Does ‘end-to-end’ speech synthesis make any sense?

Simon King, Centre for Speech Technology Research, University of Edinburgh, UK
Simon King

• Prof. of Speech Processing
• Director of the Centre for Speech Technology Research

• CSTR website:  www.cstr.ed.ac.uk

• Teaching website:  speech.zone
Motivation

text

speech
Deep Voice: Real-time Neural Text-to-Speech

Abstract

We present Deep Voice, a production-quality text-to-speech system constructed entirely from deep neural networks. Deep Voice lays the groundwork for truly end-to-end neural speech synthesis. The system comprises five major building blocks: a segmentation model for locating phoneme boundaries, a grapheme-to-phoneme conversion model, a phoneme duration prediction model, a fundamental frequency prediction model, and an audio synthesis model.

Fundamentally, it allows human-technology interaction without requiring visual interfaces. Modern TTS systems are based on complex, multi-stage processing pipelines, each of which may rely on hand-engineered features and heuristics. Due to this complexity, developing new TTS systems can be very labor intensive and difficult.

Deep Voice is inspired by traditional text-to-speech pipelines and adopts the same structure, while replacing all components with neural networks and using simpler features: first we convert text to phoneme and then use an audio synthesis model to convert linguistic features into speech (Taylor, 2009). Unlike prior work (which uses hand-engineered features such as spectral envelope, spectral parameters, aperiodic parameters, etc.), our only features are phonemes with stress annotations, phoneme durations, and fundamental frequency (F0). This choice of features makes our system more readily applicable to new datasets, voices, and domains without any manual data annotation or additional feature engineering. We demonstrate this claim by retraining our entire pipeline without any hyperparameter changes on an entirely new dataset that contains solely audio and unaligned textual transcriptions and generating relatively high quality speech. In a conventional TTS system this adaptation requires days to weeks of tuning, whereas Deep Voice allows you to do it in only a few hours of manual effort and the time it takes models to train.

Real-time inference is a requirement for a production-quality TTS system; without it, the system is unusable for most applications of TTS. Prior work has demonstrated that a WaveNet (van den Oord et al., 2016) can generate close to human-level speech. However, WaveNet inference poses a daunting computational problem due to the high-frequency, autoregressive nature of the model, and it has been hitherto unknown whether such models can be used in a production system. We answer this question in the affirmative and demonstrate efficient, faster-than-real-time WaveNet inference kernels that produce high-quality 16 kHz audio and realize a 400X speedup over previous WaveNet inference implementations (Paine et al., 2016).
In our system, the duration prediction model and the F0 prediction model are performed by a single neural network trained with a joint loss. The grapheme-to-phoneme model is used as a fallback for words that are not present in a phoneme dictionary, such as CMUDict.

Our segmentation model is trained to output the alignment between a given utterance and a sequence of target phonemes. This task is similar to the problem of aligning speech to written output in speech recognition. In that domain, the connectionist temporal classification (CTC) loss has been shown to focus on character alignments rather than single phonemes. A network trained with CTC to generate sequences of phoneme pairs rather than single phonemes.

Figure 1. System diagram depicting (a) training procedure and (b) inference procedure, with inputs on the left and outputs on the right.

For training, we use the Adam optimization algorithm with a batch size of 64, a learning rate of $10^{-4}$, a weight decay of $10^{-2}$, an initial filter size of $3 \times 3$, and 512 channels in the remaining convolutional layers. The hidden state of the corresponding encoder forward layer serves as the initial state of every decoder layer. During training, we use dropout with probability 0.95 after each recurrent layer.

The architecture is trained with teacher forcing and decoding is performed using beam search. We use 3 bidirectional recurrent layers of the same size in the encoder and 3 unidirectional layers in the decoder. To compute the phoneme-pair error rate (PPER), we decode using an additional beam search with width 50 with the constraint that neighboring phoneme pairs overlap by at least one phoneme and that the initial decoded phoneme pair is the same as the first phoneme pair in the ground-truth phoneme-pairs sequence. To overcome this, we train to predict phoneme boundaries. To illustrate our label encoding, consider the string “Hello!”. To convert this to a sequence of phoneme pair labels, convert the utterance to phonemes (using a pronunciation dictionary such as CMUDict or a grapheme-to-phoneme model) and pad the phoneme sequence on either side with silence. Then construct consecutive phoneme pairs and get “(sil, HH), (HH, EH), (EH, L), (L, OW), (OW, sil)”.

Finally, construct consecutive phoneme pairs and get “(sil, HH, EH, L, OW, sil)”. The network will then tend to output phoneme pairs at timesteps close to the boundary between two phonemes in the ground truth. Although this is sufficient to roughly align the phonemes, explicit modeling of phoneme boundaries (or phoneme-pair errors) is desirable. However, we use a multi-layer bidirectional encoder with 7 blocks in total. On top of the input layer, there are two convolution layers with 1024 channels each, a batch normalization layer, and a ReLU activation function. The convolution layers use kernels with unit stride, a gated recurrent unit (GRU) nonlinearity and an equally sized layer of the same type as the decoder inputs. The recurrent layers use 512 GRU cells (for each direction).

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
**Abstract**

A text-to-speech synthesis system typically consists of multiple stages, such as a text analysis frontend, an acoustic model and an audio synthesis module. Building these components often requires extensive domain expertise and may contain brittle heuristics and brittle design choices. Second, it more easily allows for rich conditioning on various attributes, such as speaker, character or language, or high-level features like sentiment. This is because the multi-stage model typically has various components that are trained independently, so errors from each component may compound. The complexity of modern TTS design is also a large-scale inverse problem: a highly compressed rich, expressive yet often noisy data found in the real world.

