FastDCTTS: Efficient Deep Convolutional Text-to-Speech

Minsu Kang, Jihyun Lee, Simin Kim, Injung Kim

Deep-learning Lab.

Handong Global University

HANDONG GLOBAL UNIVERSITY

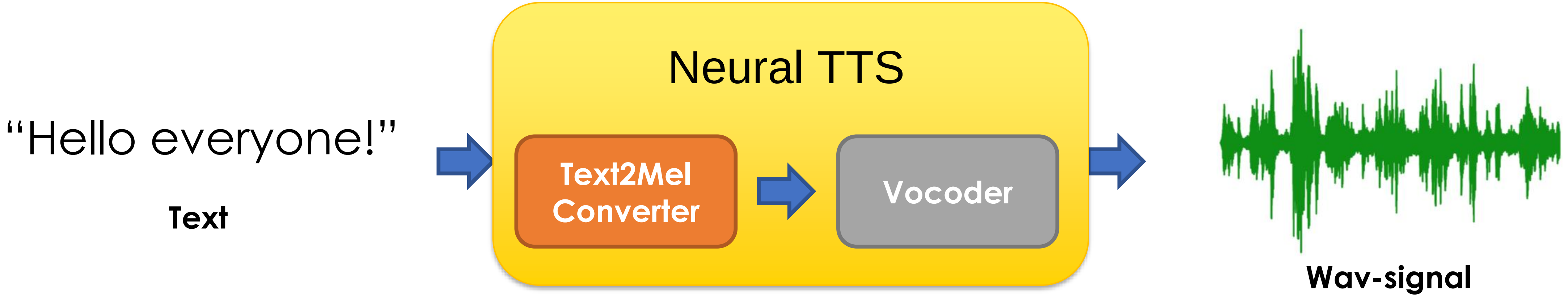
2021 TORONTO Canada June 6-11, 2021 Metro Toronto Convention Centre

Contents

- Introduction
- Quantitative fidelity metric for optimization (EMCD)
- FastDCTTS
    - Computational optimization
    - Fidelity improvement
- Experiments
- Conclusion

Neural TTS for Limited Environment

- End-to-end neural TTS



- Neural TTS **for limited environments w/o GPU**
  - Conventional encoder-decoder models ➔ **slow**
    - Tacotron[Wang17], Tacotron2[Shen17], DCTTS[Tachibana18], Transformer-TTS [Li18]
  - Non-autoregressive models: fast, but **rely on parallel computation** ➔ requires GPU
    - FastSpeech[Ren19], FastSpeech2[Ren20], AlignTTS[Zeng20]
  - TTS for limited environments **w/o GPU** requires **computational optimization**

Contributions

- **Highly-optimized neural TTS, FastDCTTS,** that generates speech signals **in real-time on a single CPU thread**
  - Multiple techniques to improve synthesis speed and fidelity
  - Compared with DCTTS, **1.76% computation, 2.75% parameters, and 7.4x faster**

- **Group highway activation**, a novel lightweight version of the Highway network

- **Elastic Mel-cepstral Distortion(EMCD),** a novel objective metric to evaluate the quality of a mel-spectrogram focusing on skipping and repeating error.

- **Quantitative and qualitative evaluation** of multiple acceleration and fidelity improvement techniques using EMCD

Baseline model: DCTTS

- **Deep convolutional Text-to-Speech [Tachibana18]**
  - Text encoder
    : Input text ➔ two character embedding sequences, K(key) and V(value)
  - Audio encoder
    : Previously generated mel-spectrogram ➔ audio embedding sequence, Q (query)
  - Attention module
    : K, V, Q ➔ alignment between text and mel (A)
  - Audio decoder
    : Generates mel-spectrogram from A, V and Q

