# Tacotron이 말을 했어요

AiFrenz

2019년04월24일, 조희철

# 목차

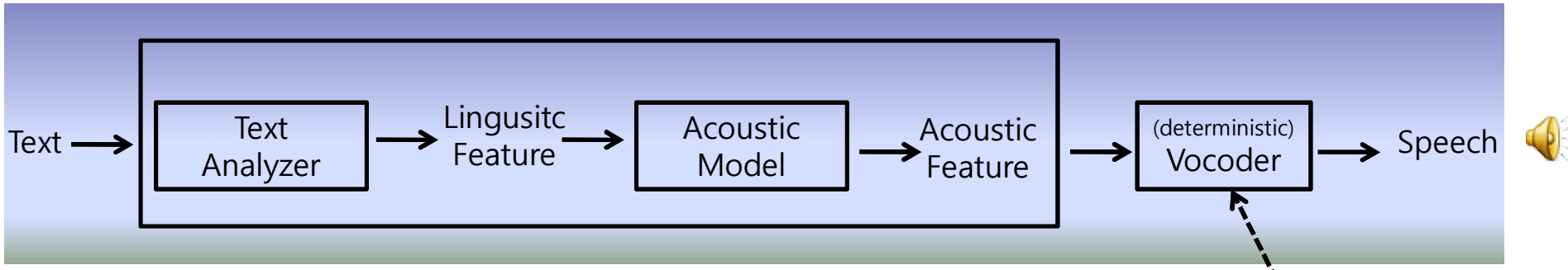
# 1.음성합성 & Tacotron

# AiFrenz 회원여러분, 반갑습니다.

# 음성합성 모델

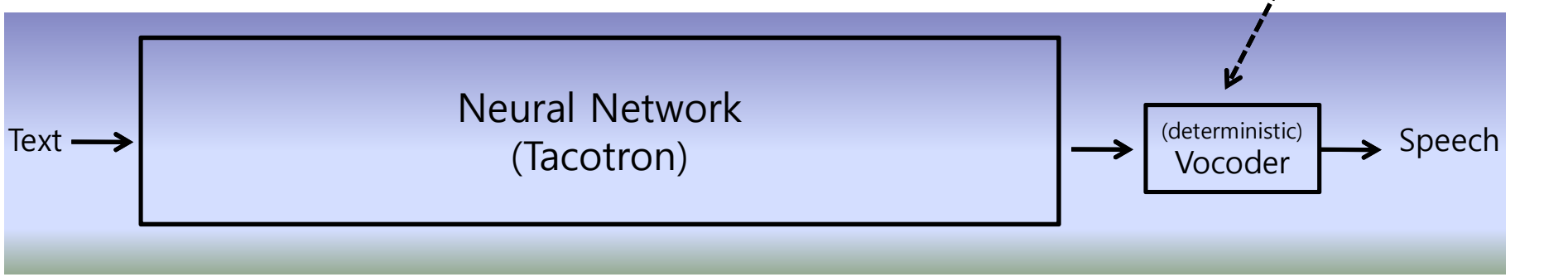
전통적 음성합성

- concatenative TTS: database에 있는 음성을 합성 ➔ 음질은 좋지만, 방대한 DB필요
- statistical parametric TTS: HMM같은 모델에 기반한 방법
  - ➢ text analyzer, F0 generator, spectrum generator, pause estimator, vocoder
  - ➢ 음질은 좋지 못하지만, Acoustic Feature 조절 가능