This makes it difficult to train and costly to adapt the model. Furthermore, the model is trained on phoneme inputs rather than only on certain components. Similarly, adaptation to new data might also be easier. Finally, a single model is likely to be more robust than a multi-stage model where each component may compound. These advantages imply that an end-to-end model: it must cope with large variations at the signal level for a given input. Moreover, unlike end-to-end speech recognition [4] or machine translation [5], TTS outputs are continuous, and output sequences are usually much longer than those of the input. These attributes cause prediction errors to accumulate directly from characters. Given that is a particularly difficult learning task for an end-to-end model.
3. Model Architecture

The backbone of Tacotron is a seq2seq model with attention [7, 14]. Figure 1 depicts the model, which includes an encoder, an attention-based decoder, and a post-processing net. At a high-level, our model takes characters as input and produces spectrogram frames, which are then converted to waveforms. We describe these components below.

3.1. CBHG module

We first describe a building block dubbed CBHG, illustrated in Figure 2. CBHG consists of a bank of 1-D convolutional filters, followed by highway networks [15] and a bidirectional gated recurrent unit (GRU) [16] recurrent neural net (RNN). CBHG is a powerful module for extracting representations from sequences. The input sequence is first convolved with $K$ sets of 1-D convolutional filters, where the $k$-th set contains $C_k$ filters of width $k$ (i.e. $k=1, 2, ..., K$). These filters explicitly model local and contextual information (akin to modeling unigrams, bigrams, up to $K$-grams). The convolution outputs are stacked together and further max pooled along time to increase local invariances. Note that we use a stride of 1 to preserve the original time resolution. We further pass the processed sequence to a few fixed-width 1-D convolutions, whose outputs are added with the original input sequence via residual connections [17]. Batch normalization [18] is used for all convolutional layers. The convolution outputs are fed into a multi-layer highway network to extract high-level features. Finally, we stack a bidirectional GRU RNN on top to extract sequential features from both forward and backward context. CBHG is inspired from work in machine translation [8], where the main differences from [8] include using non-causal convolutions, batch normalization, residual connections, and stride=1 max pooling. We found that these modifications improved generalization.

3.2. Encoder

The goal of the encoder is to extract robust sequential representations of text. The input to the encoder is a character sequence, where each character is represented as a one-hot vector and embedded into a continuous vector. We then apply a set of non-linear transformations, collectively called a "pre-net", to each embedding. We use a bottleneck layer with dropout as the pre-net in this work, which helps convergence and improves generalization. A CBHG module transforms the pre-net outputs into the final encoder representation used by the attention module. We found that this CBHG-based encoder not only reduces overfitting, but also makes fewer mispronunciations than a standard multi-layer RNN encoder (see our linked page of audio samples).

3.3. Decoder

We use a content-based tanh attention decoder (see e.g. [14]), where a stateful recurrent layer produces the attention query at each decoder time step. We concatenate the context vector and the attention RNN cell output to form the input to the decoder RNNs. We use a stack of GRUs with vertical residual connections [5] for the decoder. We found the residual connections helpful. The decoder also applies attention [7] to the encoder representations. The attention weight is calculated as a weighted sum of the encoder representations, where the weights are determined by the similarity between the decoder representation and the encoder representations. The attention weight is then used to compute the weighted average of the encoder representations, which is then fed into the decoder RNNs. The decoder RNNs then produce the output sequence, which is then fed into the Griffin-Lim reconstruction algorithm to synthesize speech.
NATURAL TTS SYNTHESIS BY CONDITIONING WAVENET ON MEL SPECTROGRAM PREDICTIONS

Jonathan Shen\textsuperscript{1}, Ruoming Pang\textsuperscript{1}, Ron J. Weiss\textsuperscript{1}, Mike Schuster\textsuperscript{1}, Navdeep Jaitly\textsuperscript{1}, Zongheng Yang\textsuperscript{*2}, Zhifeng Chen\textsuperscript{1}, Yu Zhang\textsuperscript{1}, Yuxuan Wang\textsuperscript{1}, RJ Skerry-Ryan\textsuperscript{1}, Rif A. Saurous\textsuperscript{1}, Yannis Agiomyrgiannakis\textsuperscript{1}, and Yonghui Wu\textsuperscript{1}

\textsuperscript{1}Google, Inc., \textsuperscript{2}University of California, Berkeley,
{jonathanasasdf, rpang, yonghui}@google.com

ABSTRACT

This paper describes Tacotron 2, a neural network architecture for speech synthesis directly from text. The system is composed of a recurrent sequence-to-sequence feature prediction network that maps character embeddings to mel-scale spectrograms, followed by a modified WaveNet model acting as a vocoder to synthesize time-domain waveforms from those spectrograms. Our model achieves a mean opinion score (MOS) of 4.53 comparable to a MOS of 4.58 for professionally recorded speech. To validate our design choices, we present the authors note, this was simply a placeholder for future neural vocoder approaches, as Griffin-Lim produces characteristic artifacts and lower audio quality than approaches like WaveNet.

In this paper, we describe a unified, entirely neural approach to speech synthesis that combines the best of the previous approaches: a sequence-to-sequence Tacotron-style model\cite{12} that generates mel spectrograms, followed by a modified WaveNet vocoder\cite{10, 15}. Trained directly on normalized character sequences and corresponding speech waveforms, our model learns to synthesize natural sounding speech that is difficult to distinguish from real human speech.
This paper introduces WaveNet, a deep neural network for generating raw audio waveforms. The model is fully probabilistic and autoregressive, with the predictive distribution for each audio sample conditioned on all previous ones; nonetheless we show that it can be efficiently trained on data with tens of thousands of samples per second of audio. When applied to text-to-speech, it yields state-of-the-art performance, with human listeners rating it as significantly more natural sounding than the best parametric and concatenative systems for both English and Mandarin. A single WaveNet can capture the characteristics of many different speakers with equal fidelity, and can switch between them by conditioning on the speaker identity. When trained to model music, we find that it generates novel and often highly realistic musical fragments. We also show that it can be employed as a discriminative model, returning promising results for phoneme recognition.