  ➔ Composed of **convolution operation**

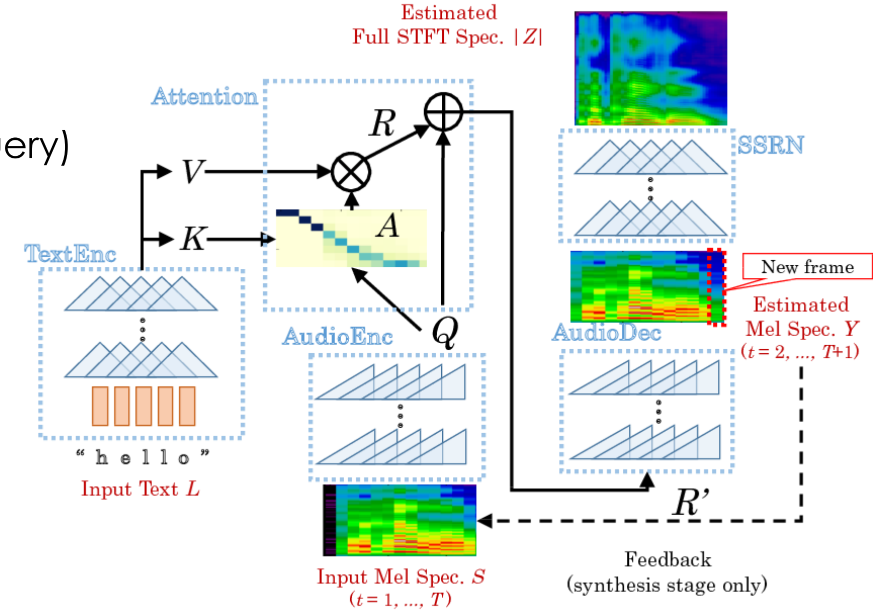


Figure: diagram of DCTTS [Tachibana18]

- **Why DCTTS?**
  (1) Many acceleration techniques available for CNNs
  ex) Depthwise separable conv.[Howard17], network pruning [Han15], etc.
  (2) Fast training and evaluation

Optimization Techniques

- **Computational optimization**
  - Depthwise separable convolution
  - **Group highway activation (proposed)**
  - Network size reduction
  - Network pruning with **weight normalization trick**
- **Fidelity improvement**
  - Positional encoding
  - Scheduled sampling
- **Quantitative evaluation** of output quality **using EMCD** during optimization

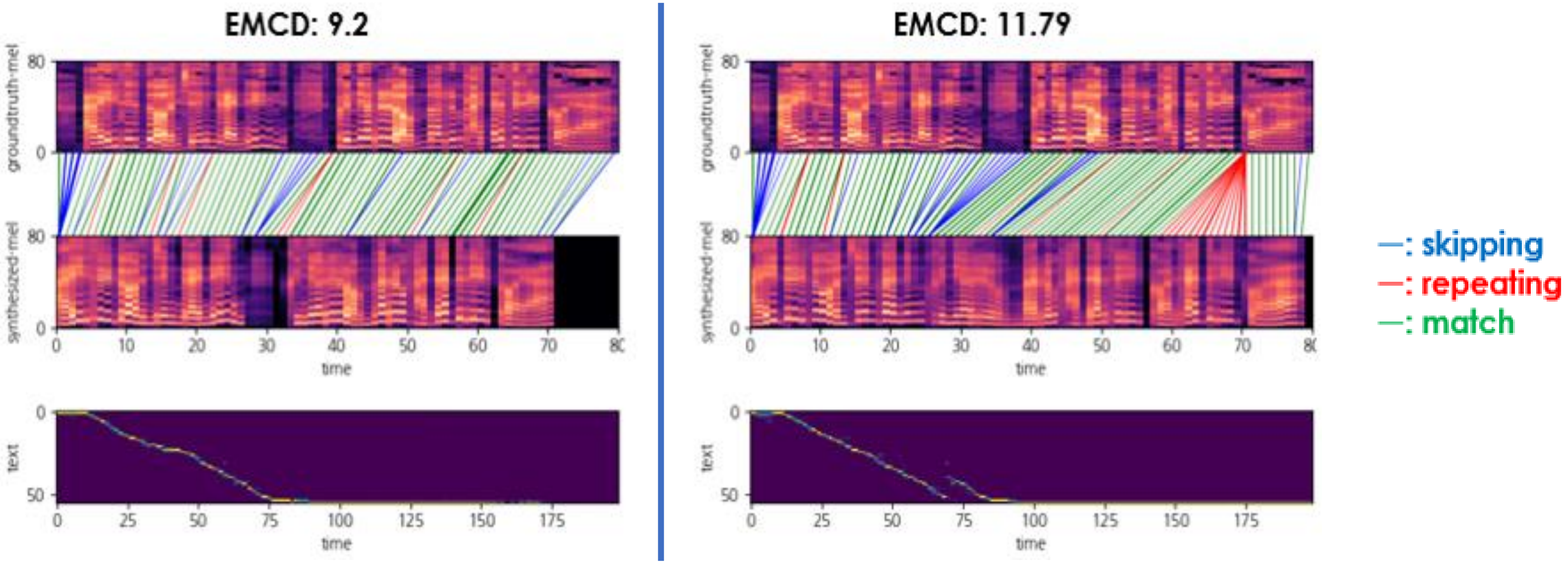
Elastic Mel-Cepstral Distortion (EMCD)

