Efficient WaveGlow: An Improved WaveGlow Vocoder with Enhanced Speed

Wei Song, Guanghui Xu, Zhengchen Zhang, Chao Zhang, Xiaodong He, Bowen Zhou

JD AI Research

{songwei11,xuguanghui,zhangzhengchen1,chao.zhang,xiaodong.he,bowen.zhou}@jd.com

Abstract

Neural vocoder, such as WaveGlow, has become an important component in recent high-quality text-to-speech (TTS) systems. In this paper, we propose Efficient WaveGlow (EWG), a flow-based generative model serving as an efficient neural vocoder. Similar to WaveGlow, EWG has a normalizing flow backbone where each flow step consists of an affine coupling layer and an invertible 1×1 convolution. To reduce the number of model parameters and enhance the speed without sacrificing the quality of the synthesized speech, EWG improves WaveGlow in three aspects. First, the WaveNet-style transform network in WaveGlow is replaced with an FFTNet-style dilated convolution network. Next, to reduce the computation cost, group convolution is applied to both audio and local condition features. At last, the local condition is shared among the transform network layers in each coupling layer. As a result, EWG can reduce the number of floating-point operations (FLOPs) required to generate one-second audio and the number of model parameters both by more than 12 times. Experimental results show that EWG can reduce real-world inference time cost by more than twice, without any obvious reduction in the speech quality.

1. Introduction

Speech synthesis, or TTS, is the task to generate speech audio automatically and is an indispensable part of many artificial intelligence applications. Deep learning based end-to-end speech synthesis approaches, such as Char2Wav [1], Tacotron [2], Tacotron2 [3], DeepVoice [4–6], Transformer TTS [7], FastSpeech [8], and ParaNet [9] *etc.*, have shown advantages in speech quality over traditional statistical parametric speech synthesis methods [10, 11]. Although these end-to-end networks could synthesize speech with expressive and natural prosody, a neural vocoder [12–17] is often necessary to synthesize high-quality speech. Compared with the classical source-filter-based WORLD vocoder [18], neural vocoders could generate high fidelity speech with nearly the same quality as human recordings.

Two types of neural vocoders are widely studied at present: the autoregressive vocoders that generate speech in sequence, and the non-autoregressive vocoders that generate speech in parallel. Regarding the autoregressive vocoders, WaveNet [12] stacks many dilated convolutional layers to increase the receptive field over the waveform sequence, while WaveRNN [15] and LPCNet [16] rely on long short-term memory (LSTM) networks [19] to model long-distance dependencies among waveform samples. Regarding non-autoregressive vocoders, multiple samples can be synthesized in parallel, which is particularly useful to speed up the waveform generation since waveform sequences are often very long (*e.g.* an one-second waveform with 16kHz sample-rate has a length of 16,000). Non-autoregressive vocoders can be implemented using generative adversarial networks (GANs) [20], such as MelGAN [21] and WaveGAN [22]. GAN-based neural vocoders use the discriminator models to evaluate the quality of the synthesized speech during training.

Alternatively, the non-autoregressive neural vocoders can be implemented as flow-based models. Parallel WaveNet [13] and ClariNet [23] use inverse autoregressive flow (IAF) [24] to convert white noise to waveform, which can be processed in parallel since the transform of each variable only depends on its direct preceding variable in IAF. Both Parallel WaveNet and ClariNet are trained by distilling pre-trained autoregressive models using teacher-student training and need extra objective functions, which makes them not easy to train. Different from Parallel WaveNet, WaveGlow [25] synthesizes speech by Glow-style normalizing flow [26] instead of IAF and simplifies training to use a single model and a single loss function. On the other hand, WaveGlow has a large model size with 12 coupling blocks and 12 invertible 1×1 convolution layers, and each coupling block consists of a stack of 8 dilated convolution layers. This makes WaveGlow not only hard to use in applications with a constrained memory budget but also overly computational expensive for CPU-based inference.

In this paper, we propose Efficient WaveGlow (EWG), an improvement to WaveGlow that can considerably reduce the numbers of parameters and floating-point operations (FLOPs) required to generate a second of audio, without any obvious degradation in the quality of the synthesized speech. Comparing to WaveGlow, EWG uses the following three modifications:

- using the structure of FFTNet [27] instead of WaveNet for the transform networks;
- using group convolution [28] instead of the standard convolution in each transform network;
- sharing the local conditions, *a.k.a.* the 1×1 convolution kernels, across all layers of the transform network in each coupling layer.

