FastDCTTS: Efficient Deep Convolutional Text-to-Speech

Minsu Kang, Jihyun Lee, Simin Kim, Injung Kim

Deep-learning Lab.
Handong Global University
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**

"Hello everyone!"

![Neural TTS Diagram](image)
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 $\rightarrow$ two character embedding sequences, $K$(key) and $V$(value)
  - **Audio encoder**
    : Previously generated mel-spectrogram $\rightarrow$ audio embedding sequence, $Q$ (query)
  - **Attention module**
    : $K$, $V$, $Q$ $\rightarrow$ alignment between text and mel $(A)$
  - **Audio decoder**
    : Generates mel-spectrogram from $A$, $V$ and $Q$
    $\rightarrow$ Composed of **convolution operation**

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

![Diagram of DCTTS](image)
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.

\[
D(i, j) = w_m \times MCD(x_i, y_j) + \min\{D(i, j - 1), D(i - 1, j), D(i - 1, j - 1)\}
\]

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

\[
MCD(i, j) = \sqrt{2 \sum_{d=1}^{D}(x_d[i] - x_d[j])^2} \quad [\text{Kubicheck 93}]
\]

\(i = \{1, ..., T_{syn}\}, j = \{1, ..., T_{gt}\}\)

\(T_{syn}, T_{gt}\): length of syn. and GT mel

\(m = \arg\min\{D(i, j - 1), D(i - 1, j), D(i - 1, j - 1)\}\)

\(w = [w_{hor}, w_{ver}, w_{diag}]\)

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.

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

- **Highway activation**
  - Highway network [Srivastava15]
    - \[ y = T(x, W_T)H(x, W_H) + C(x, W_C)x \] (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: \( \left(1 + \frac{1}{g}\right)/2 \) of ordinary highway activation

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

- \( T(x, W_T), C(x, W_C) \): 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
Computational Optimization Techniques

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

<table>
<thead>
<tr>
<th>Attempted values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layers</td>
</tr>
<tr>
<td>Channels</td>
</tr>
</tbody>
</table>

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]]

<table>
<thead>
<tr>
<th>Attempted values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layers</td>
</tr>
<tr>
<td>Channels</td>
</tr>
</tbody>
</table>

Table: # of layers and channels reduced
Fidelity Improvement Techniques

- Positional encoding for TTS [Li 18]

\[ \begin{align*}
    x_i' &= x_i + \alpha \text{PE}(pos, i), \\
    \text{where PE}(pos, i) &= \begin{cases} 
        \sin\left(\frac{pos}{\text{base}^2k/dim}\right) & \text{for } i = 2k \\
        \cos\left(\frac{pos}{\text{base}^2k/dim}\right) & \text{for } i = 2k + 1
    \end{cases}
\end{align*} \]

\( \alpha \): trainable weight

- **To improve attention stability** 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

<table>
<thead>
<tr>
<th>Model</th>
<th>Synthesis time</th>
<th>EMCD (the lower, the better)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LJ</td>
</tr>
<tr>
<td>Highway (GPU)</td>
<td>1.28 sec (1.00 x)</td>
<td>-</td>
</tr>
<tr>
<td>Highway (CPU)</td>
<td>6.85 sec (5.35 x)</td>
<td>-</td>
</tr>
<tr>
<td>Residual (CPU)</td>
<td>3.85 sec (3.00 x)</td>
<td>43.8% reduc.</td>
</tr>
<tr>
<td>Group Highway (CPU)</td>
<td>6.37 sec (4.98 x)</td>
<td>7.0% reduc.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>GH</th>
<th>GH_C192</th>
<th>GH_C128</th>
<th>GH_C64</th>
<th>GH_L9_C64</th>
<th>GH_L6_C64</th>
</tr>
</thead>
<tbody>
<tr>
<td># of layers</td>
<td></td>
<td>14, 13, 11</td>
<td></td>
<td></td>
<td></td>
<td>14, 9, 9</td>
<td>14, 6, 6</td>
</tr>
<tr>
<td>(TextEnc, AudioEnc, AudioDec)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of channels</td>
<td>256</td>
<td>192</td>
<td>128</td>
<td></td>
<td></td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>

Figure: the effect of network size reduction on synthesis time and speech quality.

(GH: Group highway activation, La: # of layers and Cb: # of channels)

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**
  - **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**

<table>
<thead>
<tr>
<th>Model</th>
<th>Weight normalization</th>
<th>Synthesis time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GH_L6_C64</td>
<td>30.59</td>
<td>15.26</td>
</tr>
<tr>
<td>GH_L6_C64 (10% pruned)</td>
<td>Unrecognizable</td>
<td>16.24</td>
</tr>
<tr>
<td>GH_L6_C64</td>
<td>46.92</td>
<td>9.75</td>
</tr>
<tr>
<td>GH_L6_C64 (10% pruned)</td>
<td>Unrecognizable</td>
<td>10.69</td>
</tr>
</tbody>
</table>

*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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Improvement in EMCD by positional encoding</th>
<th>Scheduled sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>LJSpeech</td>
<td>16.24 ➔ 9.55 (41.19% reduction)</td>
<td>No improvement</td>
</tr>
<tr>
<td>KSS</td>
<td>10.69 ➔ 9.39 (12.16% reduction)</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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!