멀티스피커 모델 정리

~2019

논문 선정 기준

Multi-Speaker 관련

대기업 발표

유수 학회 선정

탁월한 오디오 예시

Deep Voice 2 (17.05, NIPS)

Multi-Speaker Strategy

Low dimensional speaker embedding

- Speaker-dependent parameters stored in a very low-dimensional vector
- **Near-complete weight sharing between speakers***

Using Speaker Embedding

32d speaker embedding initialized with *uniform[-0.1, 0.1]*

- Use as the initial GRU hidden states
- Concatenate to the input at every timestep
- Multiply layer activations elementwise
- Use as bias of the tanh in the content-based attention

***speaker classification/verification**과 같은 별도의 **side task** 수행을 하지 않는다.

Multi-Speaker Deep Voice 2

32d speaker embedding fc-layered and injected to

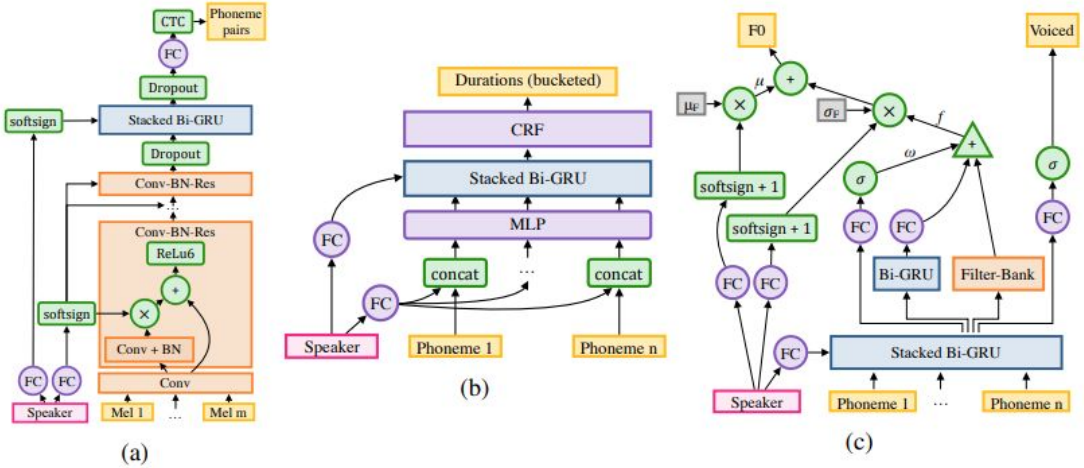


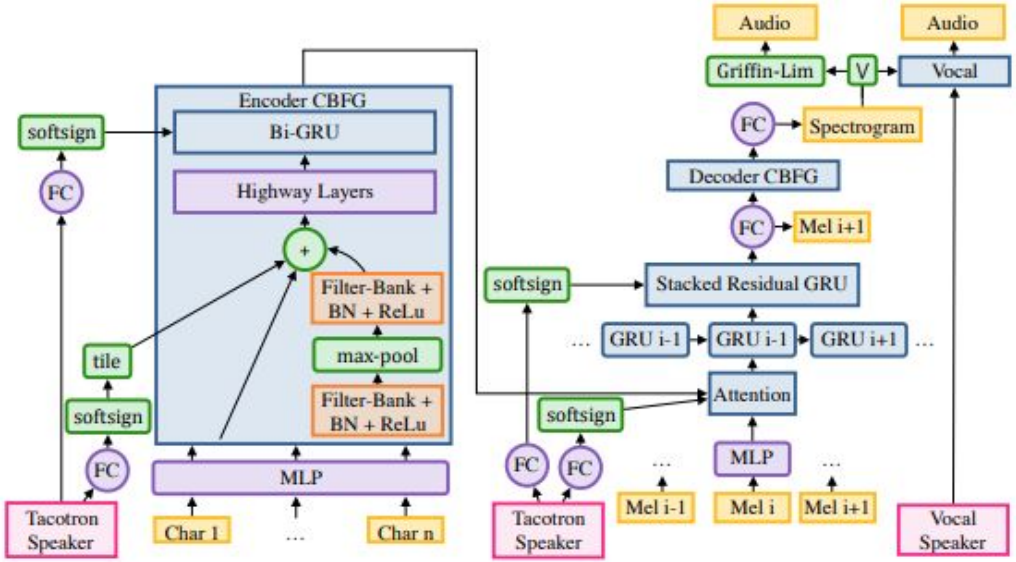
Figure 2: Architecture for the multi-speaker (a) segmentation, (b) duration, and (c) frequency model.

$$h^{(l)} = \text{relu}\left(h^{(l-1)} + \text{BN}\left(W * h^{(l-1)}\right) \cdot g_s\right)$$

$$F_0 = \mu_{F_0} \cdot \left(1 + \text{softsign}\left({V_\mu}^T g_f\right)\right) + \sigma_{F_0} \cdot \left(1 + \text{softsign}\left({V_\sigma}^T g_f\right)\right) \cdot f$$

Multi-Speaker Tacotron 1

*타코트론에는 **encoder time-step** 마다 **concat**을 하지 않는다.



Data

VCTK (108 speakers, 400 utterances per speaker)

Audiobooks (477 speakers, 30 minutes of audio per speaker)

Result

Dataset	Multi-Speaker Model	Samp. Freq.	MOS	Acc.
VCTK	Deep Voice 2 (20-layer WaveNet)	16 KHz	2.87±0.13	99.9%
VCTK	Deep Voice 2 (40-layer WaveNet)	16 KHz	3.21±0.13	100 %
VCTK	Deep Voice 2 (60-layer WaveNet)	16 KHz	3.42±0.12	99.7%
VCTK	Deep Voice 2 (80-layer WaveNet)	16 KHz	3.53±0.12	99.9%
VCTK	Tacotron (Griffin-Lim)	24 KHz	1.68±0.12	99.4%
VCTK	Tacotron (20-layer WaveNet)	24 KHz	2.51±0.13	60.9%
VCTK	Ground Truth Data	48 KHz	4.65±0.06	99.7%
Audiobooks	Deep Voice 2 (80-layer WaveNet)	16 KHz	2.97±0.17	97.4%
Audiobooks	Tacotron (Griffin-Lim)	24 KHz	1.73±0.22	93.9%
Audiobooks	Tacotron (20-layer WaveNet)	24 KHz	2.11±0.20	66.5%
Audiobooks	Ground Truth Data	44.1 KHz	4.63±0.04	98.8%

Problems

Can synthesize only the voices seen during training

Not clear if speaker-dependent info is separated from the general speech info

> “Despite training with only a generative loss,
> discriminative properties (e.g. gender and accent) are observed in the speaker embedding space”

Not clear if the discriminative properties effectively overlap with speaker id

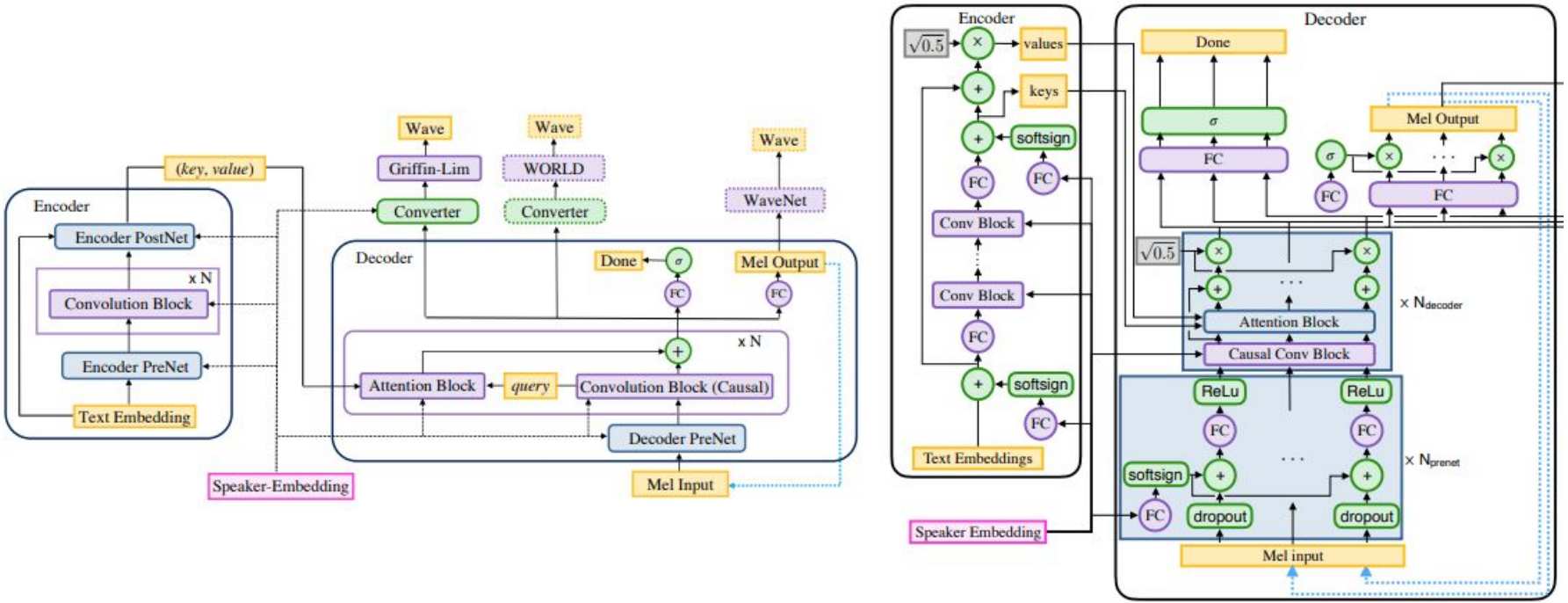
Deep Voice 3 (17.10, ICLR)