- **EMCD**: A novel quantitative metric to measure speech quality **focusing on skipping and repeating**
  - Measures MCD computed from the best alignment found by elastic matching.

$$\boldsymbol{D(i,j)} = \boldsymbol{w_m} \times \boldsymbol{MCD}\left(\boldsymbol{x_i}, \boldsymbol{y_j}\right) + \mathbf{min}\{\boldsymbol{D(i,j-1)}, \boldsymbol{D(i-1,j)}, \boldsymbol{D(i-1,j-1)}\}$$

  - Penalty weights $\boldsymbol{w_m} \in \{w_{hor} = 1, w_{ver} = 1, w_{diag} = \sqrt{2}\}$

$MCD(i,j) = \sqrt{2\sum_{d=1}^{D}(x_d[i] - x_d[j])^2}$ [Kubicheck 93]
$, where\ i = \{1, \ldots, T_{syn}\}, j = \{1, \ldots, T_{gt}\}$
T_{syn}, T_{gt}: length of syn. and GT mel
$m = argmin\{D(i,j-1), D(i-1,j), D(i-1,j-1)\}$
$w = [w_{hor}, w_{ver}, w_{diag}]$



(left) good quality speech, (right) speech with repeating

Compared to MCD-DTW[Battenberg20], EMCD assigns different penalty weights to hor, ver, and diag transitions to measure the difference caused by skipping and repeating more effectively.

Computational optimization techniques

- Depthwise separable convolution [Howard17]
  - In image processing, $O(D_K^2 M N D_F^2) \Rightarrow O(D_K^2 M D_F^2) + O(M D_F^2 D_F^2)$
    3D conv (WxHxC) ➔ 2D DW conv + 1D pointwise conv
  - **In speech processing,** $O(D_K M N D_F) \Rightarrow O(D_K M D_F) + O(M D_F D_F)$
    2D conv (time x channel) ➔ 1D DW conv + 1D pointwise conv
    ➔ Less effective in speech synthesis
  - Result
    - **In theory**, requires **only 36.3%** of operations (275B ➔ 100B)
    - **In experiments, increased synthesis time by 2.68x (6.85 sec ➔ 18.16 sec)**
    ➔ Not used in the following experiments

D_K: kernel size
M: # of input channels
N: # of output channels
D_F: feature map size
$*$: convolution operation