Experimental results showed that EWG with 8 group convolutions and a bidirectional LSTM (BLSTM) Mel-spectrogram encoder can synthesize speech with a similar quality to WaveGlow, while reducing both numbers of model parameters and FLOPs by 12 times. Both GPU- and CPU-based inference time cost is reduced by more than twice without further optimization.

The remainder of this paper is as follows: Section 2 reviews the normalizing flow and WaveGlow. The proposed EWG is introduced in detail in Section 3. Experimental setup and results are presented in Section 4 and 5, followed by conclusions.

2. Preliminaries

2.1. Normalizing flow

A normalizing flow [26, 29–31] converts a probability density to a target probability density by a sequence of invertible mappings. When using an invertible mapping f to transform a random variable $\boldsymbol{z}$ with a distribution $p(\boldsymbol{z})$, the resulting random

Thanks to Xin Yuan for managing the scoring procedure.

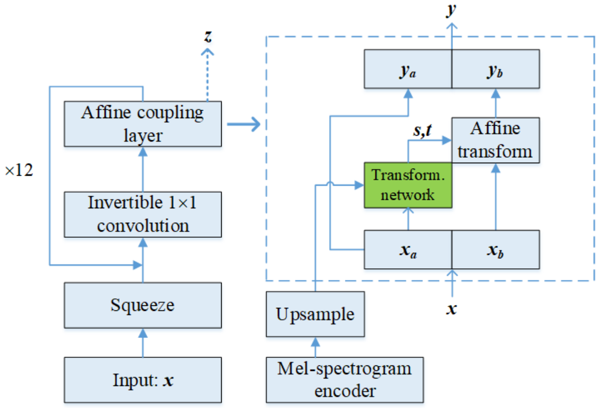

Figure 1: *A sketch map of WaveGlow. The transform network (colored green) is associated with the coupling layers, which is WaveNet-style for WaveGlow and FFTNet-style for EWG.*

variable $\boldsymbol{z}' = f(\boldsymbol{z})$ has a log probability as Eqn. (1) by *change of variables rule*.

$$\log p(\boldsymbol{z}') = \log p(\boldsymbol{z}) + \log \left|\det(\partial \boldsymbol{z}/\partial \boldsymbol{z}')\right| \tag{1}$$

where $\det(\cdot)$ refers to the determinant of a Jacobian matrix.

A random variable $\boldsymbol{x}$ can be obtained by successively transforming a random variable $\boldsymbol{z}_0$ by a chain of invertible mappings:

$$\boldsymbol{z}_0 \sim \mathcal{N}(\boldsymbol{z}_0; \boldsymbol{0}, \mathbf{I}) \tag{2}$$

$$\boldsymbol{x} = f_K ... f_2 \circ f_1(\boldsymbol{z}_0) \tag{3}$$

where $\mathcal{N}(\boldsymbol{z}_0; \boldsymbol{0}, \mathbf{I})$ refers to a multivariate Gaussian distribution with zero mean and unit variance, K is the number of invertible mappings, Eqn. (3) uses a shorthand for a sequence of invertible mappings $f_K(...f_2(f_1(\boldsymbol{z}_0)))$, and we denote $\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1})$.

The log-likelihood of variable $\boldsymbol{x}$ could be calculated directly by using the change of variables rule:

$$\log p(\boldsymbol{x}) = \log p(\boldsymbol{z}_0) + \sum\nolimits_{k=1}^{K} \log |\det(\partial \boldsymbol{z}_{k-1}/\partial \boldsymbol{z}_k)| \tag{4}$$

2.2. WaveGlow

In this paper, WaveGlow [25] is used as our network backbone. WaveGlow is a flow-based generative model, which converts random white noise to speech in parallel. WaveGlow uses the flow structure proposed in Glow [26]. Each flow step contains an affine coupling layer followed by an invertible 1×1 convolution [32]. The structure of WaveGlow is shown in Fig. 1.

