Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики факультету інформатики

Modelling prosody in the task of human speech synthesis with the use of machine learning

Теоретична частина

Курсова робота

за спеціальністю 113 „Прикладна математика ”

Керівник курсової роботи
к.ф.-м.н., доц. Крюкова Г.В.

_____

(підпис)

“____” _____ 2020 р.

Виконав
студент 4 курсу
факультету інформатики
Процик О.І.

Table of Contents

Кафедра математики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри математики
проф., д.ф.-м.н. Олійник Б.В.
_____
(підпис)
“____”_____ 2020р.

Індивідуальне завдання

на курсову роботу

студенту Процику О.І. факультету інформатики 4 курсу БП.
Тема: Modelling prosody in the task of human speech synthesis with the use of machine learning
Зміст ТЧ до курсової роботи:

- Індивідуальне завдання
- Календарний план
- Вступ
- Теорія
- Висновки
- Список використаної літератури

Дата видачі “____”_____ 2020р. Керівник _____
(підпис)

Work Schedule

Number	Coursework writing stages	Date
1	Started researching Speech synthesis	28.09.2019
2	First working prototype	04.11.2019
3	Received coursework topic	18.10.2019
4	Completed training of the model	18.03.2020
5	Completed writing underlying theory	12.04.2020
6	Completed coursework draft	17.04.2020
7	Completed final coursework draft	17.04.2020

Introduction

Generating high fidelity speech using a text-to-speech (TTS) system remains a challenging task despite the decades of research and investigations. Modern TTS systems are very complex. For example, it is a common practice for a statistical TTS system to have a linguistic extractor in the front, which extracts different linguistic features. It is followed by a duration model to estimate the speech length in time of a given text and an acoustic feature prediction model. Given these features, it is all fed into a vocoder, which synthesizes speech out of acoustic features. All these components are trained independently and require extensive field knowledge to be sophisticated enough and produce considerable results. Because it has a modular design, it is prone to errors which will proceed in the following modules and can accumulate.

There are many pros to an end-to-end system, which is compounded of just a text-audio pair. First of all, it alleviates the need for complex acoustic and linguistic features which may bottleneck other modules. Second of all, it is much easier to implement different conditionings such as speaker and sentiment. It also has the benefit of easily transferable onto new data. Finally, it is much more robust that a modular design and alleviates most accumulating errors.

A straightforward approach to an end-to-end system is using machine learning. Given the vast amounts of data we have today, machine learning system have been on a rise. They are able to model complex systems and behaviors without the need of extensive task-specific field expertise, although it is preferable.

In order to produce realistic speech which has clear speech (intelligibility), sounds natural (naturalness) and is expressive, the model must impute many implicit and

explicit factors which are not given in a simple text input. Such factors include the intonation, stress, rhythm and style of the speech, and are collectively referred to as **prosody**.

Speech synthesis using only text is a challenging and an undetermined task. The meaning of an utterance usually relies heavily on expressiveness, intonation, stress and rhythm. These factors also influence the naturalness of speech, because a human would easily differ a recording from a monotonic, robotic speech generated from a plain TTS system conditioned only on text. For example, let's take the phrase "John was standing under the pole". If the question this phrase answers is "Where did John stand?", then the speaker would underline and give a higher intonation to the word "pole" to indicate that it is the answer. Another example would be a question "Would you like water or cola?". If there are only two options, the speaker will start lowering his pitch towards the end of the sentence, so the word "water" will have a lower pitch and be more intoned than "cola", but if there are more options, they will have the same pitch and intonation to provide this information of choice. So we can define prosody as the variation in speech signals, which is left after accounting for variation due to phonetics and speaker.

The main problem that arises in this scenario is how can we measure speech quality and prosody. This remains an unresolved task to automate and currently the most reliable metric is mean opinion score (MOS), which involves taking a group of people and letting them score the naturalness, intelligibility and expressiveness of the speech. In this work some sophisticated automatic approaches and analysis will be used to measure speech and prosody quality.

My work is based off of a relatively new development in TTS called "Tacotron 2"

which is implemented and used as a baseline to further experiment with modeling prosody. Tacotron 2 model has text as input, and outputs an audio spectrogram, which is a two dimensional representation of audio spectra frequencies over time. More details about this model, augmentations, and audio preparation is explained in details in other sections.

Theory

Audio Preparation

Full end-to-end process is currently too hard for machine learning models, to take in text pairs and produce audio waveform as the output. This requires processing power and GPU memory that is currently very hard and/or expensive to obtain. So it is usually split into two modules: given the text synthesize a spectrogram and feed it into a vocoder to further produce a waveform.

A spectrogram is a two dimensional audio spectra frequencies by time. It is achieved by applying Short-time Fourier transform to an audio signal.

The Short-time Fourier transform is a Fourier transform that is used to determine the sinusoidal frequency and phase changes over time.

$$X(m,k) = \sum_{n=0}^{N-1} x[n+mH]\omega[n]e^{-\frac{2\pi ikn}{N}}$$

Where

$x: [0: L-1] \coloneqq \{0,1,\dots L-1\} \to \mathbb{R}$ real valued discrete time audio of length L received by sampling an audio signal at a constant sampling rate R.

$\omega: [0: N-1] \to \mathbb{R}$ window function of length $N \in \mathbb{N}$.

$H \in \mathbb{N}$ is the hop size, or by what amount we shift the window function.

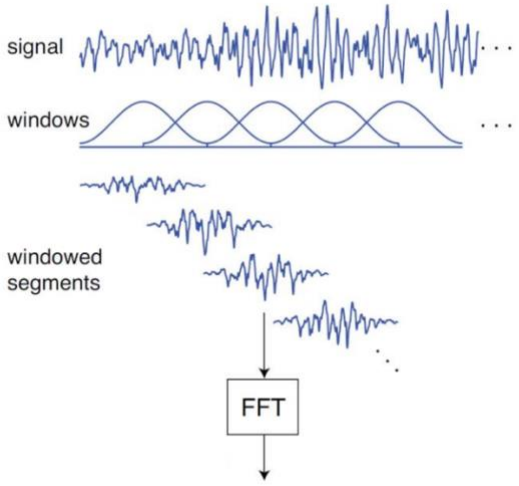
$m \in [0: M]\ where\ M \coloneqq \left\lfloor \frac{L-N}{H} \right\rfloor$ is the maximal window index.

$k \in [0: K]\ where\ K = \frac{N}{2}$ frequency index corresponding to Nyquist frequency.

The complex number $X(m,k)$ represents k_{th} Fourier coefficient for the m_{th} frame. So for each time frame we compute a vector of length K.

The main idea is instead of considering the full input signal, we only use a small section of it. It is achieved by introducing a window function, which is non-zero

for only a short period of time. The original signal is then multiplied by this windowed function. To compute the frequency information signal at different timestamps we move the window function across the whole signal by hop size steps and apply FFT to each windowed segment.



