LPCNET: IMPROVING NEURAL SPEECH SYNTHESIS THROUGH LINEAR PREDICTION

Jean-Marc Valin

Mozilla
Mountain View, CA, USA
jmvalin@jmvalin.ca

Jan Skoglund

Google LLC
San Francisco, CA, USA
jks@google.com

ABSTRACT

Neural speech synthesis models have recently demonstrated the ability to synthesize high quality speech for text-to-speech and compression applications. These new models often require powerful GPUs to achieve real-time operation, so being able to reduce their complexity would open the way for many new applications. We propose LPCNet, a WaveRNN variant that combines linear prediction with recurrent neural networks to significantly improve the efficiency of speech synthesis. We demonstrate that LPCNet can achieve significantly higher quality than WaveRNN for the same network size and that high quality LPCNet speech synthesis is achievable with a complexity under 3 GFLOPS. This makes it easier to deploy neural synthesis applications on lower-power devices, such as embedded systems and mobile phones.

Index Terms— neural audio synthesis, parametric coding, WaveRNN

1. INTRODUCTION

Neural speech synthesis algorithms have recently made it possible to both synthesize high-quality speech [1, 2, 3], and code high quality speech at very low bitrate [4]. The first generation of these algorithms, often based on algorithms like WaveNet [5], gave promising results in real time with a high-end GPU to provide the tens of billions of floating-point operations per second (GFLOPS) required. We want to perform synthesis on end-user devices like mobile phones, which do not have powerful GPUs and have limited battery capacity.

Recent work [6, 7] has focused on finding more efficient models in order to reduce the complexity of speech synthesis. In this work, we continue in that direction, providing more efficiency improvements and making it easier to synthesize speech even on slower CPUs, and with limited impact on battery life.

Low complexity parametric synthesis models such as low bitrate vocoders have existed for a long time [8, 9], but their quality has always been severely limited. While they are generally efficient at modeling the spectral envelope (vocal tract response) of the speech using linear prediction, no such simple model exists for the excitation. Despite some advances [10, 11, 12], modeling the excitation signal has remained a challenge.

In this work, we propose the LPCNet model, which takes the burden of spectral envelope modeling away from a neural synthesis network so that most of its capacity can be used to model a spectrally flat excitation. This makes it possible to match the quality of state-of-the art neural synthesis systems with fewer neurons, significantly reducing the complexity. Starting from the WaveRNN algorithm summarized in Section 2, we make improvements that reduce the model complexity, as detailed in Section 3. In Section 4, we evaluate the quality and complexity of LPCNet in a speaker-independent speech synthesis context based on the proposed model. We conclude in Section 5.

2. WAVERNN

The WaveRNN architecture proposed in [7] takes as input the previous audio sample s_{t-1}, along with conditioning parameters $\mathbf{f}$, and generates a discrete probability distribution $P\left(s_t\right)$ for the output sample. Although it is proposed as a 16-bit model (split as 8 coarse bits and 8 fine bits), we omit the coarse/fine split in this summary, both for clarity and because we do not use it in this work. The WaveRNN model mainly consists of a gated recurrent unit (GRU) [13], followed by two fully-connected layers, ending in a softmax activation. It is computed as

$$
\begin{aligned}
\mathbf{x}_t &= [s_{t-1}; \mathbf{f}] \\
\mathbf{u}_t &= \sigma\left(\mathbf{W}^{(u)}\mathbf{h}_{t-1} + \mathbf{U}^{(u)}\mathbf{x}_t\right) \\
\mathbf{r}_t &= \sigma\left(\mathbf{W}^{(r)}\mathbf{h}_{t-1} + \mathbf{U}^{(r)}\mathbf{x}_t\right) \\
\widetilde{\mathbf{h}}_t &= \tanh\left(\mathbf{r}_t \circ \left(\mathbf{W}^{(h)}\mathbf{h}_{t-1}\right) + \mathbf{U}^{(h)}\mathbf{x}_t\right) \qquad (1) \\
\mathbf{h}_t &= \mathbf{u}_t \circ \mathbf{h}_{t-1} + (1-\mathbf{u}_t) \circ \widetilde{\mathbf{h}}_t \\
P\left(s_t\right) &= \mathrm{softmax}\left(\mathbf{W}_2 \,\mathrm{relu}\left(\mathbf{W}_1 \mathbf{h}_t\right)\right)\ ,
\end{aligned}
$$