In an affine coupling layer, the input feature $\boldsymbol{x}$ is split into two halves along the channel dimension, $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$. $\boldsymbol{x}_a$ remains unchanged while $\boldsymbol{x}_b$ is updated by an affine transform taking $\boldsymbol{x}_a$ as the input. That is,

$$\boldsymbol{x}_a, \boldsymbol{x}_b = \text{split}(\boldsymbol{x}) \tag{5}$$

$$\boldsymbol{s}, \boldsymbol{t} = \text{transform\_network}(\boldsymbol{x}_a, \text{local\_condition}) \tag{6}$$

$$\boldsymbol{y} = \text{concat}(\boldsymbol{x}_a, \boldsymbol{x}_b \odot \boldsymbol{s} + \boldsymbol{t}) \tag{7}$$

where $\boldsymbol{y}$ is the output of the layer. The Jacobian matrix of an affine coupling layer is a lower triangle matrix, whose log determinant equals to the sum of the log of its diagonal elements. Since the transform of the affine coupling layer is invertible, the transform network is invertible.

An invertible 1×1 convolution is followed by each affine coupling layer, to fuse the information carried in each halve and avoid the problem that some channels may never be updated.

In WaveGlow, a WaveNet-style [12] transform network is used, which consists of 8 dilated 1-dimensional (-dim) convolution layers. Each layer uses a convolution kernel with a width of 3, and each sample has a receptive field on both left and right.

The log determinant of the Jacobian matrix of an affine coupling layer, $f^{-1}_{\text{coupling}}(\boldsymbol{x})$, is:

$$\log \left|\det(J(f^{-1}_{\text{coupling}}(\boldsymbol{x})))\right| = \log |\boldsymbol{s}| = \sum\nolimits_{c=1}^{C} \log \boldsymbol{s}_c \tag{8}$$

where C is the number of channels. The log determinant of the Jacobian matrix of an invertible 1×1 convolution layer, $f^{-1}_{1\times 1\text{conv}} = \mathbf{W}\boldsymbol{x}$, is

$$\log \left|\det(J(f^{-1}_{1\times 1\text{conv}}(\boldsymbol{x})))\right| = \log |\det(\mathbf{W})| \tag{9}$$

The training loss of WaveGlow can be presented as:

$$\begin{aligned} \log p\,(\boldsymbol{x}) = & \log p\,(\boldsymbol{z}) \\ & + \sum\nolimits_{k=1}^{K} \log |\det(\mathbf{W}_k)| \\ & + \sum\nolimits_{k=1}^{K} \log |\boldsymbol{s}_k| \end{aligned} \tag{10}$$

where K is the total number of steps in the flow.

3. Efficient WaveGlow

EWG follows the normalizing flow strucutre of Glow with an improved transform network showed in Fig. 1. Three modifications of the transform network are proposed in this paper. First, the WaveNet-style transform network is replaced with an FFTNet. Second, group convolution is used to further reduce the number of model parameters. Third, the local condition is shared among the transform network layers in each flow step.

3.1. FFTNet-style affine transform network

Inspired by the Fast Fourier Transform (FFT) [33], FFTNet [27] creates a network that follows the FFT [33] structure. Given an input audio sequence $\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_{N-1}$, each FFTNet layer clips the input sequence into two halves, $\boldsymbol{x}_L$ and $\boldsymbol{x}_R$. A separate set of 1×1 convolution kernels is used for each halve, and then the results are summed together. Specifically, there is

$$\boldsymbol{z} = \mathbf{W}_L * \boldsymbol{x}_L + \mathbf{W}_R * \boldsymbol{x}_R \tag{11}$$

where $*$ denotes a convolution operator, $\mathbf{W}_L$ and $\mathbf{W}_R$ are the 1×1 convolution kernels for $\boldsymbol{x}_L$ and $\boldsymbol{x}_R$. FFTNet stacks 11 layers to increase the receptive field and the final layer's output is used to predict sample $\boldsymbol{x}_N$. FFTNet substitutes the gated activation in WaveNet with a simple ReLU activation and removes the skip output in each layer. In fact, FFTNet is just a reversed dilated convolution with a kernel width of 2. WaveNet uses increasing dilation from the bottom to the top while FFTNet uses increasing dilation from the top to the bottom.