Multi-Speaker Strategy

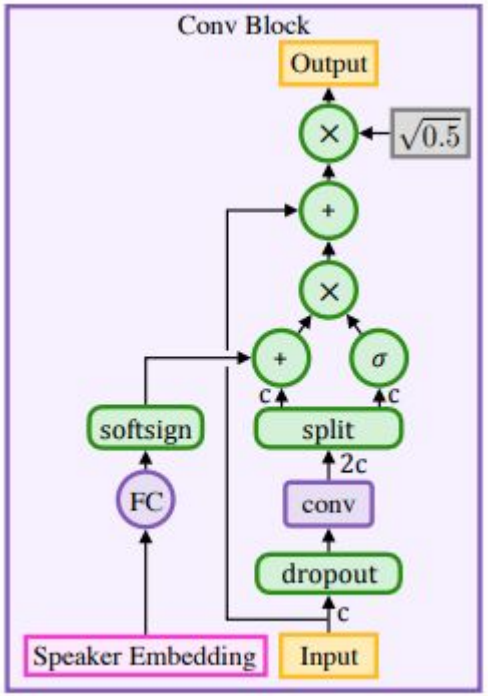
Mostly identical to Deep Voice 2

Multi-Speaker Deep Voice 3

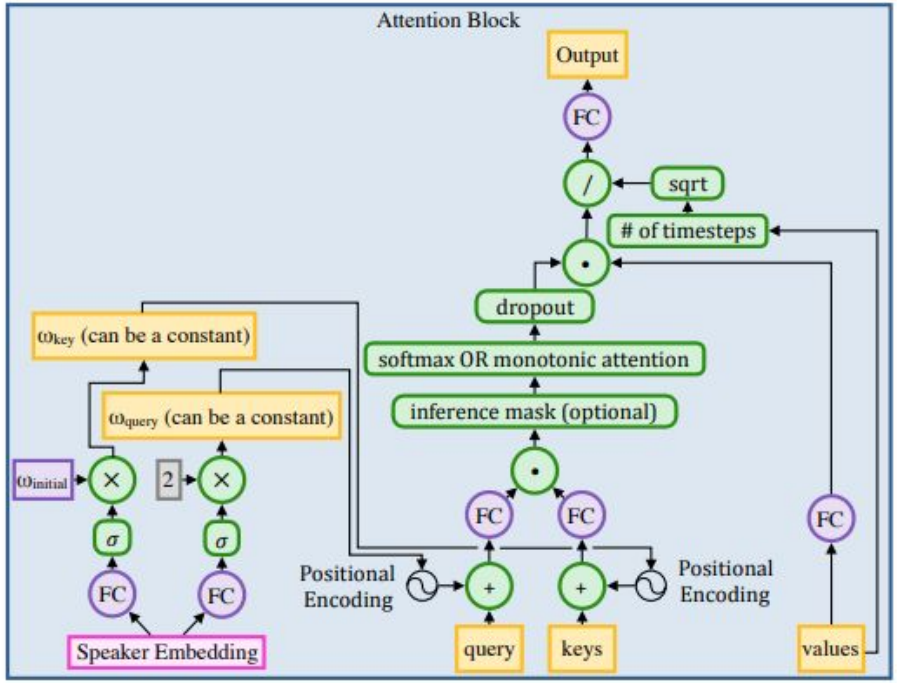
특이점: speaker embedding 16D (VCTK) → 512D (LibriSpeech, 20x)



Conv Block


Conv Block
Output
×
√0.5
+
×
+
σ
c
c
softsign
split
2c
FC
conv
dropout
c
Speaker Embedding
Input

Attention Block



Data

VCTK (108 speakers, ~44 hours (.4 hr/speaker), 48k)

LibriSpeech (2484 speakers, ~820 hours (.33 hr/speaker), 16k)

Result

Model	MOS (VCTK)	MOS (LibriSpeech)
Deep Voice 3 (Griffin-Lim)	3.01 ± 0.29	2.37 ± 0.24
Deep Voice 3 (WORLD)	3.44 ± 0.32	2.89 ± 0.38
Deep Voice 2 (WaveNet)	3.69 ± 0.23	-
Tacotron (Griffin-Lim)	2.07 ± 0.31	-
Ground truth	4.69 ± 0.04	4.51 ± 0.18

Problems

스피커 임베딩을 512까지 키우면서 원래의 철학(low-dimensional speaker embedding)과 벗어났다.

Can synthesize only the voices seen during training

Not clear if speaker-dependent info is separated from the general speech info

VoiceLoop: Voice Fitting and Synthesis via a Phonological Loop (17.07)

Loop

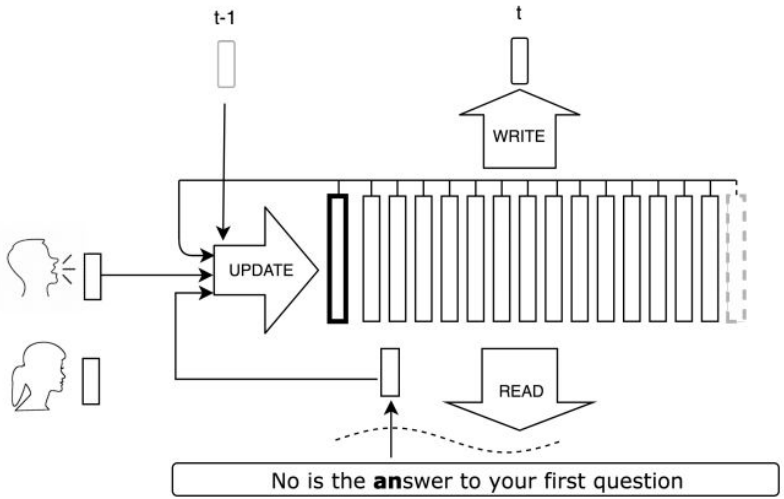


Figure 1: An overview of the VoiceLoop architecture. The reader combines the encoding of the sentence's phonemes using the attention weights to create the current context. A new representation is created by a shallow network that receives the context, the speaker ID, the previous output, and the buffer. The new representation is inserted into the buffer and the earliest vector in the buffer is discarded. The output is obtained by another shallow network that receives the buffer and the speaker as inputs. Once trained, fitting a new voice is done by freezing the network, except for the speaker embedding.

Injection Spot

Initialization of mu and sigma for GMM attention

Update the buffer (query?) for GMM attention

Concatenated to every decoder output

Speaker-related Nets

$LUT_s \in \mathbb{R}^{d_s \times N}$	embedding of the speakers
$F_u : d_s \rightarrow d_p$	projection of the speaker for update
$F_o : d_s \rightarrow d_o$	projection of the speaker for output

Encoder

```
class Encoder(nn.Module):
    def __init__(self, opt):
        super(Encoder, self).__init__()
        self.hidden_size = opt.hidden_size
        self.vocabulary_size = opt.vocabulary_size
        self.nspk = opt.nspk
        self.lut_p = nn.Embedding(self.vocabulary_size,
                                  self.hidden_size,
                                  max_norm=1.0)
        self.lut_s = nn.Embedding(self.nspk,
                                  self.hidden_size,
                                  max_norm=1.0)

    def forward(self, input, speakers):
        if isinstance(input, tuple):
            lengths = input[1].data.view(-1).tolist()
            outputs = pack(self.lut_p(input[0]), lengths)
        else:
            outputs = self.lut_p(input)
        if isinstance(input, tuple):
            outputs = unpack(outputs)[0]

        ident = self.lut_s(speakers)
        if ident.dim() == 3:
            ident = ident.squeeze(1)

        return outputs, ident
```

Decoder

```
def init_buffer(self, ident, start=True):
    mem_feat_size = self.hidden_size + self.output_size
    batch_size = ident.size(0)

    if start:
        self.mu_t = Variable(ident.data.new(batch_size, self.K).zero_())
        self.S_t = Variable(ident.data.new(batch_size,
                                           mem_feat_size,
                                           self.mem_size).zero_())

        # initialize with identity
        self.S_t[:, :self.hidden_size, :] = ident.unsqueeze(2) \
                                                 .expand(ident.size(0),
                                                         ident.size(1),
                                                         self.mem_size)
    else:
        self.mu_t = self.mu_t.detach()
        self.S_t = self.S_t.detach()
```

```
def update_buffer(self, S_tm1, c_t, o_tm1, ident):
    # concat previous output & context
    idt = torch.tanh(self.F_u(ident))
    o_tm1 = o_tm1.squeeze(0)
    z_t = torch.cat([c_t + idt, o_tm1/30], 1)
    z_t = z_t.unsqueeze(2)
    Sp = torch.cat([z_t, S_tm1[:, :, :-1]], 2)

    # update S
    u = self.N_u(Sp.view(Sp.size(0), -1))
    u[:, :idt.size(1)] = u[:, :idt.size(1)] + idt
    u = u.unsqueeze(2)
    S = torch.cat([u, S_tm1[:, :, :-1]], 2)

    return S
```

```
def forward(self, x, ident, context, start=True):
    out, attns = [], []
    o_t = x[0]
    self.init_buffer(ident, start)

    for o_tm1 in torch.split(x, 1):
        if not self.training:
            o_tm1 = o_t.unsqueeze(0)

        # predict weighted context based on S
        c_t, mu_t, alpha_t = self.attn(self.S_t,
                                       context.transpose(0, 1),
                                       self.mu_t)

        # advance mu and update buffer
        self.S_t = self.update_buffer(self.S_t, c_t, o_tm1, ident)
        self.mu_t = mu_t

        # predict next time step based on buffer content
        ot_out = self.N_o(self.S_t.view(self.S_t.size(0), -1))
        sp_out = self.F_o(ident)
        o_t = self.output(ot_out + sp_out)

        out += [o_t]
        attns += [alpha_t.squeeze()]

    out_seq = torch.stack(out)
    attns_seq = torch.stack(attns)

    return out_seq, attns_seq
```