Thus we can easily calculate the dimension of the resulting spectrogram, as it's just

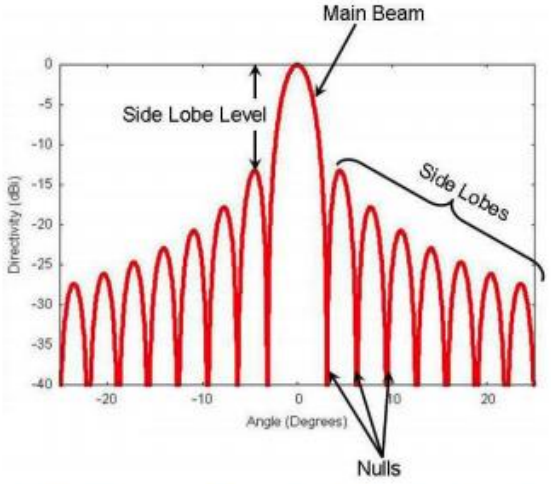
$$S \in \mathbb{R}^{d_{spectr} \times \left\lceil \frac{signal\ length}{hop\ size} \right\rceil}$$

where S is our spectrogram.

To localize speech signal in time, we introduce windowing functions $\omega[n, \tau]$ which degrades on its ends to avoid unnatural sounding discontinuities in speech.

The window function we'll be using is called Hann window and is defined as following:

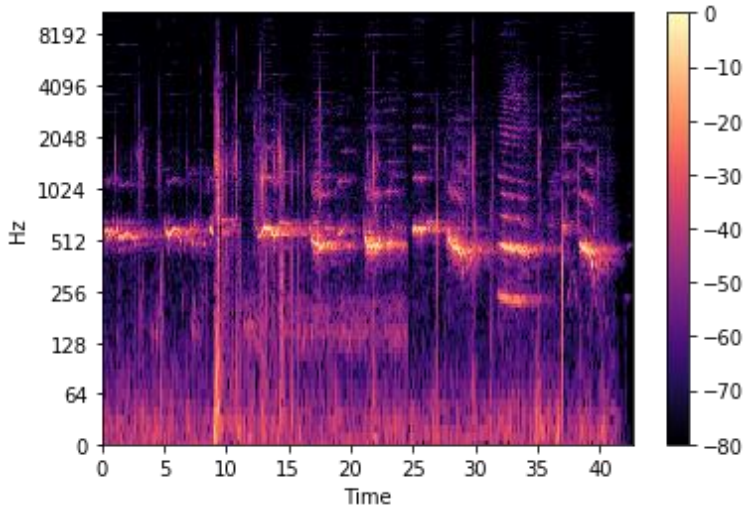
$$\omega[n, \tau] = 0.54 - 0.4\cos\left[\frac{2\pi(n-\tau)}{N_\omega - 1}\right]$$



http://en.wikipedia.org/wiki/Window_function

Although there are many more windowing functions, Hann window is by far the most popular choice in speech processing, as it provides a balance between frequency resolution and the dynamic range.

After STFT we receive a frequency spectrogram and complex phase, which is discarded. Humans can only perceive sound from 20 Hz to 20,000 Hz, so the information in a spectrogram that is outside this range becomes useless in our task. We can transform amplitudes to decibels which is just the log scale of amplitudes and finally receive a spectrogram.



The next thing we'll do is transform it to a Mel-scaled spectrogram. It is constructed such that the sounds of equal "distance" on the Mel-scale also "sound" to humans as if they are equally distant.

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

f - Hertz, m - mel

It has a logarithmic curve, because, as an example, the difference between 500Hz and 1000Hz is obvious to human perception, while the difference between 7500Hz and 8000Hz is barely noticeable.

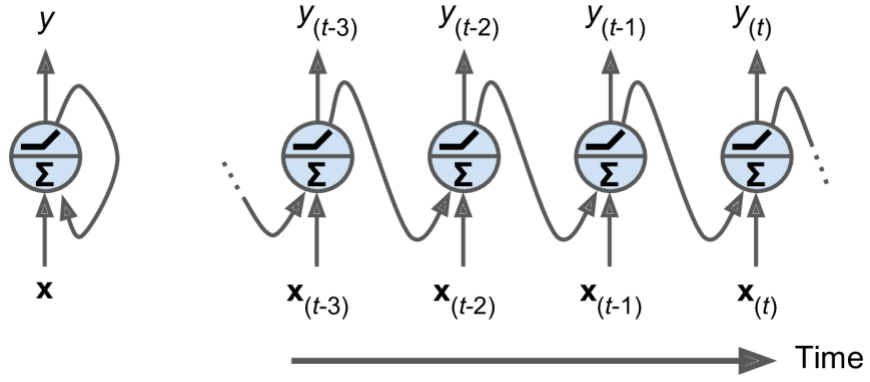
We'll be capping this spectrogram to only have 80 mel-bins, as it has been determined with as a breakpoint for optimal performance, because of a sufficient resolution along the frequency dimension.

Neural Network for TTS

In my previous coursework we went through the basics of Neural Networks, how they learn and optimize their parameters, convolutional and linear layers, representation learning and so forth. We'll continue off of that to explain all the components needed for a TTS models and how they come together.

Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a family of neural networks for processing sequential data. They are similar to a feedforward network except a few changes to process sequential data. At each time step t the recurrent neuron receives the inputs x_t as well as its own output from the last time step y_{t-1}. If t=0, meaning that it's the first token in a sequence, then the previous output is usually initialized as 0. If we unroll this RNN, we receive the following visualization



Output of a recurrent layer for a single instance is defined as follows:

$$y_t = \phi(W_x^T x_t + W_y^T y_{t-1} + b)$$

$\phi(x)$ – activation function, W_x - learnable weight matrix for inputs x,
W_y – learnable weight matrix for previous output.

After unrolling the network through time, backpropagation becomes straight-forward with a well-defined computational graph. Since the output of a recurrent neuron at time step t is a function of all the inputs from previous time steps, it can be interpreted as a kind of memory. Memory cells are denoted as h_t which refers to the memory of a sequence up to time-step t, and would be used with input x_{t+1}. Thus we have defined a Neural Network, capable of processing and storing sequence information.

Teacher forcing in RNNs

One weakness in the abovementioned algorithm to RNNs is that they lack hidden-to-hidden recurrent connections. This means that at every timestep they require all previous outputs to produce the next output, so they cannot be parallelized. Sequence-to-sequence models that depend on the previous output to generate next output can be trained with a method called **teacher-forcing.** Teacher forcing is a method that emerges from maximum likelihood criterion. At time step t + 1 the model receives the previous ground truth y_t, instead of the previous model output. Let's examine a sequence with two time steps. Conditional maximum likelihood criterion is specified as:

$$\log p(y_1, y_2|x_1, x_2) = \log p(y_2|y_1, x_1, x_2) + \log p(y_1|x_1, x_2)$$

We can see, that at t = 2, the model is trained to maximize the conditional probability of y_2 given both x sequence and the previous ground truth y_1. Another key improvement this provides is that the model can now be trained in parallel, as we've removed the dependency on the previous model output at each time step. Although training time can be parallelized, in inference mode the model is turned

back to an autoregressive form, as it doesn't have the information about ground truth.

