INTERSPEECH 2017
August 20–24, 2017, Stockholm, Sweden

Pronunciation learning with RNN-transducers

Antoine Bruguier, Danushen Gnanapragasam, Leif Johnson, Kanishka Rao, Françoise Beaufays
Google Inc, USA
{tonybruguier,nanabyte,leif,kanishkarao,fsb}@google.com

Abstract

Most speech recognition systems rely on pronunciation dictionaries to provide accurate transcriptions. Typically, some pronunciations are carved manually, but many are produced using pronunciation learning algorithms. Successful algorithms must have the ability to generate rich pronunciation variants, e.g. to accommodate words of foreign origin, while being robust to artifacts of the training data, e.g. noise in the acoustic segments from which the pronunciations are learned if the method uses acoustic signals. We propose a general finite-state transducer (FST) framework to describe such algorithms. This representation is flexible enough to accommodate a wide variety of pronunciation learning algorithms, including approaches that rely on the availability of acoustic data, and methods that only rely on the spelling of the target words. In particular, we show that the pronunciation FST can be built from a recurrent neural network (RNN) and tuned to provide rich yet constrained pronunciations. This new approach reduces the number of incorrect pronunciations learned from Google Voice traffic by up to 25% relative.

Index Terms: speech recognition, pronunciation learning

1. Introduction

Many state of the art automatic speech recognition (ASR) systems are composed of three main models: the acoustic model (AM), the pronunciation model (PM), and the language model (LM). Loosely speaking, the AM transforms a raw audio waveform into a stream of phoneme probabilities, the PM transforms a stream of phoneme probabilities into a stream of word probabilities, and the LM transforms a stream of word probabilities into a stream of sentence probabilities. The final recognition is the most probable sentence.

Typically, PMs are a combination of a word-to-pronunciation dictionary and a grapheme-to-phoneme (G2P) model [1, 2, 3] as a backoff. When a word is present in the dictionary, we use the corresponding pronunciation. Otherwise, a G2P model attempts to predict the pronunciation from the spelling.

Having an accurate G2P is critical: Without a correct pronunciation, an ASR system might not be able to recognize some uncommon words. However, even native speakers of some languages have difficulty knowing how to pronounce some words. For example, in American English, the pronunciation of words such as "Canada" (from "La Canada Flintridge" /k ə n j ɑ d ə/), "Bexar" (County in Texas, /bɛər/), "Lhuillier" (from "Monique Lhuillier" /l u i l i ɹ/) are not obvious. Building a G2P that performs better than humans is a formidable challenge.

For this reason, building an accurate pronunciation dictionary from data has been an active field of research for at least two decades. For example [4] used manually-curated rules and [5] used a decision tree approach. Pronunciation learning algorithms must balance the need for flexibility so that they can learned varied pronunciations with the requirement to be constrained enough so that the results are accurate and robust to noise. Earlier research used decision trees [5] to strike this balance. More recently, algorithms using expectation maximization algorithms [6, 7, 8], hidden Markov models [9, 10], discriminative training [11] have yielded significant improvements.

In this paper, we revisit an FST-based pronunciation learning algorithm [12, 13, 14] that in essence flips what is unknown in a speech recognition task. In a traditional speech recognition task, we fix the pronunciation of each word and the sequence of words is not known. Using the phoneme probabilities from the AM, we find the most likely path, thus finding out the most likely sequence of words. In the pronunciation learning task, we fix the sequence of words and the pronunciation of each word is not known. Using the AM, we find the most likely sequence of phonemes.

Figure 1 illustrates the current FST-based algorithm. From the transcript "good old way", we create a finite state transducer (FST, [15]) that has three sections (one for each word). We use a G2P candidate generator that gives several hypotheses for the pronunciation of each word. In the figure, we show three pronunciation candidates, but in practice we generate 20 candidates. This approach, however, suffers from several drawbacks.

First, the pronunciation candidate set lacks diversity. Often times, the correct pronunciation is not in the top 20 candidates, making it impossible to learn, no matter how much audio data the algorithm has at its disposal. Increasing the number of candidates is not efficient because their number grows exponentially with the number of graphemes in the word.