where the $\mathbf{W}^{(\cdot)}$ and $\mathbf{U}^{(\cdot)}$ matrices are the GRU weights, $\sigma\left(x\right) = \frac{1}{1+e^{-x}}$ is the sigmoid function, and $\circ$ denotes an element-wise vector multiply. Throughout this paper, biases are omitted for clarity. The synthesized output sample s_t is obtained by sampling from the probability distribution $P\left(s_t\right)$. As a way of reducing complexity, the matrices used by the GRU can be made sparse and [7] proposes using non-zero blocks of size 4x4 or 16x1 to ensure that vectorization is still possible and efficient.

3. LPCNET

This section presents the LPCNet model, our proposed improvement on WaveRNN. Fig. 1 shows an overview of its architecture, which is explained in more details in this section. It includes a sample rate network that operates at 16 kHz, and a frame rate network that processes 10-ms frames (160 samples). In this work, we limit the input of the synthesis to just 20 features: 18 Bark-scale [14] cepstral coefficients, and 2 pitch parameters (period, correlation). For low-bitrate coding applications, the cepstrum and pitch parameters would be quantized [4], whereas for text-to-speech, they would be computed from the text using another neural network [1].

978-1-5386-4658-8/18/$31.00 ©2019 IEEE

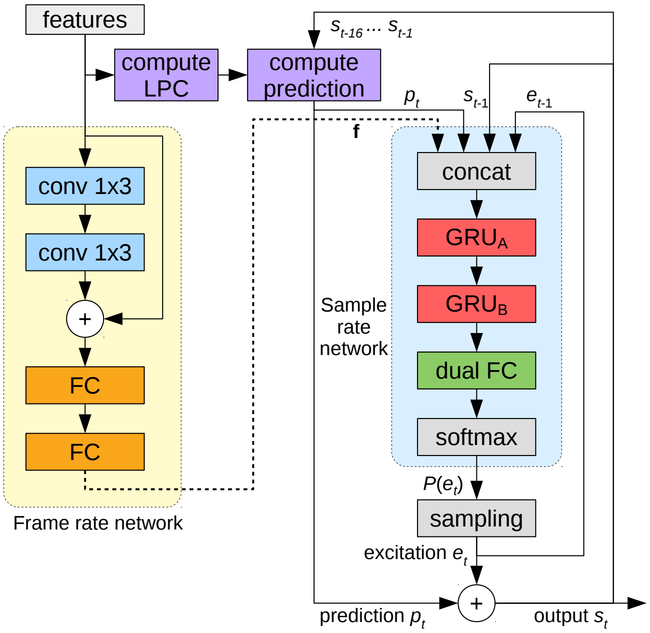

Fig. 1. Overview of the LPCNet algorithm. The left part of the network (yellow) is computed once per frame and its result is held constant throughout the frame for the sample rate network on the right (blue). The *compute prediction* block predicts the sample at time t based on previous samples and on the linear prediction coefficients. Conversions between μ-law and linear are omitted for clarity. The de-emphasis filter is applied to the output s_t.

3.1. Conditioning Parameters

As part of the frame rate network, the 20 features first go through two convolutional layers with a filter size of 3 (conv 3x1), resulting in a receptive field of 5 frames (two frames ahead and two frames back). The output of the two convolutional layers is added to a residual connection and then goes through two fully-connected layers. The frame rate network outputs a 128-dimensional conditioning vector **f** that is then used by the sample rate network. The vector **f** is held constant for the duration of each frame.

3.2. Pre-emphasis and Quantization

Some synthesis models such as WaveNet [5] use 8-bit μ-law quantization [15] to reduce the number of possible output sample values to just 256. Because the energy of speech signals tends to be mostly concentrated in the low frequencies, the μ-law white quantization noise is often audible in the high frequencies, especially for 16 kHz signals that have a higher spectral tilt. To avoid this problem, some approaches extend the output to 16 bits [7]. Instead, we propose to simply apply a first-order pre-emphasis filter $E(z) = 1 - \alpha z^{-1}$ to the training data, with $\alpha = 0.85$ providing good results. The synthesis output can then be filtered using the inverse (de-empahsis) filter

$$D(z) = \frac{1}{1 - \alpha z^{-1}} , \tag{2}$$

effectively shaping the noise such that its power at the Nyquist rate is reduced by 16 dB. This significantly reduces the perceived noise (see Section 4.3) and makes 8-bit μ-law output viable for high-quality synthesis.