LSTM

One of the appeals of RNNs is the idea that they might connect previous information and capture context. In theory, they are capable to form and handle long-term dependencies. But in practice, they fail to generalize to long sequences and can only contain short-term memory. This problem was explored in depth by Dr. W. Brauer (1991) and Bengio, et al. (1994). The basic problem is that gradients propagated over many recurrent connections tend to either vanish, or explode, both of which are a big problem in optimizing the network. Even if we manage to overcome this issue, long-term interactions will be given exponentially smaller weights because of the multiplications of many Jacobians, compared to short-term interactions.

$$h_t = (W^t)^T h_0$$

If the matrix W can be decomposed with eigendecomposition

$$W = Q\Lambda Q^T$$

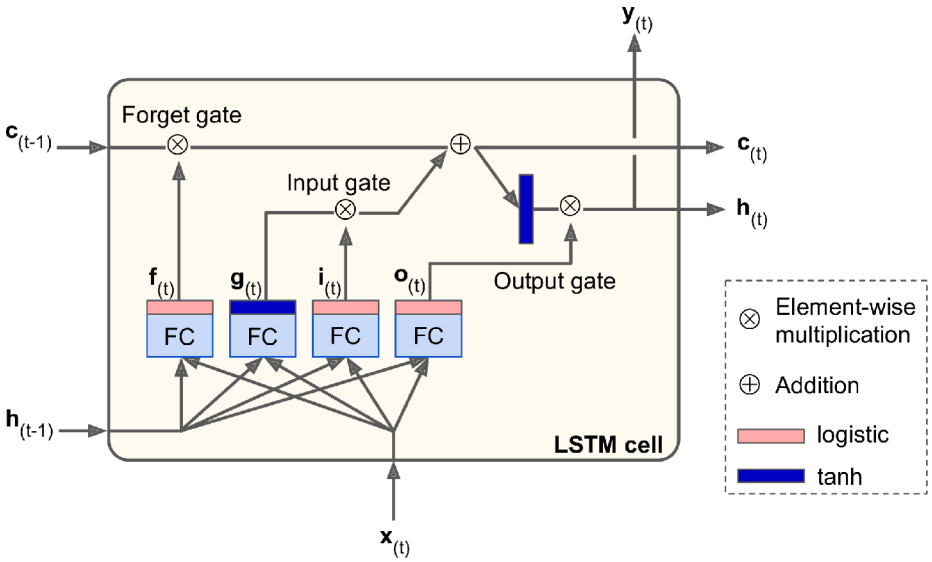
with orthogonal Q, the recurrence can be further simplified

$$h_t = Q^T \Lambda^t Q h_0$$

The eigenvalues are raised to the power t, so if t is large enough, eigenvalues that are less than 1 will decay to 0 and larger than 1 will explode. Any component of h_0 that is not aligned with a large enough eigenvalue will thus be discarded with time. This is the main issue that LSTMs proposed by Hochreiter and Schmidhuber(1997) or Long Short Term Memory cells combat by creating paths through time in which gradients can neither vanish nor explode. The key idea is to

introduce self-loops conditioned on the context, rather than fixed. By making it gated, the time scale of integration can be changed dynamically.

LSTM are defined as follows



$$i_t = \sigma(W_{xi}^T x_t + W_{hi}^T h_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf}^T x_t + W_{hf}^T h_{t-1} + b_f)$$

$$o_t = \sigma(W_{xo}^T x_t + W_{ho}^T h_{t-1} + b_o)$$

$$g_t = tanh(W_{xg}^T x_t + W_{hg}^T h_{t-1} + b_g)$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes g_t$$

$$y_t = h_t = o_t \otimes \tanh(c_t)$$

$W_{xi}, W_{xf}, W_{xo}, W_{xg}$ are learnable weight matrices of the four layers for their connection with x.

$W_{hi}, W_{hf}, W_{ho}, W_{hg}$ are learnable weight matrices of the four layers for their connection with the previous short-term state h_{t-1}.

The forget gate f_t controls which part of the long-term state should be removed.

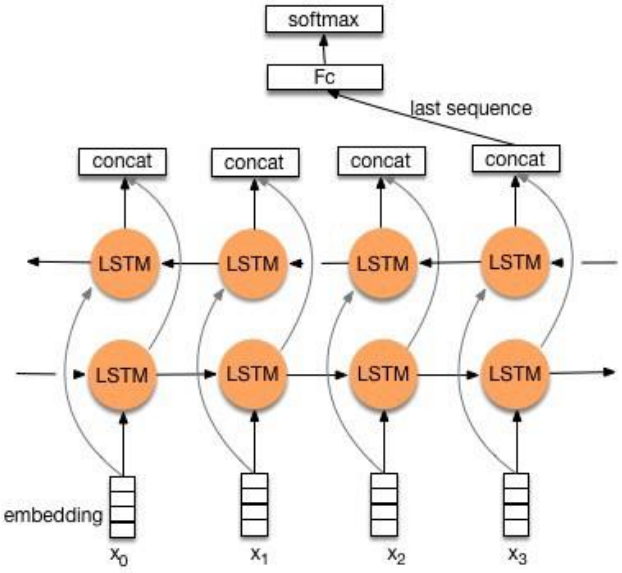
The input gate i_t controls which part of the g_t should be added to the long-term state.

The output gate o_t controls which part of the long-term state should be read and output at this time step, so $h_t = y_t$.

This architecture alleviates exploding and vanishing gradients, as well as improves and handles long-term interactions. It can learn to recognize an important input and store it in the long-term state c, and extract it whenever is needed.

Bidirectional RNNs

RNNs mentioned above have a causal structure, meaning that the state at time t captures only past and present information $X = (x_0, \dots x_t)$ before generating it's output. For many tasks, it is important to also be able to look at the next word before generating the current word, so bidirectional RNNs were invented.

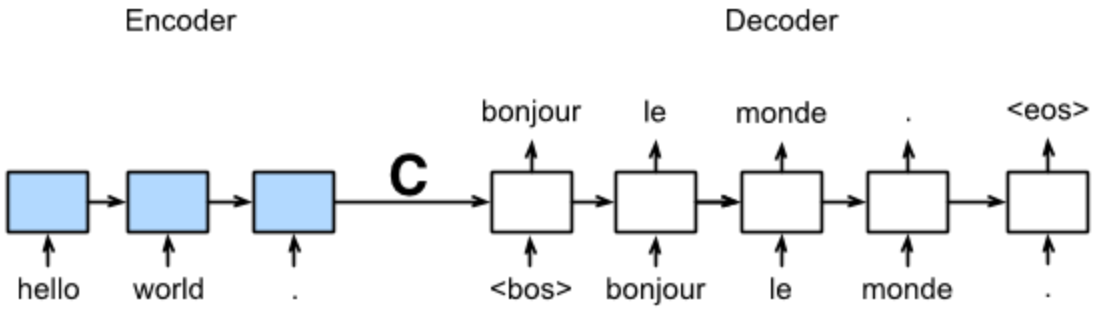


The main idea is simple, we have an $RNN_{forward}$ that has an input sequence of $X = (x_0, x_1, \dots x_n)$ and an $RNN_{backward}$ that has an input sequence $X = (x_n, x_{n-1}, \dots x_0)$. At time step t

$$y_t = \phi[W^T(RNN_{forward}(x_0, \dots x_t)||RNN_{backward}(x_n, \dots, x_t)) + b]$$