1 INTRODUCTION

This work explores raw audio generation techniques, inspired by recent advances in neural autoregressive generative models that model complex distributions such as images (van den Oord et al., 2016a;b) and text (Józefowicz et al., 2016). Modeling joint probabilities over pixels or words using neural architectures as products of conditional distributions yields state-of-the-art generation. Remarkably, these architectures are able to model distributions over thousands of random variables (e.g. $64 \times 64$ pixels as in PixelRNN (van den Oord et al., 2016a)). The question this paper addresses is whether similar approaches can succeed in generating wideband raw audio waveforms, which are signals with very high temporal resolution, at least 16,000 samples per second (see Fig. 1).

Figure 1: A second of generated speech.

This paper introduces WaveNet, an audio generative model based on the PixelCNN (van den Oord et al., 2016a;b) architecture. The main contributions of this work are as follows:

- We show that WaveNets can generate raw speech signals with subjective naturalness never before reported in the field of text-to-speech (TTS), as assessed by human raters.

\[ \text{WaveNet: A Generative Model for Raw Audio} \]

Aäron van den Oord  
Sander Dieleman  
Heiga Zen†

Karen Simonyan  
Oriol Vinyals  
Alex Graves

Nal Kalchbrenner  
Andrew Senior  
Koray Kavukcuoglu

{avdnoord, sedielem, heigazen, simonyan, vinyals, gravesa, nalk, andrewsenior, korayk}@google.com

Google DeepMind, London, UK

† Google, London, UK

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
Because models with causal convolutions do not have recurrent connections, they are typically faster to train than RNNs, especially when applied to very long sequences. One of the problems of causal convolutions is that they require many layers, or large filters to increase the receptive field. For example, in Fig. 2 the receptive field is only 5 (= #layers + filter length - 1). In this paper we use dilated convolutions to increase the receptive field by orders of magnitude, without greatly increasing computational cost.

A dilated convolution (also called ‘a trous’, or convolution with holes) is a convolution where the filter is applied over an area larger than its length by skipping input values with a certain step. It is equivalent to a convolution with a larger filter derived from the original filter by dilating it with zeros, but is significantly more efficient. A dilated convolution effectively allows the network to operate on a coarser scale than with a normal convolution. This is similar to pooling or strided convolutions, but here the output has the same size as the input. As a special case, dilated convolution with dilation 1 yields the standard convolution. Fig. 3 depicts dilated causal convolutions for dilations 1, 2, 4, and 8.

Stacked dilated convolutions enable networks to have very large receptive fields with just a few layers, while preserving the input resolution throughout the network as well as computational efficiency. In this paper, the dilation is doubled for every layer up to a limit and then repeated: e.g. 1, 2, 4, ..., 512, 1, 2, 4, ..., 512, 1, 2, 4, ..., 512.

The intuition behind this configuration is two-fold. First, exponentially increasing the dilation factor results in exponential receptive field growth with depth (Yu & Koltun, 2016). For example each 1, 2, 4, ..., 512 block has receptive field of size 1024, and can be seen as a more efficient and discriminative (non-linear) counterpart of a 1⇥1024 convolution. Second, stacking these blocks further increases the model capacity and the receptive field size.

2.2 SOFTMAX DISTRIBUTIONS

One approach to modeling the conditional distributions \(p(x_t|x_1,\ldots,x_t_{-1})\) over the individual audio samples would be to use a mixture model such as a mixture density network (Bishop, 1994) or mixture of conditional Gaussian scale mixtures (MCGSM) (Theis & Bethge, 2015). However, van den Oord et al. (2016a) showed that a softmax distribution tends to work better, even when the data is implicitly continuous (as is the case for image pixel intensities or audio sample values). One of the reasons is that a categorical distribution is more flexible and can more easily model arbitrary distributions because it makes no assumptions about their shape.

Because raw audio is typically stored as a sequence of 16-bit integer values (one per timestep), a softmax layer would need to output 65,536 probabilities per timestep to model all possible values. To make this more tractable, we first apply a \(\mu\)-law companding transformation (ITU-T, 1988) to the data, and then quantize it to 256 possible values:

\[
f(x_t) = \text{sign}(x_t) \ln (1 + \mu |x_t|) \ln (1 + \mu)\]

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
Abstract

We present DeepVoice 3, a fully-convolutional attention-based neural text-to-speech (TTS) system. DeepVoice 3 matches state-of-the-art neural speech synthesis systems in naturalness while training an order of magnitude faster. We scale DeepVoice 3 to dataset sizes unprecedented for TTS, training on more than eight hundred hours of audio from over two thousand speakers. In addition, we identify common error modes of attention-based speech synthesis networks, demonstrate how to mitigate them, and compare several different waveform synthesis methods. We also describe how to scale inference to ten million queries per day on a single GPU server.

1 Introduction

Text-to-speech (TTS) systems convert written language into human speech. TTS systems are used in a variety of applications, such as human-technology interfaces, accessibility for the visually-impaired, media and entertainment. Traditional TTS systems are based on complex multi-stage hand-engineered pipelines (Taylor, 2009). Typically, these systems first transform text into a compact audio representation, and then convert this representation into audio using an audio waveform synthesis method called a vocoder.

Recent work on neural TTS has demonstrated impressive results, yielding pipelines with simpler features, fewer components, and higher quality synthesized speech. There is not yet a consensus on the optimal neural network architecture for TTS. However, sequence-to-sequence models (Wang et al., 2017; Sotelo et al., 2017; Arık et al., 2017) have shown promising results.