TACOTRON: End-To-End Speech Synthesis, 2017년3월

**Wavenet Vocoder** 로 대체 가능



# Tacotron Model Architecture

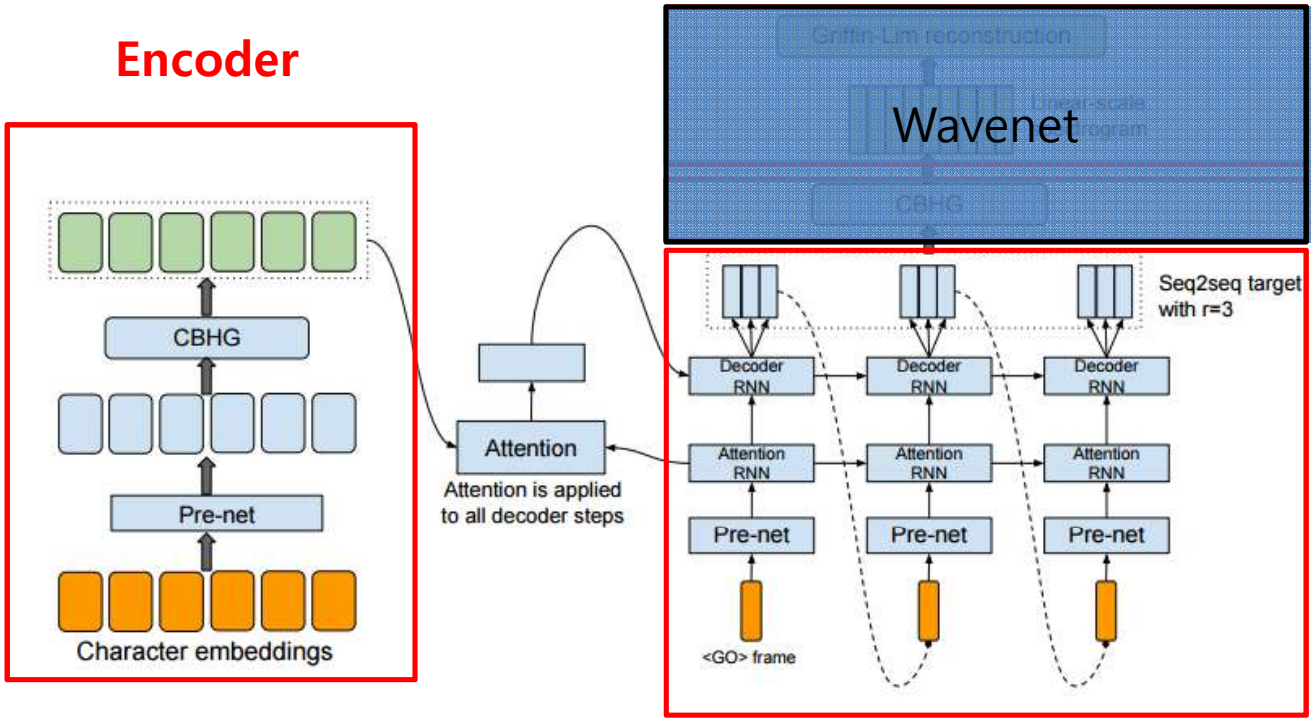



Figure 1: *Model architecture. The model takes characters as input and outputs the corresponding raw spectrogram, which is then fed to the Griffin-Lim reconstruction algorithm to synthesize speech.*

**합성 단계**

Input: Text
Outputs: Mels-pectrogram(예측) → (linear) Spectrogram(예측) → Speech(Audio)

**Train 단계**

Inputs: Text, **Mel-spectrogram, (linear) Spectrogram**
Outputs: Mels-pectrogram(예측) → (linear) Spectrogram(예측) → Speech(Audio)

# Tacotron 구현 Code

- Wavenet(2016년9월)
- ibab 코드 공개(2016년9월)
- Tacotron 논문 발표(2017년3월)

- keithito(2017년7월)
  - 대표적인 Tacotron 구현
  - https://github.com/keithito/tacotron

- carpedm20(2017년10월)
  - keithito 코드를 기반으로 Tacotron 모델로 **한국어 생성**
  - DeepVoice 2에서 제안한 **Multi-Speaker 모델로 확장**
  - Tensorflow 1.3 ➔ 최신 버전에 작동하지 않음.
  - https://github.com/carpedm20/multi-speaker-tacotron-tensorflow

- Tacotron2(2017년12월)
- r9y9 코드(wavenet vocoder) 공개(2018년1월)

- Rayhane-mamah(2018년4월)
  - keithito, r9y9 코드를 기반으로 구현된 대표적인 Tacotron 2 구현
  - Wavenet 구현도 포함
  - https://github.com/Rayhane-mamah/Tacotron-2

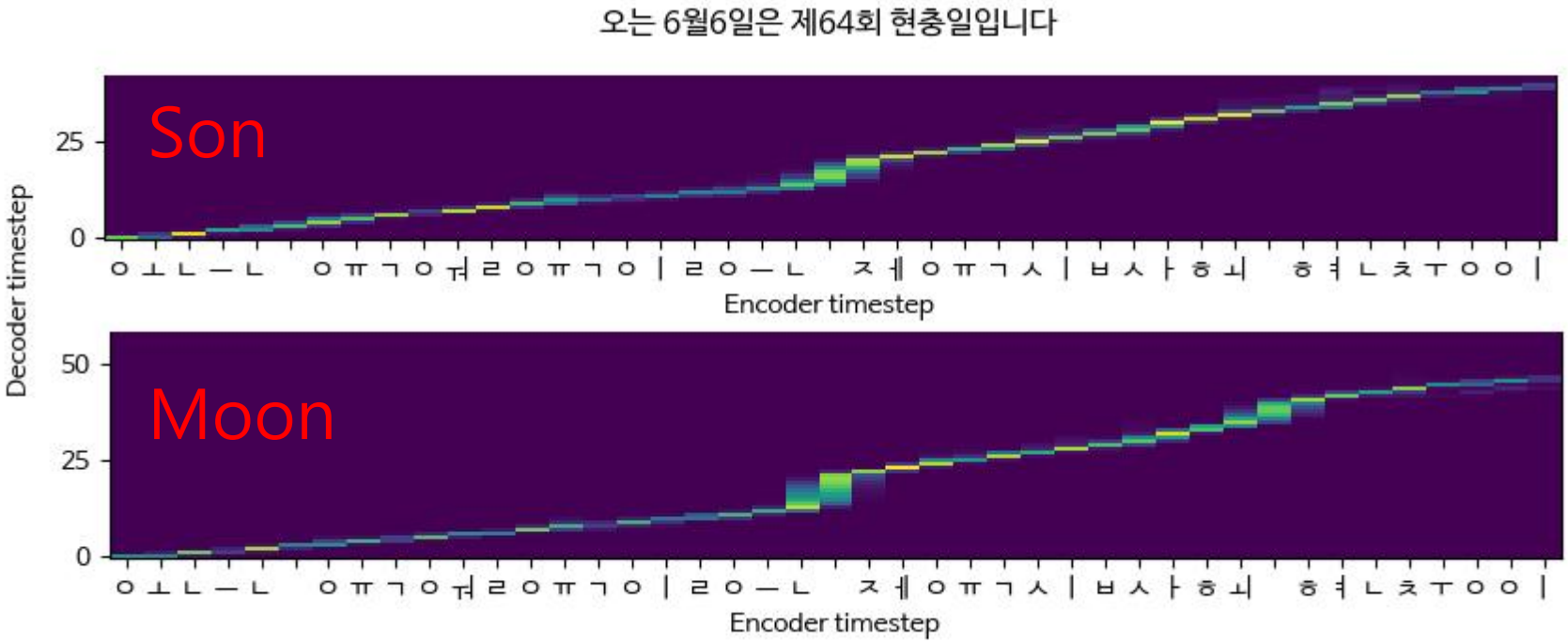
- hccho2(2018년12월)
  - 한국어 Tacotron + Wavenet, Tensorflow 최신 버전으로 실행
  - 빠른(speed up) convergence
  - https://github.com/hccho2/Tacotron-Wavenet-Vocoder