Encoder-Decoder architecture

The input to RNNs is often called the context and a good model would produce some context vector or sequence of vectors representation C that summarizes its input sequence $X = (x_1, x_2, \dots x_n)$ and generalizes to other inputs. The first RNN architecture to map a variable length sequence to another variable length sequence was proposed by Cho et al (2014) and was the first to attain state-of-the-art performance in machine translation and it was called an encoder-decoder architecture. The idea is that there are two parts: an encoder RNN that encodes some input sequence to a hidden representation, and a decoder RNN conditioned on a fixed-size C, which tries to decode the outputs of the encoder back into human-readable information $Y = (y_1, y_2, \dots y_n)$. The main contribution of this approach is the variable-length input and output sequence, as it was constrained to some constant before. In this architecture, both RNNs are trained jointly to maximize average of $\log P(y_1, y_2, \dots y_n | x_1, x_2, \dots x_n)$ over all x and y.

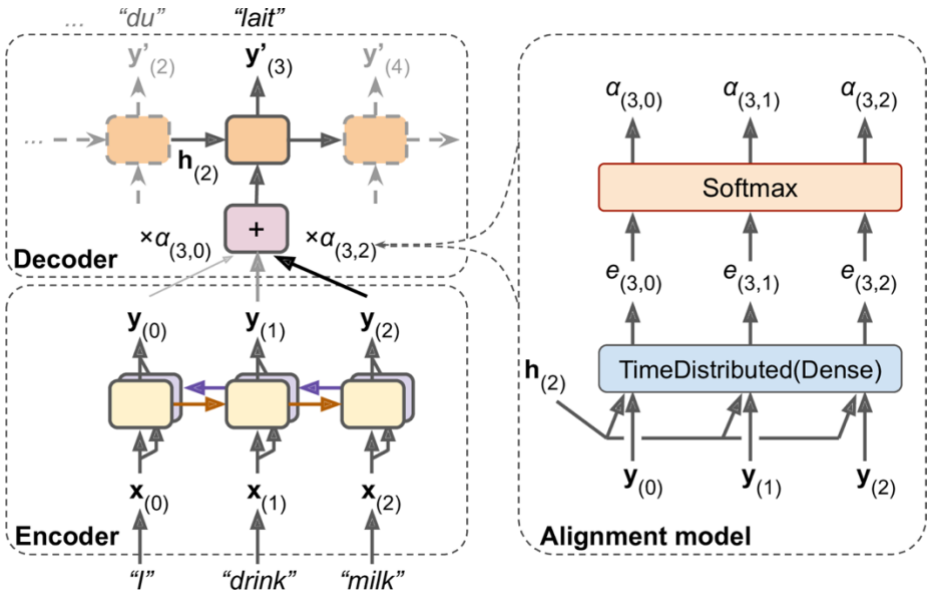


The last state h_n is typically used as the representation of the whole encoder sequence context C. Nowadays, encoder-decoder architecture is used almost everywhere. They are used in image compression-decompression, machine translation, translation alignment, TTS, representation learning, image deblurring, and the list goes on. The main limitation of this network is the way we capture context, as the encoder RNN has a dimension that is way too small to be able to

summarize a variable-length sequence into a compact representation. One key contribution that popularized encoder-decoder even more are attention mechanisms.

Attention

Attention was a groundbreaking idea in Bahdanau et al. (2014) where they introduced a technique that allowed the decoder to focus on appropriate parts of the context at each time step.



It uses an encoder-decoder architecture with an addition of the Alignment or Attention model. At each time step, the decoder computes a weighted sum of the encoder outputs and produces weights a_{ti} for decoder time step t of the i encoder output. Given that we have a softmax as the final attention layer, the model outputs a probability distribution for every time step over encoder outputs. So if $a_{(3,0)} >$

$a_{3,1}$, then the decoder will pay more attention to $a_{(3,0)}$ at time step 3. These attention weights are generated by an attention model, which is trained jointly with the encoder-decoder model. It's often a time-distributed Dense layer with a single neuron, which receives all encoder outputs, concatenated with the decoder's previous hidden state and outputs a score or energy e_{ti} for each encoder output. This energy measures how well each output is aligned with the decoder's previous state. At last, these energy scores go through a softmax to create a probability distribution and produce final weights a_{ti}, so that $\sum_{i=0}^{n} a_{t,i} = 1$.

$$a_{t,i} = \frac{\exp\left(e_{t,i}\right)}{\sum_{i'=0}^{n} \exp\left(e_{t,i'}\right)}$$

$$e_{t,i} = h_t^T W y_i$$

This exact mechanism is called Bahdanau or additive attention.

Batch Normalization

Normalization is a category of methods that seek to make different samples seen by a machine-learning model more similar to each other, which helps generalize to new data.

Batch normalization is a type of layer introduced in Ioffe and Szegedy (2015), which can adaptively normalize data even as the mean and variance change during training. The main effect it has is that it helps gradients flow through deep networks during training and alleviates the problem of exploding/vanishing gradients. It simply zero-centers and normalizes each input, then scales and shifts the result using two new parameter vectors per layer. So it lets the model learn the needed mean and shift to learn a better representation, it is also possible for the model to learn to ignore it, which makes it a good addition anyways. In order to zero-center and normalize the inputs, batch normalization needs to estimate each input's mean and standard deviation, which is done over the whole mini-batch.

$$\mu_B = \frac{1}{m_B}\sum_{i=1}^{m_B} x_i$$

$$\sigma_B^2 = \frac{1}{m_B}\sum_{i=1}^{m_B}(x_i - \mu_B)^2$$

$$\widehat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

$$z_i = \gamma \otimes \widehat{x_i} + \beta$$

Where:

μ_B is the vector of input means, evaluated over a mini-batch B.

σ_B^2 is the vector of input variances, evaluated over a mini-batch B.

m_B is the number of instances in mini-batch B.

$\widehat{x_i}$ is the zero-centered and normalized input vector for instance i.

γ is the learnable scale parameter for the layer.