Loop

```
def forward(self, src, tgt, start=True):
    x = self.init_input(tgt, start)

    context, ident = self.encoder(src[0], src[1])
    out, attn = self.decoder(x, ident, context, start)

    return out, attn
```

Learn a New Speaker

The weights of all networks and projections (N_a, N_u, N_o, LUT_p, F_u, F_o) fixed

Vector z is learned to form the embedding of the new speaker.

Audio

https://ytaigman.github.io/loop/

Generalized End-to-End Loss for Speaker Verification (17.10, ICASSP)

Notes

Multi-speaker verification (NOT classification):

Emphasizes examples that are difficult to verify

Does not require an initial stage of example selection

d-vector

3-layer LSTM (TI: 768, TD: 128) with projection (TI: 256, TD: 64)

The embedding vector (d-vector) size is the same as the LSTM projection size.

```
class SpeechEmbedder(nn.Module):

    def __init__(self):
        super(SpeechEmbedder, self).__init__()
        self.LSTM_stack = nn.LSTM(hp.data.nmels, hp.model.hidden, num_layers=hp.model.num_layer, batch_first=True)
        for name, param in self.LSTM_stack.named_parameters():
          if 'bias' in name:
             nn.init.constant_(param, 0.0)
          elif 'weight' in name:
             nn.init.xavier_normal_(param)
        self.projection = nn.Linear(hp.model.hidden, hp.model.proj)

    def forward(self, x):
        x, _ = self.LSTM_stack(x.float()) #(batch, frames, n_mels)
        #only use last frame
        x = x[:,x.size(1)-1]
        x = self.projection(x.float())
        x = x / torch.norm(x, dim=1).unsqueeze(1)
        return x
```

Embedding

L2 normalized response of the LSTM

$$\mathbf{e}_{ji} = \frac{f(\mathbf{x}_{ji};\mathbf{w})}{||f(\mathbf{x}_{ji};\mathbf{w})||_2}. \qquad \mathbf{c}_k = \mathbb{E}_m[\mathbf{e}_{km}] = \frac{1}{M}\sum_{m=1}^{M}\mathbf{e}_{km}. \qquad \mathbf{S}_{ji,k} = w \cdot \cos(\mathbf{e}_{ji}, \mathbf{c}_k) + b,$$

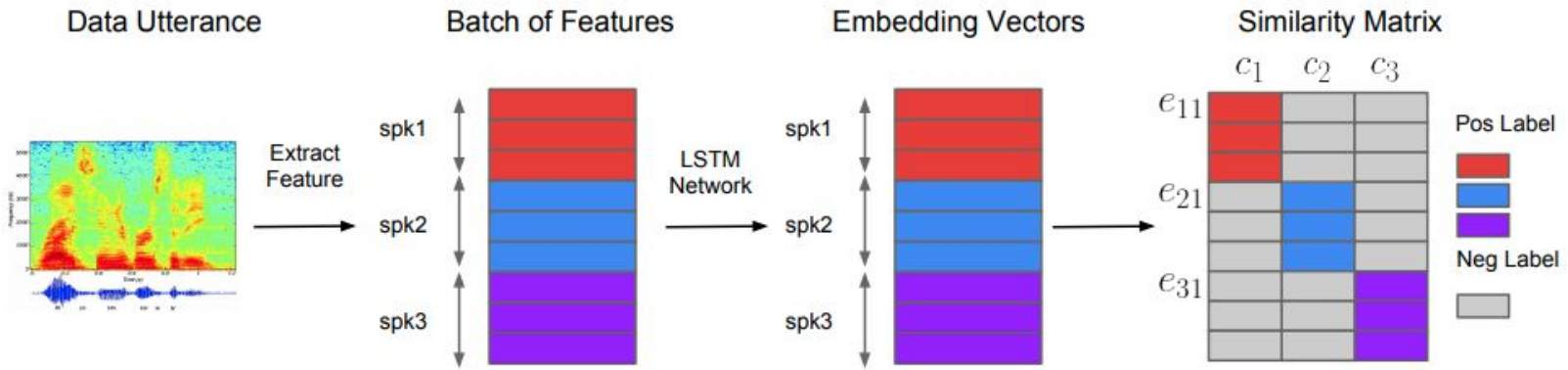


Fig. 1. System overview. Different colors indicate utterances/embeddings from different speakers.

Loss

Softmax Loss: 1 iff k=j (better for TI-SV)

$$L(\mathbf{e}_{ji}) = -\mathbf{S}_{ji,j} + \log \sum_{k=1}^{N} \exp(\mathbf{S}_{ji,k}).$$

Contrastive Loss: positive pairs vs most negative pairs (better for TD-SV)

$$L(\mathbf{e}_{ji}) = 1 - \sigma(\mathbf{S}_{ji,j}) + \max_{\substack{1 \le k \le N \\ k \neq j}} \sigma(\mathbf{S}_{ji,k}),$$

Loss (구현은 softmax)

```python
class GE2ELoss(nn.Module):

    def __init__(self, device):
        super(GE2ELoss, self).__init__()
        self.w = nn.Parameter(torch.tensor(10.0).to(device), requires_grad=True)
        self.b = nn.Parameter(torch.tensor(-5.0).to(device), requires_grad=True)
        self.device = device

    def forward(self, embeddings):
        torch.clamp(self.w, 1e-6)
        centroids = get_centroids(embeddings)
        cossim = get_cossim(embeddings, centroids)
        sim_matrix = self.w*cossim.to(self.device) + self.b
        loss, _ = calc_loss(sim_matrix)
        return loss
```

```python
def get_cossim(embeddings, centroids):
    # Calculates cosine similarity matrix. Requires (N, M, feature) input
    cossim = torch.zeros(embeddings.size(0),embeddings.size(1),centroids.size(0))
    for speaker_num, speaker in enumerate(embeddings):
        for utterance_num, utterance in enumerate(speaker):
            for centroid_num, centroid in enumerate(centroids):
                if speaker_num == centroid_num:
                    centroid = get_centroid(embeddings, speaker_num, utterance_num)
                output = F.cosine_similarity(utterance,centroid,dim=0)+1e-6
                cossim[speaker_num][utterance_num][centroid_num] = output
    return cossim

def calc_loss(sim_matrix):
    # Calculates loss from (N, M, K) similarity matrix
    per_embedding_loss = torch.zeros(sim_matrix.size(0), sim_matrix.size(1))
    for j in range(len(sim_matrix)):
        for i in range(sim_matrix.size(1)):
            per_embedding_loss[j][i] = -(sim_matrix[j][i][j] - ((torch.exp(sim_matrix[j][i]).sum()+1e-6).log_()))
    loss = per_embedding_loss.sum()
    return loss, per_embedding_loss
```

Clustering Effect

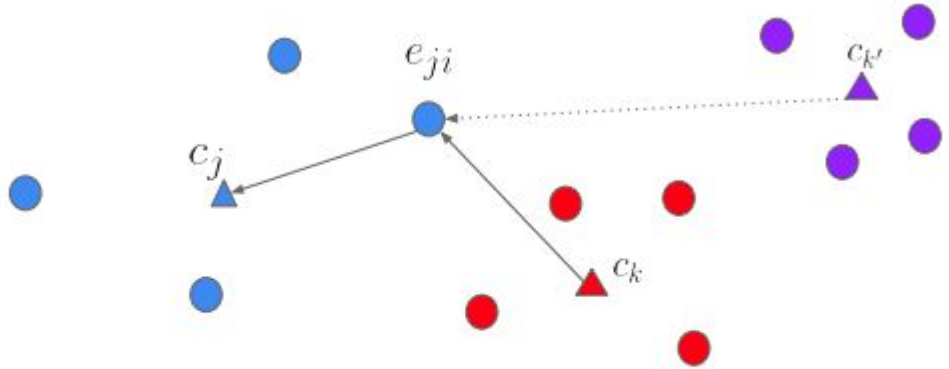


Fig. 2. GE2E loss pushes the embedding towards the centroid of the true speaker, and away from the centroid of the most similar different speaker.