An FFTNet-style network is used as the transform network to reduce the computational complexity. In EWG, the causal convolution in FFTNet is replaced with the symmetrical convolution with a kernel of width 3, which enlarges the receptive field by twice. In the parallel speech synthesis, the convolution causal is not kept.

$$\boldsymbol{z}_i = \mathbf{W}_L * \boldsymbol{x}_{i-d} + \mathbf{W}_M * \boldsymbol{x}_i + \mathbf{W}_R * \boldsymbol{x}_{i+d} \tag{12}$$

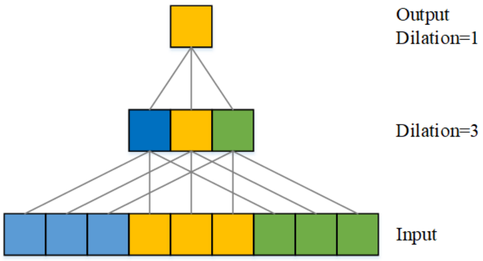

Figure 2: *An example of the FFTNet-style dilated convolution network. Each layer contains a dilated convolution with ReLU activation and a* 1×1 *convolution layer with ReLU activation.*

where $\mathbf{W}_L, \mathbf{W}_M$ and $\mathbf{W}_R$ are $1{\times}1$ convolution kernels for $\boldsymbol{x}_{i-d}, \boldsymbol{x}_i$ and $\boldsymbol{x}_{i+d}$, d is the dilation for current layer, and i is the layer index. Different from FFTNet where separate local condition convolution kernels are used for each halve, only a single $1{\times}1$ convolution kernel is used in our method for the local condition in each layer. That is,

$$\boldsymbol{z}_i = (\mathbf{W}_L * \boldsymbol{x}_{i-d} + \mathbf{W}_M * \boldsymbol{x}_i + \mathbf{W}_R * \boldsymbol{x}_{i+d}) + \mathbf{V}_i * \boldsymbol{h}_i \quad (13)$$

where $\mathbf{V}_i$ is $1{\times}1$ convolution kernel and $\boldsymbol{h}_i$ is the local condition for the i th audio sample.

Our transform network structure is illustrated in Fig. 2, whose output sample has a receptive field of 9 samples. A ReLU [34] function is used for the dilated convolution before using a $1{\times}1$ convolution with another ReLU function, namely $\boldsymbol{x} = \text{ReLU}(1 \times 1\text{conv}(\text{ReLU}(\boldsymbol{z})))$. A residual connection [35] is added in each dilated convolution layer of the transform network to alleviate the gradient vanishing problem.

3.2. Group convolution

The widely used group convolution [28] in the computer vision domain is applied in our experiments to further reduce the numbers of model parameters and FLOPs. We apply group convolution to both audio feature convolution and local condition convolution. If n group is used in group convolution, the FLOPs and number of parameters in one convolution layer could be reduced by n times. Table 1 shows a detailed comparison in the reductions of FLOPs and the number of model parameters for models with different group convolutions.

3.3. Local condition

Mel-spectrogram is used as local condition features for both WaveGlow and EWG, which is encoded by a local condition encoder to get the contextualized features. Two different Mel-spectrogram encoders are used in our experiments. The first one is a BLSTM encoder, which generates a local condition with global and bi-directional contextual information. The second one is a Conv1d encoder where 1-dim convolution layers are used to encode the Mel-spectrogram to extract local contextual information. The performance with both types of encoders is given in Table 3. As the Mel-spectrogram encoder is a frame rate network and the normalizing flow is a sample rate network, the output from the Mel-spectrogram encoder is upsampled by repeating to the sample rate.

Since the 1×1 convolution kernels for local condition accounts for a large part of the total number of parameters, sharing the local condition between the transform network layers in each flow step is also studied. The same as in Eqn. (13), the upsampled local condition is transformed by a 1×1 convolution and then shared across all layers in the transform network. By using such a method of shared local condition (SLC), the computational complexity and number of model parameters are reduced by a large margin, as shown in Table 1.

4. Experimental Setup

We use LJSpeech [36] as our training corpus, which contains 13,100 sentences, encoded in 22,050Hz. We randomly select 200 sentences for test and use all remaining sentences for training. Mel-spectrogram is extracted by using 80 channel filterbank, with a hop size of 256 and a window size of 1024 samples. The Adam [37] learning rate scheduler is used for network optimization with an initial learning rate of 0.001, and the learning rate is exponentially decayed every 50,000 steps. Each model is trained for 1 million steps on 4 Nvidia Tesla P40 GPUs, with synchronized gradient update.