In this paper, we propose a novel, fully-convolutional architecture for speech synthesis, scale it to very large audio data sets, and address several real-world issues that arise when attempting to deploy an attention-based TTS system. Specifically, we make the following contributions:

1. We propose a fully-convolutional character-to-spectrogram architecture, which enables fully parallel computation and trains an order of magnitude faster than analogous architectures using recurrent cells (e.g., Wang et al., 2017).

2. We show that our architecture trains quickly and scales to the LibriSpeech ASR dataset (Panayotov et al., 2015), which consists of 820 hours of audio data from 2484 speakers.

3. We demonstrate that we can generate monotonic attention behavior, avoiding error modes commonly affecting sequence-to-sequence models.

4. We compare the quality of several waveform synthesis methods, including WORLD (Morise et al., 2016), Griffin-Lim (Griffin & Lim, 1984), and WaveNet (Oord et al., 2016).

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
The Deep Voice 3 architecture consists of three components:

- **Encoder**: A fully-convolutional encoder, which converts textual features to an internal learned representation.
- **Decoder**: A fully-convolutional causal decoder, which decodes the learned representation with a multi-hop convolutional attention mechanism into a low-dimensional audio representation (mel-scale spectrograms) in an autoregressive manner.
- **Converter**: A fully-convolutional post-processing network, which predicts final vocoder parameters (depending on the vocoder choice) from the decoder hidden states. Unlike the decoder, the converter is non-causal and can thus depend on future context information.

The overall objective function to be optimized is a linear combination of the losses from the decoder (Section 3.5) and the converter (Section 3.7). We separate decoder and converter and apply multi-task training, because it makes attention learning easier in practice. To be specific, the loss for mel-spectrogram prediction guides training of the attention mechanism, because the attention is trained with the gradients from mel-spectrogram prediction besides vocoder parameter prediction.

In multi-speaker scenario, trainable speaker embeddings as in Arık et al. (2017) are used across encoder, decoder and converter. Next, we describe each of these components and the data preprocessing in detail. Model hyperparameters are available in Table 4 within Appendix C.

**3.1 Text Preprocessing**

Text preprocessing is crucial for good performance. Feeding raw text (characters with spacing and punctuation) yields acceptable performance on many utterances. However, some utterances may have mispronunciations of rare words, or may yield skipped words and repeated words. We alleviate these issues by normalizing the input text as follows:

1. We uppercase all characters in the input text.
2. We remove all intermediate punctuation marks.
3. We end every utterance with a period or question mark.
4. We replace spaces between words with special separator characters which indicate the duration of pauses inserted by the speaker between words. We use four different word separators, indicating (i) slurred-together words, (ii) standard pronunciation and space characters, (iii) a short pause between words, and (iv) a long pause between words. For example, the sentence "Either way, you should shoot very slowly," with a long pause after "way" and a short pause after "shoot", would be written as "Either way%you should shoot/very slowly%." with % representing a long pause and / representing a short pause for encoding convenience.

The pause durations can be obtained through either manual labeling or by estimated by a text-audio aligner such as Gentle (Ochshorn & Hawkins, 2017). Our single-speaker dataset is labeled by hand and our multi-speaker datasets are annotated using Gentle.
Contents

1. “Traditional” methods
2. Here comes machine learning
3. The best of both
Part 1 — “Traditional” methods — the text-to-speech pipeline
The classic two-stage pipeline of unit selection

Front end → Waveform generator

text → linguistic specification → waveform

Author of the...
The end-to-end problem we want to solve

Text-to-Speech

Author of the...
A problem we can actually solve with machine learning

linguistic specification

acoustic features

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
The classic three-stage pipeline of statistical parametric speech synthesis

1. **Front end**
2. **Statistical model**
3. **Waveform generator**

*Text* → *linguistic specification* → *acoustic features* → *waveform*
The classic three-stage pipeline of statistical parametric speech synthesis

Text → Feature Extraction → Linguistic Specification → Statistical Model → Acoustic Features → Feature Extraction → Waveform
We can describe the core problem as **sequence-to-sequence regression**.

Different lengths, because of differing ‘clock rates’.

**output sequence**
(acoustic features)

**input sequence**
(linguistic features)
From text to speech

- Text processing
  - pipeline architecture
  - linguistic specification
- Regression
  - duration model
  - acoustic model
- Waveform generation
  - acoustic features
  - signal processing
From text to speech

- **Text processing**
  - pipeline architecture
  - linguistic specification
- **Regression**
  - duration model
  - acoustic model
- **Waveform generation**
  - acoustic features
  - signal processing
The linguistic specification

Author

sil

syl₁

syl₀

ao

other

of

DT

of

the

syl₀

syl₀

ah

f

dh

ax

...
Extracting features from text using the front end

Author of the...
Text processing pipeline

**Front end**
- tokenize
- POS tag
- LTS
- Phrase breaks
- intonation

Individually learned from **labelled** data

**linguistic specification**
Text processing pipeline

Front end

- tokenize
- POS tag
- LTS
- Phrase breaks
- intonation
Tokenize & Normalize

• Step 1: divide input stream into tokens, which are potential words

• For English and many other languages
  • rule based
  • whitespace and punctuation are good features

• For some other languages, especially those that don’t use whitespace
  • may be more difficult
  • other techniques required (out of scope here)
Tokenize & Normalize

• Step 2: classify every token, finding **Non-Standard Words** that need further processing

In 2011, I spent £100 at IKEA on 100 DVD holders.