3.3. Linear Prediction

The neural networks in many neural speech synthesis approaches [4, 5, 6, 7, 16] have to model the entire speech production process, including glottal pulses, noise excitation, as well as the response of the vocal tract. Although some of these are indeed very hard to model, we know that the vocal tract response can be represented reasonably well by a simple all-pole linear filter [17]. Let s_t be the signal at time t, its linear prediction based on previous samples is

$$p_t = \sum_{k=1}^{M} a_k s_{t-k} , \tag{3}$$

where a_k are the M^{th} order linear prediction coefficients (LPC) for the current frame.

The prediction coefficients a_k are computed by first converting the 18-band Bark-frequency cepstrum into a linear-frequency power spectral density (PSD). The PSD is then converted to an auto-correlation using an inverse FFT. From the auto-correlation, the Levinson-Durbin algorithm is used to compute the predictor. Computing the predictor from the cepstrum ensures that no additional information needs to be transmitted (in a speech coding context) or synthesized (in a text-to-speech context). Even though the LPC analysis computed in this way is not as accurate as one computed on the input signal (due to the low resolution of the cepstrum), the effect on the output is small because the network is able to learn to compensate. This is an advantage over *open-loop* filtering approaches [12].

As an obvious extension of using a linear predictor to help the neural network, we can also have the network directly predict the excitation (prediction residual) rather than the sample values. This is not only a slightly easier task for the network, but it also slightly reduces the μ-law quantization noise, since the excitation generally has a smaller amplitude than the pre-emphasized signal. The network takes as input the previously sampled excitation e_{t-1}, but also the past signal s_{t-1}, and the current prediction p_t. We still include s_{t-1} and p_t because we find that *open-loop* synthesis based only on e_{t-1} produces bad quality speech (see Section 3.8).

3.4. Output Layer

To make it easier to compute the output probabilities without significantly increasing the size of the preceding layer, we combine two fully-connected layers with an element-wise weighted sum. The layer, which we refer to as *dual fully-connected* (or DualFC) is defined as

$$\mathrm{dual\_fc}(\mathbf{x}) = \mathbf{a}_1 \circ \tanh\left(\mathbf{W}_1\mathbf{x}\right) + \mathbf{a}_2 \circ \tanh\left(\mathbf{W}_2\mathbf{x}\right) , \tag{4}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are weight matrices, and $\mathbf{a}_1$ and $\mathbf{a}_2$ are weighting vectors. While not strictly necessary for the proposed approach to work, we have found that the DualFC layer slightly improves quality when comparing to a regular fully-connected layer at equivalent complexity. The intuition behind the DualFC later is that determining whether a value falls withing a certain range (μ-law quantization interval in this case) requires two comparisons, with each fully-connected $\tanh$ layer implementing the equivalent of one comparison. Visualizing the weights on a trained network supports that intuition. The output of the DualFC layer is used with a softmax activation to compute the probability $P\left(e_t\right)$ of each possible excitation value for e_t.

3.5. Sparse Matrices

To keep the complexity low, we use sparse matrices for the largest GRU (GRU$_\mathrm{A}$ in Fig. 1). Rather than allowing general element-by-element sparseness – which prevents efficient vectorization – we use block-sparse matrices as proposed in [7]. Training starts with dense matrices and the blocks with the lowest magnitudes are progressively forced to zero until the desired sparseness is achieved. We find that 16x1 blocks provide good accuracy, while making it easy to vectorize the products.

In addition to the non-zero blocks, we also include all the diagonal terms in the sparse matrix, since those are the most likely to be non-zero. Even though they are not aligned horizontally or vertically, the diagonal terms are nonetheless easy to vectorize since they result in an element-wise multiplication with the vector operand. Including the diagonal terms avoids forcing 16x1 non-zero blocks only for a single element on the diagonal.