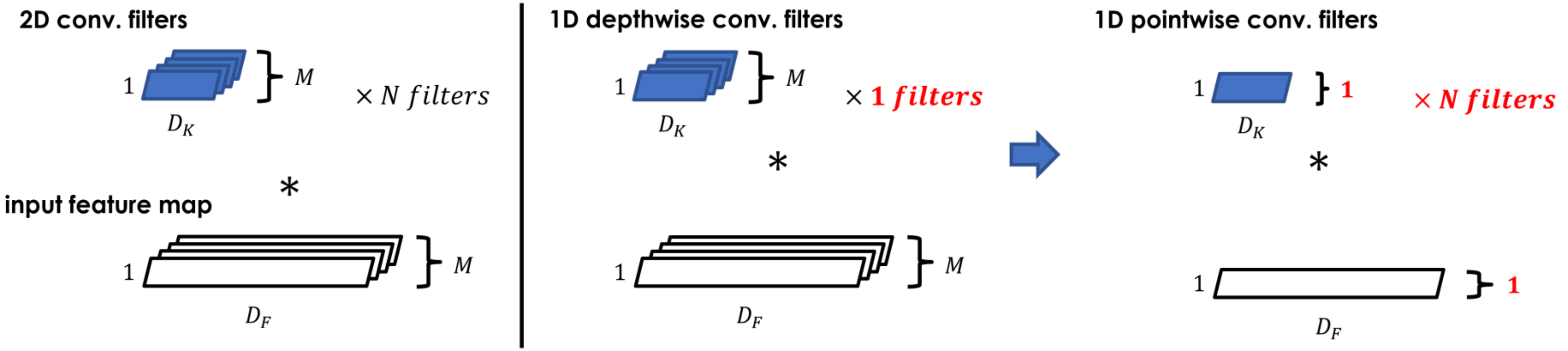


Figure: 2D convolution (left) vs. 1D depthwise + 1D pointwise convolution (right)

Computational optimization techniques

- $T(x, W_T)$, $C(x, W_C)$: transformation and carry gate
- x, y: input and output feature map
- W_T, W_C, W_H: parameters of $T(\cdot)$, $C(\cdot)$, $H(\cdot)$
- g: group size

- Highway activation
  - Highway network [Srivastava15]
    - $y = \boldsymbol{T}(\boldsymbol{x}, \boldsymbol{W_T})H(x, W_H) + \mathbf{C}(\mathbf{x}, \boldsymbol{W_C})x$ (usually, $C(x, W_C) = 1 - T(x, W_T)$)

    ➔ **increases computations by 2 or 3 times**

- **Group highway activation: a simplified form of Highway activation**
  - **A group of elements share the same gate value**

$$y = T_G(x, W_{T_G})H(x, W_H) + (1 - T_G(x, W_{T_G}))x$$

- Computation: $\left(\mathbf{1} + \frac{\mathbf{1}}{\mathbf{g}}\right)/\mathbf{2}$ **of ordinary highway activation**

