat's`) and plural possessive (e.g. `cats'`) forms in addition to words that are concatenations of subwords (e.g. `activeminds` in the URL `activeminds.org`). Moreover, if we want to scale the approach to multiple languages, writing special rules will become increasingly expensive.

The hand-crafting of rules is even more complex for synthetic languages (languages where forming new words from base words/morphemes is very common) e.g. Germanic languages with productive compounding and Slavic languages with intricate morphology. If the base words are already rare, the combined word will be extremely rare but a native speaker might still know how to pronounce it by using the pronunciation of its components. For traditional G2Ps, a compound word appears like a completely new word, devoid of relationship to its subwords. This is both counterintuitive because it does not mimic humans but also counterproductive because it ignores existing data.

Thus, we would like to have a single G2P model where we can *inject* prior knowledge. We would want this model to be end-to-end in the sense that it is trained all at once, without having to fine-tune some manually written rules. We would want this model to be able to read new knowledge, so for example if an expert linguist or a pronunciation learning algorithm adds a new pronunciation for a base form we can expand our lexicon dictionary and have the G2P use the new data without retraining the neural network.

2. Prior work

There is an extensive history of work on G2Ps [6]. Some pioneering work used finite-state transducers (FSTs) [8] and LSTM neural networks [3]. Recent approaches however, have focused on using sequence-to-sequence [9] models using either recursive neural network transducers (RNN-T, [4, 5]) or listen attend and spell (LAS, [10]). Sequence-to-sequence models have found a wide range of applications beyond G2Ps, including machine translation [11].

Machine translation is of particular interest because of the

10.21437/Interspeech.2018-2061

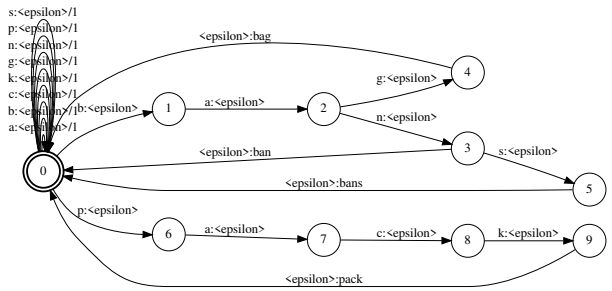

Figure 1: *Word splitter*

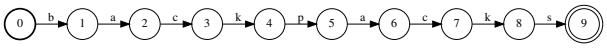

Figure 2: *Input FST I for the word "backpack"*

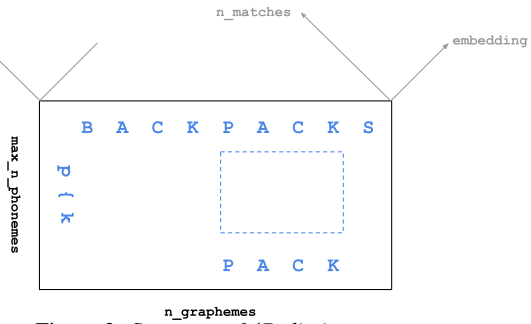

Figure 3: *Structure of 4D dictionary tensor*

issue of tail word distribution. Given the Zipf distribution of words in a language, even the largest data sets will not have complete coverage. The challenge is similar to the one faced by G2Ps that have to predict pronunciations of tail words and we thus would like to find an efficient way of adding a memory bank to a neural network.

Another general approach has been provided by Neural Turing machines [12], but this approach may not be suited for our problem. In our case, we do not need to read and write to a comparatively small memory bank, but rather only read from a large dictionary. Similarly, networks that use a general memory approach [13, 14] might be able to perform the task at hand but might be too slow for real-time systems.

The approach that was the closest to our needs was an augmentation of a machine translation network [15]. Our approach follows the idea of augmenting the input of a sequence-to-sequence model to inject some additional information. We, however, had to solve two issues that only exist with G2Ps. First, while in machine translation it is possible to decompose a sentence into words, in our case, we do not a-priori know what the sub-components of a word are. We thus needed to have a way to find the sub-components. Second, we do not have a one-to-one mapping from the word in one language to the word in the other, but instead in our case, we need to force-align a sequence of graphemes to a sequence of phonemes. Nevertheless, the approach of [15] was inspiring in the design of our algorithm.

