
FastPitch: Parallel Text-to-speech with Pitch Prediction

Adrian Łańcucki
NVIDIA Corporation
alancucki@nvidia.com

Abstract

We present FastPitch, a fully-parallel text-to-speech model based on FastSpeech, conditioned on fundamental frequency contours. The model predicts pitch contours during inference, and generates speech that could be further controlled with predicted contours. FastPitch can thus change the perceived emotional state of the speaker or put emphasis on certain lexical units. We find that uniformly increasing or decreasing the pitch with FastPitch generates speech that resembles the voluntary modulation of voice. Conditioning on frequency contours improves the quality of synthesized speech, making it comparable to state-of-the-art. It does not introduce an overhead, and FastPitch retains the favorable, fully-parallel Transformer architecture of FastSpeech with a similar speed of mel-scale spectrogram synthesis, orders of magnitude faster than real-time.

1 Introduction

Recent advances in neural text-to-speech (TTS) systems brought real-time synthesis of naturally sounding, human-like speech. The leading models are auto-regressive and computationally expensive due to the high temporal resolution of the audio signal. Feed-forward models are able to synthesize mel-spectrograms orders of magnitude faster than auto-regressive models. For instance FastSpeech [21], which is based on the feed-forward Transformer [5], explicitly predicts the duration of every input symbol, and infers the entire spectrogram in parallel. Feed-forward models synthesize reasonably sounding speech even when conditioned on imperfect alignments of output frames with input symbols. However, these models still fail to match the quality of auto-regressive generation, and pose several challenges in training [21, 31].

Text-to-speech models are often conditioned on additional qualities of speech such as linguistic features and fundamental frequency [26, 27]. Introduction of neural networks into TTS made possible accurate modeling of these qualities. In particular, fundamental frequency modeled with a neural network has been repeatedly shown to improve the quality of synthesized speech in a concatenative model [6], and later fully neural models [11, 12]. Conditioning on fundamental frequency with voiced/unvoiced decisions is a common approach to augmenting the model with singing capabilities [24, 13], or adaptability to other speakers [11].

In this paper we propose FastPitch, a feed-forward model based on FastSpeech that improves FastSpeech and matches the state-of-the-art auto-regressive TTS models by conditioning on fundamental frequency estimated for every input symbol, which we refer to simply as a pitch contour. We show that explicit modeling of such pitch contours addresses the quality shortcomings of feed-forward Transformer. These shortcomings most likely arise from collapsing different pronunciations of the same phonetic units in the absence of enough linguistic information in the textual input alone. By conditioning on fundamental frequency, we provide the model with more linguistic information and prevent this collapse. In addition, conditioning on fundamental frequency improves convergence, and eliminates the need for knowledge distillation of mel-spectrogram targets used in FastSpeech.

Because the model learns to predict pitch, it gains new practical applications like shifting the fundamental frequency, putting emphasis, adding expressiveness, and interactive editing of the pitch contour during synthesis. Constant offsetting of F_0 values with FastPitch produces naturally sounding low- and high-pitched variations of voice that preserve the perceived identity of the person, and outperforms the ordinary pitch shifting algorithms. We conclude that the model is expressive and learns to mimic the action of vocal chords, which happens during the voluntary modulation of voice.

Combined with WaveGlow [19], FastPitch is able to synthesize mel-spectrograms over $30\times$ faster than real-time, using PyTorch library functions and not resorting to kernel-level optimizations [18]. In Mean Opinion Score evaluations, FastPitch scored higher than our implementation of Tacotron2 [23].

We would like to note that a concurrently developed idea of pitch prediction in FastSpeech [20] has been published in between the releases of our source code and this paper describing FastPitch.