# Audio Samples

**train step: 106000(GTX1080ti-18h)**
**moon data: 1,125 examples (0.89 hours)**
**son data: 20,105 examples (19.10 hours)**

- 이런 논란은 타코트론 논문 이후에 사라졌습니다.

- 오는 6월6일은 제64회 현충일입니다.



# Audio Samples(Tacotron2+ Griffin-Lim)

**train step: 100,000(27h)**
**moon data: 1,125 examples (0.89 hours)**
**son data: 20,105 examples (19.10 hours)**

- 이런 논란은 타코트론 논문 이후에 사라졌습니다.

- 오는 6월6일은 제64회 현충일입니다.

| Model | # of trainable_variables() | sec/step (GTX1080ti) |
|---|---|---|
| Tacotron 1 | 7M | 0.60 |
| Tacotron 2(Griffin-Lim) | 29M | 0.98 |

# 한국어 Data 준비: 음성/Text

- (약 12초 이하 길이의 음성 파일, Script) 쌍이 필요하다.
- 긴 음성 파일 ➔ 약 12초 이하의 길이로 잘라야 한다.
  - ✓ 문장 단위로 자르지 않아도 된다. 침묵 구간을 기준으로 자른다.
  - ✓ 잘라진 음성 파일과 script의 sync를 맞추는 것은 고단한 작업.
  - ✓ 잘라진 음성 파일 ➔ Google Speech API(STT)로 script 생성
  - ✓ STT로 생성한 script를 원문과 비교하여 수정(수작업 vs programming)

| 원문 | Google STT 결과 |
|---|---|
| 세수실적이 좋아 | 최순실 저 좋아 |
| 추경예산안의 | 조병일 전화해 |
| 대폭 늘리겠습니다 | 아이폰 6s |
| 국회를 존중하면서 | 쿠키런 존중하며서 |
| 세계 평화와 안보에 기여해 온 | 세계 평화와 안 보여 귀여워요 |

# 한글 Text 분해

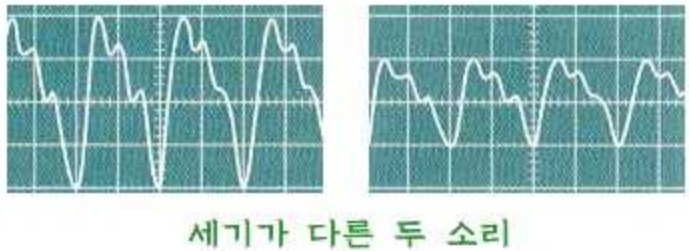
- 한글 text를 (초성/중성/종성)으로 나누어진 sequence로 만들어야 한다.
  - jamo package를 이용하면 된다.
  - *'존경하는'* ➔ *['ㅈ', 'ㅗ', 'ㄴ', 'ㄱ', 'ㅕ', 'ㅇ', 'ㅎ', 'ㅏ', 'ㄴ', 'ㅡ', 'ㄴ', ~']*
    ➔ [14, 29, 45, 2, 27, 62, 20, 21, 4, 39, 45, 1]
  - 초성과 종성의 자음은 각각 다른 character로 처리

*'_': 0, '~': 1, 'ㄱ': 2, 'ㄲ': 3, 'ㄴ': 4, 'ㄷ': 5, 'ㄸ': 6, 'ㄹ': 7, 'ㅁ': 8, 'ㅂ': 9, 'ㅃ': 10,*
*'ㅅ': 11, 'ㅆ': 12, 'ㅇ': 13, 'ㅈ': 14, 'ㅉ': 15, 'ㅊ': 16, 'ㅋ': 17, 'ㅌ': 18, 'ㅍ': 19, 'ㅎ': 20,*
*'ㅏ': 21, 'ㅐ': 22, 'ㅑ': 23, 'ㅒ': 24, 'ㅓ': 25, 'ㅔ': 26, 'ㅕ': 27, 'ㅖ': 28, 'ㅗ': 29, 'ㅘ': 30,*
*'ㅙ': 31, 'ㅚ': 32, 'ㅛ': 33, 'ㅜ': 34, 'ㅝ': 35, 'ㅞ': 36, 'ㅟ': 37, 'ㅠ': 38, 'ㅡ': 39, 'ㅢ': 40,*
*'ㅣ': 41, 'ㄱ': 42, 'ㄲ': 43, 'ㄳ': 44, 'ㄴ': 45, 'ㄵ': 46, 'ㄶ': 47, 'ㄷ': 48, 'ㄹ': 49, 'ㄺ': 50,*
*'ㄻ': 51, 'ㄼ': 52, 'ㄽ': 53, 'ㄾ': 54, 'ㄿ': 55, 'ㅀ': 56, 'ㅁ': 57, 'ㅂ': 58, 'ㅄ': 59, 'ㅅ': 60,*
*'ㅆ': 61, 'ㅇ': 62, 'ㅈ': 63, 'ㅊ': 64, 'ㅋ': 65, 'ㅌ': 66, 'ㅍ': 67, 'ㅎ': 68, '!': 69, '"': 70,*
*'(': 71, ')': 72, ',': 73, '-': 74, '.': 75, ':': 76, ';': 77, '?': 78, ' ': 79*