3. Model

The first step of the algorithm is to split a word into its components. We first build an FST that contains all the words in the lexicon with paths shared for as long as the prefixes of the words match.

3.1. Word splitter

Figure 1 shows a simplified example of such an FST for a lexicon that only contains the words `bag`, `ban`, `bans`, and `pack`. The FST accepts the sequence of letters `b`, `a`, and `g` through states 0, 1, 2, and 4. Then a loop back from state 4 to 0 emits the word `bag`. The words `ban` and `bans` are represented by states 0, 1, 2, and 3 and 0, 1, 2, 3, and 5, respectively. We can see that since the words `bag`, `ban` and `bans` share a 2-letter prefix, their paths are shared up to state 2.

In addition to the sub-word candidates, we also have free-grapheme loops that start and end at state 0, with arcs that are weighted. The resulting FST is called S. To decompose a word like `backpacks`, we first construct an input FST I as shown in figure 2.

Finally, we find the max-covering of the input words. We compute the composition of the two previous FSTs: $I \circ S$ and find the top N lowest cost best paths. In the example above, the best path goes through the loop graphemes `b`, `a`, `c`, and `k`. Then, it goes through the word `pack`, and finally again through the loop grapheme `s`. The total cost of this path is the total number of loop graphemes: 5. If we had included the word `back` in the S FST, we would have had a lower of 1 (single loop grapheme `s` at the end).

In practice, we filter lexicon entries that are too short (3 characters or below). In particular, this prevents single-letter entries from being candidates for subwords. Note that this does not mean we won't learn affixes. If the model sees a lot of examples for a highly productive affix, say German `-er`, then we'd see a lot of training examples of the form base+`er` with the prior knowledge of the pronunciation for base. The model will learn how to pronounce a base+`er` word given the pronounciation of the base. We also filter out words that match the entire input, because in practice, if we have a pronunciation for the word itself, the G2P will not be called and we want the network to train with more than just copying its hint.

3.2. Force-align tensor

We build a 4D tensor to encode the matches, as shown in figure 3. Its dimensions are:

- `I` Number of matches: number of subwords that are matches of the input. This dimension could be zero if there is no match.
- `J` Number of graphemes in the input word.
- `K` Maximum number of phonemes in all the matches.
- `L` Embedding of the data.

For a given match index, whenever the word grapheme does not correspond to the subword, the value of all the embedding entries is 0. Otherwise, the embedding encodes the following information:

- $\log(\text{len}(subword))$. For "pack" that value is $\log 4$.
- $\text{distance\_from\_start}(grapheme)/\text{len}(subword)$. For "pack" that value goes from $0/4$ to $3/4$.
- $\text{distance\_from\_end}(grapheme)/\text{len}(subword)$. For "pack" that value goes from $3/4$ to $0/4$.
- $\log(\text{len}(pronunciation))$. For "pack" that has pronunciation "p { k" the value is $\log 3$.
- $\text{distance\_from\_start}(phoneme)/\text{len}(pronunciation)$. For "pack" that value goes from $0/3$ to $2/3$.
- $\text{distance\_from\_end}(phoneme)/\text{len}(pronunciation)$. For "pack" that value goes from $2/3$ to $0/3$.
- One-hot encoding of the grapheme.
- One-hot encoding of the phoneme.

Note that the value of I will vary depending on the input. If there are no matches (determined as in section 3.1), then I will be zero. For each example, the value is different. Once the training is done, we can still add new words to the lexicon, so that the value of I depends on the input data, and we don't need to retrain the network when we add pronunciations to the dictionary (although if we add many of them, a retrain might still be beneficial). Section 3.3 will show how we handle a varying number of matches.

We hypothesized that the number of possible splits is small for tail words. For example, in English the word `carshow` could be decomposed into `car` and `show` or `cars` and `how`. This could likely be improved with additional morphology and frequency data, but our model currently does not support this. Thus, the performance is dependent on the number of splits, which we hypothesized is small for tail words.