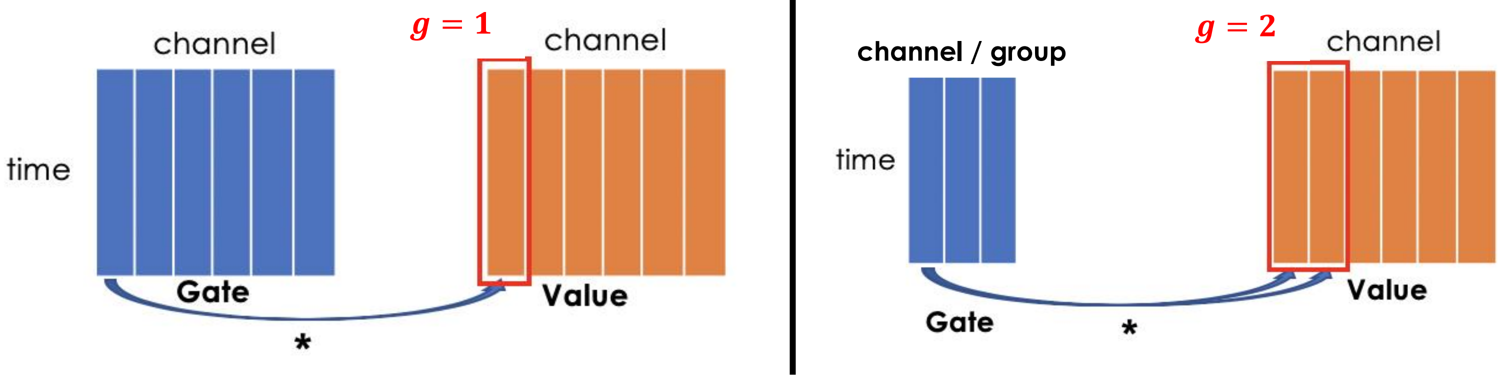


Figure: elementwise representation of gating mechanism in Highway(left) vs. Group Highway(right, g =2)

Computational Optimization Techniques

- Network size reduction
  - **Reduce the number of layers and channels** measuring output quality by EMCD

	Attempted values			
Layers	12	9	6	
Channels	256	192	128	64

Table: # of layers and channels reduced

- Network pruning for CNN [Li16]
  - Remove 10% of less important filters (by L1-norm)
  - Modified for group highway activation
  - **Weight normalization trick**
    1. **Train model applying weight normalization**
    2. **Pruning (reduces model capacity)**
    3. **Deactivate weight normalization and fine-tune reduced model**

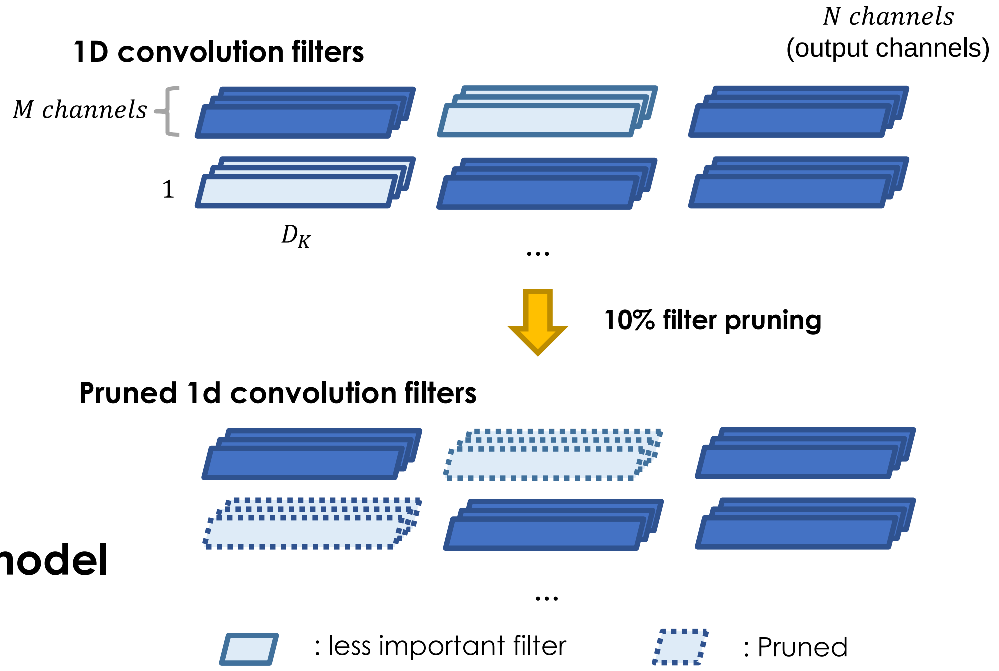


Figure: pruning filters for CNN [Li16]