Config

Each batch (N = 64 speakers & M = 10 utterances per speaker)

Training

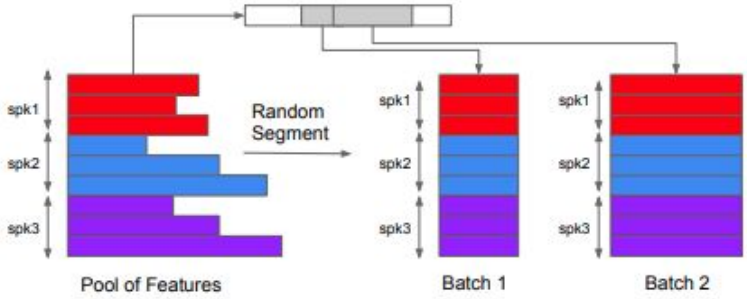


Fig. 3. Batch construction process for training TI-SV models.

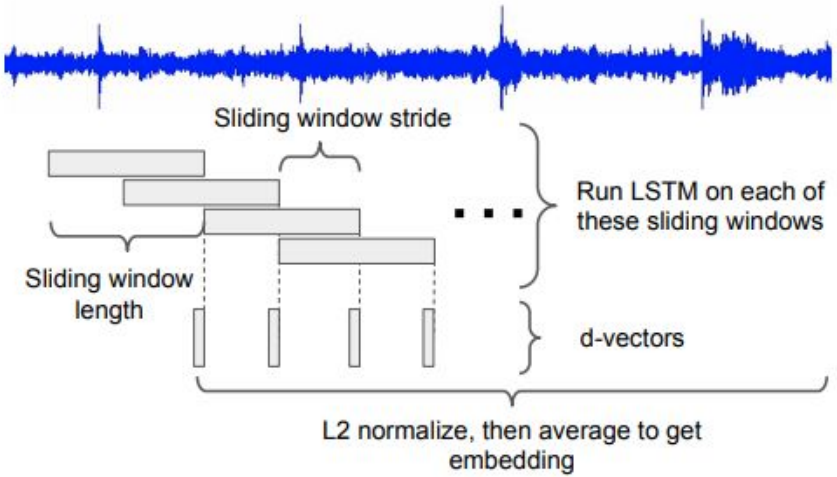


Fig. 4. Sliding window used for TI-SV.

Data

1) OK Google set (~ 150 M utterances, ~ 630 K speakers)

2) OK/Hey Google set (~ 1.2 M utterances, ~ 18 K speakers)

Result

Table 2. Text-dependent speaker verification EER.

Model Architecture	Embed Size	Loss	Multi Reader	Average EER (%)
$(512,)$ [13]	128	TE2E	No	3.30
			Yes	2.78
$(128, 64) \times 3$	64	TE2E	No	3.55
			Yes	2.67
$(128, 64) \times 3$	64	GE2E	No	3.10
			Yes	2.38

Table 3. Text-independent speaker verification EER (%).

Softmax	TE2E [13]	GE2E
4.06	4.13	3.55

X-Vectors: Robust DNN Embeddings for Speaker Recognition

Neural Voice Cloning with a Few Samples (18.02, NIPS)

According to Transfer...

Deep Voice 3 + VoiceLoop

the model parameters are fine-tuned on a small amount of adaptation data

Adaptation vs Encoding

Speaker adaptation

- fine-tuning a multi-speaker generative model (e.g. Deep Voice, GST)

Speaker encoding

- training a separate model to directly infer a new speaker embedding (e.g. Transfer)

Adaptation Strategies

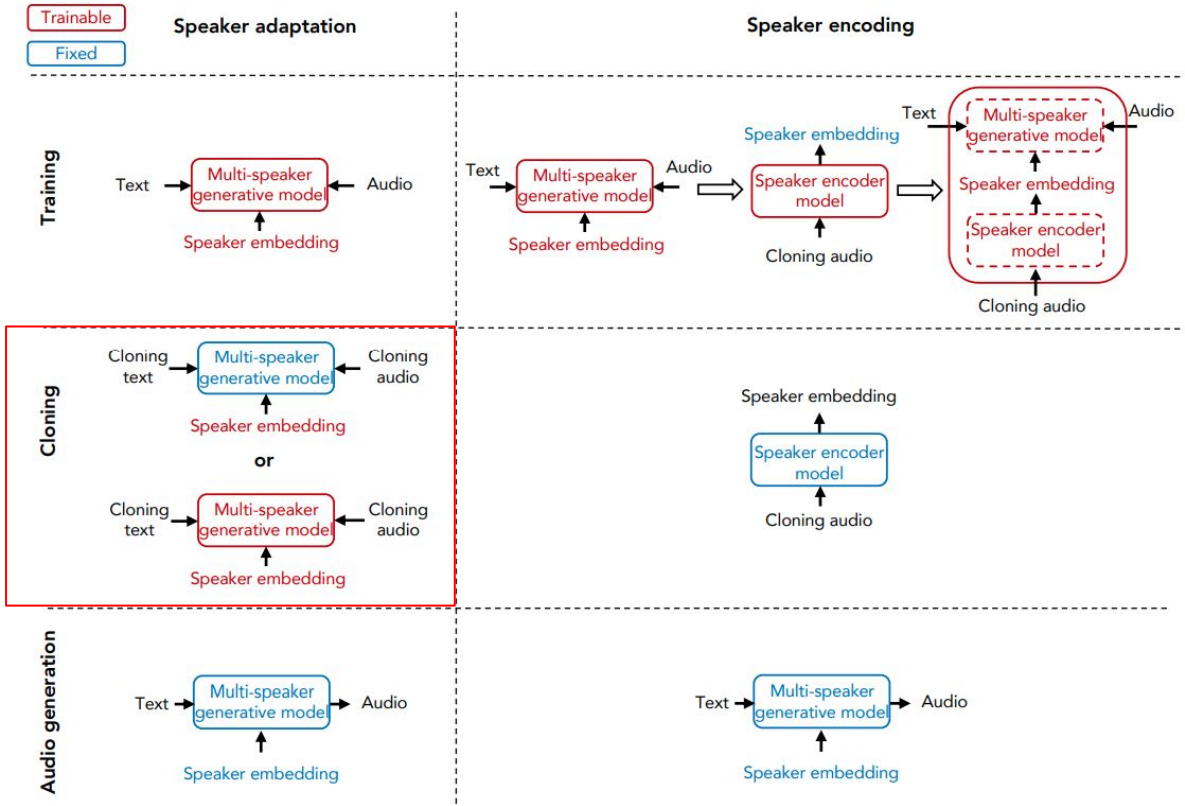
embedding-only adaptation

$$\min_{\mathbf{e}_{s_k}} \mathbb{E}_{(\mathbf{t}_{k,j},\mathbf{a}_{k,j})\sim\mathcal{T}_{s_k}} \left\{ L\left(f(\mathbf{t}_{k,j}, s_k; \widehat{W}, \mathbf{e}_{s_k}), \mathbf{a}_{k,j} \right) \right\},$$

whole model adaptation (prone to overfitting)

$$\min_{W,\mathbf{e}_{s_k}} \mathbb{E}_{(\mathbf{t}_{k,j},\mathbf{a}_{k,j})\sim\mathcal{T}_{s_k}} \left\{ L\left(f(\mathbf{t}_{k,j}, s_k; W, \mathbf{e}_{s_k}), \mathbf{a}_{k,j} \right) \right\}.$$

Speaker Adaptation vs Speaker Encoding



Speaker Encoder

A Detailed speaker encoder architecture

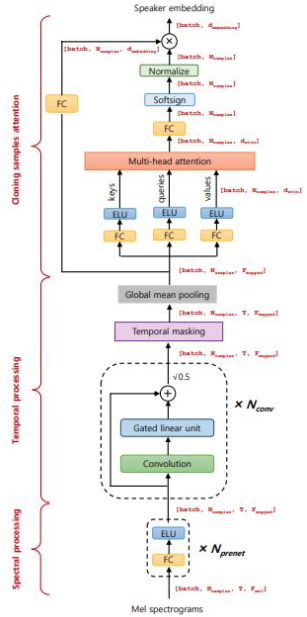
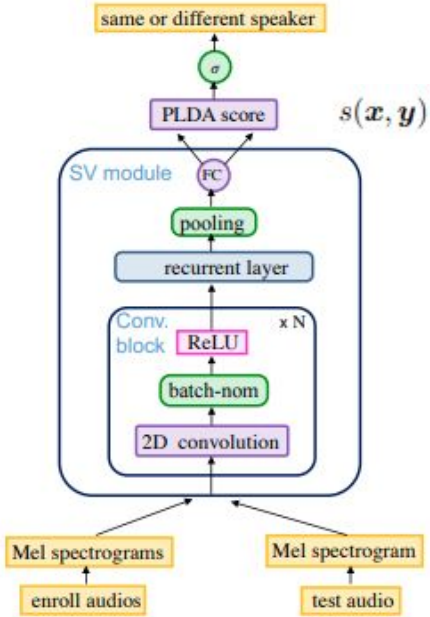


Figure 6: Speaker encoder architecture with intermediate state dimensions. ($batch$: batch size, $N_{samples}$: number of cloning audio samples $|A_{s_k}|$, T: number of mel spectrograms timeframes, F_{mel}: number of mel frequency channels, F_{mapped}: number of frequency channels after prenet, $d_{embedding}$: speaker embedding dimension). Multiplication operation at the last layer represents inner product along the dimension of cloning samples.