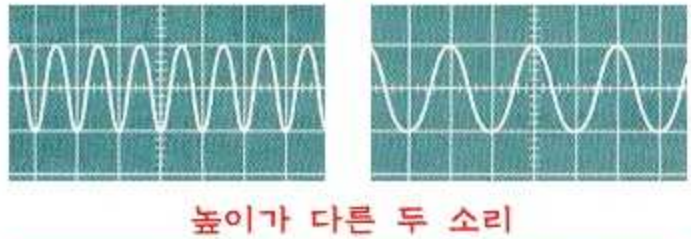
80개 token

# 소리의 3요소(3 elements of sound)

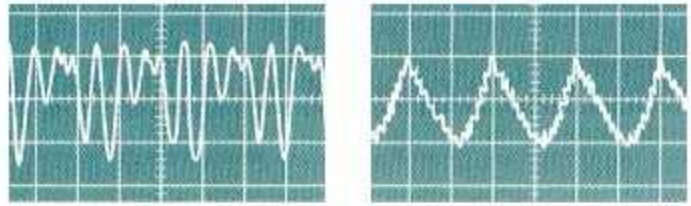
- 소리의 세기(loudness)
  - 소리의 세기는 물체가 진동하는 폭(진폭)에 의하여 정해지는데, 센(강한) 소리는 진폭이 크고, 약한 소리는 진폭이 작다. 그리고 소리의 세기가 변하더라도 진동수는 달라지지 않으며, 소리의 세기 단위로는 dB(데시벨)을 사용한다.
- 소리의 높이(음정, 고음/저음, frequency or pitch)
  - 소리의 높낮이는 음원의 진동수에 의해 정해지며, 진동수가 많을수록 높은 소리가 나며, 적을수록 낮은 소리가 난다. 그리고 단위는 진동수와 같은 단위인 Hz.
  - ✓ Pitch Extraction(Detection) Algorithm
- 소리의 음색(timbre, 맵시)
  - 소리의 맵시는 파형(파동의 모양)에 따라 구분된다.
  - 사람마다 목소리가 다른 것은 소리의 맵시가 다르기 때문이다.
  - MFCC는 음색의 특징을 잘 잡아낸다.



세기가 다른 두 소리



높이가 다른 두 소리



맵시가 다른 두 소리

**어떤 Feature를 사용할 것인가?**

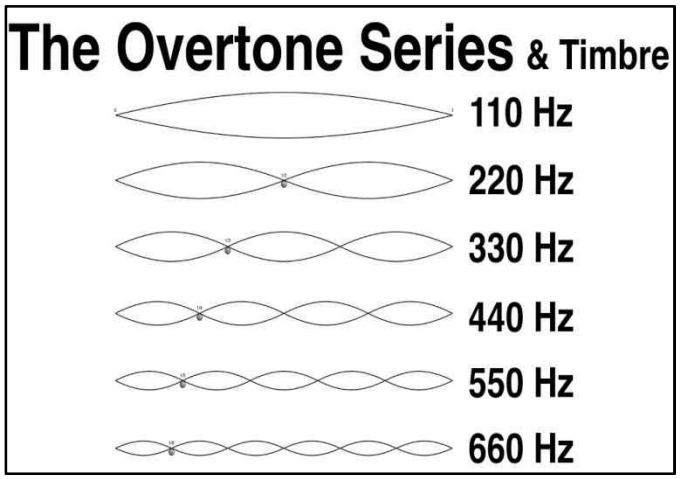
# MFCC(Mel-frequency cepstral coefficients)