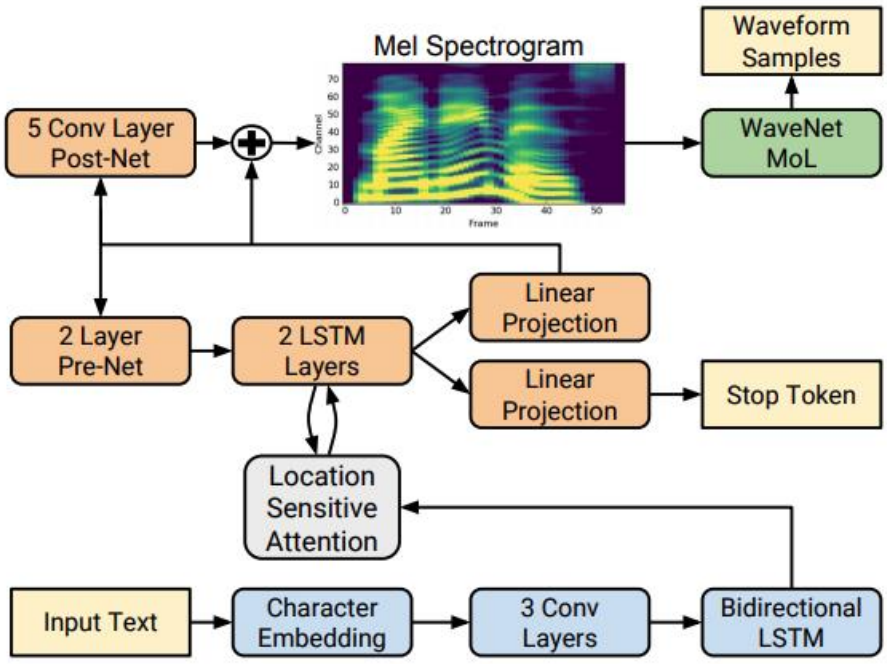
β is the learnable shift parameter for the layer.

z_i is the output of Batch Normalization for instance i.

This approach adds some complexity to the model, but it's often possible to integrate these parameters into the previous layer at test times, further improving speed. During training the algorithm is clear, but the problem comes at test times when we need to normalize a single input, instead of a batch, so we have no way of computing it's mean and standard deviation. One approach of fixing this issue is by learning a moving mean and average during training and freeze it on inference stage, thus alleviating this problem. Although this approach is not optimal for every task. It will fail for a task like stock prediction if the stocks are highly volatile, so we will have to constantly update the mean and standard deviation for the model to be adequate.

Tacotron 2

A modern state-of-the-art TTS system is usually a modified Tacotron 2 spectrogram generator, coupled with a vocoder. We'll be basing off of a Tacotron 2 model which is known of producing robotic, but clear speech and further modifying it by coupling it with a prosody encoder to further enhance the results. This is a model based off of RJ Skerry-Ryan et al. (2018) which is a neural network architecture for speech synthesis conditioned on text. It is composed of two components: a recurrent sequence-to-sequence encoder-decoder network with attention, which predicts a sequence of mel spectrogram frames from input character sequence, and a Wavenet vocoder, which will be explained in a later section.



The network is composed of an **encoder** and a **decoder**.

Encoder produces a context over input text $X = (x_0, \dots x_n)$ which is preprocessed to only contain lower-case letters, punctuation and numbers. It is then fed into a character embedding matrix, which stores and learns the representation of each

letter in a 512 dimensional latent space for the network to use further on. Each character is encoded and passed on to a stack of convolution layers containing 512 filters with the shape of 5×1, so that their feature maps for every character contain information about the previous two and next two characters, encoded in a single representation and is then followed by batch normalization and ReLU activation. These convolutional layers model N-grams in the input sequence, which is then passed on to a bidirectional LSTM layer containing 256 units for each direction, with 512 in total, which generates the encoded feature set. It could be thought of as a smoothing network over all features which captures all character information with a smooth transition in-between character embeddings. This encoded feature set is then passed into a Location sensitive attention, which is a modification of additive attention, to include the cumulative previous decoder states as additional feature to generate current attention. This encourages the model to move forward without the regard for potential failures in the decoding process like repeating vowels or incorrectly generated mel-spectrogram frames. Attention probabilities are computed after projecting inputs and location features to a 128-dimensional representation. Location features are computed using 32 1-D convolution filters.

Decoder is a teacher-forced autoregressive recurrent neural network composed of two stacked LSTM layers, which produce a mel-spectrogram frame for each time step. Prediction from the previous time step is passed through a pre-net, which is a small network containing two Dense layers with 256 neurons each, and a ReLU activation. It acts as an information bottleneck and could be theoretically upgraded to a VAE (Variational Autoencoder) which can also act as an information flow bottleneck. Prenet output and attention context are both concatenated and passed into the LSTM stack. Both of the LSTM layers are uni-directional with 1024 units each, because the model is autoregressive and on inference time there is no

information regarding the future frames. LSTM stack and attention outputs are then concatenated to produce a single spectrogram frame at each time step using a projection layer, which has a dimension of 80 in our case as we are using an 80-bin mel-spectrogram. Finally, the resulting mel-spectrogram is passed through a 5 layer post-net comprised of convolutional layers, with 512 filters and 5×1 kernels, batch normalization and tanh activation on all but last layer. Post-net computes a residual to add to the resulting spectrogram. In theory, this should make it less smooth and natural, but in practice it has been shown that it has little to no effect further on and can be discarded in a powerful model. The main metric used is Mean Squared Error between the predicted and target mel-spectrogram. In parallel a small network is trained to output the probability of a generated spectrogram frame of being the last, so at inference time we can automatically detect when we need to stop generating frames. All convolutional layers are normalized with Dropout and LSTM layers are normalized using Zoneout, which is a modified Dropout algorithm for LSTMs.

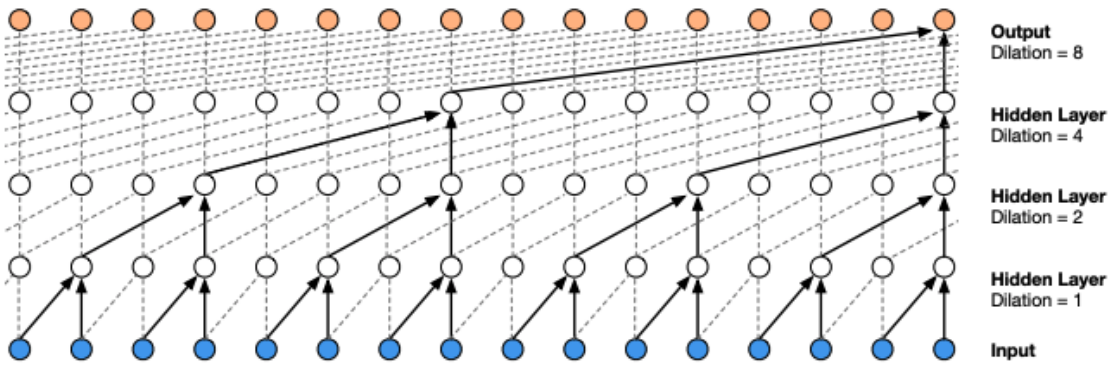
During training, we use teacher-forcing in LSTM stack. It is essential, as the task is too hard for the model to excel at without it. Instead of previously generated frames, previous target frames are passed on into pre-net on the decoder side, so at every time step the LSTM stack is given the attention text context and pre-net transformed previous mel-spectrogram frames. Now that we've received a mel-spectrogram representation, we want to transform it into a waveform, which is not as easy as inverting it because we don't have the phase information. This is where the need of a vocoder comes, which would take a mel-spectrogram as it's input and output a waveform.