Figure 2: Speaker encoder architecture. See Appendix A for details.

Speaker Encoder Task

Speaker Verification



$$s(\boldsymbol{x}, \boldsymbol{y}) = w \cdot \boldsymbol{x}^\top \boldsymbol{y} - \boldsymbol{x}^\top S \boldsymbol{x} - \boldsymbol{y}^\top S \boldsymbol{y} + b$$

Figure 8: Architecture of speaker verification model.

Possible Problems

Joint encoder training → voice averaged to minimize the overall generative loss

We observed optimization challenges with joint training from scratch: the speaker encoder tends to estimate an average voice to minimize the overall generative loss. One possible solution is to introduce discriminative loss functions for intermediate embeddings[4] or generated audios[5]. In our case, however, such approaches only slightly improve speaker differences. Instead, we propose a separate training procedure for speaker encoder. Speaker embeddings $\widehat{\mathbf{e}}_{\mathbf{s_i}}$ are extracted from a trained multi-speaker generative model $f(\mathbf{t}_{i,j}, s_i; W, \mathbf{e}_{s_i})$. Then, the speaker encoder $g(\mathcal{A}_{s_k}; \mathbf{\Theta})$ is trained with an L1 loss to predict the embeddings from sampled cloning audios:

$$\min_{\mathbf{\Theta}} \mathbb{E}_{s_i \sim \mathcal{S}} \left\{ |g(\mathcal{A}_{s_i}; \mathbf{\Theta}) - \widehat{\mathbf{e}}_{\mathbf{s_i}})| \right\}. \tag{5}$$

Components

```
class PreNet(nn.Module):
    '''
    2-layer prenet
    1st is the linear layer.2nd is the elu activation layer
    '''

    def __init__(self , f_mel=80,f_mapped=128):
        super(PreNet,self).__init__()
        self.linear_1 = nn.Linear(f_mel,f_mapped)

    def forward(self,x):
        x = F.elu(self.linear_1(x))
        return x
```

```
class SpectralProcessing(nn.Module):
    '''
    Spectral Transformation layer that transforms mel
    spectogram to size 128
    '''

    def __init__(self,f_mel=80):
        super(SpectralProcessing,self).__init__()
        self.prenet_1  = PreNet(f_mel,128)

    def forward(self,x):
        mapped_x = self.prenet_1(x)

        return mapped_x
```

```
class TemporalProcessing(nn.Module):
    '''
    Implementation of Temporal Processing Layers
    '''

    def __init__(self,in_channels=128, out_channels=128,padding = None,
                 dilation = 2,kernel_size=12):
        super(TemporalProcessing,self).__init__()
        self.conv1d_glu = Conv1dGLU(in_channels,out_channels,padding,dilation,
                                    kernel_size)



    def forward(self,x):
        batch_size = x.size(0)
        # transpose to do operation on the temporal dimension
        x = x.view(batch_size*N_samples, x.size(2), x.size(3)).transpose(1,2)
        x = self.conv1d_glu(x)
        x = x.transpose(1,2)

        x.contiguous()
        x = x.view(batch_size,N_samples,x.size(1),x.size(2))
        #x = librosa.decompose.hpss(x)[0]
        # temporal masking on x
        x = x.mean(dim=2)

        return x
```

Result

Approach	Sample count				
	1	2	3	5	10
Ground-truth (16 KHz sampling rate)	4.66±0.06				
Multi-speaker generative model	2.61±0.10				
Speaker adaptation (embedding-only)	2.27±0.10	2.38±0.10	2.43±0.10	2.46±0.09	2.67±0.10
Speaker adaptation (whole-model)	2.32±0.10	2.87±0.09	2.98±0.11	2.67±0.11	3.16±0.09
Speaker encoding (without fine-tuning)	2.76±0.10	2.76±0.09	2.78±0.10	2.75±0.10	2.79±0.10
Speaker encoding (with fine-tuning)	2.93±0.10	3.02±0.11	2.97±0.1	2.93±0.10	2.99±0.12

Table 2: Mean Opinion Score (MOS) evaluations for naturalness with 95% confidence intervals (training with LibriSpeech speakers and cloning with 108 VCTK speakers).

Approach	Sample count				
	1	2	3	5	10
Ground-truth (same speaker)	3.91±0.03				
Ground-truth (different speakers)	1.52±0.09				
Speaker adaptation (embedding-only)	2.66±0.09	2.64±0.09	2.71±0.09	2.78±0.10	2.95±0.09
Speaker adaptation (whole-model)	2.59±0.09	2.95±0.09	3.01±0.10	3.07±0.08	3.16±0.08
Speaker encoding (without fine-tuning)	2.48±0.10	2.73±0.10	2.70±0.11	2.81±0.10	2.85±0.10
Speaker encoding (with fine-tuning)	2.59±0.12	2.67±0.12	2.73±0.13	2.77±0.12	2.77±0.11

Table 3: Similarity score evaluations with 95% confidence intervals (training with LibriSpeech speakers and cloning with 108 VCTK speakers).

Result

Approach	Sample count				
	1	5	10	20	100
Speaker adaptation (embedding-only)	3.01±0.11	-	3.13±0.11	-	3.13±0.11
Speaker adaptation (whole-model)	2.34±0.13	2.99±0.10	3.07±0.09	3.40±0.10	3.38±0.09

Table 4: Mean Opinion Score (MOS) evaluations for naturalness with 95% confidence intervals (training with 84 VCTK speakers and cloning with 16 VCTK speakers).

Approach	Sample count				
	1	5	10	20	100
Speaker adaptation (embedding-only)	2.42±0.13	-	2.37±0.13	-	2.37±0.12
Speaker adaptation (whole-model)	2.55±0.11	2.93±0.11	2.95±0.10	3.01±0.10	3.14±0.10

Table 5: Similarity score evaluations with 95% confidence intervals (training with 84 VCTK speakers and cloning with 16 VCTK speakers).

Conclusion

whole-model > embedding-only

discriminative speaker info exists in the model besides speaker embeddings

The benefit of compact representation via embeddings is fast cloning and small footprint per speaker. We observe **drawbacks of training the multi-speaker generative model using a speech recognition dataset with low-quality audios and limited speaker diversity**.

Audio

https://audiodemos.github.io/

Fitting New Speakers Based on a Short Untranscribed Sample (18.02, Facebook)

According to Transfer...

Extended VoiceLoop to utilize a target speaker encoding network to predict a speaker embedding, trained jointly with the synthesis network

contrastive triplet loss

> to ensure that embeddings predicted from utterances by the same speaker are closer than embeddings computed from different speakers.

a cycle-consistency loss

> to ensure that the synthesized speech encodes to a similar embedding as the adaptation utterance.

Towards End-to-End Prosody Transfer for Expressive Speech Synthesis with Tacotron (18. 03)

Prosody

The variation in speech signals that remains after accounting for variation due to phonetics, speaker identity, and channel effects (i.e. the recording environment)

Overall Architecture

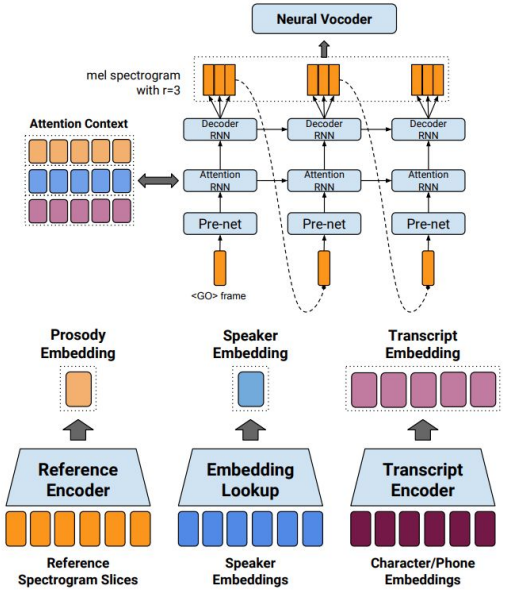


Figure 1. The full Tacotron architecture for prosody control. The autoregressive decoder is conditioned on the result of the reference encoder, transcript encoder, and speaker embedding via an attention module.

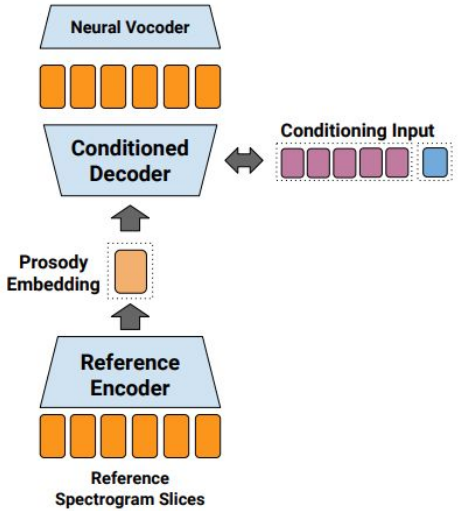


Figure 3. An interpretation of the Tacotron architecture for prosody control from Figure 1 as an RNN encoder-decoder with speaker and phonetic conditioning input.

