End-to-End Neural Speech Synthesis

Alex Barron
Stanford University
admb@cs.stanford.edu

Abstract
In recent years, end-to-end neural networks have become the state of the art for speech recognition tasks and they are now widely deployed in industry (Amodei et al., 2016). Naturally, this has led to the creation of systems to do the opposite – end-to-end speech synthesis from raw text. Very recently, neural TTS systems have become highly competitive with their conventional counterparts, showing high naturalness scores in a variety of incarnations. The flexibility and relative simplicity of these models is compelling and their potential for transfer learning (Arik et al., 2017) suggests the future existence of high quality voice mimicking algorithms. Here, we present an open source end-to-end TTS system, based on Google’s Tacotron, which uses freely available datasets to train a working voice with reasonable naturalness.

1 Introduction
The text-to-speech task consists of transforming a string of input characters to a waveform representing the corresponding output speech. Until recently, concatenation systems have led field in terms of naturalness but as of late, parametric and neural systems have begun to outperform them. Despite their potential for excellent naturalness, there are still a number of bottlenecks preventing the widespread adoption of neural TTS. Firstly, the best performing neural systems have been auto-regressive (van den Oord et al., 2016) and prohibitively slow. Secondly, these architectures have typically replaced only part of the text-to-speech pipeline and often their components must be trained separately (Arik et al., 2017), making them difficult to implement. The newly released Tacotron architecture (Wang et al., 2017) makes significant strides to overcoming both of these obstacles and it is this system that we explore in depth.

2 Related Work
There currently exist four main approaches which use neural networks for all or most of the text-to-speech pipeline. The first is DeepMind’s WaveNet (van den Oord et al., 2016), a flexible model for audio generation which uses dilated, causal convolutions (with residual/skip connections) to form a conditional probability for the next time step value. For TTS tasks, WaveNet was conditioned on linguistic features from an existing TTS system and so is not fully end-to-end. In addition, its conditional model is auto-regressive and thus prohibitively slow for many applications. In return for these limitations, WaveNet produces very high quality audio samples, surpassing strong concatenative and parametric baselines in naturalness. DeepVoice (Arik et al., 2017) from Baidu implemented the entire TTS pipeline with neural networks in contrast to WaveNet. They also achieve production-ready speeds with many stacked QRNN (Quasi-recurrent network) layers which execute in a single batch, greatly reducing compute time. The biggest issue with their approach is that it requires separate training for the many different steps of their TTS pipeline - they train grapheme-to-phoneme, segmentation, duration and audio synthesis modules. This greatly increases the complexity of training and deploying their model and likely makes it harder to adapt existing models to new contexts. Their final naturalness scores (while not perfectly comparable) are significantly lower than WaveNet...
as a price for the greatly increased speed of their system.

A more end-to-end approach can be found in Char2Wav (Sotelo et al., 2017), but they still split their network into two separately trained components - a predictor of vocoder parameters and neural vocoder. They use a standard seq2seq attention encoder-decoder paradigm for the initial stage and a novel SampleRNN to compute the final synthesized signal. The SampleRNN uses a hierarchical structure to extract a more accurate signal from the vocoder features.

The only fully end-to-end model in existence as of this paper, is Tacotron (Wang et al., 2017) and it also has the significant advantage of being frame-level and thus highly efficient. For these reasons we settled on recreating the Tacotron architecture for our neural TTS efforts.

3 Approach

The Tacotron architecture (Wang et al., 2017) we deploy resembles a RNN-based seq2seq model used in machine translation. The model has 5 major components which we describe below.

3.1 CBHG

To extract rich features from text and spectrogram frames, Tacotron uses a combination of a convolution bank, multi-layer highway network and GRU dubbed a “CBHG” layer.

The convolution bank consists of $K$ convolutions of kernel size 1...$K$ respectively each with $C$ filters and followed by a ReLU activation. “Same” padding is used to preserve the original number of time steps across all convolutions. These outputs are then concatenated along the channels dimension, batch normalized and max-pooled with stride 1, kernel 2. This is followed by two 3x3 convolutions to project the output back to the same size as the input, so that a residual connection can be formed by adding the original inputs to our convolution bank outputs.

The output is then passed through 4 highway layers (Srivastava et al., 2015) and the final representation is extracted with a bi-directional GRU. This allows the network to construct features using both the forward and backward contexts.