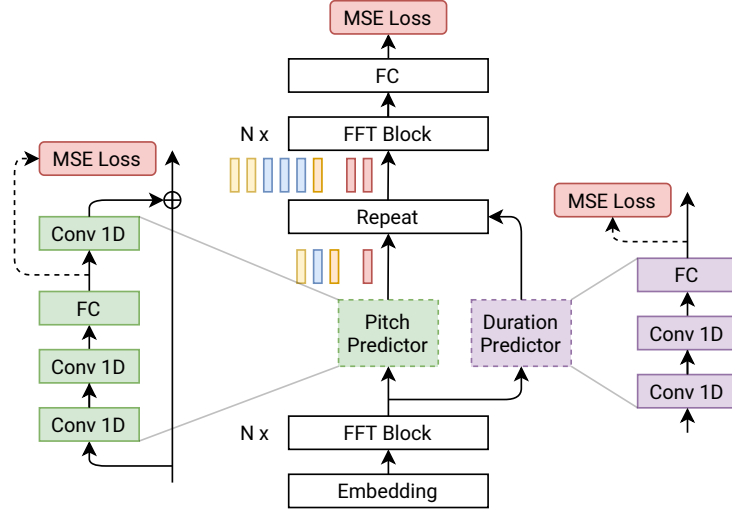


Figure 1: **Architecture of FastPitch** follows FastSpeech [21]. The additional PitchPredictor module has a similar architecture as DurationPredictor and predicts a single scalar for every temporal location. Those values are embedded with a 1-D convolution, and added to the signal through a residual connection. The model is trained on ground truth input symbol durations \mathbf{d} and pitch values \mathbf{p} . The predicted values $\hat{\mathbf{d}}$ and $\hat{\mathbf{p}}$ are used during inference.

2 Model Description

The architecture of FastPitch is shown in Figure 1. It is based on FastSpeech and composed mainly of two feed-forward Transformer (FFT) stacks [5]. The first one operates in the resolution of input tokens, the second one in the resolution of the output frames. Let $\mathbf{x} = (x_1, \dots, x_t)$ be the sequence of input lexical units, and $\mathbf{y} = (y_1, \dots, y_T)$ be the sequence of target mel-scale spectrogram frames. The first FFT stack produces the hidden representation

$$\mathbf{h} = \text{FFTransformer}(\mathbf{x}). \quad (1)$$

The hidden representation \mathbf{h} is used to make predictions about the duration and average pitch of every character with a 1-D CNN

$$\begin{aligned} \hat{\mathbf{d}} &= \text{DurationPredictor}(\mathbf{h}) \\ \hat{\mathbf{p}} &= \text{PitchPredictor}(\mathbf{h}) \end{aligned} \quad (2)$$

where $\mathbf{d} \in \mathbb{N}^t$ and $\mathbf{p} \in \mathbb{R}^t$. Next, the pitch is projected to match the dimensionality of the hidden representation $\mathbf{h} \in \mathbb{R}^{t \times d}$ and added to \mathbf{h} . The resulting sum \mathbf{g} is discretely up-sampled and passed

to the output FFT, which produces the output mel-spectrogram sequence

$$\begin{aligned} g &= \mathbf{h} + \text{PitchEmbedding}(\mathbf{p}) \\ \hat{\mathbf{y}} &= \text{FFTransformer}(\underbrace{[g_1, \dots, g_1]}_{d_1}, \underbrace{[g_2, \dots, g_2]}_{d_2}, \dots, \underbrace{[g_t, \dots, g_t]}_{d_t}). \end{aligned} \quad (3)$$

The model optimizes mean-squared error (MSE) between the predicted and ground-truth modalities

$$\mathcal{L} = \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 + \alpha \|\hat{\mathbf{p}} - \mathbf{p}\|_2^2 + \gamma \|\hat{\mathbf{d}} - \mathbf{d}\|_2^2 \quad (4)$$

Ground truth \mathbf{p} and \mathbf{d} are used during training. During inference, predicted $\hat{\mathbf{p}}$ and $\hat{\mathbf{d}}$ are used during inference.

2.1 Duration of Input Symbols

Durations of input symbols are estimated with a Tacotron2 model trained on LJSpeech-1.1. Let $\mathbf{A} \in \mathbb{R}^{t \times T}$ be the final Tacotron2 attention matrix. The duration of the i th input symbol is [21]:

$$d_i = \sum_{c=1}^T [\arg \max_r \mathbf{A}_{r,c} = i]. \quad (5)$$

Because Tacotron2 has a single attention matrix, we do not need to choose between attention heads, as it would be necessary with a multi-head Transformer model.

FastPitch is robust to the quality of alignments. We observe that durations extracted with distinct Tacotron2 models tend to differ. E.g., for the same input utterance, the longest durations are assigned approximately at the same locations, but possibly to different characters (Figure 2.1). Surprisingly, those different alignment models produce FastPitch models, which synthesize speech of similar quality. We found that even constant durations, e.g., five output frames per input character, still produce intelligible speech.