3.3. Force-align network

The goal of the force-align network is to reduce the dimensionality of the 4D tensor from section 3.2. We want to remove the first dimension (of size I) and the third dimension (of size K).

To remove the third dimension, we use a quadridirectional LSTM on the dimensions J and K, with dimension I used as a batch. This could be thought of as a force-align, because the network has access to both the graphemes and phonemes, and can learn which phonemes correspond to which graphemes. Additionally, the embeddings of section 3.2 give hint of the approximate location of the match. Note that the force-align is not strict because a single phoneme can correspond to multiple graphemes (like `k` in the word `chiral`), or multiple phonemes can correspond to a single grapheme (like `d V b @ l j u` in `W`).

After the quadridirectional LSTM, we have another 4D tensor of dimensions (I, J, K, L'). The last dimension L' is a parameter of the model. We only take the first slice for the third dimension, and get a 3D tensor of dimensions (I, J, L').

Finally, to remove the first dimension I, we average over all matches. We thus get a 2D tensor of dimension (J, L'). If there are no matches, the tensor is set to be zero.

We can now augment the input of traditional sequence-to-sequence models. Both RNN-T and LAS models for G2Ps have the sequence of one-hot encoded graphemes as an input. Thus, the dimension of the input is (J, M) where M is the number of grapheme symbols used. By concatenating this tensor with the output of the force-align network, we get a new input of dimension $(J, M + L')$ which can be transparently used by sequence-to-sequence models. The data encoded in the output of the force-align network is a hint of what the pronunciation of that letter (in this neighborhood) should be.

4. Experiments

4.1. Model parameters and training data

We trained two RNN-T models one with dictionary augmentation, and one without. Both models used an encoder with 3 LSTM layers, each with 256 units. We used dropout [16] with a keep value of 0.9.

The decoder used RNN transducers [4, 9, 5] where the decoder network used 3 LSTM layers, each with 256 units. We used dropout with a keep value of 0.6 for both models.

In the base condition, we did not augment the input with any force-align data. In the test condition, the force-align network had 3 quadridirectional LSTM layers, each with 128 units and a dropout keep probability of 0.6. The output of the force align (of dimension `input_length` x 128) was concatenated to the input graphemes (of dimension `input_length` x `number_letters_in_alphabet`) along the first dimension. Thus, the only difference betweeen the two models is the presence/absence of the dictionary augmentation network. All other parameters were identical.

We chose to investigate the performance of the model on a G2P for German words. German has both morphological variation and a productive agglutinative aspect, so it is a good example of how using our model can result in significant differences between the base and test case. In addition, it is a language that has a wide enough usage that we would both have enough lexicon data and any improvement would benefit many users. We had at our disposal a lexicon consisting of hundreds of thousands of entries, split randomly in 80% train set, 10% test set, and 10% dev set.

We had the lexicon available in two formats, a version with pure phonemes and another which had phonemes along with stress markers and syllable boundaries annotated. Getting the latter right is harder, but it is not clear to what extent they impact speech applications (we add listening test evaluations to remedy that). Also, often an important aspect of agglutination of words is that the stress and syllabification might be affected. Thus, we decided to train G2Ps on the stressed syllabified phonemes.

4.2. Results on held-out set

We first looked at the performance results on the test set. After the same number of iterations, the model that used the dictionary input had better performance than the one that did not (see table 1); We saw a relative reduction of about 28% in the number of errors made. While encouraging, this measure does not fully represent the performance of the new model in its intended use. Indeed, here we have compared performance on a random subset of our lexicon, which, by design, focuses on head words. However, the G2P is intended to be used precisely when there is no entry in the lexicon, typically for tail words.

Baseline model	86%
Dictionary-augmented model	90%

Table 1: *Performance on test set. Accuracy (higher is better).*

4.3. Side-by-side analyses

We ran side-by-side comparisons of the performance of the base and test models. We took a random sample of our TTS traffic and synthesized sentences using both models. The synthesis used the entire TTS stack (text normalization, verbalization, etc...) including the lexicon data. Thus, this was a faithful representation of the impact of the new model on voice quality.