Speaker Embedding

Just a look-up

Concatenated to encoder output every step

No additional changes or loss metrics

***Dimension size?**

Prosody Embedding

Reference encoder

6-Layer Conv2D:

filter (3x3),

strides (2x2),

32-32-64-64-128-128

128d GRU

Linear projection to 128

Concatenated to encoder output every step

No additional loss?*

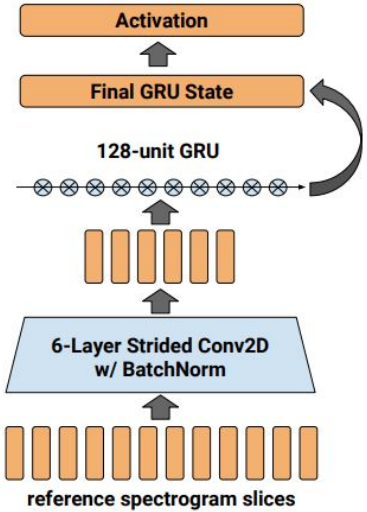


Figure 2. The prosody reference encoder module. A 6-layer stack of 2D convolutions with batch normalization, followed by "recurrent pooling" to summarize the variable length sequence, followed by an optional fully connected layer and activation.

Result

“This suggests that the prosody and speaker representations are somewhat entangled.”

Result

Table 1. A summary of quantitative and subjective metrics (Section 4.2) used to evaluate the prosody transfer. Lower is better for both MCD_k and FFE. Higher subjective scores are better, and indicate whether human raters believe the voice is closer in prosody to the reference than the corresponding baseline model on a 7 point (-3 to 3) scale, where 0 is "about the same".

VOICE	MODEL	REFERENCE	MCD_{13}	FFE	SUBJECTIVE
SINGLE-SPEAKER	BASELINE	SAME SPEAKER	10.63	53.2%	
SINGLE-SPEAKER	TANH-128	SAME SPEAKER	**7.92**	**28.1%**	$\mathbf{1.611 \pm 0.164}$
SINGLE-SPEAKER	BASELINE	UNSEEN SPEAKER	11.22	59.6%	
SINGLE-SPEAKER	TANH-128	UNSEEN SPEAKER	**8.89**	**38.0%**	$\mathbf{1.465 \pm 0.132}$
MULTI-SPEAKER	BASELINE	SAME SPEAKER	9.93	48.5%	
MULTI-SPEAKER	TANH-128	SAME SPEAKER	**6.99**	**27.5%**	$\mathbf{1.307 \pm 0.127}$
MULTI-SPEAKER	BASELINE	SEEN SPEAKER	12.37	64.2%	
MULTI-SPEAKER	TANH-128	SEEN SPEAKER	**9.51**	**37.1%**	$\mathbf{0.871 \pm 0.138}$
MULTI-SPEAKER	BASELINE	UNSEEN SPEAKER	11.84	60.0%	
MULTI-SPEAKER	TANH-128	UNSEEN SPEAKER	**10.87**	**41.3%**	$\mathbf{1.146 \pm 0.246}$

Audio

https://google.github.io/tacotron/publications/end_to_end_prosody_transfer/

Style Tokens: Unsupervised Style Modeling, Control and Transfer in End-to-End Speech Synthesis (18.03, Google)

Global Style Tokens

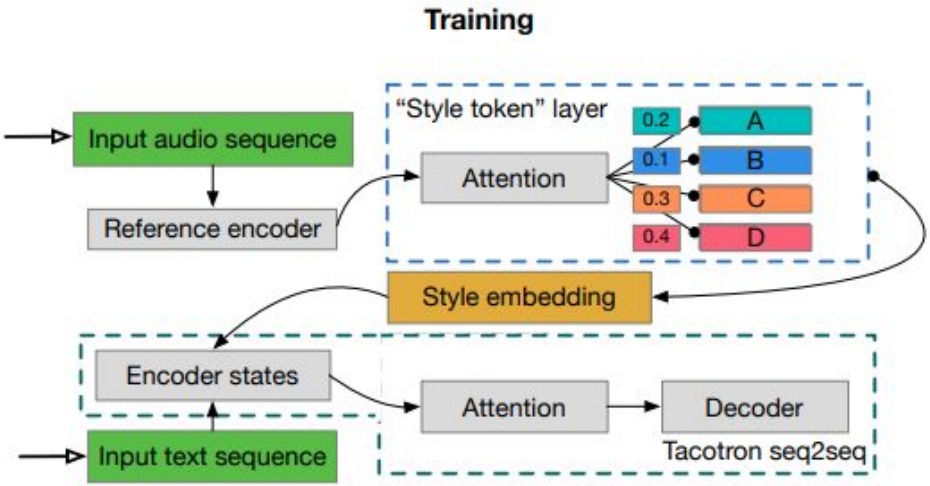
A bank of embeddings that are JOINTLY trained within Tacotron

Neither SINGLE (e.g. Mellotron) nor SITE-SPECIFIC (e.g. DeepVoice)

Control prosody (factorizing noise, speaker id, speed etc.)

GST Architecture

특이점: BOTH Encoder & Embedding
Concat with encoder output



Reference Encoder

Encoder (Conv + RNN)

Log-mel

Six 2D conv [32, 32, 64, 64, 128, 128] (3×3 kernel, 2×2 stride) + BN + ReLU

Shaped back to 3d (preserving the output time resolution)

A single-layer 128-unit unidirectional GRU

The last GRU state: the “reference” embedding (NOT “style” or “token”)

Fed to the style layer (= Query)

Style Token Layer

A bank of 10 "token" embeddings and an attention

Each token embedding is 256D (the same as the text encoder state)

Tanh activation to GSTs before applying attention (greater token diversity)

The content-based tanh attention uses a softmax activation to output a set of combination weights over the tokens

특이점: attention은 다양한 종류 사용해도 좋고 특히 Multi-head attention이 효과적

Loss

The style token layer is jointly trained with the rest of the model,
driven only by the reconstruction loss from the Tacotron decoder
(no need for labels)

Injection Spot

Replicate the "style" embedding

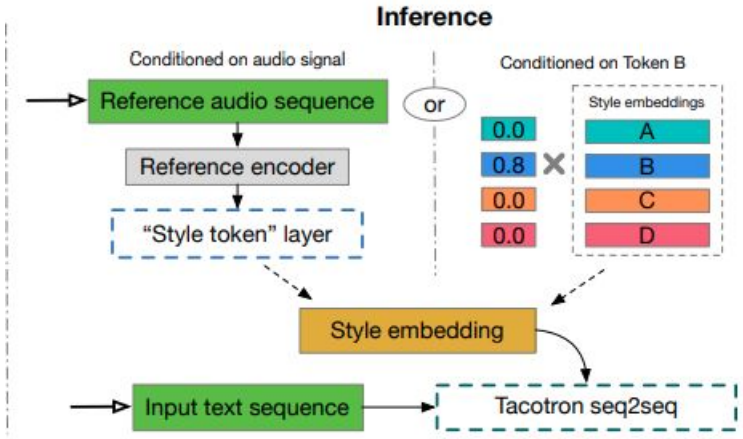
Add it to every text encoder state

*특이점: 다른 여러 포지션도 테스트한 결과 **text encoder state** 쪽이 가장 효과적

Inference

Feed untranscribed audio

Control token weights



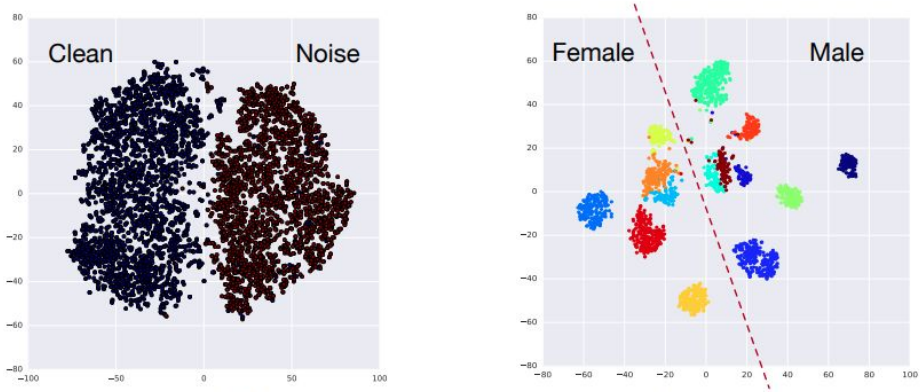
Result

Table 3. WER for the Spanish to English unsupervised language transfer experiment. Note that WER is an underestimate of the true intelligibility score; we only care about the relative differences.

MODEL	WER (INS/DEL/SUB)
GST	18.68 (6.13/2.37/10.18)
MULTI-SPEAKER	56.18 (3.75/20.27/32.14)

Table 4. Classification accuracy (noise-vs-clean and TED speaker ID) using GSTs and i-vectors. Despite being trained within a generative model, GSTs encode rich discriminative information.

EMBEDDING	ARTIFICIAL DATA	TED (431 SPEAKERS)
GST	99.2%	75.0%
I-VECTOR	/	73.4%



(a) 50% noisy data (b) Multi-speaker TED

Figure 8. Style embedding visualization using t-SNE.