A second drawback is that the current approach does not weigh the various pronunciations. While having diversity of candidates is important, the graphemes still convey some information, and, as we will show below, using them improves the quality of the pronunciations learned. Other drawbacks include lack of independence between each pronunciation learned for a given word, inefficiency of aggregating examples, strict dependence on graphemes, inability to learn from multiple transcripts, and awkward handling of verbalization.

The proposed general algorithm has an infinite number of candidates and at the same time retains (and controls the amount of) importance given to the graphemes. It naturally extends to multiple transcripts and multiple verbalizations. It also can learn pronunciations in the case where graphemes are not known.

2. A more general architecture

The insight of the proposed algorithm comes from observing the two types of arcs on the FST built in the old algorithm (figure 1). The first type of arcs are those that accept epsilons and output words. The second type of arcs are those that accept a phoneme and output the same phoneme. In the proposed approach, we split the construction of the FST in two steps: first construct the FST with arcs that outputs words, and second construct the various FSTs with arcs that output phonemes. There are many ways to build the second kind of FSTs; we describe some of them, but the approach is general and it is easy to add new ways.

http://dx.doi.org/10.21437/Interspeech.2017-47

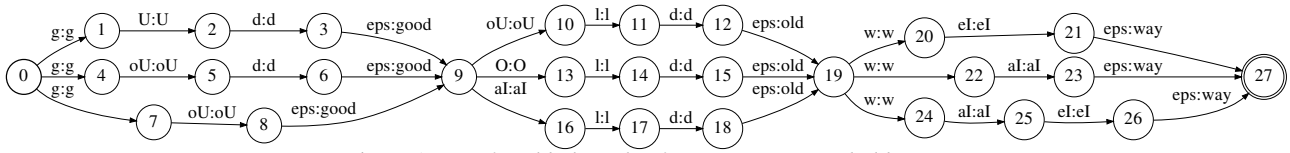

Figure 1: *FST for old algorithm for transcript "good old way"*

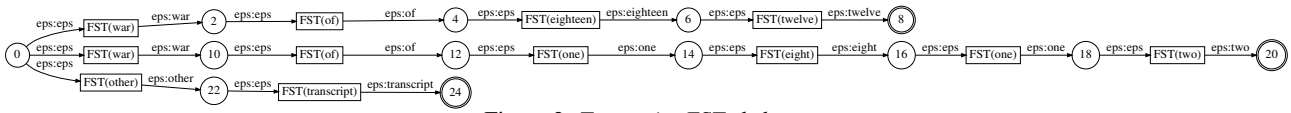

Figure 2: *Transcript FST skeleton*

2.1. Building the transcript FST

In this section, we describe how to build the FST with arcs that output words (figure 2). The approach is an extension of the multiple-transcript approach proposed in [16]. For each possible transcript, we create one or more verbalized transcripts. Then, for each verbalized transcript, we create a branch of an FST that consists of a sequence of phoneme-to-phoneme-FST (described below) and epsilon-to-word FST.

For clarity, consider an example where it is unknown whether the transcript is "war of 1812" or "other transcript". We first verbalize both transcripts, resulting in transcripts "war of eighteen twelve", "war of one eight one two", and "other transcript". We then built the FST of figure 2 where the FSTs for the words (rectangular boxes) still need to be built.

2.2. Building the word FSTs

In this section, we describe various ways of building the word level FSTs. All these FSTs are acceptors: the inputs and outputs are the same. These acceptors allow only a subset of phoneme sequences and/or give a weight to phoneme sequences.

2.2.1. Lexicon FST

For a given word, this FST accepts either the pronunciation in the dictionary, if present, or the top G2P pronunciation, otherwise. On its own, this FST has very little use, because with it we would simply learn what we already know or use the current best guess we have. In other words, using this FST is the same as doing a force-alignment. We will show its usefulness in section 2.3.

2.2.2. N-best FST

This FST used a candidate generator to create of list of N pronunciations. The resulting FST accepts only these pronunciations. Using this FST, we replicate the behavior of the prior algorithm, except that we can handle multiple transcripts and verbalizations.