3.2 Encoder

The encoder first embeds the raw characters as 256-vectors which are learned during training. These embeddings are fed to a ‘pre-net’ which consists of a 256 and 128 hidden unit ReLU fully connected layer each with dropout of probability 0.5.

This is passed to a CBHG module with $K=16$, $C=128$ and 128 hidden unit highway layers and GRU. This gives a final representation of the input with the number of time steps preserved.

3.3 Decoder

Tacotron uses a 3-layer residual GRU decoder with Bahdanau style attention. At each time step of the decoder, we compute the attention over the entire input text. We also feed the input frame at that time step (ground truth at train time, previous decoder output at test time) through another pre-net identical to that in the encoder. The attention and processed input frame are concatenated and passed through a linear layer to form the input to the residual GRUs. In our best architecture, (and in the original paper) we do not use scheduled sampling so dropout in the frame pre-net is crucial for generalization.

The choice of seq2seq target for the decoder is essential for the efficacy of the architecture. Tacotron uses 80 mel filters which provide more compact and relevant representation of speech compared to the linear spectrogram. This is what allows the decoder to learn a generalizable attention alignment.

Another important aspect of the decoder is that it outputs multiple spectrogram frames per time step. As in the original paper, we denote this number of frames as $r$. Particularly with the smaller datasets we were forced to work with, using a relatively high $r=5$ was very helpful in getting the model to learn an alignment. Outputting multiple frames meant the model was less likely to get stuck on one particular syllable in the speech in the attention alignment. It also increased training speed, reduced gradient explosion and seemed to have a regularizing effect.
3.4 Post-Process

To convert the mel filters into a linear spectrogram, Tacotron uses a second CBHG module with K=8, C=128 and 128 hidden unit highway layers and GRU.

3.5 Spectrogram Inverter

Since it is trained using only the log-magnitudes of the spectrogram, Tacotron uses Griffin-Lim (Griffin and Lim, 1984) to invert the spectrogram by iteratively recovering phase information through a series of stft and istft transformations. In the original paper they implement this algorithm in TensorFlow, but since performance was not the main goal of this project, we use the Librosa Python library. As noted in the original paper, there is considerable room for improvement in this spectrogram inversion portion of the model – it is the only portion of the pipeline not trained as an end-to-end neural network (Griffin-Lim has no parameters). Recent work from Baidu (Arik et al., 2017), replaced Griffin-Lim with a WaveNet style, separately trained neural network for spectrogram inversion and achieved higher naturalness as a result. One possible avenue for the continuation of the project would be to implement this spectrogram inverter and potentially attempt to train the entire system end-to-end, perhaps replacing the linear spectrogram loss with one on the final wav file.

3.6 Training

We train Tacotron with Adam and batch size 32. We use a L1 loss combining with equal weight the seq2seq loss on the mel filters and output loss on the linear spectrogram. Equal weighting works here since the L1 loss encourages optimization equally regardless of the size of the losses.

Gradient explosion was a serious issue due to the long decoder lengths and high dimensionality of hidden state in the Attention GRU so we globally clip gradients before every step to a value of 5 to mitigate the problem. While the original paper uses 0.001 initial learning rate for the first 500K steps, we found that beginning with 0.0005 learning rate significantly reduced gradient explosion while still overfitting the training set in a reasonable time. Since our datasets are smaller than those used in the original Tacotron paper, we train for only 120K steps (in practice until the gradient explodes), then reduce the learning rate further to 0.0003 and then 0.0001 after another 100K steps.

While training, we also found it crucial to normalize the mel and linear spectrograms by subtracting their mean and dividing by their standard deviation along each frequency bucket. We save the means and standard deviations of the linear spectrogram at train time and then use them to recover the unnormalized signal on new inputs when testing.

A large bottleneck in the training speed of Tacotron is the feeding of the speech inputs into the GPU. To combat this, we pre-compute the mel filters and linear spectrogram for each dataset and store them as numpy arrays on CPU RAM. We then pass them to TensorFlow through placeholders using the new Dataset class. This reduces training time over reading the inputs from disk as protobufs by a factor of 3. The largest dataset used contained around 60GB of spectrogram data so we ended up casting these to float16 for storage so that they fit in our servers’ 32GB of CPU RAM. The casting to reduced precision had no noticeable effect on the outputs of the spectrogram inverter. A better alternative in the future may be to compute spectrograms on the fly using TensorFlow’s very new audio APIs but these were not yet released when we implemented this part of the project.