NYER   MONEY   ASWD   NUM   LSEQ

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
Tokenize & Normalize

- Step 3: a set of specialised modules to process NSWs of each type

\[
\begin{align*}
2011 & \Rightarrow \text{NYER} & \Rightarrow \text{twenty eleven} \\
£100 & \Rightarrow \text{MONEY} & \Rightarrow \text{one hundred pounds} \\
\text{IKEA} & \Rightarrow \text{ASWD} & \Rightarrow \text{apply letter-to-sound} \\
100 & \Rightarrow \text{NUM} & \Rightarrow \text{one hundred} \\
\text{DVD} & \Rightarrow \text{LSEQ} & \Rightarrow \text{D. V. D.} \Rightarrow \text{dee vee dee}
\end{align*}
\]
POS tagging

- Part-of-speech tagger
- Accuracy can be very high
- Trained on annotated text data
- **Categories** are designed for text, not speech
Pronunciation / LTS

• Pronunciation model
  • dictionary look-up, *plus*
  • letter-to-sound model

• But
  • need deep **knowledge** of the language to design the phoneme set
  • human **expert** must write dictionary

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
The linguistic specification

The diagram illustrates a linguistic structure with the following components:

- **Author**
  - **NN** (Noun Phrase) labeled as Author
  - **syl** (syllable) nodes labeled as 0, 1
  - **sil** (syl) nodes labeled as ao, th, er

- **of**
  - **of** labeled as DT (Determiner)
  - **syl** (syllable) nodes labeled as 0
  - **syl** (syllable) nodes labeled as ah, f

- **the**
  - **the** labeled as DT (Determiner)
  - **syl** (syllable) nodes labeled as 0
  - **syl** (syllable) nodes labeled as dh, ax
Linguistic feature engineering

Author of the...
Flatten & encode: convert linguistic specification to vector sequence

“linguistic timescale”
Upsample: add duration information

linguistic timescale

predict durations

[0 0 1 0 0 1 0 1 1 0 ... 0.2 0.0]
[0 0 1 0 0 1 0 1 1 0 ... 0.2 0.1]
...
[0 0 1 0 0 1 0 1 1 0 ... 0.2 1.0]
[0 0 1 0 0 1 0 1 1 0 ... 0.4 0.0]
[0 0 1 0 0 1 0 1 1 0 ... 0.4 0.5]
[0 0 1 0 0 1 0 1 1 0 ... 0.4 1.0]
...
[0 0 1 0 0 1 0 1 1 0 ... 1.0 1.0]
[0 0 0 1 1 1 0 1 0 0 ... 0.2 0.0]
[0 0 0 1 1 1 0 1 0 0 ... 0.2 0.2]
[0 0 0 1 1 1 0 1 0 0 ... 0.2 0.4]
...

acoustic framerate

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
From text to speech

- Text processing
  - pipeline architecture
  - linguistic specification
- Regression
  - duration model
  - acoustic model
- Waveform generation
  - acoustic features
  - signal processing
Acoustic model: a simple feed-forward neural network
Synthesis with a simple neural network — frame-by-frame
From text to speech

- Text processing
  - pipeline architecture
  - linguistic specification
- Regression
  - duration model
  - acoustic model
- Waveform generation
  - acoustic features
  - signal processing
What are the acoustic features?
What are the acoustic features?

STRAIGHT is a VOICED input signal analysis.

- F0 analysis
- Spectral envelope analysis
- Non-periodicity analysis
- Non-periodic component generator
- Filter
- Shaper and mixer
- Periodic pulse generator
- Non-periodic component generator
- Output signal
Putting it all together: text-to-speech with a neural network

“Author of the …”

Front end

- tokenize
- POS tag
- LTS
- Phrase breaks
- intonation
Putting it all together: text-to-speech with a neural network
Putting it all together: text-to-speech with a neural network

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
Putting it all together: text-to-speech with a neural network
Part 2 — Here comes machine learning

the so-called ‘end-to-end’ approaches
text

speech
Stages of processing

- feature extraction
- regression
- waveform generation
Tacotron 2

Front end

Regression

Waveform generator
Tacotron 2

Encoder

Decoder

Vocoder
The Deep Voice 3 architecture consists of three components:

- **Encoder**: A fully-convolutional encoder, which converts textual features to an internal learned representation.
- **Decoder**: A fully-convolutional causal decoder, which decodes the learned representation with a multi-hop convolutional attention mechanism into a low-dimensional audio representation (mel-scale spectrograms) in an autoregressive manner.
- **Converter**: A fully-convolutional post-processing network, which predicts final vocoder parameters (depending on the vocoder choice) from the decoder hidden states. Unlike the decoder, the converter is non-causal and can thus depend on future context information.

The overall objective function to be optimized is a linear combination of the losses from the decoder (Section 3.5) and the converter (Section 3.7). We separate decoder and converter and apply multi-task training, because it makes attention learning easier in practice. To be specific, the loss for mel-spectrogram prediction guides training of the attention mechanism, because the attention is trained with the gradients from mel-spectrogram prediction besides vocoder parameter prediction.

In a multi-speaker scenario, trainable speaker embeddings as in Arık et al. (2017) are used across encoder, decoder and converter. Next, we describe each of these components and the data preprocessing in detail. Model hyperparameters are available in Table 4 within Appendix C.
Part 3 — The best of both
Traditional approach

Front end

- tokenize
- POS tag
- LTS
- Phrase breaks
- intonation

Encoder

Decoder

Vocoder

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
Traditional — explicit pronunciation dictionary + letter-to-sound model

from 20k up to 200k entries (unique types) + a statistical model learned from this data
New — encoder learns a character sequence embedding
Comparing traditional and new approaches to pronunciation

• **The traditional approach**
  - explicit representation of pronunciation (syllables + phonemes + lexical stress)
  - write a pronunciation dictionary (not usually speaker-specific)
  - learn to extrapolate from that with a statistical model