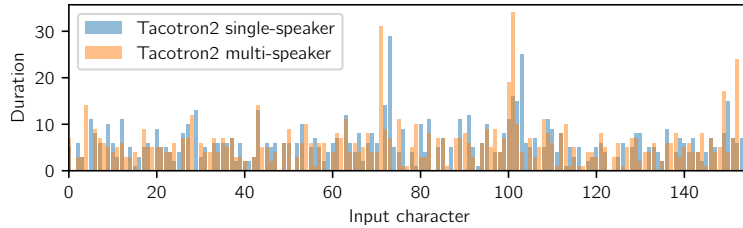


Figure 2: **Varying character durations** extracted with different Tacotron2 models allow to train FastPitch checkpoint of comparable quality (cf. MOS scores in Table 1 and Table 2). The examined utterance comes from the LJSpeech-1.1 training subset.

2.2 Pitch of Input Symbols

We obtain ground truth pitch values through acoustic periodicity detection using the accurate autocorrelation method [3]. Let \mathbf{a} be the windowed signal, calculated using Hann windows. The algorithm finds an array of maxima of the normalized autocorrelation function r_x

$$\begin{aligned} r_a(\tau) &= \frac{\int_0^{T-\tau} a_t a_{t+\tau} dt}{\int_0^T a_t^2 dt} \\ r_x(\tau) &= \frac{r_a(\tau)}{r_w(\tau)}, \end{aligned} \quad (6)$$

where r_a denotes autocorrelation of the windowed signal, and r_w autocorrelation of the Hann window, which has a closed form [3]. These maxima become the candidate frequencies. The unvoiced candidate is present at every time step. Then, the lowest-cost path through the array of candidates

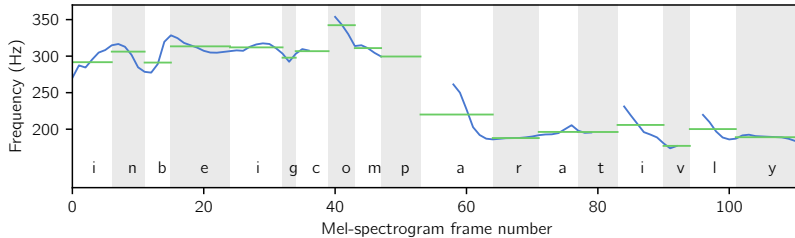


Figure 3: **Fundamental frequency** estimated for the utterance *In being comparatively*. Raw values are shown in blue, values averaged over input characters in green. Character durations have been extracted with a Tacotron2 model. Silent characters have zero duration and no fundamental frequency.

is calculated with the Viterbi algorithm. The path minimizes the transitions between the candidate frequencies

$$TransitionCost(F_1, F_2) = \begin{cases} 0 & \text{if } F_1 = 0 \text{ and } F_2 = 0 \\ VoicedUnvoicedCost & \text{if } F_1 = 0 \text{ xor } F_2 = 0 \\ OctaveJumpCost & \text{if } F_1 \neq 0 \text{ and } F_2 \neq 0. \end{cases} \quad (7)$$

In order to get one F_0 value for every mel-scale spectrogram frame, we change the default window size from 0.01 s to 0.0116 s, which corresponds to the input span for a single mel-scale spectrogram frame, calculated using Short-time Fourier Transform (STFT) with hop size 256 and sampling rate 22 050 Hz.

F_0 values are averaged over every input symbol using the extracted durations d (Figure 3). Unvoiced values are excluded from the calculation. For training, the values are standardized to mean of 0 and standard deviation of 1. If there are no voiced F_0 estimates for a particular symbol, its pitch is being set to 0.

Training on non-averaged F_0 values, i.e., one value per every mel-spectrogram frame, introduced artifacts and inconsistencies into the synthesized speech, even when the model has been conditioned on ground-truth pitch contours, meaning that the model was not able to learn to predict, nor synthesize in a coherent manner. Similar artifacts and behavior have been observed in the models which synthesize singing voices [24, 13]

Following [11], we tried averaging to three pitch values per every symbol, in the hope to capture the beginning, middle and ending pitch for every symbol. The quality of speech with three pitch values per input character did not have the artifacts introduced by one pitch value per every output frame. However, during a blind test it was judged inferior to having one pitch value for every input character (Section 3.2).