Fidelity Improvement Techniques

- Positional encoding for TTS [Li 18]

$$x_i' = x_i + \alpha PE(pos, i) \quad , where\ PE(pos,i) = \begin{cases} \sin(\frac{pos}{base^{2k/dim}}) \ for\ i = 2k \\ \cos(\frac{pos}{base^{2k/dim}}) \ for\ i = 2k+1 \end{cases}$$

α: trainable weight

  - **<u>To improve attention stability</u>** by helping to learn the temporal relation

- Scheduled sampling [Bengio15]
  - Learns mainly from ground truth ➔ increase portion of generated mel-spec. as learning progresses

Experiments

- Settings
  - Dataset
    - **LJ-speech**[Ito17]
      - English, a female single speaker, about 24 hours
    - **Korean Single Speaker (KSS)** [Park19]
      - Korean, a female single speaker, 12+ hours

    ➔ 70% for training, **10% for validation, and 20% for test**
      - **A large portion of validation and test set for reliable evaluation.**
      - Relatively small portion for training

  - Experimental setting
    - Training: NVIDIA GTX-1080 GPU
    - Synthesis: **single thread** of Intel Xeon E3-1240 v3 CPU (3.40 GHz), **batch_size = 1**

Experiments

- **Group highway activation**
  - Amount of computation
    ➔ In theory, **reduced to 75%** of highway convolution
  - Synthesis time and speech quality
    ➔ Residual DCTTS: ½ of synthesis time of baseline, but increased EMCD
    ➔ **Group Highway DCTTS: reduced syn-time by 7% of baseline, decreased EMCD**

Model	Synthesis time		EMCD (the lower, the better) LJ	EMCD (the lower, the better) KSS
Highway (GPU)	1.28 sec (1.00 x)	-	9.45	10.36
Highway (CPU)	6.85 sec (5.35 x)	-		
Residual (CPU)	**3.85 sec (3.00 x)**	**43.8% reduc.**	**12.93**	**13.59**
Group Highway (CPU)	**6.37 sec (4.98 x)**	**7.0% reduc.**	**9.10**	**9.29**

Table: comparison of the baseline model(Base(HC)), residual DCTTS(ResDCTTS), and Group highway DCTTS(GH DCTTS)

Experiments

- **Network size reduction**
  - Reducing # of channels and layers increases speed and degrades output quality.
    - GH_C128 exhibited a good trade-off
    - **GH_L6_C64 was the fastest, but poor output quality ➔ improve by fidelity improvement techniques**

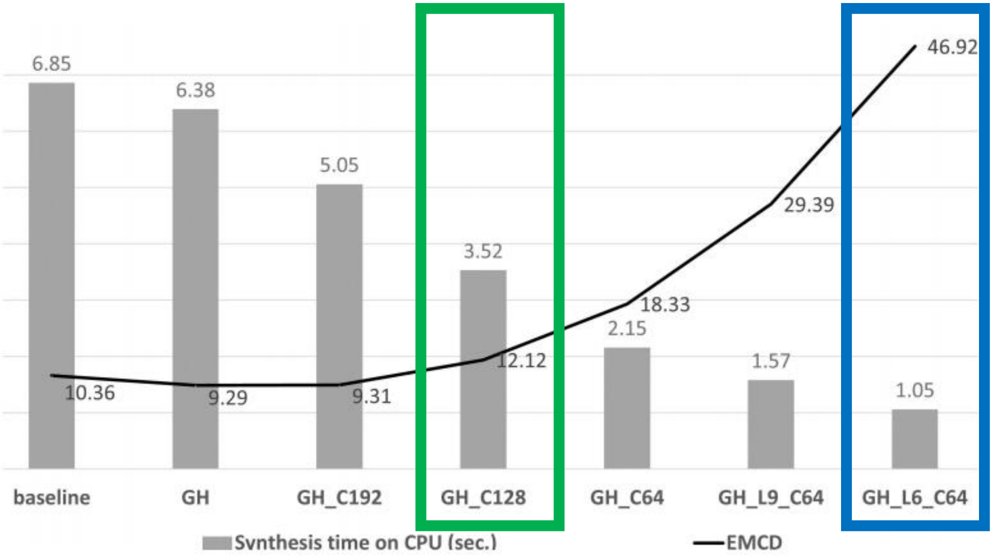


Figure: the effect of network size reduction on synthesis time and speech quality.
(GH:Group highway activation, La: # of layers and Cb: # of channels)