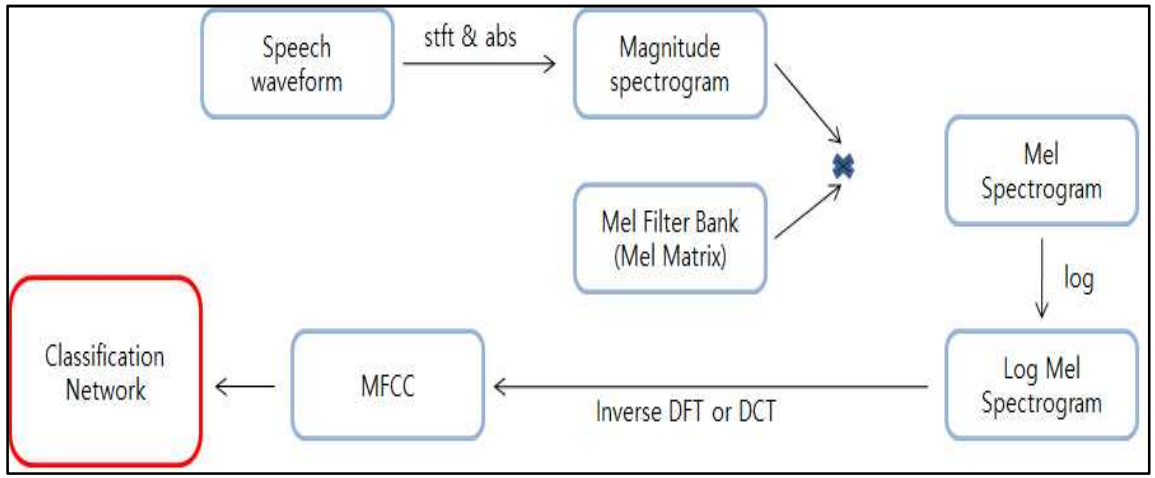
- 음색의 특징을 잘 나타낼 수 있는 feature.
- 음소(phoneme)을 구분한다.
- 배음(overtone, 밑음+부분음) 구조를 잘 파악한다.
- 음정의 차이는 무시 ➔ 악보를 그리는 데는 부적절.
- 악기의 소리 구분, 사람의 목소리 구분에 적합함.

순음 

배음 

- MFCC values are not very robust in the presence of additive noise, and so it is common to normalize their values in speech recognition systems to lessen the influence of noise.(wikipedia)
- Tacotron 모델에서는 MFCC보다는 Mel-Spectrogram을 활용한다.
  - MFCC는 만드는 과정에서 소리의 많은 정보를 잃어버린다. ➔ 복원을 고려해서 Mel-Spectrogram
  - Linear, Mel-Spectrogram, MFCC 모두 Phase(위상)에 대한 정보를 가지고 있지 않다. ➔ Griffin-Lim으로 복원