• **The end-to-end approach**
  - annotate a **small** quantity of text with speech using a **single** human talker
    - e.g., 40 hours of speech ≈ 2400 minutes ≈ 360k word tokens
    - which may have about 40k unique word types (random newswire text in English Gigaword)
  - number of word types observed in the speech corpus is substantial
    - so it should be feasible to learn a general pronunciation model from spoken data
Tacotron 2

While our samples sound great, there are still some difficult problems to be tackled. For example, our system has difficulties pronouncing complex words (such as “decorum” and “merlot”),


• Are “decorum” and “merlot” really complex words?
• The Oxford British English dictionary says

  DECORUM  
  MERLOT  

• Which doesn’t seem particularly difficult …
Traditional approach to Non-Standard Words — detect+classify+expand

2011 ⇒ NYER ⇒ twenty eleven
£100 ⇒ MONEY ⇒ one hundred pounds
IKEA ⇒ ASWD ⇒ apply letter-to-sound
100 ⇒ NUM ⇒ one hundred

Sproat et al, “Normalization of non-standard words”
Computer Speech and Language (2001) 15, 287–333
doi:10.1006/csla.2001.0169

| Table I. Taxonomy of non-standard words used in hand-tagging and in the text normalization models |
|----------------------------------|-------------------------------|-------------------|
| EXPN abbreviation                 | adv, N.Y, mph, gov’t          |
| alpha LSEQ letter sequence        | CIA, D.C, CDs                 |
| ASWD read as word                 | CAT, proper names             |
| MSPL misspelling                  | geography                     |
| NUM number (cardinal)             | 12, 45, 1/2, 0·6              |
| NORD number (ordinal)             | May 7, 3rd, Bill Gates III    |
| NTEL telephone (or part of)       | 212 555-4523                  |
| NDIG number as digits             | Room 101                      |
| NIDE identifier                   | 747, 386, 15, pc110, 3A       |
| NADDR number as street address    | 5000 Pennsylvania, 4523 Forbes|
| NZIP zip code or PO Box           | 91020                         |
| NTIME a (compound) time           | 3·20, 11·45                   |
| NDATE a (compound) date           | 2/2/99, 14/03/87 (or US)      |
| NYER year(s)                      | 1998, 80s, 1900s, 2003        |
| MONEY money (US or other)         | $3·45, HK$300, Y20,000, $200K |
| BMONEY money tr/m/billions        | $3·45 billion                 |
| PRCT percentage                  | 75%, 3·4%                     |
| SPLT mixed or “split”             | WS99, x220, 2-car             |
| SLNT not spoken, word boundary    | M.bath, KENT*RLTY, really_   |
| PUNC not spoken, non-standard punctuation: “***” in | $99,9K***Whites, “…” in DECIDE...Year |
| FNSP funny spelling               | slloooooww, sh*t              |
| URL url, pathname or email        | http://apj.co.uk, /usr/local, phj@tpt.com |
| NONE should be ignored ascii art, formatting junk |
A machine-learning approach to Non-Standard Words (arXiv:1611.00068v2)

RNN Approaches to Text Normalization: A Challenge

Richard Sproat, Navdeep Jaitly
Google, Inc.
{rws,ndjaitly}@google.com

Abstract

This paper presents a challenge to the community: given a large corpus of written text aligned to its normalized spoken form, train an RNN to learn the correct normalization function. We present a data set of general text where the normalizations were generated using an existing text normalization component of a text-to-speech system. This data set will be released open-source in the near future.

1 Introduction

Within the last few years a major shift has taken place in speech and language technology: the field has been taken over by deep learning approaches. For example, at a recent NAACL conference well more than half the papers related in some way to word embeddings or deep or recurrent neural networks.

This change is surely justified by the impressive performance gains to be had by deep learning, something that has been demonstrated in a range of areas from image processing, handwriting recogni-
Comparing traditional and new approaches Non-Standard Words (NSWs)

• The traditional approach
  • **explicit** capture of human knowledge
  • annotate a **large** quantity of NSWs with **categories** using **many** human labellers
  • learn an automatic NSW classifier from that data
  • write a specialised expander for each type (simple, deterministic rules are often enough)

• The end-to-end approach *
  • annotate a relatively **small** quantity of text with **speech** using a **single** human talker
  • **implicit** capture of human knowledge

* actually, most end-to-end systems don’t even attempt this; they require normalised text
The best of both

Tokenise, normalise, letter-to-sound

Encoder

Decode

Encoder

Decoder

Vocoder

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
3.1 Text Preprocessing

Text preprocessing is crucial for good performance. Feeding raw text (characters with spacing and punctuation) yields acceptable performance on many utterances. However, some utterances may have mispronunciations of rare words, or may yield skipped words and repeated words. We alleviate these issues by normalizing the input text as follows:

1. We uppercase all characters in the input text.
2. We remove all intermediate punctuation marks.
3. We end every utterance with a period or question mark.
4. We replace spaces between words with special separator characters which indicate the duration of pauses inserted by the speaker between words. We use four different word separators, indicating (i) slurred-together words, (ii) standard pronunciation and space characters, (iii) a short pause between words, and (iv) a long pause between words. For example, the sentence “Either way, you should shoot very slowly,” with a long pause after “way” and a short pause after “shoot”, would be written as “Either way%you should shoot/very slowly%.” with % representing a long pause and / representing a short pause for encoding convenience. 2
The best of both — characters and phonemes

3.2 Joint Representation of Characters and Phonemes

Deployed TTS systems (e.g., Capes et al., 2017; Gonzalvo et al., 2016) should include a way to modify pronunciations to correct common mistakes (which typically involve proper nouns, foreign words, and domain-specific jargon). A conventional way to do this is to maintain a dictionary to map words to their phonetic representations.