Wavenet

Wavenet is a vocoder proposed by Aaron van den Oord et al. (2016). The main task for this generative model is to model raw audio waveforms. The joint probability of a waveform $x = \{x_1, \dots x_n\}$ is factorized as a product of conditional probabilities:

$$p(x) = \prod_{n=1}^{N} p(x_n | x_1, \dots x_{n-1})$$

This conditional probability is modeled by a stack of convolutional layer. No pooling layers are used as the goal is to receive the same output dimensionality as the input. By stacking 1D dilated convolutional layers, they doubled the dilation rate (how spread apart neurons are) at every layer. A dilated convolutional layer is a convolution where filter is applied over an area larger than its length by skipping input values by a specified step. First convolutional layer analyses two samples, second convolutional layer analyses the outputs of first, so it receives 4 samples, next one up receives 8 and so on, thus greatly increasing the receptive field with every layer.



With this simple technique, they were able to model long sequences using convolutions and thus removing RNN dependency. This way they were able to make their training session parallel and in return it converged faster. By using causal convolutions, we make sure that we don't violate time conditioning

$p(x_n|x_1, \dots x_n)$. Although during training we know all inputs and can thus be trained in parallel, but during inference we have to revert back to an autoregressive model that uses its previous output as a part of the next input. By creating a separate upsampling network for our spectrogram $S = (s_1, .. s_t)$ with t frames, we stretch this spectrogram to the length of the waveform it represents $X = (x_1, \dots x_n)$ so that the new transformation $S_{upsampled} \in \mathbb{R}^{n \times mel\ bins}$. Next we concatenate $S_{upsampled}$ with X and feed it into the model to receive desired waveform.

$$p(x|S_{upsampled}) = \prod_{n=1}^{N} p(x_n|x_1, \dots, x_{n-1}, S_{upsampled})$$

We have now built our entire TTS pipeline with STFT creating mel-spectrograms, Tacotron 2 producing mel-spectrograms conditioned on text and Wavenet vocoder transforming them to waveforms for us to listen to.

Prosody conditioning network

With a defined meaning of prosody, we can now model it with a separate sub-model such that we want max log $(p(S|y_T, y_P))$, maximize log-likelihood of our spectrogram given $y_T = encoder_{text}(text)$ and $y_P = model_{prosody}(S)$.

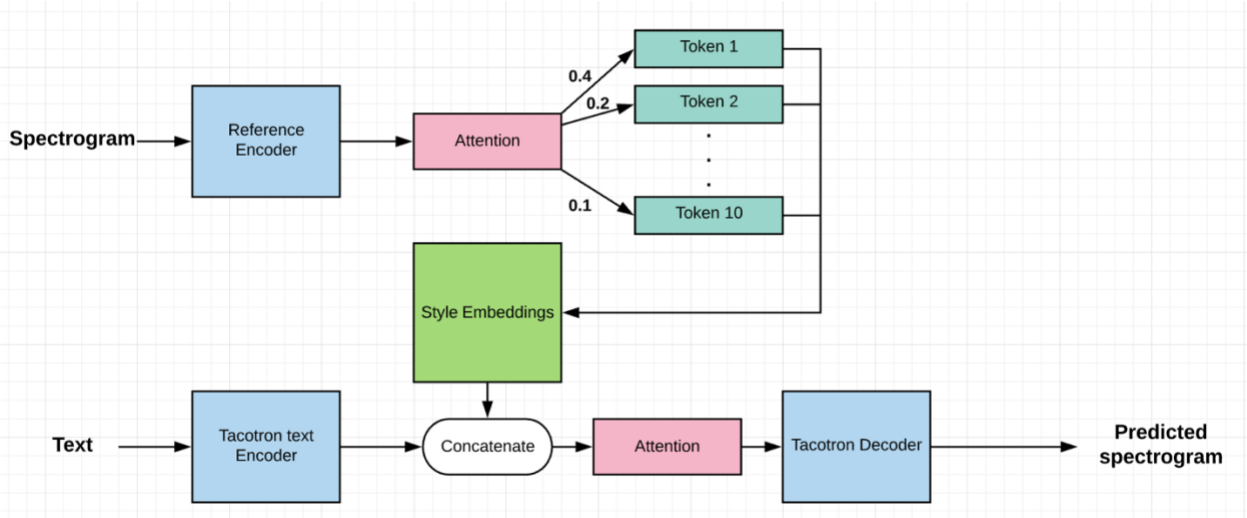
We can define our loss function as

$$L(S, y_T, y_P) = -\log\left(p(S|y_T, y_P) = \left|\left|f(y_T, y_P) - S\right|\right|_2\right.$$

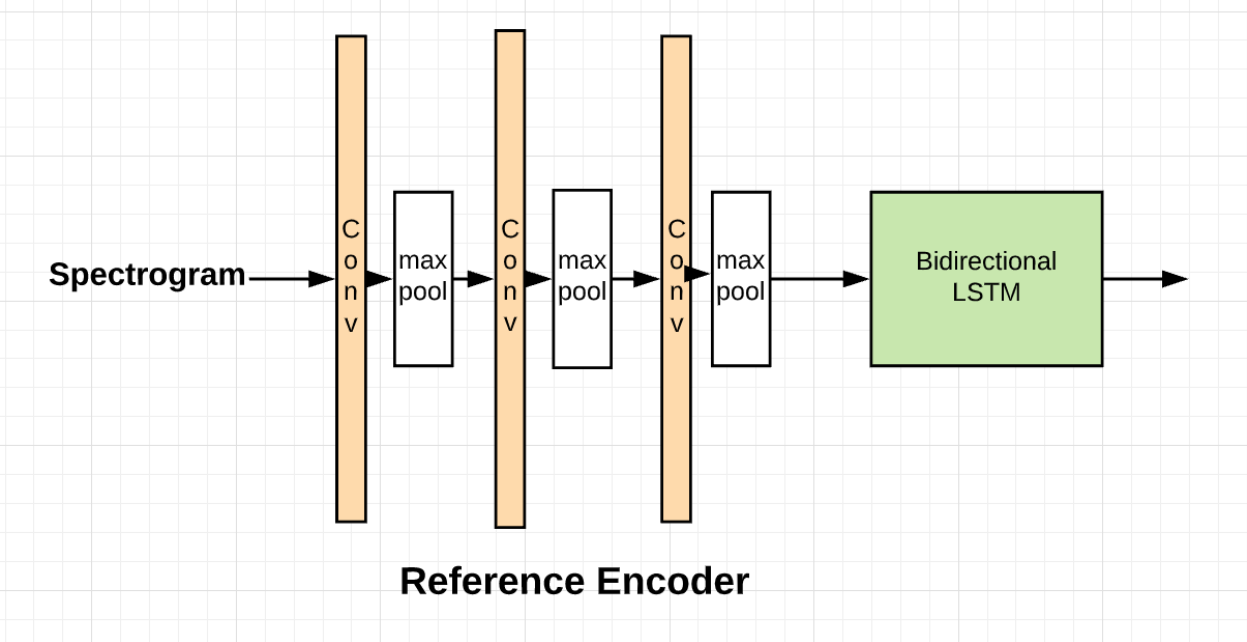
Where f – is Tacotron 2 model, with an integrated prosody module, which is trained with Tacotron and is end-to-end. This means our model learns to differentiate between prosody and text.