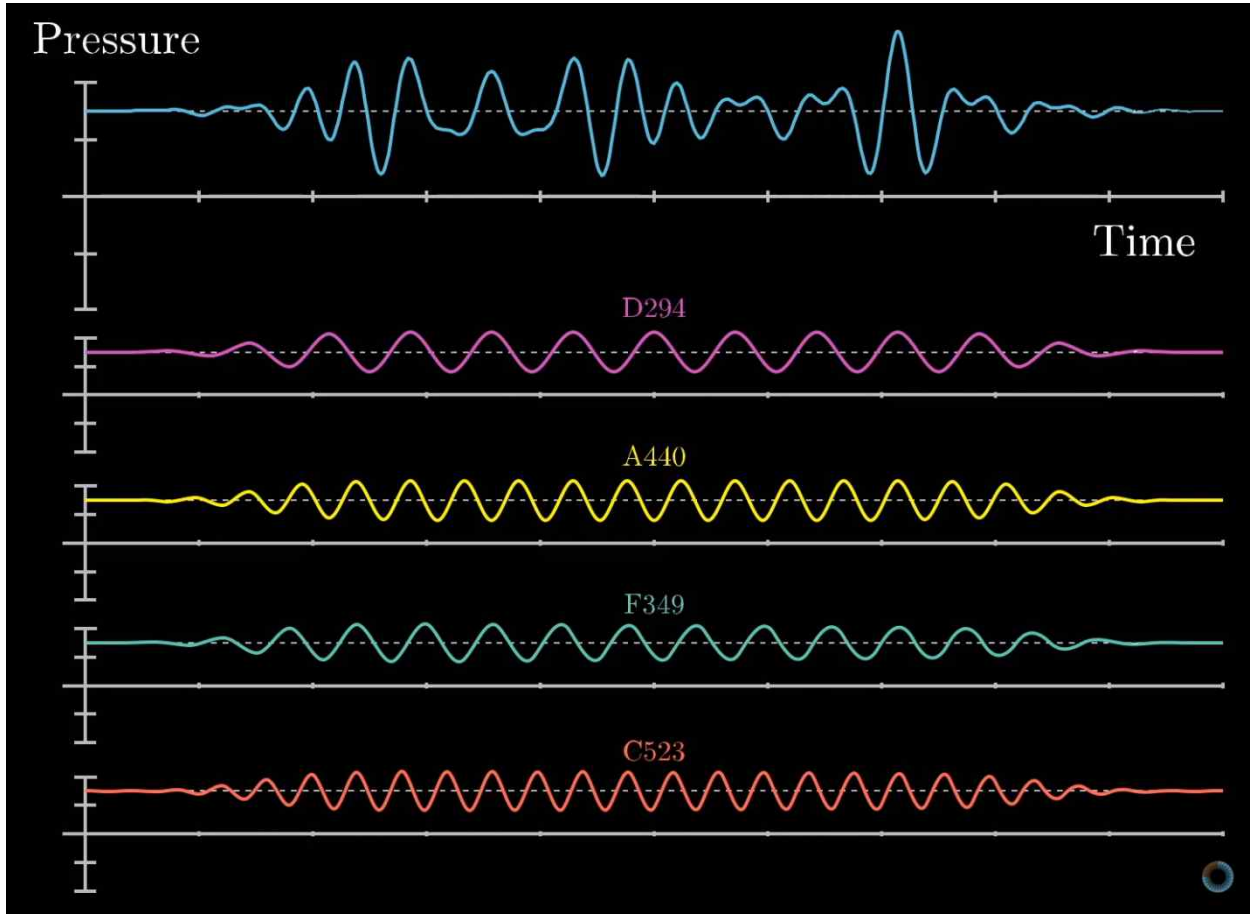



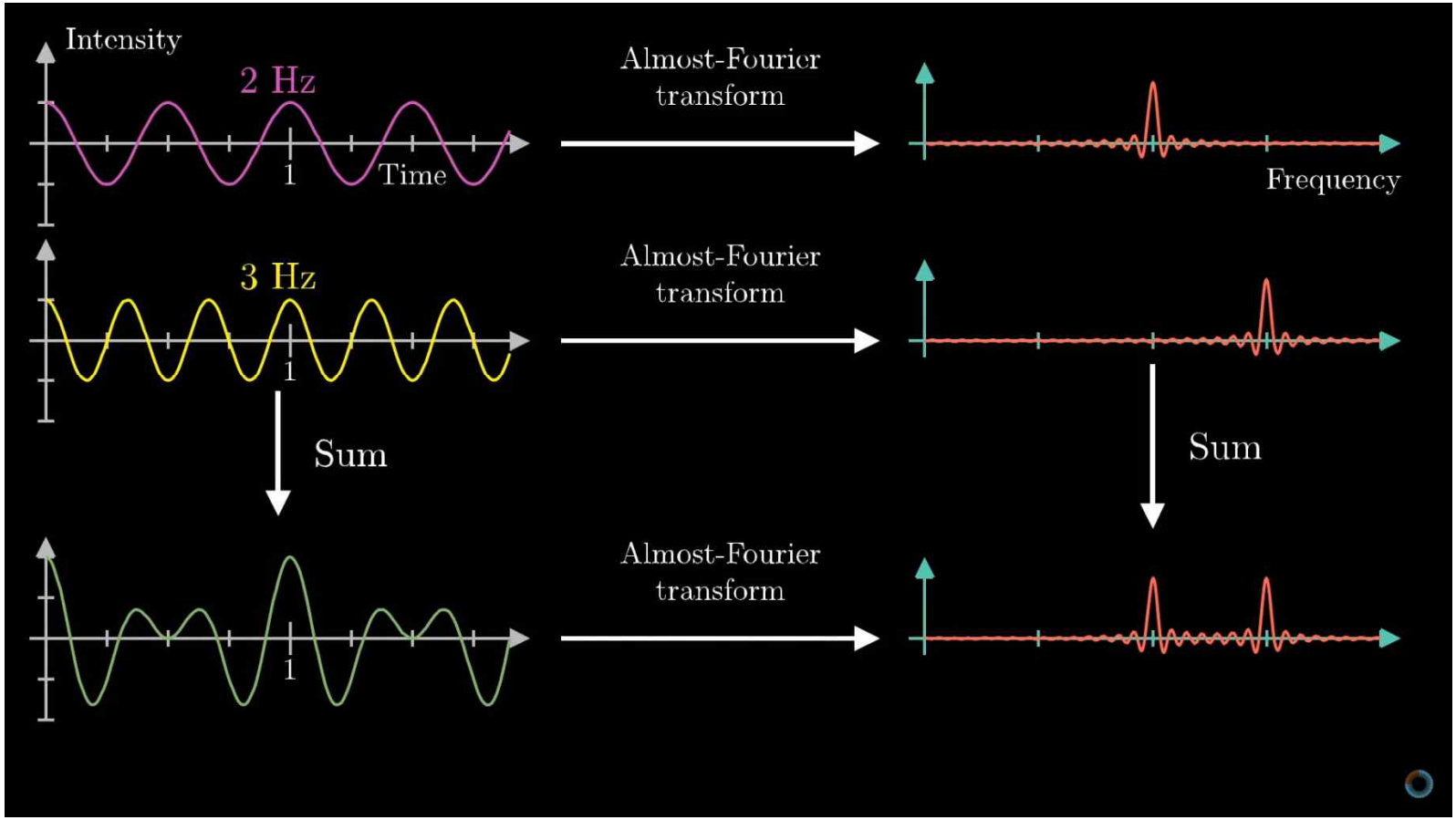
# Fourier Transform

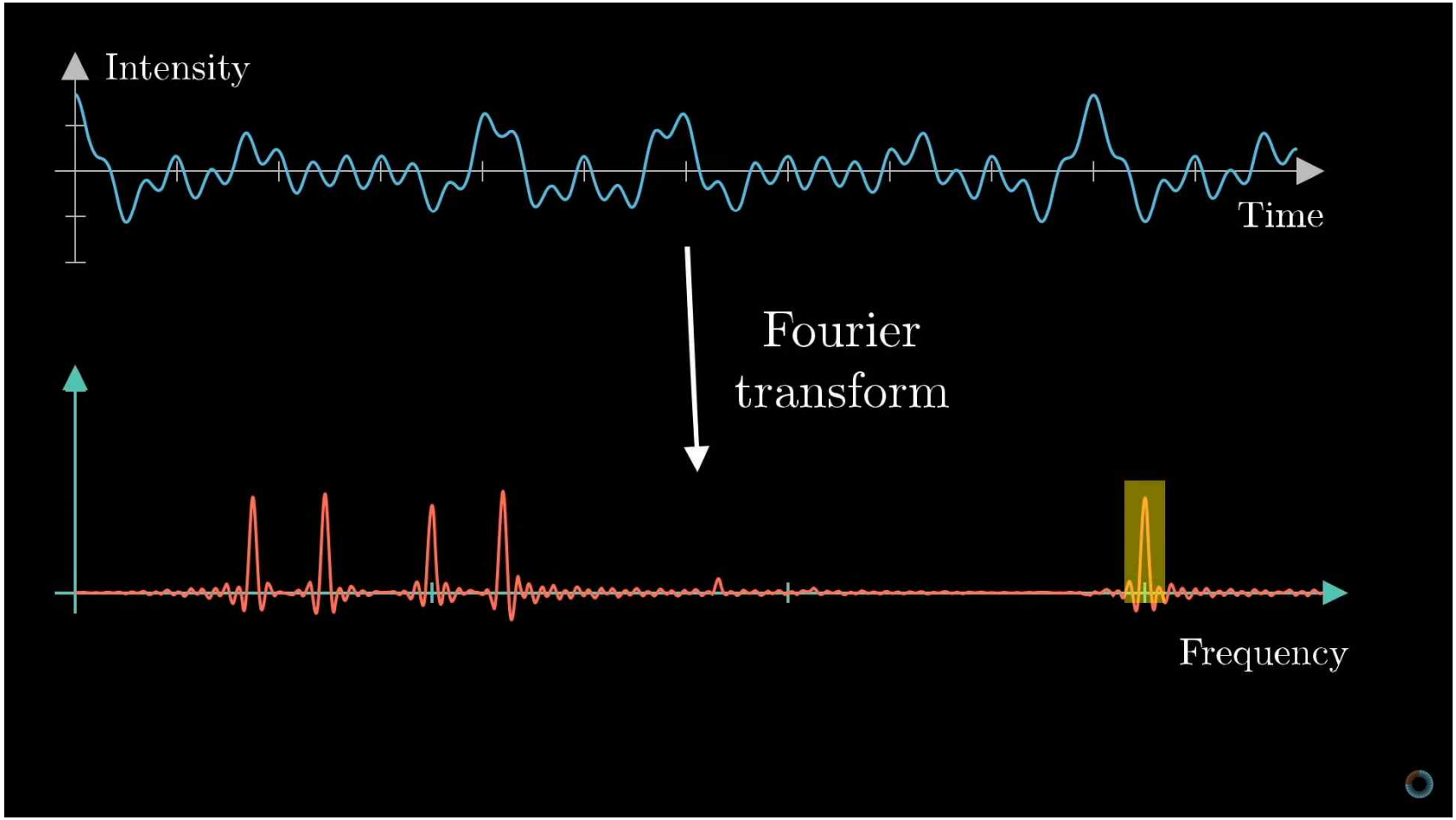
- 시간에 대한 함수 (or 신호)를 주파수 성분으로 분해하는 작업이다.
- 시간의 함수가 푸리에 변환이 되면, 주파수의 복소함수가  된다.
- 이것의 절대값은 원래 함수를 구성하는 주파수 성분의 양을, 편각은 기본 사인 곡선과의 위상차 (phase offset) 을 나타낸다.

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i x \xi}\, dx, \qquad (Eq.1)$$

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)\, e^{2\pi i x \xi}\, d\xi, \qquad (Eq.2)$$