Our model can directly convert characters (including punctuation and spacing) to acoustic features, and hence learns an implicit grapheme-to-phoneme model. This implicit conversion is difficult to correct when the model makes mistakes. Thus, in addition to character models, we also train phoneme-only models and mixed character-and-phoneme models by allowing phoneme input option explicitly. These models are identical to character-only models, except that the input layer of the encoder sometimes receives phoneme and phoneme stress embeddings instead of character embeddings.
Traditional decoder operates frame-by-frame on upsampled linguistic features
New decoders that bridge linguistic and acoustic timescales
Ronanki, Watts & King. Interspeech 2017

Figure 1: Schematic diagram of a hierarchical encoder-decoder for SPSS. The lower part of the network is the hierarchical encoder, with each layer operating at a particular linguistic level, and phone-level recurrence as its final encoded output. The upper part of the network is the decoder, generating speech parameters using frame-level recurrence. Solid black lines indicate the propagation of hidden activations between layers, and dashed colored lines indicate the injection of linguistic features at the appropriate level. The patterns of connections between word, syllable and phone layers is determined by the known linguistic structure of the current utterance. Each block of green units represents a phone, with the number of units corresponding to its duration in frames (although not drawn to scale).

2. The Usual Approach
2.1. Pre-processing input features
The alignments between words, syllables, and phones are given by the linguistic specification, provided by the TTS front end. The usual approach in neural network-based TTS – regardless of neural network architecture – is to pre-process the input representation by flattening followed by upsampling [2, 4].

Flattening: attaching linguistic features to the phone, creating a linear sequence of context-dependent phones, and discarding explicit structure (e.g., that phones belong to syllables).

Upsampling: duplicating linguistic features for a number of consecutive acoustic frames, to map from linguistic timescale to vocoder frame rate (or possibly to waveform sampling rate, if directly generating a waveform). Note that upsampling cannot add information; in fact, it results in the same amount of information being represented less efficiently.

It is common practice to add within-phone positional features, derived from existing features, when upsampling to compensate for limitations of the regression model.

2.2. Input-to-output alignment in the usual approach
By making an assumption that is almost universal in speech technology – viz. that a speech signal is a sequence of non-overlapping units – the input-output alignment can be precomputed using HMM-based forced alignment for the training data, and can be determined during synthesis of test utterances using a duration model learned from the same data.

2.3. Regression
There are many possibilities for the architecture of the network used to perform regression from flattened-and-upsampled linguistic features to either frame-level vocoder speech parameters [2, 7, 9, for example] or to waveform samples [18]. Nevertheless, what all of these architectures have in common is the requirement for input and output to be at the same rate, meaning that the input must be upsampled.

3. Proposed Hierarchical Encoder-Decoder
Figure 1 provides a schematic diagram of the proposed hierarchical encoder-decoder neural network. The key ideas are to avoid the flattening pre-processing step entirely, and to integrate the upsampling into the model itself.

3.1. Hierarchical encoder
Figure 1 shows how the proposed method employs a hierarchical encoder that accepts input at the original linguistic timescales of word, syllable and phone. The upsampling between these levels is performed progressively, rather than all at once. Features at each level are injected into the model at the appropriate timescale by appending them to the hidden representation at that level. This has a variety of possible advantages over the usual approach.

Features from longer timescales (e.g., word), have a potentially weakened or diluted effect in the usual approach because they are constant for many consecutive frames, yet must share...
New decoders are true sequence models — Tacotron 2
Traditional vocoders use carefully crafted signal processing

```
Encoder

STRAIGHT is a VOCODER

Decoder

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
```
New vocoders are learned from data but contain sub-components that mimic traditional approaches.
WAVENET: A GENERATIVE MODEL FOR RAW AUDIO

Aäron van den Oord
Karen Simonyan
Nal Kalchbrenner
Sander Dieleman
Oriol Vinyals
Andrew Senior
Heiga Zen†

{avdnoord, sediele, heigazen, simonyan, vinyals, gravesa, nalk, andrewsenior, korayk}@google.com

Google DeepMind, London, UK
† Google, London, UK

ABSTRACT

This paper introduces WaveNet, a deep neural network for generating raw audio waveforms. The model is fully probabilistic and autoregressive, with the predictive distribution for each audio sample conditioned on all previous ones; nonetheless we show that it can be efficiently trained on data with tens of thousands of samples per second of audio. When applied to text-to-speech, it yields state-of-the-art performance, with human listeners rating it as significantly more natural sounding than the best parametric and concatenative systems for both English and Mandarin. A single WaveNet can capture the characteristics of many different speakers with equal fidelity, and can switch between them by conditioning on the speaker identity. When trained to model music, we find that it generates novel and often highly realistic musical fragments. We also show that it can be employed as a discriminative model, returning promising results for phoneme recognition.
Because models with causal convolutions do not have recurrent connections, they are typically faster to train than RNNs, especially when applied to very long sequences. One of the problems of causal convolutions is that they require many layers, or large filters to increase the receptive field. For example, in Fig. 2 the receptive field is only 5 (= #layers + filter length - 1). In this paper we use dilated convolutions to increase the receptive field by orders of magnitude, without greatly increasing computational cost.

A dilated convolution (also called \textit{\`a trous}, or convolution with holes) is a convolution where the filter is applied over an area larger than its length by skipping input values with a certain step. It is equivalent to a convolution with a larger filter derived from the original filter by dilating it with zeros, but is significantly more efficient. A dilated convolution effectively allows the network to operate on a coarser scale than with a normal convolution. This is similar to pooling or strided convolutions, but here the output has the same size as the input. As a special case, dilated convolution with dilation 1 yields the standard convolution. Fig. 3 depicts dilated causal convolutions for dilations 1, 2, 4, and 8. Dilated convolutions have previously been used in various contexts, e.g. signal processing (Holschneider et al., 1989; Dutilleux, 1989), and image segmentation (Chen et al., 2015; Yu & Koltun, 2016).