Each convolution kernel is normalized by weight normalization [38] to stabilize the model training procedure, we found the model training process would be easy to fail without weight normalization. The same model structure configuration as WaveGlow with 12 flow steps is used. Each transform network consists of 8 dilated convolution layers, with each of them having 256 filters and a filter width of 3. The dilation numbers for the eight layers in each transform network are set to 128, 64, 32, 16, 8, 4, 2, and 1 respectively. We output 2 channels for every 4 coupling layers, which is the same as WaveGlow.

Regarding the BLSTM Mel-spectrogram encoder, two bi-directional LSTM layers with a hidden size of 128 are used in this paper. For the Conv1d Mel-spectrogram encoder, two 1-dimensional convolution layers with ReLU activations are used, with each of them having 128 filters and a filter width of 5.

5. Experimental Results

5.1. FLOPs and number of model parameters

In Table 1, FLOPs is computed using 86 frames of Mel-spectrograms (about 1 seconds). Compared to WaveGlow with BLSTM Mel-spectrogram encoder (WaveGlow BLSTM), when FFTNet-style transform network is used, EWG BLSTM reduces both FLOPs and number of model parameters by more than 50%. When group convolution with 8 groups is further incorporated, model EWG BLSTM G8 reduces the FLOPs and number of model parameters by more than 12 times. When shared local condition (SLC) is used in EWG with BLSTM encoder, model EWG BLSTM SLC reduces the FLOPs and number of model parameters by more than 4 times compared with the WaveGlow BLSTM model. And further more, if group convolution with 8 groups is combined into model EWG BLSTM SLC, model EWG BLSTM SLC G8 achieves 16 times FLOPs reduction and 15 times parameters reduction.

The same FLOPs and number of model parameter reduction patterns could be found for models with Conv1d Mel-spectrogram encoder.

5.2. Inference speed analysis

We use 400 frames (4.64 seconds) Mel-spectrograms to evaluate the network inference speed both on CPU [1] and GPU (Nvidia Tesla P40) without any inference optimization. Regrading the inference on CPU, we restrict the computation to only 4

[1] Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz

Table 1: *The numbers of FLOPs and of model parameters (#Params) for models with BLSTM/Conv1d Mel-spectrogram encoder. M and B stand million and billion respectively.*

Model	FLOPs		#Params	
	BLSTM	Conv1d	BLSTM	Conv1d
WaveGlow	833B	551B	152M	101M
EWG	417B	278B	76M	51M
EWG G4	96B	96B	24M	18M
EWG G8	65B	65B	12M	12M
EWG SLC	174B	156B	32M	29M
EWG SLC G8	52B	52B	10M	10M

Table 2: *The inference time cost on a CPU or a P40 GPU for models with BLSTM/Conv1d Mel-spectrogram encoder, and s stands for second.*

Model	BLSTM		Conv1d	
	CPU	P40	CPU	P40
WaveGlow	31.50s	0.80s	19.40s	0.60s
EWG	21.63s	0.50s	12.80s	0.36s
EWG G4	16.53s	0.38s	8.76s	0.27s
EWG G8	15.20s	0.37s	8.03s	0.24s
EWG SLC	7.39s	0.23s	6.30s	0.21s
EWG SLC G8	4.70s	0.15s	4.00s	0.13s

cores to mimic limited computation resources.

In Table 2, it could be see that EWG models reduce the inference time by more than 1.6 times on P40, both for Conv1d and BLSTM Mel-spectrogram encoders. When group convolution with 8 groups is used, model EWG Conv1d G8 achieves a 2.5 times speedup and model EWG BLSTM G8 achieves a 2.2 times speedup on P40. With shared local condition and group convolution, model EWG Conv1d SLC G8 and EWG BLSTM SLC G8 speed up the inference speed by 4.6 and 5.3 times respectively.

Regarding the inference speed on CPU, compared with the WaveGlow baseline model, model EWG BLSTM SLC G8 and EWG Conv1d SLC G8 reduce the time cost by 6.7 and 4.9 times separately and achieve real-time speech synthesis on CPU.