4 Datasets

We train our TTS system separately on three female single speaker corpora.

4.1 CMU Arctic

The smallest dataset used contains 1132 utterances for a combined 50 minutes of single speaker audio. The neural network overfits quickly on this dataset without learning any recognizable alignment with the output loss decreasing far more aggressively than the sequence to sequence loss. Thus even on training set examples, when we remove the ground truth inputs at each frame, it is impossible to discern the original speech. Instead the output is gibberish although distinctly English.
4.2 Nancy

The “Nancy” corpus was the data used for the 2011 Blizzard challenge and contains approximately 12000 utterances with corresponding prompts. The speaker is a professional voice coach, and delivers each utterance with the same tone and accent. We throw out any prompts whose audio recordings surpass 350 stft frames (approximately 6 seconds of real time) to reduce training time and combat gradient explosion which is aggravated by long decoder lengths. This gives us around 9000 examples on which to train.

4.3 Byers

Finally we test on the audiobook data from the 2013 Blizzard Challenge. The segmented dataset provided consists of only about 5000 samples with corresponding prompts that are not too long to be practical. Thus we synthesized approximately 20 hours of audio-prompt pairs from the 100 hours of unsegmented speech provided. Using the open source Gentle aligner and the audiobook texts from Gutenberg, we were able to obtain accurate alignments. We then segmented each text into sentences and clauses to obtain discrete utterances, splitting on periods, question/exclamation marks and semi-colons and throwing out any segments over 6 seconds long.

Since these are sourced from audiobooks rather than independent prompt readings (as in the Arctic or Nancy corpus), this dataset has much less consistent tone and the speaker varies their voice drastically when voicing the dialogue of different characters.

5 Experiments

In our experiments, we found the most reliable indicator of generalization to be the monotonicity of the attention alignment learned by the decoder. We depict this in figures 1, 2 and 3 for various hyperparameter configurations and corpora as a heat map of encoder steps (in the text) vs decoder steps (in the output spectrogram). Monotonicity is a very good sign for generalization, since it indicates that the model is learning that text and speech are sequential and likely focusing on the correct word when producing speech. Since we have no pre-trained/computed alignments, this step is crucial if Tacotron is to produce any discernible output on new examples. As we discuss below, getting the model to learn an alignment rather than just overfitting turned out to be non-trivial.

5.1 Baseline

As a baseline, we implemented a simple attention seq2seq model on the Arctic dataset. This had no encoder or post-processor and directly trained on the linear spectrogram with a simple character em-
bedding for the text inputs. This model was able to overfit Arctic within 50K training steps and produce reasonably high quality replications of training inputs but failed miserably in generalization, producing only static. We attempted to address this by adding scheduled sampling during training, but this yielded very minor improvements.

5.2 Tacotron

Next we implemented the full Tacotron model on Arctic and trained for 100K iterations with \( r=5 \) (number of frames per decoder output) until results on the training set were clear. Unfortunately, we still could not learn an alignment on the training set as shown below and this translated into output on new examples which sounded distinctly human and English, but was merely unconnected, unintelligible syllables.

Such results were not unexpected given the diminutive size of the Arctic corpus of less than 1 hour of audio, but it was encouraging that the model was at least producing speech sounding outputs at test time and this confirmed that the auxiliary parts of the implementation such as spectrogram inversion were working correctly.

We then trained our model on the Nancy corpus. We trained for 120K iterations at learning rate 0.0005 before reducing to 0.0003 and training for a further 100K iterations. The results obtained with \( r=5 \) on this dataset ended up being the best we achieved with the 9 times increase in dataset scale greater improving the outputs obtained from the system. Most of the words in a new prompt were discernible and, though inconsistent, there was evidence of natural prosody.