2.2 \textit{S}OFTMAX DISTRIBUTIONS

One approach to modeling the conditional distributions $p(x_t | x_1, ..., x_t-1)$ over the individual audio samples would be to use a mixture model such as a mixture density network (Bishop, 1994) or mixture of conditional Gaussian scale mixtures (MCGSM) (Theis & Bethge, 2015). However, van den Oord et al. (2016a) showed that a softmax distribution tends to work better, even when the data is implicitly continuous (as is the case for image pixel intensities or audio sample values). One of the reasons is that a categorical distribution is more flexible and can more easily model arbitrary distributions because it makes no assumptions about their shape.

Because raw audio is typically stored as a sequence of 16-bit integer values (one per timestep), a softmax layer would need to output 65,536 probabilities per timestep to model all possible values. To make this more tractable, we first apply a $\mu$-law companding transformation (ITU-T, 1988) to the data, and then quantize it to 256 possible values:

$$f(x_t) = \text{sign}(x_t) \ln \left(1 + \mu|x_t|\right) / \ln(1 + \mu),$$
Because models with causal convolutions do not have recurrent connections, they are typically faster to train than RNNs, especially when applied to very long sequences. One of the problems of causal convolutions is that they require many layers, or large filters to increase the receptive field. For example, in Fig. 2 the receptive field is only 5 (= #layers + filter length - 1). In this paper we use dilated convolutions to increase the receptive field by orders of magnitude, without greatly increasing computational cost.

A dilated convolution (also called *a trous*, or convolution with holes) is a convolution where the filter is applied over an area larger than its length by skipping input values with a certain step. It is equivalent to a convolution with a larger filter derived from the original filter by dilating it with zeros, but is significantly more efficient. A dilated convolution effectively allows the network to operate on a coarser scale than with a normal convolution. This is similar to pooling or strided convolutions, but here the output has the same size as the input. As a special case, dilated convolution with dilation 1 yields the standard convolution. Fig. 3 depicts dilated causal convolutions for dilations 1, 2, 4, and 8.

Stacked dilated convolutions enable networks to have very large receptive fields with just a few layers, while preserving the input resolution throughout the network as well as computational efficiency. In this paper, the dilation is doubled for every layer up to a limit and then repeated: e.g. 1, 2, 4, ..., 512, 1, 2, 4, ..., 512, 1, 2, 4, ..., 512.

The intuition behind this configuration is two-fold. First, exponentially increasing the dilation factor results in exponential receptive field growth with depth (Yu & Koltun, 2016). For example each 1, 2, 4, ..., 512 block has receptive field of size 1024, and can be seen as a more efficient and discriminative (non-linear) counterpart of a 1⇥1024 convolution. Second, stacking these blocks further increases the model capacity and the receptive field size.

### 2.2 SOFTMAX DISTRIBUTIONS

One approach to modeling the conditional distributions \( p(x_t|x_1,\ldots,x_{t-1}) \) over the individual audio samples would be to use a mixture model such as a mixture density network (Bishop, 1994) or mixture of conditional Gaussian scale mixtures (MCGSM) (Theis & Bethge, 2015). However, van den Oord et al. (2016a) showed that a softmax distribution tends to work better, even when the data is implicitly continuous (as is the case for image pixel intensities or audio sample values). One of the reasons is that a categorical distribution is more flexible and can more easily model arbitrary distributions because it makes no assumptions about their shape.

Because raw audio is typically stored as a sequence of 16-bit integer values (one per timestep), a softmax layer would need to output 65,536 probabilities per timestep to model all possible values. To make this more tractable, we first apply a \( \mu \)-law companding transformation (ITU-T, 1988) to the data, and then quantize it to 256 possible values:

\[
\text{f}(x_t) = \text{sign}(x_t) \ln (1 + \mu |x_t|) \\
\ln (1 + \mu)
\]

“one-hot” coding of 8 bit quantised waveform sample = 1-of-256
The best of both

**Wavenet conditioned** not on text, but on spectrogram

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
Are you traditional, purist, or pragmatic?

- Traditional methods learn from a **wide variety of data**
  - but there are many inconsistencies between these data sources
    - e.g., dictionary does not match the speaker’s accent

- Purist end-to-end approaches learn **only** from **parallel text + speech data**
  - but they cannot make use of additional text-only (or speech-only) data

- “Best of both” approaches take a **pragmatic view**
  - traditional approaches to normalise the text into “phones++”
  - end-to-end learning to regress from that to the spectrogram
  - neural vocoder to generate the waveform, conditioned on that spectrogram

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
Opportunities

(1) Traditional text processing works well
   • unfortunately, these methods are not differentiable
   • and therefore not learnable end-to-end

(2) Even the best end-to-end models still use flat input sequences
   • but linguistic structures are not flat
   • e.g., phrases - words - syllables - phones

(3) Even in end-to-end approaches, we have many choices about what we optimise
Opportunity — differentiable traditional approaches

traditional and differentiable?

Encoder

Decoder

Vocoder

© Copyright Simon King, University of Edinburgh, 2018. Personal use only. Not for re-use or redistribution.
Opportunity — make use of linguistic structure
Opportunity — choose what to optimise

Encoder → Decoder → Vocoder
What we are optimising vs. what do we want to optimise

• explicit intermediate representations imply minimisation of error in that domain
  • e.g., regression model minimises error in the spectrogram domain

• intermediate representations are therefore a critical design choice, at every stage
  • so, if we care about minimising pronunciation errors, perhaps an explicit representation of pronunciation is not a bad idea
  • or, if we wish to minimise perceived error in the speech output, then a perceptually-relevant representation would be nice (the log Mel spectrogram is on the right lines, but too simplistic)