3 Experiments

The source code¹ for FastPitch and synthesized samples² are available online. We synthesize waveforms for evaluation for all models with a single pre-trained WaveGlow vocoder [19].

3.1 Setup

The model is trained on the publicly available LJSpeech 1.1 dataset [8] which contains approximately 24 hours of single-speaker speech recorded at 22 050 Hz. We manually correct erroneous transcriptions for samples LJ034-0138 and LJ031-0175, which we have discovered during an inspection with a speech recognition model.

Different authors tend to use custom training/development/test splits of LJSpeech 1.1. We are cautious to use the same split, which was used to train WaveGlow. Overlooking this detail, especially

¹<https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/SpeechSynthesis/FastPitch>

²<https://fastpitch.github.io/>

Table 1: **Single speaker Mean Opinion Scores** gathered on Amazon Turk. Both models have been trained in a similar setup on the LJSpeech-1.1 dataset and character-level input.

Model	MOS
Tacotron2 (Mel + WaveGlow)	3.946 \pm 0.134
FastPitch (Mel + WaveGlow)	4.080 \pm 0.133

when using an off-shelf WaveGlow checkpoint, can easily leak the training data during evaluation and inflate the results. We calculate 80-band mel-scale spectrograms with Short-time Fourier transform using window length of 1024 samples and hop size of 256.

Parameters of the model mostly follow FastSpeech [21]. The embedded and hidden vectors in the network have $d = 384$ dimensions. The input and output FF Transformer modules have 6 layers each, and a single 64-dimensional attention head. Each feed-forward block of a FF Transformer layer is composed of a 1-D convolution with kernel size 3 and 384/1536 input/output channels, ReLU activation, a second 1-D convolution with kernel size 3 and 1536/384 input/output filters, followed by Dropout and Layer Norm.

Duration Predictor and Pitch Predictor have the same convolutional architecture. They are composed of a 1-D convolution with kernel size 3 and 384/256 channels, and a second 1-D convolution with 256/256 channels, each followed by ReLU, Layer Norm and Dropout layers. The last layer is linear and projects every 256-channel vector into a single scalar. We use Dropout rate of 0.1 in every Dropout layer, including Dropout on attention heads.

The described models were trained on characters. We tried the mixed approach of training simultaneously on phonemes and characters [9]. Both word- and sentence-level mixing introduced unpleasant artifacts into the synthesized speech. Standard input cleaning has been applied: numerals and common abbreviations have been heuristically expanded (*20* \rightarrow *twenty*, *dr.* \rightarrow *doctor*, etc.). The sentences have been converted to lower-case ASCII characters.

FastPitch has been trained on $8 \times$ NVIDIA V100 GPUs with automatic mixed precision [16], although similar results can be achieved with longer, full precision training. The training converges after 2 hours, and full training takes 5.5 hours. We use the LAMB optimizer [30] with learning rate 0.1, $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 1e-9$. In comparison with ADAM, it stabilizes the training at higher learning rates. Learning rate is increased for a warmup period of 1000 steps, and then decayed according to the Transformer schedule [28]. We apply weight decay of $1e-6$, and train with batch size of 256, split into 32 samples per GPU.

3.2 Evaluation

3.2.1 MOS Scores

We have compared our FastPitch model with Tacotron2 (Table 1). The samples have been scored on Amazon Turk with the Crowdsourced Audio Quality Evaluation Toolkit [4]. We have generated speech from the first 30 samples from our development subset of the LJSpeech-1.1. At least 250 scores have been gathered per every model, with the total of 60 unique Turkers participating in the study. In order to qualify, the Turkers were asked to pass a hearing test.

3.2.2 Pairwise Comparisons

Generative models pose difficulties when it comes to hyperparameter tuning. The quality of generated samples is highly subjective, and running large-scale studies time-consuming and costly. While major differences in quality of the outputs are captured by the developers, minor adjustments are challenging. In order to efficiently score a higher number of models and avoid score drift when the same person scores samples over a long period of time, we have investigated the approach of comparing pairs of samples. Pairwise comparisons allow to construct a global ranking assuming that skill ratings are transitive [1]. The approach is best known from building large-scale rankings of human players in chess, sports and on-line games.