2.2.3. Free-form FST

This FST is a simple loop on itself that accepts any and an arbitrary number of phonemes (figure 3). Thus, this simply reads in the most likely phonemes from the acoustic model.

2.2.4. Phonotactic FST

Using special linguistic knowledge, a *phonotactic* FST can be manually built. These phonotactic constraints are language-specific and are built from linguistic knowledge [17]. The constraints are the same regardless of the graphemes, but the FST is more restrictive than the free-form one of 2.2.3. For example, for American English, phonotactic constraints would prohibit the pronunciation /p z b s s p p/.

Figure 3: *Free-form FST. For simplicity, we omitted most phonemes*

2.2.5. G2P FST

Previous work [18] on building a G2P used an FST. This FST, F, accepts graphemes and outputs phonemes. The arcs of F have weights. Instead of using it to generate candidates for each word we can create another FST, G, that accepts epsilons and outputs graphemes. We then compute the composition $G \circ F$, resulting in an FST that accepts epsilons and outputs phonemes. By doing a projection that copies the output symbols of each arc to the input symbols, we get a weighted FST that can be readily used for pronunciation learning.

We call this approach *FST-based phonographic* pronunciation learning.

2.2.6. Neural language model expansion FST

Recent progress [19] in the design of neural networks simplified sequence to sequence modelling and in [20], the authors use this approach to create a new G2P. A full description of the technique is beyond the scope of this paper; what we show here is how to use this new G2P for pronunciation learning.

The essence of the approach of [20] is to encode the grapheme sequence into a state vector. Then, using a decoder, the model recursively outputs phoneme symbols. For each step, the input of the decoding network is the previously predicted phoneme and the current state. The decoding starts with a special `sos` symbol. The decoding stops once the network predicts a special `eos` phoneme (figure 4). For simplicity, we only show the first two unrolling steps and only for a subset of the phonemes. The resulting tree-shaped FST is infinite. The weight of each arc is the probability assigned by the neural network for this transition, as computed with a softmax.

In [20] the authors find the most likely path from `sos` to an `eos`. However, in pronunciation learning, we keep the entire FST. Each arc has a weight according to its likelihood and we combine this weight with the AM weight during decoding. Due to the very large size of the FST, we only explore sections of it during the decoding, using a lazy FST expansion API [21].

This approach has the additional advantage of having two tunable hyperparameters. The first hyperparameter is the Boltzmann temperature τ of the softmax output:

$$p_i = \frac{e^{z_i/\tau}}{\sum_j e^{z_j/\tau}} \tag{1}$$

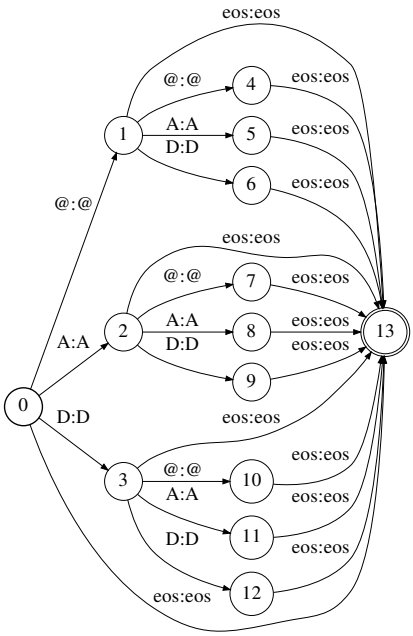

Figure 4: *FST built from sequence-to-sequence model.*

The temperature τ serves to control the "spikiness" of the output, thus controlling candidate diversity. Set to 1.0, it preserves the original model outputs; set to ∞, it makes each transition equiprobable; and in the limit near 0, the output is a pure maximum function.

The second hyperparameter is the relative weight α of the AM and PM models. The beam search algorithm minimizes the total cost of the path, computed as the sum of the AM and PM costs:

$$TotalCost = \alpha.AMCost + (1 - \alpha).PMCost \qquad (2)$$

By tuning α, we can control the importance of the AM versus the PM.