3.6. Embedding and Algebraic Simplifications

Rather than scale the scalar sample values to a fixed range before feeding them to the network, we use the discrete nature of the μ-law values to learn an embedding matrix $\mathbf{E}$. The embedding maps each μ-law level to a vector, essentially learning a set of non-linear functions to be applied to the μ-law value. Visualizing the embedding matrix of trained networks, we have been able to confirm that the embedding has learned – among other things – the function that converts the μ-law scale to linear.

As long as the embedding is sent directly to the GRU, it is possible to avoid increasing the complexity by pre-computing the product of the embedding matrices with the corresponding submatrices of the GRU's non-recurrent weights ($\mathbf{U}^{(\cdot)}$). Let $\mathbf{U}^{(u,s)}$ be the submatrix of $\mathbf{U}^{(u)}$ composed of the columns that apply to the embedding of the s_{t-1} input sample, we can derive a new embedding matrix $\mathbf{V}^{(u,s)} = \mathbf{U}^{(u,s)}\mathbf{E}$ that directly maps the sample s_{t-1} to the non-recurrent term of the update gate computation. The same transformation applies for all gates (u, r, h) and all embedded inputs (s, p, e), for a total of 9 pre-computed $\mathbf{V}^{(\cdot,\cdot)}$ matrices. In that way, the embedding contribution can be simplified to only one add operation per gate, per embedded input. Since only a single entry per embedding matrix is used for each sample, the large size of these matrices is not an issue, even if they do not fit in cache.

In a similar way to the embedding, the contributions of the frame conditioning vector $\mathbf{f}$ can also be simplified since it is constant over an entire frame. Once per frame, we can compute $\mathbf{g}^{(\cdot)} = \mathbf{U}^{(\cdot)}\mathbf{f}$, the contribution of $\mathbf{f}$ to each of the GRU gates.

The simplifications above essentially make the computational cost of all the non-recurrent inputs to the main GRU negligible, so the calculations in (1) become

$$\begin{aligned}
\mathbf{u}_t &= \sigma\left(\mathbf{W}_u\mathbf{h}_t + \mathbf{v}^{(u,s)}_{s_{t-1}} + \mathbf{v}^{(u,p)}_{p_{t-1}} + \mathbf{v}^{(u,e)}_{e_{t-1}} + \mathbf{g}^{(u)}\right) \\
\mathbf{r}_t &= \sigma\left(\mathbf{W}_r\mathbf{h}_t + \mathbf{v}^{(r,s)}_{s_{t-1}} + \mathbf{v}^{(r,p)}_{p_{t-1}} + \mathbf{v}^{(r,e)}_{e_{t-1}} + \mathbf{g}^{(r)}\right) \qquad (5) \\
\widetilde{\mathbf{h}}_t &= \tanh\left(\mathbf{r}_t \circ (\mathbf{W}_h\mathbf{h}_t) + \mathbf{v}^{(h,s)}_{s_{t-1}} + \mathbf{v}^{(h,p)}_{p_{t-1}} + \mathbf{v}^{(h,e)}_{e_{t-1}} + \mathbf{g}^{(h)}\right) \\
\mathbf{h}_t &= \mathbf{u}_t \circ \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \circ \widetilde{\mathbf{h}}_t \\
P(e_t) &= \mathrm{softmax}\left(\mathrm{dual\_fc}\left(\mathrm{GRU_B}\left(\mathbf{h}_t\right)\right)\right) \ ,
\end{aligned}$$

where the $\mathbf{v}^{(\cdot,\cdot)}_i$ vectors are lookups of column vector i into the corresponding $\mathbf{V}^{(\cdot,\cdot)}$ matrix, and $\mathrm{GRU_B}(\cdot)$ is a regular, non-sparse GRU used in place of the fully-connected layer with ReLU activation in (1).