We have run an internal study, in which over 50 participants scored randomly selected pairs of samples, although more efficient selection strategies than random exist [29]. We used the Glicko-2

rating system [7] to build a ranking based on those scores. It is known in the context of automatic scoring of generative models [17]. Every participant scored 20 pairs on average. We present the scores for a relevant subset of scored models (Figure 4): FastPitch with 1, 2 and 4 attention heads, 6 and 10 transformer layers, and pitch predicted in the resolution of three values per input token. The method allowed us to assess the effect of a number of hyper-parameters on the subjective quality of the synthesized speech. In addition, we found it crucial for efficient model development, even with a handful of evaluators. An evolving ranking kept during development facilitates tracking multiple hyperparameter settings.

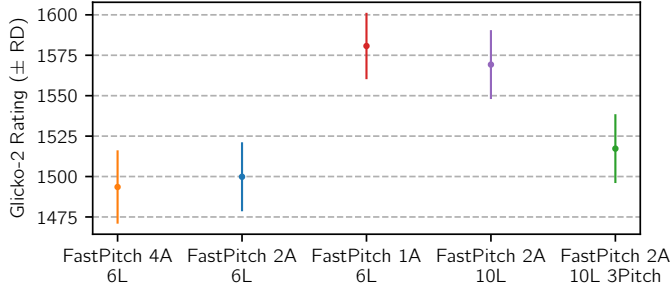


Figure 4: **Glicko-2 ranking** of different configurations of FastPitch. The compared models have different numbers of attention heads (A), layers (L), and pitch averaged to three values per input symbol instead of one (3Pitch).

3.3 Pitch and Speaker Conditioning

The pitch contour can be modified during inference to control certain perceived qualities of the generated speech. For instance, it can be used to increase or decrease F_0 , put emphasis, or increase the variance of pitch. Among the published audio samples accompanying this paper we demonstrate the effects of increasing, decreasing or inverting the frequency around the mean value for a single utterance. We encourage the reader to listen to them.

Figure 5 shows an example of shifting the frequency by 50 Hz. Compared to simple shifting in the frequency domain, FastPitch preserves the perceived identity of the speaker, and models the action of vocal chords that happens during voluntary modulation of voice.

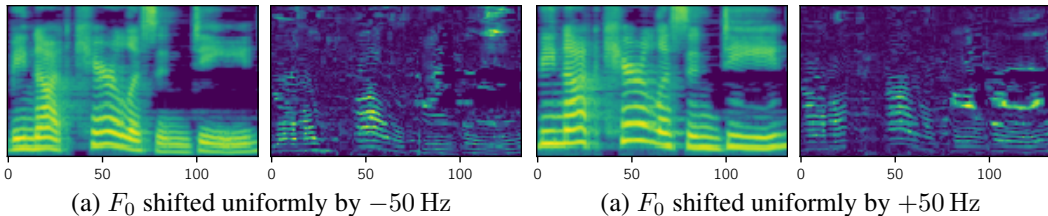


Figure 5: **Shifting F_0 with FastPitch** by adding a constant to the predicted pitch \hat{p} during inference. Pairs are displayed: a shifted spectrogram, and the absolute difference between shifted and unshifted spectrograms. FastPitch demonstrates a high degree of expressiveness, which makes the shifted samples sound natural.

Further examples in which we use FastPitch to modify pitch are presented in Figure 6. These have been generated by applying simple, pre-defined transformations to predicted pitch \hat{p} . FastPitch exhibits a wide range of expressiveness, making non-trivial changes to the output mel-spectrograms when conditioned on the transformer pitch vectors.

On a single NVIDIA V100 GPU, FastPitch easily achieves real-time factor over $30\times$ for a complete synthesis from text to audio, when combined with a WaveGlow vocoder, which still is the bottleneck. Because one pitch value per input symbol is easy to interpret by a human, FastPitch is suitable to applications like real-time editing of synthesized samples. Pitch contours displayed in Figure 6

show how a simple interface for this kind of editing might look like, where the pitch values could be manually adjusted.