Every word where the final phonemic representation was identical between base and test was discarded. This could have been because either the word was in the lexicon or the two models, while different, predicted the same pronunciation for that word. We then synthesized the audio waveform from the phonemic representation and played both to raters. These raters were native German speakers, but had otherwise no profesional background in linguistics.

The rating task was in two steps. Native German speakers were asked to listen to a TTS rendering of both pronunciations. Then, for each, they could indicate whether the pronunciation was "correct", "incorrect", or they were "unsure". In case they indicated that both pronunciations were "correct", then they were asked a follow-up question as to which pronunciation they preferred, if any. For all questions, we randomly flipped the display so that raters were blind to which side was the base and test algorithm. We used this setup for two different experiments described in the sections below.

4.3.1. Dictionary-augmented sequence to sequence versus plain sequence to sequence

The goal of this experiment was to isolate the effect of dictionary augmentation. The difference was in the G2P. In the base condition, we used the sequence-to-sequence model that did not use dictionary augmentation (the base model of section 4.2). In the test condition, we used the sequence-to-sequence model that had dictionary augmentation (the test model of section 4.2).

	Correct	Incorrect
Baseline model	53.5%	36.9%
Dictionary-augmented model	65.0%	25.4%

Table 2: *Side-by-side to measure the effect of dictionary augmentation (N=2,973)*

Test model better	About the same	Base model better
19.6%	66.9%	13.5%

Table 3: *Preference when both pronunciations are judged correct (N=2,973)*

The results are shown in tables 2 and 3. We had N=2,973 rating tasks, and we see that the algorithm that used dictionary augmentation produced correct pronunciations more often (65.0%) than the algorithm that did not use dictionary augmentation (53.5%). Note that the percentages do not add up to 100% because the raters had the option to indicate that they were unsure (the words being rare, it is not surprising to have a non-negligible number of ratings that fall in this category). For the ratings where both pronunciations were judged as correct, we still see a preference for the model that uses dictionary augmentation (19.6% versus 13.5%) even if both variants are often equally liked.

4.3.2. Dictionary-augmented sequence to sequence versus previous algorithm

The goal of this experiment is to test whether we can replace hand-crafted rules with our new model. We compared two algorithms to generate pronunciations. The base model was our current production system, that used manually-curated subword rules and our current G2P. The test model did not have any manually-curated rules and used our dictionary-augmented G2P. The rules of the base model were determined by a native speaker engineer who decided how to split words into subcomponents, refer some pronunciations from lexicon and recombine the individual pronunciations to form the final output using a combination of handwritten grammars and manual blacklisting/whitelisting. While the measurements we report in this section are less crisp and tightly controlled than in section 4.3.1 we felt that they could be insightful because they measure the quality of the new algorithm precisely how it is intended to be used in a real production system. This model also has the advantage of being end-to-end in the sense that it handles both subword discovery and pronunciation prediction.

	Correct	Incorrect
Manually written rules	69.4%	21.7%
Dictionary-augmented model	75.1%	16.0%

Table 4: *Side-by-side to measure the effect of dictionary augmentation (N=2,985)*

Test model better	About the same	Base model better
21.0%	64.4%	14.6%

Table 5: *Preference when both pronunciations are judged correct (N=2,985)*

The results are shown in tables 4 and 5. We had N=2,985 rating tasks, and we see that the algorithm that used dictionary augmentation produced correct pronunciations more often (75.1%) than our current production system (69.4%). We observed a similar pattern in ratings that are not definitive. For the ratings where both pronunciations were judged as correct, we still see a preference for our new model (21.0% versus 14.6%) with still a high amount of variants that are equally liked.

5. Discussion

A G2P is a critical component of both ASR and TTS systems. While we usually rely on pronunciation lexicons to know the correct pronunciation of words, they typically cannot cover the entire vocabulary and we have to use a G2P as a fallback. Thus, by definition, a G2P is typically used to predict the pronunciation of tail words.