Problems

Prosody = Speaker ID? (this also concerns Mellotron, Prosody Transfer by Skerry-Ryan)

Audio

https://google.github.io/tacotron/publications/global_style_tokens/

Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis (18.06, Google)

Three INDEPENDENT Models

Speaker Encoder: Trained on TI-SV

Tacotron2

WaveNet

Architecture

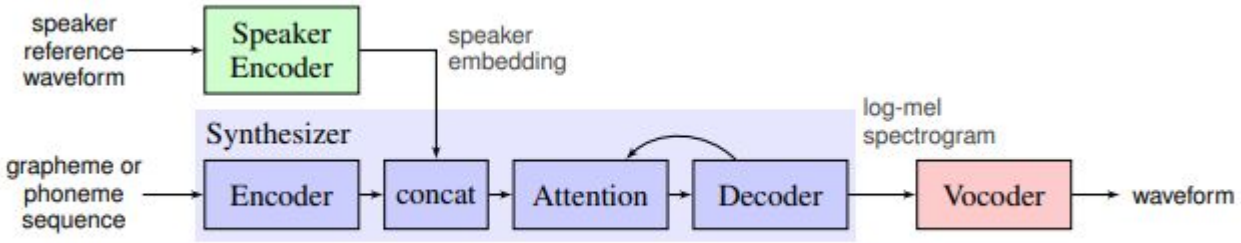


Figure 1: Model overview. Each of the three components are trained independently.

Speaker Encoder

40 mels

3 LSTM + proj (768 → 256)

L2-normalization (final hidden)

800 ms windows

36M utterances, 18k speakers

Embedding_dim 64

GE2E

```
class SpeakerEncoder(nn.Module):
    def __init__(self, device, loss_device):
        super().__init__()
        self.loss_device = loss_device

        # Network defition
        self.lstm = nn.LSTM(input_size=mel_n_channels,
                            hidden_size=model_hidden_size,
                            num_layers=model_num_layers,
                            batch_first=True).to(device)
        self.linear = nn.Linear(in_features=model_hidden_size,
                                out_features=model_embedding_size).to(device)
        self.relu = torch.nn.ReLU().to(device)

        # Cosine similarity scaling (with fixed initial parameter values)
        self.similarity_weight = nn.Parameter(torch.tensor([10.])).to(loss_device)
        self.similarity_bias = nn.Parameter(torch.tensor([-5.])).to(loss_device)

        # Loss
        self.loss_fn = nn.CrossEntropyLoss().to(loss_device)
```

Concat to Encoder Output in Synthesizer

```
### SV2TT2 ###

# Append the speaker embedding to the encoder output at each timestep
tileable_shape = [-1, 1, self._hparams.speaker_embedding_size]
tileable_embed_targets = tf.reshape(tower_embed_targets[i], tileable_shape)
tiled_embed_targets = tf.tile(tileable_embed_targets,
                              [1, tf.shape(encoder_outputs)[1], 1])
encoder_cond_outputs = tf.concat((encoder_outputs, tiled_embed_targets), 2)

###############
```

Ind. Encoder vs. Joint Embedding

One computes the embedding using the speaker encoder (Google style)

Baseline optimizes a fixed embedding for each speaker in the training set (Deep Voice Style)

Otherwise, **IDENTICAL**

Result

Table 1: Speech naturalness Mean Opinion Score (MOS) with 95% confidence intervals.

System	VCTK Seen	VCTK Unseen	LibriSpeech Seen	LibriSpeech Unseen
Ground truth	4.43 ± 0.05	4.49 ± 0.05	4.49 ± 0.05	4.42 ± 0.07
Embedding table	4.12 ± 0.06	N/A	3.90 ± 0.06	N/A
Proposed model	4.07 ± 0.06	4.20 ± 0.06	3.89 ± 0.06	4.12 ± 0.05

Result

Table 2: Speaker similarity Mean Opinion Score (MOS) with 95% confidence intervals.

System	Speaker Set	VCTK	LibriSpeech
Ground truth	Same speaker	4.67 ± 0.04	4.33 ± 0.08
Ground truth	Same gender	2.25 ± 0.07	1.83 ± 0.07
Ground truth	Different gender	1.15 ± 0.04	1.04 ± 0.03
Embedding table	Seen	4.17 ± 0.06	3.70 ± 0.08
Proposed model	Seen	4.22 ± 0.06	3.28 ± 0.08
Proposed model	Unseen	3.28 ± 0.07	3.03 ± 0.09

Audio

https://google.github.io/tacotron/publications/speaker_adaptation/

Mellotron: Multispeaker Expressive Voice Synthesis by Conditioning on Rhythm, Pitch and Global Style Tokens (19.10, Nvidia)

Multi-Speaker Strategy

SINGLE speaker embedding (NOT site-specific)

channel-wise concatenated with the encoder outputs over every token

n_speaker = 123,

speaker_embedding _dim = 128

```
self.speaker_embedding = nn.Embedding(
    hparams.n_speakers, hparams.speaker_embedding_dim)
```

Mellotron Forward

```
def forward(self, inputs):
    inputs, input_lengths, targets, max_len, \
        output_lengths, speaker_ids, f0s = inputs
    input_lengths, output_lengths = input_lengths.data, output_lengths.data

    embedded_inputs = self.embedding(inputs).transpose(1, 2)
    embedded_text = self.encoder(embedded_inputs, input_lengths)
    embedded_speakers = self.speaker_embedding(speaker_ids)[:, None]
    embedded_gst = self.gst(targets)
    embedded_gst = embedded_gst.repeat(1, embedded_text.size(1), 1)
    embedded_speakers = embedded_speakers.repeat(1, embedded_text.size(1), 1)

    encoder_outputs = torch.cat(
        (embedded_text, embedded_gst, embedded_speakers), dim=2)

    mel_outputs, gate_outputs, alignments = self.decoder(
        encoder_outputs, targets, memory_lengths=input_lengths, f0s=f0s)

    mel_outputs_postnet = self.postnet(mel_outputs)
    mel_outputs_postnet = mel_outputs + mel_outputs_postnet

    return self.parse_output(
        [mel_outputs, mel_outputs_postnet, gate_outputs, alignments],
        output_lengths)
```

Audio

https://nv-adlr.github.io/Mellotron

Learning to Speak Fluently in a Foreign Language: Multilingual Speech Synthesis and Cross-Language Voice Cloning (19. 07, Interspeech)

Lit Review: Multi-Speaker

Neural voice cloning with a few samples*

Transfer learning from speaker verification to multispeaker text-to-speech synthesis*

Fitting new speakers based on a short untranscribed sample

Sample efficient adaptive text-to-speech

Contributions

Multi-Language Strategy

- Phonemic input

Multi-Speaker Strategy

- Adversarial loss to disentangle the representation of speaker ID
- Explicit language embedding

Overall Architecture

의문점: residual encoder도 독립적 학습을 요하는지

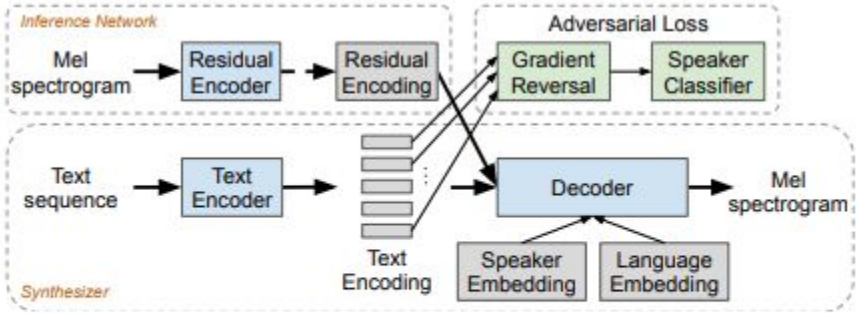


Figure 1: *Overview of the components of the proposed model. Dashed lines denote sampling via reparameterization [21] during training. The prior mean is always use during inference.*

Residual Encoder

VAE-like: maps a variable length mel spec to two vectors

- mean
- log variance of the Gaussian posterior

Residual other than

- text representation
- speaker
- language embeddings

Embedding

Speaker: 64-dim, Language: 3-dim

Injection (정확한 설명 X)

> “Concatenated and passed to the decoder at each step”
> 아마도 torch.cat(speaker_emb, language_emb, encoder_output)

*특이점: 멜을 128bin 전부 사용

DATA

3 Languages

92 Speakers

EN	84EN 3ES 5CN language ID fixed to EN
ES	84EN 3ES 5CN
CN	84EN 3ES 5CN

Result

Table 4: *Naturalness and speaker similarity MOS of cross-language voice cloning of the full multilingual model using phoneme inputs.*

Source Language	Model	EN target		ES target		CN target	
		Naturalness	Similarity	Naturalness	Similarity	Naturalness	Similarity
-	Ground truth (self-similarity)	4.60±0.05	4.40±0.07	4.37±0.06	4.39±0.06	4.42±0.06	3.51±0.12
EN	84EN 3ES 5CN	4.37±0.12	4.63±0.06	4.20±0.07	3.50±0.12	3.94±0.09	3.03±0.10
	language ID fixed to EN	-	-	3.68±0.07	4.06±0.09	3.09±0.09	3.20±0.09
ES	84EN 3ES 5CN	4.28±0.10	3.24±0.09	4.37±0.04	4.01±0.07	3.85±0.09	2.93±0.12
CN	84EN 3ES 5CN	4.49±0.08	2.46±0.10	4.56±0.08	2.48±0.09	4.09±0.10	3.45±0.12

Audio

https://google.github.io/tacotron/publications/multilingual/

Multi-Speaker End-to-End Speech Synthesis (19. 07)

Basic Strategy

***Similar to Deep Voice 3**

low dimensional speaker embeddings

- shared across each component of ClariNet and trained jointly
- As a bias to each part of the network (= element-wise addition?)