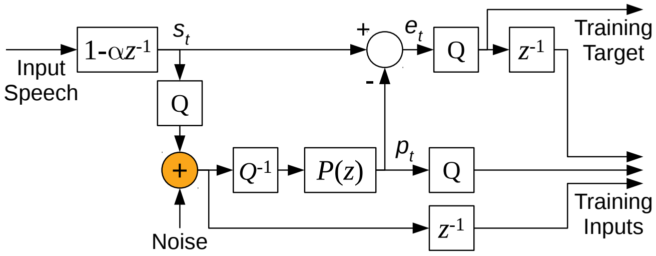

Fig. 2. Noise injection during the training procedure, with Q denoting μ-law quantization and Q^{-1} denoting conversion from μ-law to linear. The prediction filter filter $P(z) = \sum_{k=1}^{M} a_k z^{-k}$ is applied to the noisy, quantized input. The excitation is computed as the difference between the clean, unquantized input and the prediction. Note that the noise is added in the μ-law domain.

3.7. Sampling from Probability Distribution

Directly sampling from the output distribution can sometimes cause excessive noise. This is addressed in [6] by multiplying the logits by a constant $c = 2$ for voice sounds, which is equivalent to lowering the "temperature" of the sampling process. Rather than making a binary voicing decision, we instead set

$$c = 1 + \max\left(0, 1.5g_p - 0.5\right) \ , \qquad (6)$$

where g_p is the pitch correlation ($0 < g_p < 1$). As a second step, we subtract a constant from the distribution to ensure that any probability below that constant threshold T becomes zero. This prevents impulse noise caused by low probabilities. The modified probability distribution becomes

$$P'(e_t) = \mathcal{R}\left(\max\left[\mathcal{R}\left(\left[P(e_t)\right]^c\right) - T, 0\right]\right) \ , \qquad (7)$$

where the $\mathcal{R}(\cdot)$ operator renormalizes the distribution to unity, both between the two steps and on the result. We find that $T = 0.002$ provides a good trade-off between reducing impulse noise and preserving the naturalness of the speech.

3.8. Training Noise Injection

When synthesizing speech, the network operates in conditions that are different from those of the training because the generated samples are different (more imperfect) than those used during training. This mismatch can amplify and cause excessive distortion in the synthesis. To make the network more robust to the mismatch, we add noise to the input during training, as suggested in [6].

The use of linear prediction makes the details of the noise injection particularly important. When injecting noise in the signal, but training the network on the clean excitation, we find that the system produces artifacts similar to those of the pre-analysis-by-synthesis vocoder era, where the noise has the same shape as the synthesis filter $\frac{1}{1-P(z)}$. Instead, we find that by adding the noise as shown in Fig. 2, the network effectively learns to minimize the error in the signal domain because even though its output is the prediction residual, one of its input is the same prediction that was used to compute that residual. This is similar to the effect of analysis-by-synthesis in CELP [18, 19] and greatly reduces the artifacts in the synthesized speech.

To make the noise proportional to the signal amplitude, we inject it directly in the μ-law domain. We vary its distribution across the training data from no noise to a uniform distribution in the $[-3, 3]$ range.

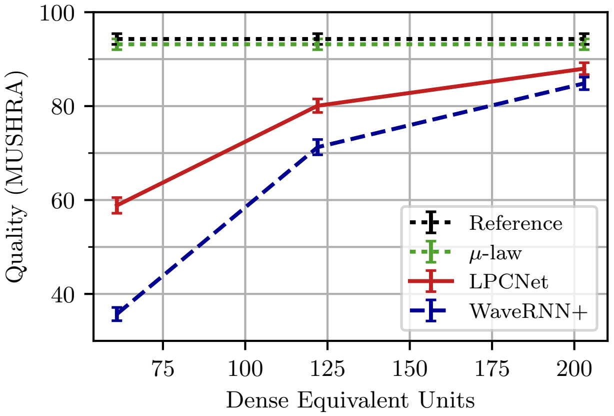

Fig. 3. Subjective quality (MUSHRA) results as a function of the dense equivalent number of units in $\mathrm{GRU_A}$.

4. EVALUATION

The source code for this work is available under an open-source license at `https://github.com/mozilla/LPCNet/`. The evaluation in this section is based on commit `0ddcda0`.