Usually, lexicon and G2P are only loosely coupled. While we use the lexicon data to train a G2P, once the training is done, the two systems operate independently. If we add words to the lexicon, the new information can only be propagated to the G2P by doing a full retraining.

Another limitation of loosely coupled G2Ps is the intent of the training task. We typically intend the G2P to learn *general* rules about how words are pronounced. We hypothesized that it is not a full picture of how languages work and that small variants, or compositions of words would result in pronunciations that would be highly dependent on the pronunciation of the base words. In these cases, the G2P needs to learn *specific* rules about how one word is pronounced given its base components' pronunciations. This is contrary the way G2Ps are currently approached.

Our new model demonstrates how to implement such G2Ps. It is based on an existing sequence-to-sequence model, where we augment the input. This model has several advantages. It is flexible and the augmentation is compatible with a wide variety of sequence-to-sequence models, since only the input is modified. This addition is orthogonal to the architecture of the rest of the model, and thus we do not need additional modification. Another advantage is that we can inject new knowledge without having to retrain the G2P.

We measured the quality of the new G2P both on a test set and by doing side-by-side experiments. In all cases, we saw an improvement of the pronunciations that were predicted. A strict comparison that highlights the differences of augmentation shows the benefits of our approach.

A comparison against our current production system also reveals improvements. Not only can we obtain better performance with our new model, we were also able to remove the need to manually curate and write rules for how to predict the pronunciation of a word given its sub-components. Thus, our new model also greatly simplifies the architecture of our system, reduces the engineering effort, and allows us to scale to more languages, especially those where finding engineers familiar with the linguistics is difficult. We thus plan to extend the approach to other languages such as Dutch, Swedish, Norwegian, and Finnish.

While the approach here was solely focused on G2Ps, we believe that the idea of injecting knowledge has application in many parts of ASR and TTS systems because the distribution of words is such that being able to inject knowledge about tail words is useful. Thus, future work will also focus on applying the results presented here to a wider range of systems.

6. References

[1] Y. Zhang, M. Pezeshki, P. Brakel, C. L. Saizheng Zhang, Y. Bengio, and A. Courville, "Towards end-to-end speech recognition

with deep convolutional neural networks," *Interspeech*, 2016.

[2] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," *ICML*, 2014.

[3] K. Rao, F. Peng, H. Sak, and F. Beaufays, "Grapheme-to-phoneme conversion using long short-term memory recurrent neural networks," *ICASSP*, 2015.

[4] K. Yao and G. Zweig, "Sequence-to-sequence neural net models for grapheme-to-phoneme conversion," *Interspeech*, 2015.

[5] M. Bisani and H. Ney, "Joint-sequence models for grapheme-to-phoneme conversion," *Journal Speech Communication*, 2008.

[6] S. Hahn, P. Vozila, and M. Bisani, "Comparison of grapheme-to-phoneme methods on large pronunciation dictionaries and lvcsr tasks," in *Interspeech*, 2012.

[7] A. Bruguier, D. Gnanapragasam, L. Johnson, K. Rao, and F. Beaufays, "Pronunciation learning with RNN-transducers," *Interspeech*, 2017.

[8] M. Jansche, "Computer-aided quality assurance of an icelandic pronunciation dictionary," in *LREC*, 2014.

[9] A. Graves, "arxiv," in *Sequence Transduction with Recurrent Neural Networks*. [Online]. Available: https://arxiv.org/abs/1211.3711

[10] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, attend and spell," *ICASSP*, 2016.

[11] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *NIPS*, 2014.

[12] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines." [Online]. Available: https://arxiv.org/pdf/1410.5401.pdf

[13] C. Gulcehre, S. Chandar, and Y. Bengio, "Memory augmented neural networks with wormhole connections." [Online]. Available: https://arxiv.org/pdf/1701.08718.pdf

[14] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," *NIPS*, 2015.

[15] Y. Feng, S. Zhang, A. Zhang, D. Wang, and A. Abel, "Memory-augmented neural machine translation," *EMNLP*, 2017.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, 2014.