We will define $model_{prosody}$ as a network with a trainable prosody embedding matrix $X_{prosody} \in \mathbb{R}^{10 \times 256}$, which is initialized using Xavier initialization. It's an embedding matrix with 10 tokens, each with a 256 element vector representing

prosody. To extract prosody from speech during training, we need to first feed in a spectrogram to reference encoder network to extract information from it, with an attention module on top of it. The attention module helps us choose the right style token and the chosen style token is used with the outputs of our text encoder to synthesize a new spectrogram with a given prosody and text.



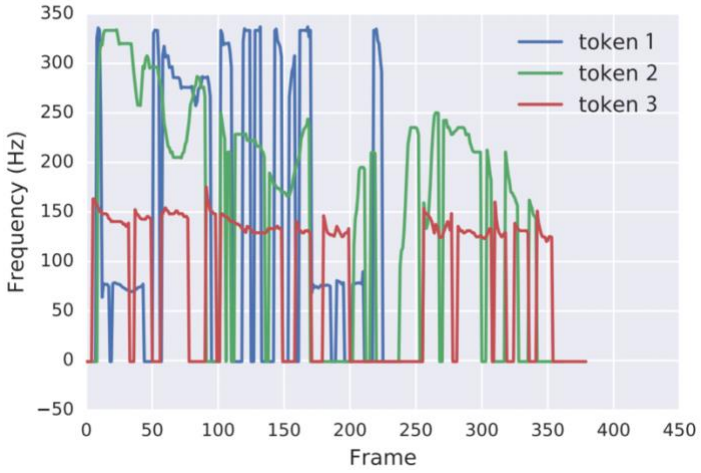
By keeping a small amount of tokens, we want them to capture the most essential information in speech. Given that our decoder is also conditioned on text, it will not encode text information to our style embedding given that our text encoder is strong enough to encode all sequence information. All we are left with is prosody, and speaker. But we are training with a single speaker dataset called Blizzard Challenge 2013, which is composed of several hours of a expressive single-speaker audio books. Given that we have a single speaker scenario, there is no need to encode any speaker specific information. If it were a multi speaker dataset, we could also create a speaker encoder, which would encode all speaker information and be concatenated with text and prosody encoder outputs. Thus, we leave out any phonetic leakage. A reference encoder is composed of stacked convolutional layers with pooling layers and batch normalization. To further encode the time component, we have an LSTM layer on top of the CNN stack.

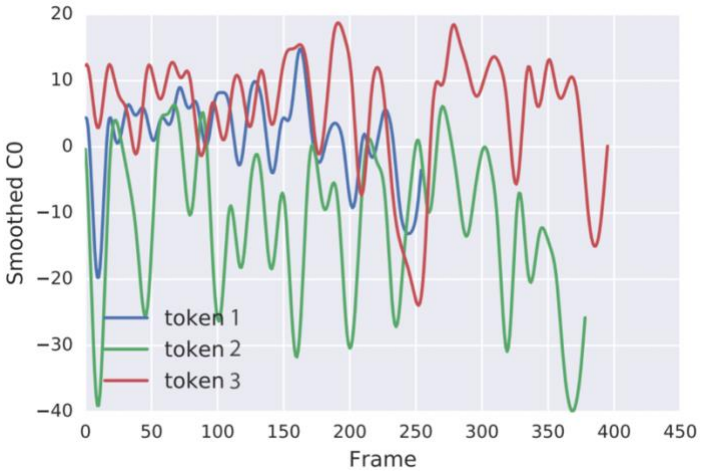


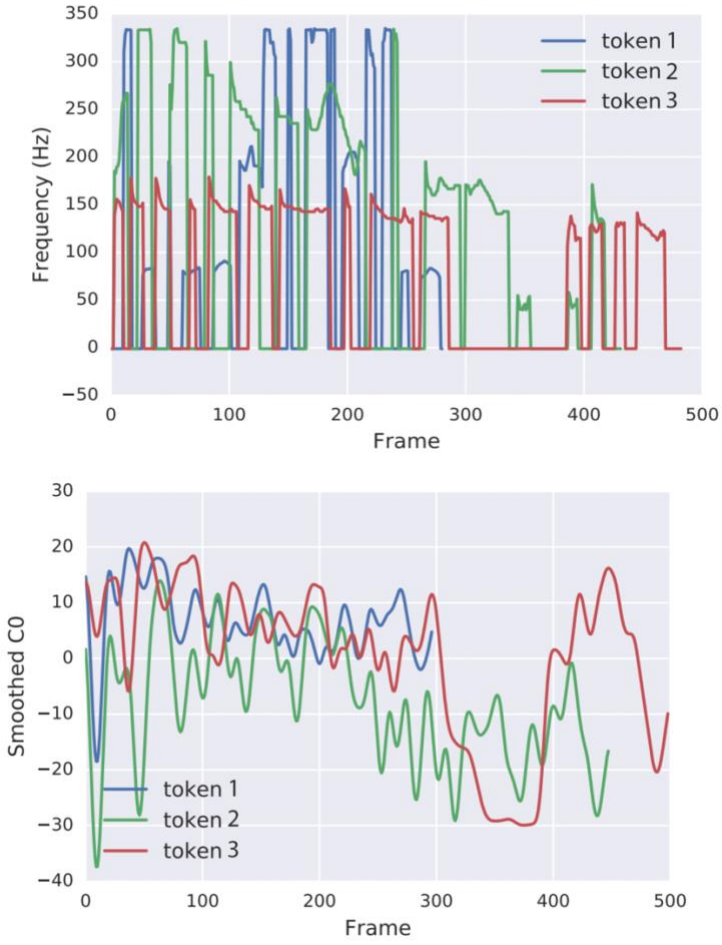
Now we have a defined Neural Network for speech synthesis with a module for prosody extraction. At inference time, we have a choice of extracting prosody from a spectrogram the prosody of which we want to copy, or we can manually choose a token from the embedding matrix and use it for synthesis. We will further explore the effect of these embeddings on generated speech.

Generating and analyzing prosody embeddings

With a trained model to maximize the log likelihood of our data conditioned on text and prosody. We have a learned prosody embedding matrix, which we will further call tokens. By synthesizing a single sentence with different tokens, we can explore there fundamental frequency $F0$ and energy $C0$. With a distinct difference in tokens, we will see drastic changes using different tokens. If they have no effect, their graphs will be the same or close to same.