Using either of these parameters, we can smoothly tune the algorithm from a grapheme-free pronunciation learning to a force-alignment (no pronunciation learned, we only use the graphemes to get the top candidate). If we either make the PM cost relatively more important or if we make the softmax output more spiky, we pay more attention to the graphemes and become more like a force-alignment. Conversely, if we make the AM cost relatively more important or if we make the softmax output more smooth, we pay more attention to the acoustic data and become more like a grapheme-free pronunciation learning.

The correct tradeoff between pure G2P and grapheme-free is task-dependent. For tasks where the AM has high accuracy, the audio is of high quality, the prediction of pronunciation from graphemes is difficult, or the PM is of low quality, we should favor the AM. Conversely, in case of low-accuracy AM, noisy audio, or easy to predict pronunciation from graphemes and accurate PM, we should favor the PM.

We call this approach *RNN-based phonographic* pronunciation learning.

2.3. Learn one at a time or all at once

While the algorithm works by replacing the boxes $FST(word)$ in figure 2 with one FST build according to one algorithm of section 2.2, it has to be done carefully.

Consider, for example, the simple transcript "good old way" where we learn the pronunciations by using a free-form FST (section 2.2.3) for all the words. Even if the AM is perfect, there will be ambiguity because we don't know how to place word boundaries in the sequence of phonemes "g U d oU l d w eI" and we could end up learning that the word "good" is pronounced "g U d oU". For some word FSTs, such as N-Best FST (section 2.2.2), there is almost never any ambiguity.

We thus need to resort to two approaches, depending on whether the word FST creates some word-boundary ambiguity or not. If there is no ambiguity, then we can substitute *all* the boxes of the form $FST(word)$ in figure 2 with the appropriate FST.

However, if there is some ambiguity, we need to learn words one at a time. In this case, we loop over all the words in the verbalized transcript. For the word we are currently interested in, we use the FST that enables us to learn a pronunciation (for example, a free-form FST). For the other words, we fix the pronunciation using the lexicon FST (section 2.2.1). This means sometimes we don't learn any pronunciation at all. For example, if the transcript is "war of 1812" and the word-to-learn is "eight" (figure 2), it is very likely that given the audio data, that branch is not taken, and we do not learn anything.

Learning one word at a time is computationally more demanding. We can speed up the computation by skipping very common words: There is no point in learning the pronunciation for the word "the". However, we cannot skip all the words that are already in our dictionary, because we might miss new pronunciations for existing words.

3. Experiments and results

3.1. Building two evaluation sets

To measure the performance of the new algorithm, we used two data sets. The first data set is a random sampling of Google anonymized voice traffic. We used it to measure the quality of the pronunciation learning algorithms on general traffic. The second data set comes from voice traffic that was followed within 30 seconds by a typed correction by the user. If that typed correction only changed a single word, we keep the utterance and use the typed transcript. We can use this set to magnify the differences between the algorithms by purposely choosing traffic that contains difficult to recognize words. For both sets, we needed to get a phonetic transcription of the words.

Since the phonetic transcription is expensive, we used a combination of untrained raters and trained linguists to transcribe all the words. For the untrained raters, we synthesized audio from the learned pronunciations and asked them to compare it to the original audio.

After rating, we divided the words in our dataset into three categories:

- Words for which the pronunciation learned by one algorithm differs from the pronunciation learned by the other algorithms.
- Words for which all the pronunciation learning algorithms agree and the non-linguist raters say it is different from the synthesized audio.
- Words for which all the pronunciation learning algorithms agree and the non-linguist raters say the word in the synthesized audio has the same pronunciation as in the original audio used in pronunciation learning.

We then asked trained linguists to transcribe the correct pronunciations for words in the first two categories. For the words in the third category, we took the learned pronunciation as correct. This approach guarantees that the relative strength was always correctly evaluated, while saving some transcription cost by having a slightly noisier absolute performance measurement.

Table 1 shows the results on the random sampling set. We compare the performance of the prior algorithm with the phonographic algorithm based on a neural network model. The new algorithm has 25% fewer errors (relative).