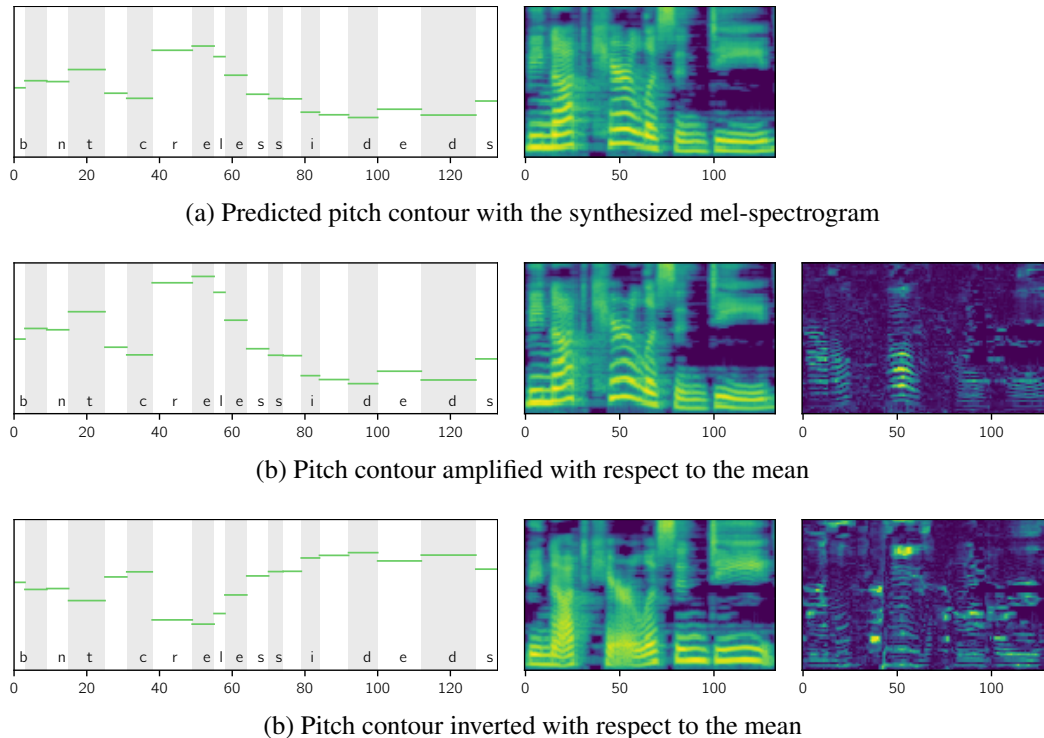


Figure 6: **The effect of modifying the pitch contour** for the phrase *Be not careless in deeds*. The predicted pitch values, at the resolution of a single value per input character, are suitable for building simplified interfaces for manual adjustments. Here automatic adjustments have been applied: (b) amplifying the pitch by linearly increasing the distance from the mean over the whole phrase, and (c) inverting with respect to the mean.

3.3.1 Multiple Speakers

FastPitch is easy to extend to multiple speakers. We have trained a model on the LJSpeech-1.1 dataset with additional internal training data coming from two female speakers: Sally (8330 samples with the total of 13.6 h), and Helen (18995 samples with the total of 17.3 h). We condition the model on the speaker by adding a global speaker embedding to the input tokens x . To compare, we have chosen multi-speaker Tacotron2 and FlowTron [25]. The latter is an auto-regressive flow-based model. All models have been trained on the same data, and the multi-speaker Tacotron2 has been used to extract training alignments for FastPitch. The results are summarized in Table 2.

Table 2: **Multi-speaker Mean Opinion Scores**, evaluated on samples from the LJSpeech development set. Tacotron2 and FlowTron models have been trained on mixed grapheme and phoneme inputs, FastPitch on only character inputs. The models were trained on a three-speaker dataset with roughly 57 h of training data.

Model	MOS
Tacotron2 (Mel + WaveGlow)	3.707 \pm 0.218
FlowTron (Mel + WaveGlow)	3.882 \pm 0.159
FastPitch (Mel + WaveGlow)	4.071 \pm 0.164

3.4 Discussion

The success of the Transformer architecture in domains of relatively low bandwidth such as language modelling or translation, in part comes from the ability to memorize and combine small fragments of the training data, even more so than other architectures. The output predictions of the FF Transformer are made in parallel and independently, which pushes the model towards modeling the mean pronunciation, rather than focusing on a specific way of pronunciation chosen beforehand.