4.1. Complexity

The complexity of the proposed LPCNet model mostly comes from the two GRUs as well as the dual fully-connected layer. It corresponds to two operations (one add, one multiply) per weight, for each sample produced and is given by

$$C = \left(3dN_A^2 + 3N_B\left(N_A + N_B\right) + 2N_B Q\right) \cdot 2F_s\ , \qquad (8)$$

where N_A and N_B are the sizes of the two GRUs, d is the density of the sparse GRU, Q is the number of μ-law levels and F_s is the sampling rate. Using $N_A = 384$, $N_B = 16$ and $Q = 256$ for wideband speech ($F_s = 16000$), and considering around 0.5 GFLOPS complexity for the neglected terms (biases, conditioning network, activation functions, ...), we obtain a total complexity around 2.8 GFLOPS. Real-time synthesis can be achieved on a single core of an Apple A8 (iPhone 6) or with 20% of a 2.4 GHz Intel Broadwell core.

As a comparison, the speaker-dependent FFTNet model – which claims a lower complexity than the original WaveNet [5] algorithm – has a complexity around 16 GFLOPS [6]. The complexity of the original WaveRNN – evaluated as a speaker-dependent model – is not explicitly stated, but our interpretation of the data provided in the WaveRNN [7] paper suggests a complexity around 10 GFLOPS for the sparse, mobile version. The complexity of SampleRNN is not explicitly stated either, but from the paper, we estimate around 50 GFLOPS (mostly due to the 1024-unit MLP layers).

4.2. Experimental Setup

Although the proposed system can be either speaker-dependent or speaker-independent, we evaluate it in the more challenging speaker-independent context. To isolate the quality of the vocoder itself, we compute the features directly from recorded speech samples. The cepstrum uses the same band layout as [20] and the pitch estimator is based on an open-loop cross-correlation search.

The training data consists of only 4 hours of speech from the NTT Multi-Lingual Speech Database for Telephonometry (21 languages), from which we excluded all samples from the speakers used in testing. Each network was trained for 120 epochs (230k updates), with a batch size size of 64, each sequence consisting of 15 10-ms frames. Training was performed on an Nvidia GPU with Keras[1]/Tensorflow[2] using the CuDNN GRU implementation and the AMSGrad [21] optimization method (Adam variant) with a step size $\alpha = \frac{\alpha_0}{1+\delta \cdot b}$ where $\alpha_0 = 0.001$, $\delta = 5 \times 10^{-5}$, and b is the batch number.

We compare LPCNet to an improved version of WaveRNN (denoted WaveRNN+) that includes all of the improvements in Section 3 except for the LPC part, i.e. WaveRNN+ predicts sample s_t only from s_{t-1} and the conditioning parameters. Each model is evaluated with a main GRU of size N_A equal to 192, 384, and 640 units, and with a non-zero density $d = 0.1$. These have the same number of non-zero weights as equivalent dense GRUs of size 61, 122, and 203, respectively (those GRU sizes match the "equivalent" sizes in [7]). In all cases, the size of the second GRU is $N_B = 16$. For each test sample, we compute the input features from the original audio (ground truth), and then synthesize new audio with each of the models under test. We also evaluate the effect of the μ-law quantization with pre-emphasis, as an upper-bound for the achievable quality.

4.3. Quality Evaluation

As reported in [4], objective quality metrics such as PESQ and POLQA cannot adequately evaluate non-waveform, neural vocoders. We conducted a subjective listening test with a MUSHRA-derived methodology [22], where 8 utterances (2 male and 2 female speakers) were each evaluated by 100 participants. The results in Fig. 3 show that the quality of LPCNet significantly exceeds that of WaveRNN+ at equal complexity. Alternatively, it shows that the same quality is possible at a significantly reduced complexity. The results also validate our assumption that pre-emphasis makes the effect of μ-law quantization noise negligible compared to the synthesis artifacts. Being able to compute only a single 256-value distribution also reduces complexity compared to the original 16-bit WaveRNN.

The main audible artifact in the generated samples – from both WaveRNN+ and LPCNet – is some roughness due to noise between pitch harmonics. One possible remedy – which we did not yet investigate – consists of using post-denoising techniques, as suggested in [6]. A subset of the samples used in the listening test is available at `https://people.xiph.org/~jm/demo/lpcnet/`.