Baseline algorithm (section 2.2.2)	84%
Neural model (section 2.2.6)	88%

Table 1: *Performance on randomly-sampled traffic. Accuracy (higher is better).*

To magnify the differences between the algorithms, we built a data set that comes from typed corrections. We also filtered out about 500,000 pronunciations that were already in our dictionary, thus replicating the conditions in which we would run the algorithm in production.

Force-alignment (section 2.2.1)	30%
Baseline algorithm (section 2.2.2)	48%
From G2P FST (section 2.2.5)	57%
Neural model (section 2.2.6)	57%
Neural model, tuned τ and α	62%
Phonotactic (section 2.2.4)	31%

Table 2: *Performance on typed-after-voice traffic. Accuracy (higher is better).*

Table 2 shows the results on typed-after-voice traffic. The force-alignment shows the perforance if we don't do any pronounciation learning: We get only 30% of the words correct. The baseline algorithm improves that figure by getting 48% correct. By comparing this number to the FST from the G2P or the neural model, we see the improvement from not being limited to a fixed number of candidates. We see further gains by tuning the parameters τ and α. Finally, the phonotactic approach (which relies principally on the acoustic model) shows that we do need to take into account the graphemes, and that the optimal weight given to them should be tuned.

3.2. Impact on ASR

As an additional way to evaluate the performance of the new algorithm, we ran the neural model with tuned τ and α (table 2) on 90 days of typed corrections.

We filtered out many newly learned pronunciations. First, we filtered out about 500,000 pronunciations that we already had in our dictionary. We also excluded pronunciations that had been already learned from the same set, but using our old algorithm. We aggregated the learned pronunciations across several utterances and filtered them by frequency (e.g., a threshold on the ratio of word-pronunciation pair occurrences over word occurrences). The frequency filters were identical for both algorithms. Finally, non-linguist raters checked the pronunciations learned by listening to the original utterance and compared it with a TTS-rendered version of the newly-learned pronunciation. We only kept the pronunciations that were marked as correct. This yielded about 2,000 new pronunciations.

We observed that the proportion of pronunciations that humans marked correct went up to 71% from 64%, despite the fact that we had already mined this data set using the old pronunciation learning algorithm. In other words, we were able to learn new pronunciations from an already used dataset and still get more of them correct.

To confirm that the pronunciations learned were helpful, we measured their impact on ASR performance. We added the 2,000 new pronunciations to our dictionary and rebuilt the LG

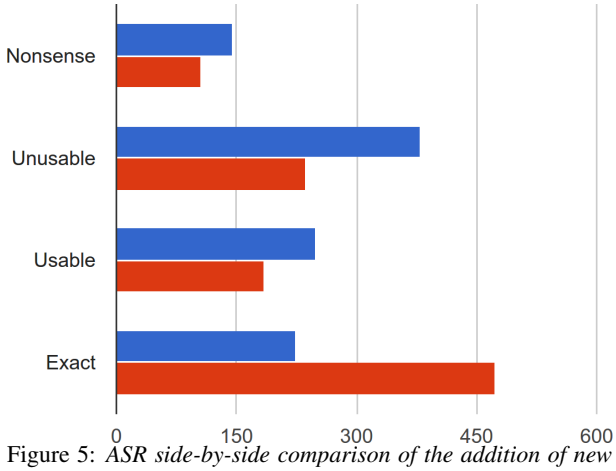

Figure 5: *ASR side-by-side comparison of the addition of new pronunciations. The x-axis is the number of utterances. The y-axis is the quality of the recognition, as judged by human raters for the base recognizier (blue) and the one with the added pronounciations (red).*

FST. Since our existing pronounciation dictionary already contains many words, evaluating the impact the word-error rate on test sets would require an extremely large set. Instead, we ran the recognizer over audio traffic not overlapping with the traffic used to learn the pronunciations. For every utterance where the recognition was different between the two models, we asked non-linguist raters to judge the quality of the recognition. We observed clear improvement of the recognition (figure 5). The new model (in red) produces many fewer transcriptions that were judged "nonsense" or "unusable" and many more "usable" and "exact". On a per-utterance basis, about 4.75 recognitions were improved for each worsened recognition. The differences were statistically significant at $p < 0.001$.