Buoyed by the good results on the Nancy corpus, expectations were high for the same architecture trained on the Byers corpus, which contained about 3 times as many examples again. Unfortunately, the inconsistent accent and prosody in the audiobook samples severely impeded Tacotron’s ability to learn an alignment. Since the architecture is end-to-end, it has no explicit model for phone alignment and is thus very sensitive to changes in the pace of speech. The internal Google dataset used by the paper is similar in length to the Byers corpus, but is spoken by a professional female speaker whose prosody and pace are likely very consistent. Of course the only hard evidence for this is provided by the few samples released by the paper’s authors but our experiments seem to corroborate this hypothesis.
Figure 3: We show three examples from our best model, all entirely unseen during training. The first generates speech almost perfectly. The second entirely loses its alignment halfway through the sentence but recovers well at the end. The third loses its alignment towards the end of the prompt and is unable to recover. On all three examples (and on every example we tested), the decoder is unable to decide when to stop the output and we see the last few words repeated continuously.

5.3 Further Experimentation

We hypothesized that reducing r (the number of frames outputted per time step) on the Byers dataset, might allow the model to better adapt to the fast prosody of some of the samples. To test this, we ran Tacotron on both the Nancy and Byers corpora with r=2 and measured the results. Instead of improving the alignment learned, setting r=2 causes the model to overtrain much more quickly and, as shown in figure 2, learn no alignment at all. This may be due to it focusing on finer grain reconstructions which distracts the algorithm from learning a good overall alignment.

To aid this problem, we tried combining r=2 with scheduled sampling of 0.5 to encourage the model to learn to output consecutive training. Although this produced an excellent alignment on the training set, for reasons still somewhat mysterious the model was not generalizing well on new examples.

The addition of scheduled sampling, which was not used in the original Tactron model, deserves a more thorough investigation than we were able to give here due to the long training times required.

6 Evaluation

We test our best model on a variety of prompts, many inspired by those used in the original paper. Given that this model is trained on a corpus approximately a quarter of the size of that used by Google in the original paper, our version of Tacotron performs reasonably well. On unseen inputs, we find that most of the words are recognizable in the outputted speech and on simple sequences, the prosody can sound quite natural.

Our version of Tacotron was not able to differentiate words with identical spelling used in different contexts. For example “read” in “he reads books” and “he has read the whole thing” are both spoken with the first pronunciation.

Although we struggle on some more complicated pronunciations such as “alignment” or “intermittent”, the end-to-end training of Tacotron allows it to deal well with many out of vocabulary words. Its rendition of “otolaryngology” is surprisingly convincing given that it likely saw no similar words in training.

A problem with our current implementation is that the decoder is unable to determine when it should stop outputting syllables at test time and repeats, somewhat eerily, the last few words until the maximum decoder length is reached. We attempted to address this problem by reconstructing the zero frames during training so that silence would also be produced at test time, but our efforts have so far come to no avail. We were reassured by an advisor that training with larger datasets alleviates this problem significantly so it may not be the result of a specific bug.

In general, the speech produced by our model
demonstrates the fragility of architectures trained end-to-end on audio-text pairs with no alignment labels. At test time, when we do not provide the input at each frame, the decoder often gets thrown completely off track by a difficult word or phrase. It is then hard, though not impossible, for it to recover since each time step is receiving an increasingly corrupted input and we rely on the robustness of the attention mechanism in these situations to correct the outputs. Some examples of such behavior are illustrated in figure 3.

7 Conclusion

Our results demonstrate that it is possible to train Tacotron to produce samples of reasonable naturalness on publicly available TTS datasets. This is despite these datasets’ significant shortcomings in scale and quality when compared to their proprietary counterparts so these results bode well for the application of Tacotron in industry and open source TTS systems.

Now that we have a working implementation, there are many exciting paths for extension, particularly in the direction of multi-speaker, conditional TTS. Large multi-speaker datasets exist in the public domain (Panayotov et al., 2015) and recent work has demonstrated that it is possible to augment Tacotron with trained speaker embeddings at various points in the architecture to train on such corpora (Arik et al., 2017). With access to these much larger datasets, an open source neural TTS system is much more likely to be competitive with industry implementations.

With the appropriate datasets one could also investigate the addition of embeddings for emotion or tone into the Tacotron architecture. It is feasible that a system similar to Tacotron could soon be able to generate anyone’s voice with any emotion convincingly with minimal new training data. That would be a major step forward for the field of speech synthesis.

The code for this project is available on GitHub. I hope it will inspire future work in neural TTS and serve as a starting point for improvements and extensions.

Thank you to Andrew Maas and Ziang Xie for their advice throughout the project – it was invaluable.

References