5. CONCLUSION

This work demonstrates that the efficiency of speaker-independent speech synthesis can be improved by combining newer neural synthesis techniques with the much older technique of linear prediction. In addition to the main contribution of combining linear prediction with WaveRNN, we make other contributions, such as the embedding of the signal values, the improved sampling, as well as the pre-emphasis prior to μ-law quantization. We believe that the proposed model is equally applicable to text-to-speech and to low-bitrate speech coding.

Future work on LPCNet includes investigating whether long-term (pitch) prediction can be incorporated as a way to further reduce the complexity.

[1] `https://keras.io/`

[2] `https://www.tensorflow.org/`

6. REFERENCES

[1] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan, et al., "Natural TTS synthesis by conditioning WaveNet on mel spectrogram predictions," in *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2018, pp. 4779–4783.

[2] S. Arik, G. Diamos, A. Gibiansky, J. Miller, K. Peng, W. Ping, J. Raiman, and Y. Zhou, "Deep voice 2: Multi-speaker neural text-to-speech," *arXiv:1705.08947*, 2017.

[3] J. Sotelo, S. Mehri, K. Kumar, J. F. Santos, K. Kastner, A. Courville, and Y. Bengio, "Char2wav: End-to-end speech synthesis," in *Proc. ICLR Workshop*, 2017.

[4] W. B. Kleijn, F. SC Lim, A. Luebs, J. Skoglund, F. Stimberg, Q. Wang, and T. C. Walters, "WaveNet based low rate speech coding," in *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2018, pp. 676–680.

[5] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," *arXiv:1609.03499*, 2016.

[6] Z. Jin, A. Finkelstein, G. J. Mysore, and J. Lu, "FFTNet: A real-time speaker-dependent neural vocoder," in *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2018, pp. 2251–2255.

[7] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. van den Oord, S. Dieleman, and K. Kavukcuoglu, "Efficient neural audio synthesis," *arXiv:1802.08435*, 2018.

[8] B. S. Atal and S. L. Hanauer, "Speech analysis and synthesis by linear prediction of the speech wave," *The journal of the acoustical society of America*, vol. 50, no. 2B, pp. 637–655, 1971.

[9] J. Markel and A. Gray, "A linear prediction vocoder simulation based upon the autocorrelation method," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 22, no. 2, pp. 124–134, 1974.

[10] D. Griffin and J. Lim, "A new model-based speech analysis/synthesis system," in *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1985, vol. 10, pp. 513–516.

[11] A. McCree, K. Truong, E. B. George, T. P. Barnwell, and V. Viswanathan, "A 2.4 kbit/s MELP coder candidate for the new us federal standard," in *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1996, vol. 1, pp. 200–203.

[12] L. Juvela, V. Tsiaras, B. Bollepalli, M. Airaksinen, J. Yamagishi, and P. Alku, "Speaker-independent raw waveform model for glottal excitation," in *Proc. Interspeech*, 2018, pp. 2012–2016.

[13] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *Proc. Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8)*, 2014.

[14] B.C.J. Moore, *An introduction to the psychology of hearing*, Brill, fifth edition, 2012.

[15] ITU-T, *Recommendation G.711: Pulse Code Modulation (PCM) of voice frequencies*, International Telecommunications Union, 1988.

[16] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. Courville, and Y. Bengio, "SampleRNN: An unconditional end-to-end neural audio generation model," *arXiv:1612.07837*, 2016.

[17] J. Makhoul, "Linear prediction: A tutorial review," *Proc. IEEE*, vol. 63, no. 4, pp. 561–580, 1975.

[18] B. S. Atal and J. Remde, "A new model of LPC excitation for producing natural-sounding speech at low bit rates," in *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1982, vol. 7, pp. 614–617.

[19] M. Schroeder and B.S. Atal, "Code-excited linear prediction (CELP): High-quality speech at very low bit rates," in *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1985, vol. 10, pp. 937–940.

[20] J.-M. Valin, "A hybrid DSP/deep learning approach to real-time full-band speech enhancement," in *Proc. Multimedia Signal Processing Workshop (MMSP)*, 2018.

[21] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *Proc. ICLR*, 2018.

[22] ITU-R, *Recommendation BS.1534-1: Method for the subjective assessment of intermediate quality level of coding systems*, International Telecommunications Union, 2001.