The inference speedup gain is not as large as gains with the reduction of the FLOPs and number of model parameters, this is due to the highly optimized matrix multiplication implementation and inner parallelization of PyTorch. With the huge reduced FLOPs and number of model parameters, it could be much easier for engineers to optimize the inference process, and it would also be possible for model deployment with restricted memory or CPU environment.

5.3. Subjective evaluation

Mean opinion score (MOS) test [2] is conducted to evaluate the quality of the synthesized speech. We randomly select 20 sentences in the test set and synthesize audios by different models, each audio is listened by 20 testers that consist of native English speakers and professional bilinguals.

Table 3: *The MOSs of models with BLSTM/Conv1d Mel-spectrogram encoder.*

Model	BLSTM	Conv1d
Ground Truth	4.55±0.11	
WaveGlow	3.82±0.06	3.49±0.13
EWG	3.79±0.07	2.81±0.08
EWG G4	3.66±0.08	3.17±0.04
EWG G8	3.69±0.07	3.15±0.07
EWG SLC	3.65±0.07	3.32±0.05
EWG SLC G8	3.65±0.07	3.26±0.05

In Table 3, models with BLSTM Mel-spectrogram encoder surpass models with Conv1d Mel-spectrogram encoder, this shows that BLSTM Mel-spectrogram encoder could extract more robust and representative local condition features which improves the model quality both for WaveGlow and EWG. Compared to model WaveGlow BLSTM with a score of 3.82, model EWG BLSTM has a nearly the same score of 3.79 and model EWG BLSTM G8 has a score of 3.69. The quality degradation for model EWG BLSTM G8 is acceptable but with more than 12 times fewer number of model parameters. Model EWG BLSTM SLC G8 has a nearly same score with model EWG BLSTM G8, this indicates that the shared local condition could also be leveraged to further reduce model parameters.

5.4. Discussion

FFTNet [27] synthesizes high-quality speech with a more simple network structure compared with WaveNet, so in this paper, we use the FFTNet style transform network, experimental results showed that model EWG BLSTM performs identically to model WaveGlow BLSTM.

Each transform network consists of 8 dilated convolution layers and EWG has 12 transform networks, we assume that there are lots of redundant local condition features if each dilated convolution layer in each transform network has a separate 1×1 convolution kernel. So we use shared local condition among transform network layers in each flow step to remove redundant features and reduce the number of model parameters, experimental results showed that with shared local condition model EWG BLSTM SLC G8 has a similar MOS score as model EWG BLSTM G8.

6. Conclusions

EWG, an efficient flow-based generative model, is proposed and used as a neural vocoder in this paper. Compared with WaveGlow, EWG can synthesize speech with a similar high-quality, while significantly reducing both the number of FLOPs for generating the same piece of audio and the number of model parameters. Based on the WaveGlow architecture, to reduce the computation and storage complexities, the FFTNet-style instead of WaveNet-style transform network is used in the coupling layers, and group convolution is further applied in the transform networks. For the vocoder models with BLSTM Mel-spectrogram encoder and group convolution, both FLOPs and the number of model parameters can be reduced by more than 12 times. Further improvement could be achieved when the shared local condition is incorporated. Experiments showed that the use of EWG can reduce the real-world time cost for speech synthesis by at least 50% with either CPU or GPU, and can synthesize speech without an obvious degradation in the mean opinion scores.

[2] The audio samples can be found at `http://weixsong.github.io/demos/EfficientWaveGlow/index.html`

7. References

[1] 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*, Toulon, 2017.

[2] Y. Wang, R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, L. Quoc, and C. R. S. R. Agiomyrgiannakis, Yannis, "Tacotron: Towards end-to-end speech synthesis," in *Proc. Interspeech*, Stockholm, 2017.

[3] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu, "Natural TTS synthesis by conditioning wavenet on Mel spectrogram predictions," in *Proc. ICASSP*, Calgary, 2018.

[4] S. Ö. Arik, M. Chrzanowski, A. Coates, G. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, A. Ng, J. Raiman *et al.*, "Deep voice: Real-time neural text-to-speech," in *Proc. ICML*, Sydney, 2017.