ClariNet

TTWave (NOT TTMel)

TTS+Vocoder trained jointly at once

Embedding to Four Different Locations

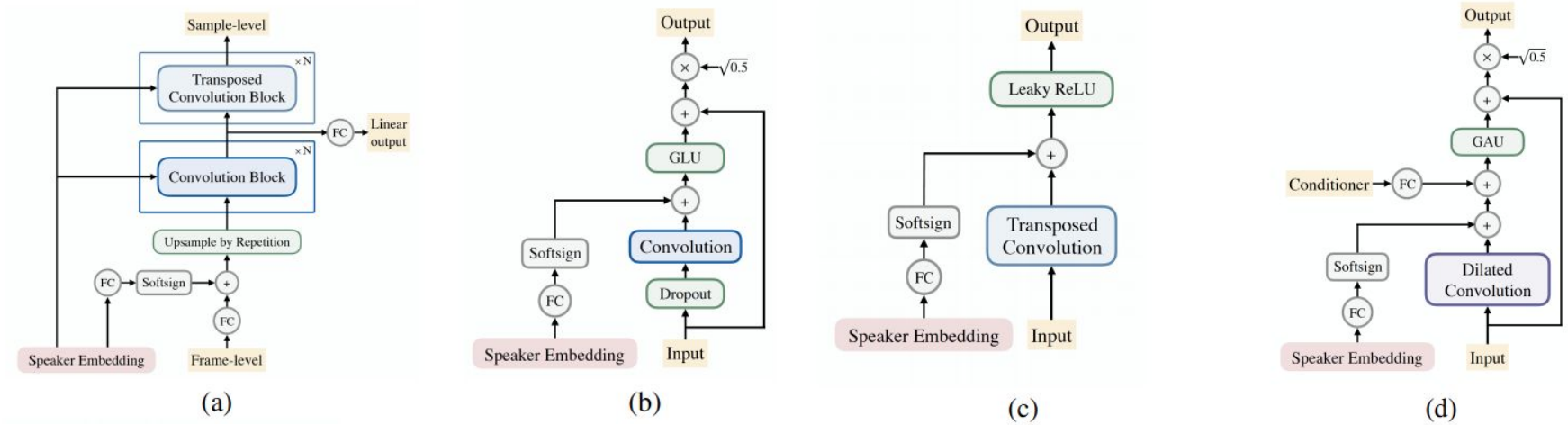


Figure 3: Architectures of (a) Bridge-net, (b) Convolution block, (c) Transposed convolution block, and (d) Dilated convolution layer.

Overall Architecture

*보코더에도 스피커 임베딩

*보코더까지 한 번에 학습

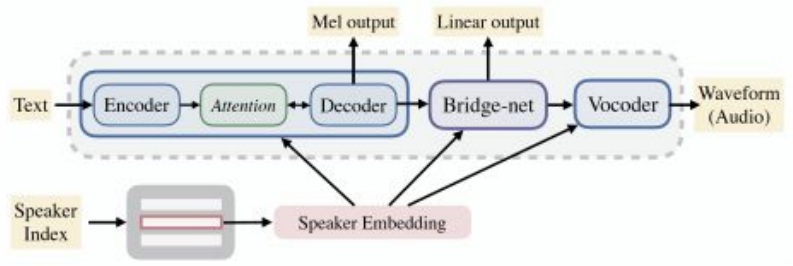


Figure 1: Overall architecture of multi-speaker ClariNet.

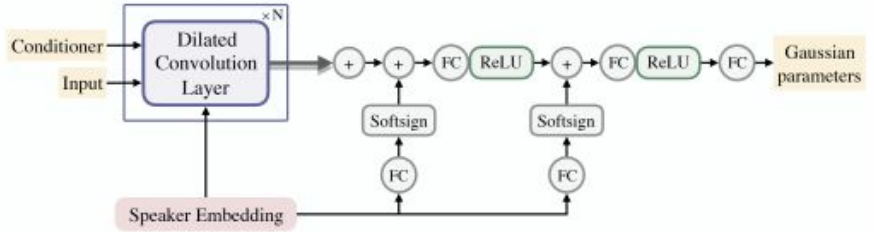


Figure 2: Overall architecture of the Gaussian autoregressive WaveNet. Shadowed arrow after the dilated convolution layers depicts the skip connections, and GAU stands for gated activation unit.

Audio

https://multi-speaker-clarinet-demo.github.io/

Zero-Shot Multi-Speaker Text-To-Speech with State-Of-The-Art Neural Speaker Embeddings (19. 10, ICASSP)

Literature Review: Multi-Speaker

1. Joint Training of Speaker Encoder with TTS
    Multi-speaker end-to-end speech synthesis*

2. Speaker-Embedding + Fine-Tuning
    Neural voice cloning with a few samples*

3. Transfer Learning (Speaker encoder trained independently for SC, SV, etc.)
    Transfer learning from SV to multispeaker text-to-speech synthesis

Literature Review: Speaker-Verification

x-vector based (over i-vector)

New encoding methods

End-to-end loss functions

Learnable dictionary encoding

Angular softmax

Speaker Recognition

Encoder Network (frame-level)

Statistical Pooling Layer (fixed-d)

Classifier

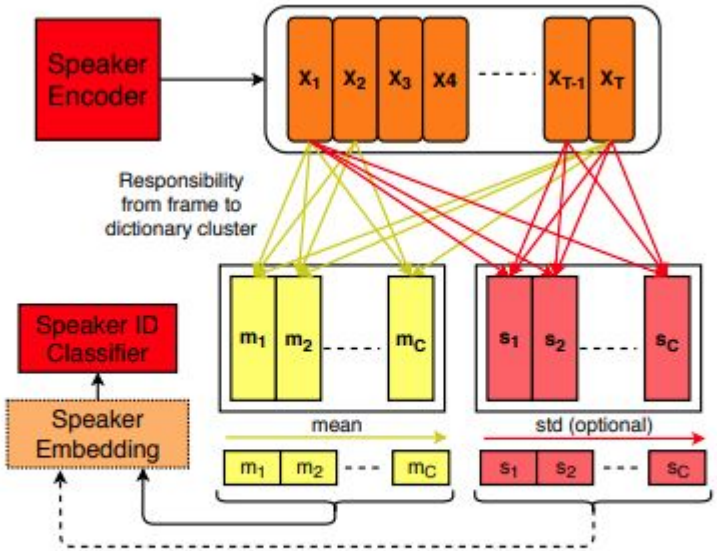


Fig. 1. Learnable dictionary encoding (LDE) pooling method in an end-to-end speaker recognition system.

Speaker Embedding

Projected down to 64d (SINGLE embedding for any location)

Injection: where and how

1 Concat to each encoder output

2 input to the prenet

3 input to the postnet

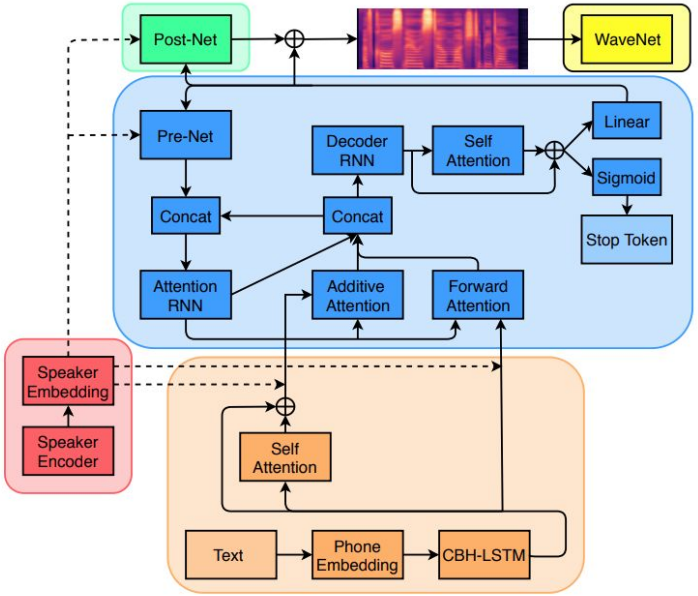


Fig. 2. Proposed multi-speaker TTS system. Encoder blocks are in *orange*, decoder blocks in *blue*, pre-net block in *green*, speaker encoder blocks in *red*, and vocoder block in *yellow*.

Result

*post-only terrible

Input location	Gender-ind		Gender-dep	
	train	dev	train	dev
pre	0.357	0.402	0.438	0.361
attn	0.709	0.490	0.711	0.476
pre+attn	0.676	0.489	0.708	**0.533**
pre+attn+post	0.684	0.480	0.717	0.477

- Concatenate with encoder output only and input to attention mechanism (`attn`)
- Prenet + concatenate with encoder output (`pre+attn`)
- Prenet + concatenate with encoder output + postnet (`pre+attn+post`)

Audio

https://nii-yamagishilab.github.io/samples-multi-speaker-tacotron/