We observed this behavior, that is the reliance on interpolation between memorized samples and collapsing between different ways of a pronunciation, in the original FastSpeech. Conditioning on the aggregated pitch seems resolve much of these problems, making the model outputs more coherent, and focused sharply on a single pronunciation. FastSpeech was reported to have an improved quality with knowledge distillation of mel-spectrogram targets from a pre-trained Transformer-TTS model [15]. We conjecture that the curated modes of pronunciation of certain phrases, learned by the Transformer-TTS model, were easier training targets for the parallel FastSpeech, and in some cases prevented the collapse to the mean pronunciation.

4 Related Work

The predominant paradigm in text-to-speech is two-stage synthesis: first producing mel-scale spectrograms from text, and then the actual sound waves with a vocoder model [18, 23, 14]. In attempts to speed up the synthesis, parallel models have been explored. In addition to Transformer models investigated in this work [21], convolutional GAN-TTS [2] is able to synthesize raw audio waveforms with state-of-the-art quality. It is conditioned on linguistic and pitch features.

The efforts in parallelizing existing models include duration prediction similar to FastSpeech, applied to Tacotron [22] or a generative flow model Glow-TTS [10]. Explicit modeling of duration has rekindled the interest in automatic alignment in order to relieve from bootstrapping the models with forced alignments. These approaches typically use dynamic programming algorithms associated with inference and training of HMMs. Glow-TTS aligns with Viterbi paths, and FastSpeech has been improved with a variant of the forward-backward algorithm [31].

Lastly, we describe the details of explicit neural modeling of pitch introduced alongside a neural TTS voice conversion model [11], which shares similarities with other models from IBM Research [6, 12]. An LSTM-based Variational Autoencoder generation network modeled prosody, and pitch was calculated with a separate tool prior to the training. Prosody information was encoded in vectors of four values: log-duration, start log-pitch, end log-pitch, and log-energy. There were three prosody vectors for every phoneme, corresponding to the HMM states from the auxiliary model which supplied alignments. Apart from prosody vectors, sparse textual features were used as inputs.

5 Conclusions

We have presented FastPitch, a parallel text-to-speech model based on FastSpeech, able to rapidly synthesize high-fidelity mel-scale spectrograms with a high degree of control over the prosody. The model demonstrates how conditioning on prosodic information can significantly improve the convergence and quality of synthesized speech in a feed-forward model, enabling more coherent pronunciation across its independent outputs, and lead to state-of-the-art results. Our pitch conditioning method is simpler than many of the approaches known from the literature. It does not introduce an overhead, and opens up possibilities for practical applications in adjusting the prosody interactively, as the model is fast, highly expressive, and presents potential for multi-speaker scenarios.

Acknowledgements

The author would like to thank Dabi Ahn, Alvaro Garcia, and Grzegorz Karch for their help with the experiments and evaluation of the model, and Jan Chorowski, João Felipe Santos, James Sohn, Przemek Strzelczyk, and Rafael Valle for helpful discussions and support in preparation of this paper.