4. Conclusions

We have described a more general approach for learning pronunciations from audio. Instead of the rigid construction of an LG FST, we show a generic approach to building the FST. This generic approach is a superset of the previous approach, but it can also do force-alignment and text-free pronunciation learning.

One of the possible algorithms uses a RNN-transducer neural network. This approach has an infinite number of candidates, while at the same time uses the graphemes to guide the learning. This frees us from the systematic error where the correct candidate is not in the candidate list. We have at our disposal two knobs to tune the amount of importance given to the graphemes.

We used the tuned algorithm to learn pronunciations and showed improvements in several ways. On a golden set, we reduced the number of errors by about 25% relative. On a large set of typed corrections, we improved the number of accepted new pronunciations despite the fact that we were re-mining the set. Finally, we showed that the newly learned pronunciations had a highly statistically significant impact on ASR quality, with a 4.75 to 1 improvement over regression ratio.

5. Acknowledgements

We would like to thank Tom Babgy for his help with the lazy FST expansion, David Rybach for his help with FST beam search, and Alexa Cohen, Bobby Bermudez, Evan Crew, Shayna Lurya and Morgan T'Felt for the transcription of sets.

6. References

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

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

[3] K. Rao, F. Peng, H. Sak, and F. Beaufays, "Grapheme-to-phoneme conversion using long short-term memory recurrent neural networks," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[4] E. Fosler, M. Weintraub, S. Wegmann, Y.-H. Kao, S. Khudanpur, C. Galles, and M. Saraclar, "Automatic learning of word pronunciation from data," 1996.

[5] B. Byrne, M. Finke, S. Khudanpur, J. Mcdonough, H. Nock, M. Riley, M. Saraclar, C. Wooters, and G. Zavaliagkos, "Pronunciation modelling for conversational speech recognition: A status report from ws97," 1997.

[6] I. Badr, "Pronunciation learning for automatic speech recognition," Ph.D. dissertation, 2011.

[7] F. Beaufays, A. Sankar, S. Williams, and M. Weintraub, "Learning linguistically valid pronunciations from acoustic data," 2003.

[8] L. Lu, A. Ghoshal, and S. Renals, "Acoustic data-driven pronunciation lexicon for large vocabulary speech recognition," 2013.

[9] R. Rasipuram, M. Razavi, and M. Magimai-Doss, "Integrated pronunciation learning for automatic speech recognition using probabilistic lexicon modeling," 2015.

[10] R. Rasipuram and M. Magimai-Doss, "Acoustic data-driven grapheme-to-phoneme conversion using kl-hmm," 2012.

[11] O. Vinyals, L. Deng, D. Yu, and A. Acero, "Discriminative pronunciation learning using phonetic decoder and minimum-classification-error criterion," 2009.

[12] A. Rutherford, F. Peng, and F. Beaufays, "Pronunciation learning for named-entities through crowd-sourcing," in *Proceedings of Interspeech*, 2014.

[13] Z. Kou, D. Stanton, F. Peng, F. Beaufays, and T. Strohman, "Fix it where it fails: Pronunciation learning by mining error corrections from speech logs," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[14] K. Rao, F. Peng, and F. Beaufays, "Automatic pronunciation verification for speech recognition," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[15] M. Mohri, F. C. N. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech and Language*, vol. 16, no. 1, pp. 69–88, 2002.

[16] A. Bruguier, F. Peng, and F. Beaufays, "Learning personalized pronunciations for contact name recognition," in *Interspeech*, 2016.

[17] B. Hayes, R. Kirchner, and D. Steriade, *Phonetically Based Phonology*, 2008.

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

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

[20] S. Toshniwal and K. Livescu, "Read, attend and pronounce: An attention-based approach for grapheme-to-phoneme conversion."

[21] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "Invited talk," in *OpenFst: A General and Efficient Weighted Finite-State Transducer Library*. [Online]. Available: http://cs.nyu.edu/~allauzen/pdf/openfst.pdf