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

[6] W. Ping, K. Peng, A. Gibiansky, S. O. Arik, A. Kannan, S. Narang, J. Raiman, and J. Miller, "Deep voice 3: Scaling text-to-speech with convolutional sequence learning," in *arXiv:1710.07654*, 2017.

[7] N. Li, S. Liu, Y. Liu, S. Zhao, and M. Liu, "Neural speech synthesis with Transformer network," in *Proc. AAAI*, Honolulu, 2019.

[8] Y. Ren, Y. Ruan, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, "FastSpeech: Fast, robust and controllable text to speech," in *Proc. NIPS*, Vancouver, 2019.

[9] K. Peng, W. Ping, Z. Song, and K. Zhao, "Parallel neural text-to-speech," in *arXiv:1905.08459*, 2019.

[10] Z. Wu, O. Watts, and S. King, "Merlin: An open source neural network speech synthesis system," in *Proc. SSW*, Sunnyvale, 2016.

[11] H. Zen, K. Tokuda, and A. Black, "Statistical parametric speech synthesis," *Speech Communication*, vol. 51, no. 11, pp. 1039–1064, 2009.

[12] 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," in *arXiv:1609.03499*, 2016.

[13] A. van den Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. v. d. Driessche, E. Lockhart, L. C. Cobo, F. Stimberg, N. Casagrande, D. Grewe, S. Noury, S. Dieleman, E. Elsen, N. Kalchbrenner, H. Zen, A. Graves, H. King, T. Walters, D. Belov, and D. Hassabis, "Parallel WaveNet: Fast high-fidelity speech synthesis," in *Proc. ICML*, Sydney, 2017.

[14] 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," in *arXiv:1612.07837*, 2016.

[15] 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," in *arXiv:1802.08435*, 2018.

[16] J.-M. Valin and J. Skoglund, "LPCNet: Improving neural speech synthesis through linear prediction," in *Proc. ICASSP*, Brighton, 2019.

[17] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, "GANSynth: Adversarial neural audio synthesis," *arXiv:1902.08710*, 2019.

[18] M. Morise, F. Yokomori, and K. Ozawa, "WORLD: A vocoder-based high-quality speech synthesis system for real-time applications," *IEICE Transactions on Information and Systems*, vol. 99, no. 7, pp. 1877–1884, 2016.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. NIPS*, Montreal, 2014.

[21] K. Kumar, R. Kumar, T. de Boissiere, L. Gestin, W. Z. Teoh, J. Sotelo, A. de Brébisson, Y. Bengio, and A. C. Courville, "MelGAN: Generative adversarial networks for conditional waveform synthesis," in *Proc. NIPS*, Vancouver, 2019.

[22] C. Donahue, J. McAuley, and M. Puckette, "Adversarial audio synthesis," *arXiv:1802.04208*, 2018.

[23] W. Ping, K. Peng, and J. Chen, "ClariNet: Parallel wave generation in end-to-end text-to-speech," *arXiv:1807.07281*, 2018.

[24] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, "Improved variational inference with inverse autoregressive flow," in *Proc. NIPS*, Barcelona, 2016.

[25] R. Prenger, R. Valle, and B. Catanzaro, "WaveGlow: A flow-based generative network for speech synthesis," in *Proc. ICASSP*, Brighton, 2019.

[26] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Proc. NIPS*, Montreal, 2018.

[27] Z. Jin, A. Finkelstein, G. J. Mysore, and J. Lu, "FFTNet: A real-time speaker-dependent neural vocoder," in *Proc. ICASSP*, Calgary, 2018.

[28] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. CVPR*, Salt Lake City, 2018.

[29] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proc. ICML*, Lille, France, 2015.

[30] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using Real NVP," in *arXiv:1605.08803*, 2016.

[31] L. Dinh, D. Krueger, and Y. Bengio, "NICE: Non-linear independent components estimation," in *arXiv:1410.8516*, 2014.

[32] M. Lin, Q. Chen, and S. Yan, "Network in network," in *arXiv:1312.4400*, 2013.

[33] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[34] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. ICML*, Haifa, 2010.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, Las Vegas, 2016.

[36] K. Ito, "The LJ Speech Dataset," https://keithito.com/LJ-Speech-Dataset, 2017.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, San Diego, 2015.

[38] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Proc. NIPS*, Barcelona, 2016.