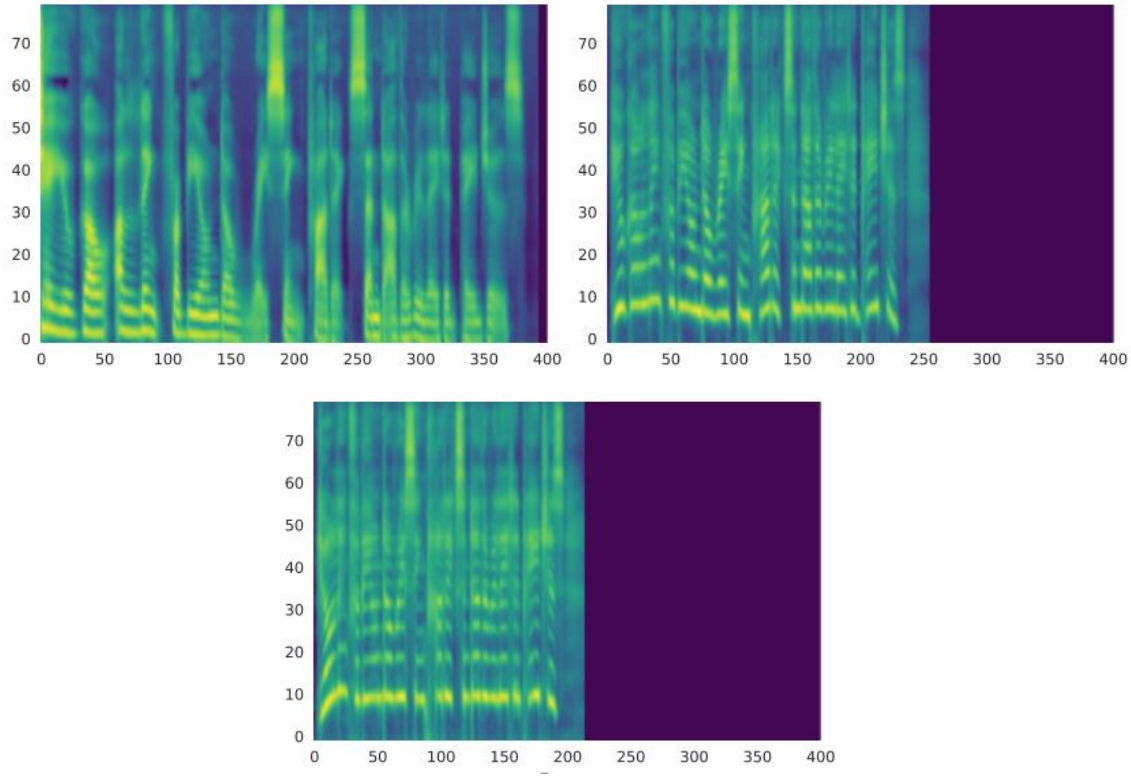

token 1
token 2
token 3
Frequency (Hz)
Frame


token 1
token 2
token 3
Smoothed C0
Frame



Two sentences are synthesized with three different tokens, scaled by 0.3. We will explore scaling effect later. Both fundamental frequency and energy are completely different for each token. We see a distinct trend of first token being shorter, meaning that it controls the speaking rate. If we scale it by a larger number, speech will become longer, but if we scale it by a lower constant, it will become even shorter. Third token represents a lower-pitched speech and green token represents decreasing pitch as it starts with high energy and gradually decays. Another key insight is that these tokens are not completely disentangled. Although first token describes speaking rate, we see that second token is a bit faster, than red. Meaning

that either red or green tokens contain some information on speaking rate. It would be interesting to create a disentangled representation using VAEs in the future. We can further explore scaling tokens. I'll take the first token and scale it by 3 factors: -0.3, 0.3, 0.5



As we can see, it indeed controls the speed and a positive value speeds up speech, while a lower value elongates it. These transformations are just not possible using vanilla Tacotron 2. As a side bonus, it also produces higher fidelity speech as opposed to vanilla and lets us model and control speech prosody directly or indirectly with the use of tokens and scaling them manually.

Conclusion

Machine learning is on the rise, showing impressive results even on challenging tasks such as text-to-speech. With quite a simple architecture we were able to create a system, that synthesizes a spectrogram representation of audio out of speech, and combined with a vocoder we produced audio output to further examine. After that, we undertook a prosody modeling network which was able to successfully model and encode prosody into our text-to-speech pipeline. We've shown that even with a simple network used to encode prosody we could indeed produce prosody embeddings that had a significant effect on our synthesis.

Future work would include working with VAEs instead of the token architecture. By disentangling each prosody dimension we could have a fine-grained prosody control system with each component representing some specific change in prosody like speaking rate, accents, loudness, pitch and much more.

Literature

[1] Aurelien Geron. [book] Hand-On Machine Learning with Scikit-Learn and TensorFlow.

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courvile. [book]Deep Learning.

[3] Sebastian Raschka. [book] Python Machine Learing.

[4] Professor Dr. W. Brauer [article] Untersuchungen zu dynamischen neuronalen Netzen. http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf

[5] Yoshua Bengio et al [article] Learning long-term-dependencies with gradient descent is difficult. http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf

[6] Paarth Neekhara et al. [article] Expediting TTS Synthesis with Adversarial Vocoding. https://arxiv.org/pdf/1904.07944.pdf

[7] [website] THE SHORT-TIME FOURIER TRANSFORM. https://www.dsprelated.com/freebooks/sasp/Short_Time_Fourier_Transform.html

[8] Ricardo Gutierrez-Osuna[article] Short-time Fourier analysis and synthesis. http://research.cs.tamu.edu/prism/lectures/sp/l6.pdf

[9] Kartik Chaudhary [article] Understanding Audio data, Fourier Transform, FFT and Spectrogram features for a Speech Recognition System. https://towardsdatascience.com/understanding-audio-data-fourier-transform-fft-spectrogram-and-speech-recognition-a4072d228520

[10] Dalya Gartzman [article] Getting to Know the Mel Spectrogram. https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0

[11] Christopher Olah [article] Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[12] Ashish Vaswani [article] Attention Is All You Need. https://arxiv.org/pdf/1706.03762.pdf

[13] Lilian Weng [article] Attention? Attention! https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

[14] Dzmitry Bahdanau, Y. Bengio [article] NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE. https://arxiv.org/pdf/1409.0473.pdf

[15] RJ Skerry-Ryan et al. [article] TACOTRON: TOWARDS END-TO-END SPEECH SYNTHESIS. https://arxiv.org/pdf/1703.10135.pdf

[16] RJ Skerry-Ryan et al. [article] NATURAL TTS SYNTHESIS BY CONDITIONING WAVENET ON MEL SPECTROGRAM PREDICTIONS. https://arxiv.org/pdf/1712.05884.pdf

[17] RJ Skerry-Ryan et al. [article] Towards End-to-End Prosody Transfer for Expressive Speech Synthesis with Tacotron. https://arxiv.org/pdf/1803.09047.pdf

[18] RJ Skerry-Ryan et al. [article] EFFECTIVE USE OF VARIATIONAL EMBEDDING CAPACITY IN EXPRESSIVE END-TO-END SPEECH SYNTHESIS. https://arxiv.org/pdf/1906.03402.pdf

[19] Aaron van den Oord [article] WAVENET: A GENERATIVE MODEL FOR RAW AUDIO. https://arxiv.org/pdf/1609.03499.pdf

[20] Diederik P. Kingma, Max Welling [article] An Introduction to Variational Autoencoders. https://arxiv.org/pdf/1906.02691.pdf