	Baseline	GH	GH_C192	GH_C128	GH_C64	GH_L9_C64	GH_L6_C64
# of layers (TextEnc, AudioEnc, AudioDec)	14, 13, 11					14, 9, 9	14, 6, 6
# of channels	256		192	128	64		

Table: comparison of parameters between various network parameters of models

Experiments

- **Network pruning with weight normalization trick**
  - Removed **10% of convolution filters** by network pruning [Li16]
    - Synthesis time was **reduced by 18.09%** (1.05sec ➔ 0.86 sec)
    - Often produced unrecognizable speech
  - Weight normalization trick
    - Train model **applying weight-norm**
    - **Pruning**
    - **Deactivate weight-norm to compensate the reduced capacity** and fine-tune the reduced model

  ➔ **Significantly improves the output quality of small capacity models**

	model	Weight normalization		Synthesis time
		on → on	on → off	
LJ Speech	GH_L6_C64	30.59	**15.26**	1.05
	GH_L6_C64 (10% pruned)	Unrecognizable	**16.24**	**0.86**
KSS	GH_L6_C64	46.92	**9.75**	1.05
	GH_L6_C64 (10% pruned)	Unrecognizable	**10.69**	**0.86**

Table: the effect of the pruning and weight normalization trick

Experiments

- **Positional encoding and scheduled sampling**
  - Positional encoding improves speech quality
    - ➔ **Decreases EMCD values to 9.55 (LJSpeech) and 9.39 (KSS)**
  - Scheduled sampling did not lead to any improvement

Dataset	Improvement in EMCD by positional encoding	Scheduled sampling
LJSpeech	16.24 ➔ **9.55** **(41.19% reduction)**	**No improvement**
KSS	10.69 ➔ **9.39** **(12.16% reduction)**	

Table: the effect of the positional encoding and scheduled sampling

Experiments

- **FastDCTTS**
  - Synthesis time on a single CPU thread: **0.92 sec**
    ➔ Faster than $baseline_{CPU}$ (**6.85 sec.**) and $baseline_{GPU}$ (**1.28 sec.**)
  - Speech quality is comparable to the baseline model, DCTTS

	Baseline model	**FastDCTTS**
# of computations	275,098,419,200	**4,835,728,000 (1.6% of baseline)**
# of params	23,896,094	**657,728 (2.75% of baseline)**
$synthesis\ time_{cpu}$	6.85 sec.	**0.92 sec. (7.45x faster)**
EMCD (LJ, KSS)	9.45, 10.36	**9.55, 9.39**
MOS (LJ, KSS)	2.42, 2.62	**2.45, 2.74**
Speech samples KSS script: "한국은 천연자원이 풍부하지 않습니다." ("Korea is not rich in natural resources.") LJ script: "The most trifling acts were magnified into offenses."	**KSS-90%** **KSS-70%** **LJ-90%** **LJ-70%**	**KSS-90%** **KSS-70%** **LJ-90%** **LJ-70%**

Table: comparison between baseline model and FastDCTTS

Conclusion

- A novel lightweight neural TTS, FastDCTTS that synthesizes speech in real-time without CPU.
  - Based on DCTTS, apply multiple acceleration and fidelity improvement techniques.
  - 1.76% computation, 2.75% parameters, and 7.4x faster

- A few novel techniques
  - A novel objective metric EMCD
  - Group highway activation
  - Weight normalization trick

Thank you for attention!