References

- [1] Timo Baumann. Large-Scale Speaker Ranking from Crowdsourced Pairwise Listener Ratings. In *INTERSPEECH*, pages 2262–2266, 2017.
- [2] Mikołaj Bińkowski, Jeff Donahue, Sander Dieleman, Aidan Clark, Erich Elsen, Norman Casagrande, Luis C. Cobo, and Karen Simonyan. High fidelity speech synthesis with adversarial networks. *arXiv:1909.11646*, 2019. [arXiv:1909.11646](https://arxiv.org/abs/1909.11646).
- [3] Paul Boersma. Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound. *IFA Proceedings*, pages 97–110, 1993.
- [4] M. Cartwright, B. Pardo, G. J. Mysore, and M. Hoffman. Fast and easy crowdsourced perceptual audio evaluation. In *ICASSP 2016*, pages 619–623, 2016.
- [5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL 2019*, pages 4171–4186, 2019.
- [6] Raul Fernandez, Asaf Rendel, Bhuvana Ramabhadran, and Ron Hoory. Using deep bidirectional recurrent neural networks for prosodic-target prediction in a unit-selection text-to-speech system. In *INTERSPEECH*, pages 1606–1610, 2015.
- [7] Mark E. Glickman. *Example of the Glicko-2 system*, 2013. URL: <http://www.glicko.net/glicko.html>.
- [8] Keith Ito. The LJ Speech Dataset. <https://keithito.com/LJ-Speech-Dataset>, 2017.
- [9] Kyle Kastner, Joao Felipe Santos, Yoshua Bengio, and Aaron Courville. Representation Mixing for TTS Synthesis. In *ICASSP*, pages 5906–5910, 2019.
- [10] Jaehyeon Kim, Sungwon Kim, Jungil Kong, and Sungroh Yoon. Glow-TTS: A Generative Flow for Text-to-Speech via Monotonic Alignment Search. *arXiv:2005.11129*, May 2020.
- [11] Z. Kons, S. Shechtman, A. Sorin, R. Hoory, C. Rabinovitz, and E. Da Silva Morais. Neural tts voice conversion. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 290–296, 2018.
- [12] Zvi Kons, Slava Shechtman, Alex Sorin, Carmel Rabinovitz, and Ron Hoory. High quality, lightweight and adaptable TTS using LPCNet. *Conference of the International Speech Communication Association*, 2019.
- [13] Juheon Lee, Hyeong-Seok Choi, Chang-Bin Jeon, Junghyun Koo, and Kyogu Lee. Adversarially trained end-to-end korean singing voice synthesis system. In *INTERSPEECH*, pages 2588–2592, 2019.
- [14] Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, and Ming Liu. Neural speech synthesis with transformer network. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:6706–6713, 07 2019.
- [15] Shujie Liu, Yanqing Liu, Sheng Zhao, and Ming Liu. Neural speech synthesis with transformer network. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:6706–6713, 2019.
- [16] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *ICLR*, 2018.
- [17] Catherine Olsson, Surya Bhupatiraju, Tom Brown, Augustus Odena, and Ian Goodfellow. Skill rating for generative models. *arXiv:1808.04888*, 2018. [arXiv:1808.04888](https://arxiv.org/abs/1808.04888).
- [18] Wei Ping, Kainan Peng, Andrew Gibiansky, Sercan Ömer Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller. Deep voice 3: Scaling text-to-speech with convolutional sequence learning. In *ICLR*, 2018.
- [19] R. Prenger, R. Valle, and B. Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP*, pages 3617–3621, 2019.
- [20] Yi Ren, Chenxu Hu, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. FastSpeech 2: Fast and high-quality end-to-end text to speech. *arXiv:2006.04555*, 2020. [arXiv:2006.04555](https://arxiv.org/abs/2006.04555).
- [21] Yi Ren, Yangjun Ruan, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. FastSpeech: Fast, robust and controllable text to speech. In *NeurIPS 2019*, 2019.

- [22] Christian Schäfer. *ForwardTacotron*, 2020. URL: <https://github.com/as-ideas/ForwardTacotron>.
- [23] Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, and et al. Natural TTS synthesis by conditioning wavenet on mel spectrogram predictions. *ICASSP*, 2018.
- [24] Rafael Valle, Jason Li, Ryan Prenger, and Bryan Catanzaro. Mellotron: Multispeaker expressive voice synthesis by conditioning on rhythm, pitch and global style tokens. *arXiv:1910.11997*, 2019.
- [25] Rafael Valle, Kevin Shih, Ryan Prenger, and Bryan Catanzaro. Flowtron: an autoregressive flow-based generative network for text-to-speech synthesis. *arXiv:2005.05957*, 2020. [arXiv:2005.05957](https://arxiv.org/abs/2005.05957).
- [26] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv:1609.03499*, 2016.
- [27] Aäron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis Carlos Cobo Rus, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alexander Graves, Helen King, Thomas Walters, Dan Belov, and Demis Hassabis. Parallel wavenet: Fast high-fidelity speech synthesis. *arXiv:1711.10433*, 2017.
- [28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008. 2017.
- [29] Fabian Wauthier, Michael Jordan, and Nebojsa Jojic. Efficient ranking from pairwise comparisons. In *ICML*, pages 109–117, 2013.
- [30] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *ICLR*, 2020.
- [31] Zhen Zeng, Jianzong Wang, Ning Cheng, Tian Xia, and Jing Xiao. AlignTTS: Efficient Feed-Forward Text-to-Speech System without Explicit Alignment. *arXiv:2003.01950*, 2020.