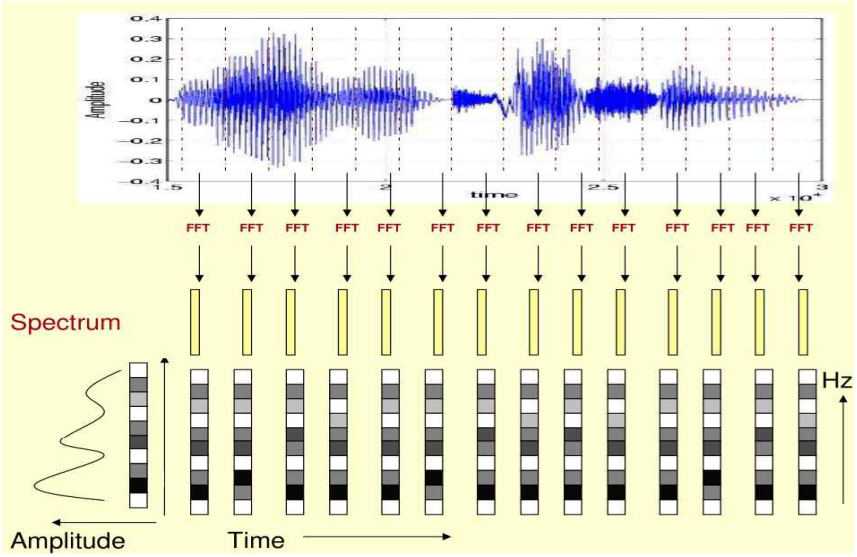


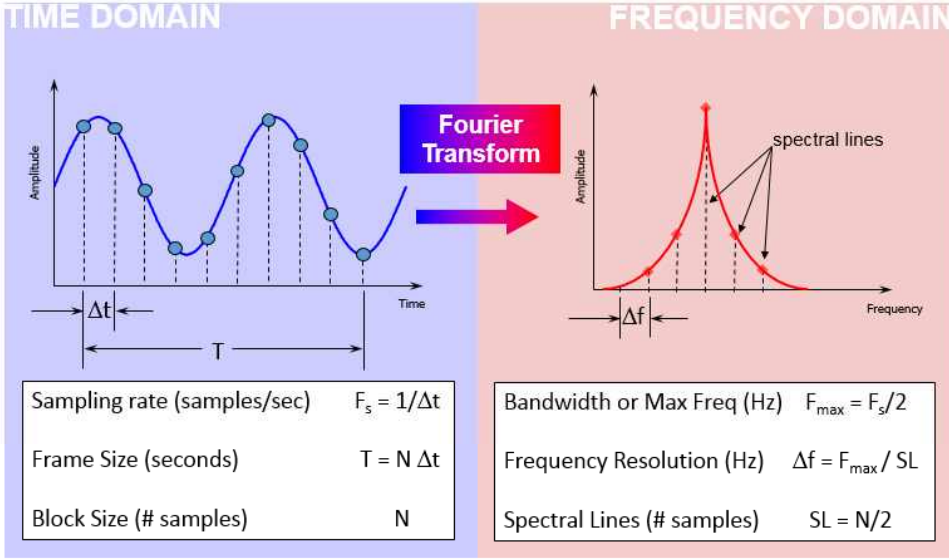

Intensity
2 Hz
1
Time
Almost-Fourier transform
Frequency
3 Hz
1
Almost-Fourier transform
Sum
Sum
1
Almost-Fourier transform




3Blue1Brown

# STFT(Short Time Fourier Transform)



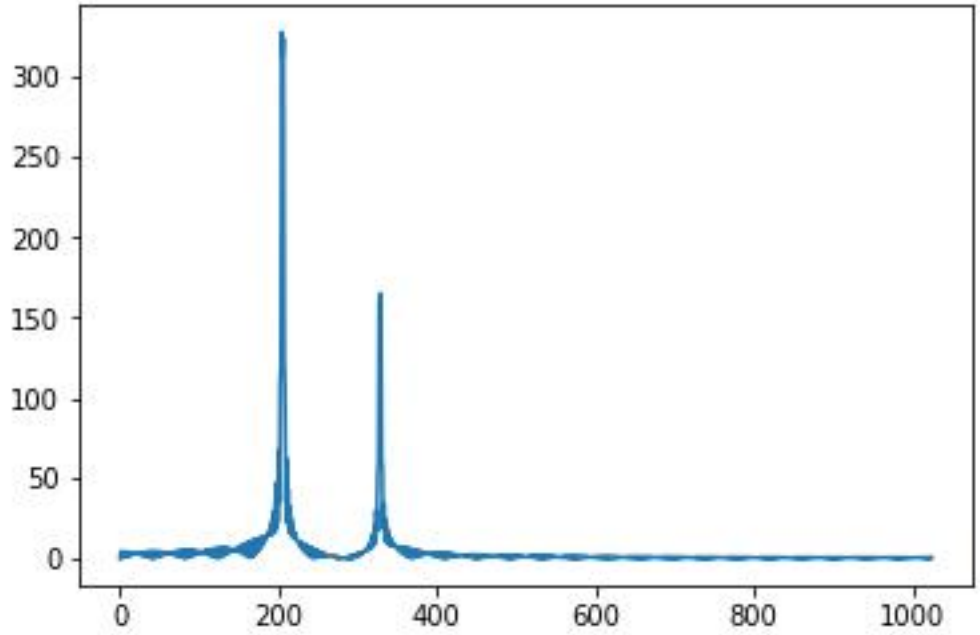

```
y = librosa.load(audio_clip, sampling_rate)

STFT(y, window_size, hop_size, fft_size)
```

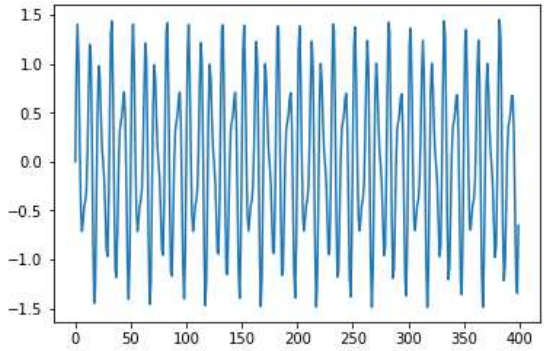
audio_clip(1D data) ➔ 2D data ( T, fft_size/2 +1)

e.g.
sampling_rate = 24000/1sec
window_size = 1200(=0.05sec)
hop_size = 300 ➔ 길이 결정
fft_size = 2048 ➔ output 크기 결정

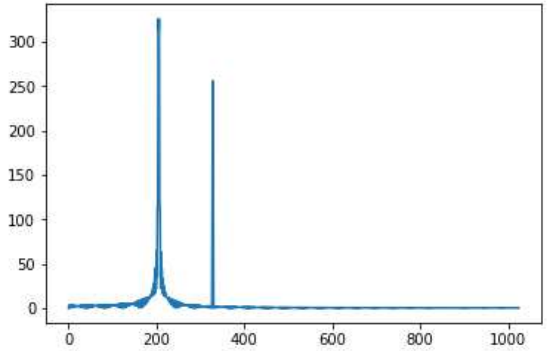
# STFT(Short Time Fourier Transform)

$$y = \sin(100 \times 2\pi x) + 0.5\sin(160 \times 2\pi x)$$



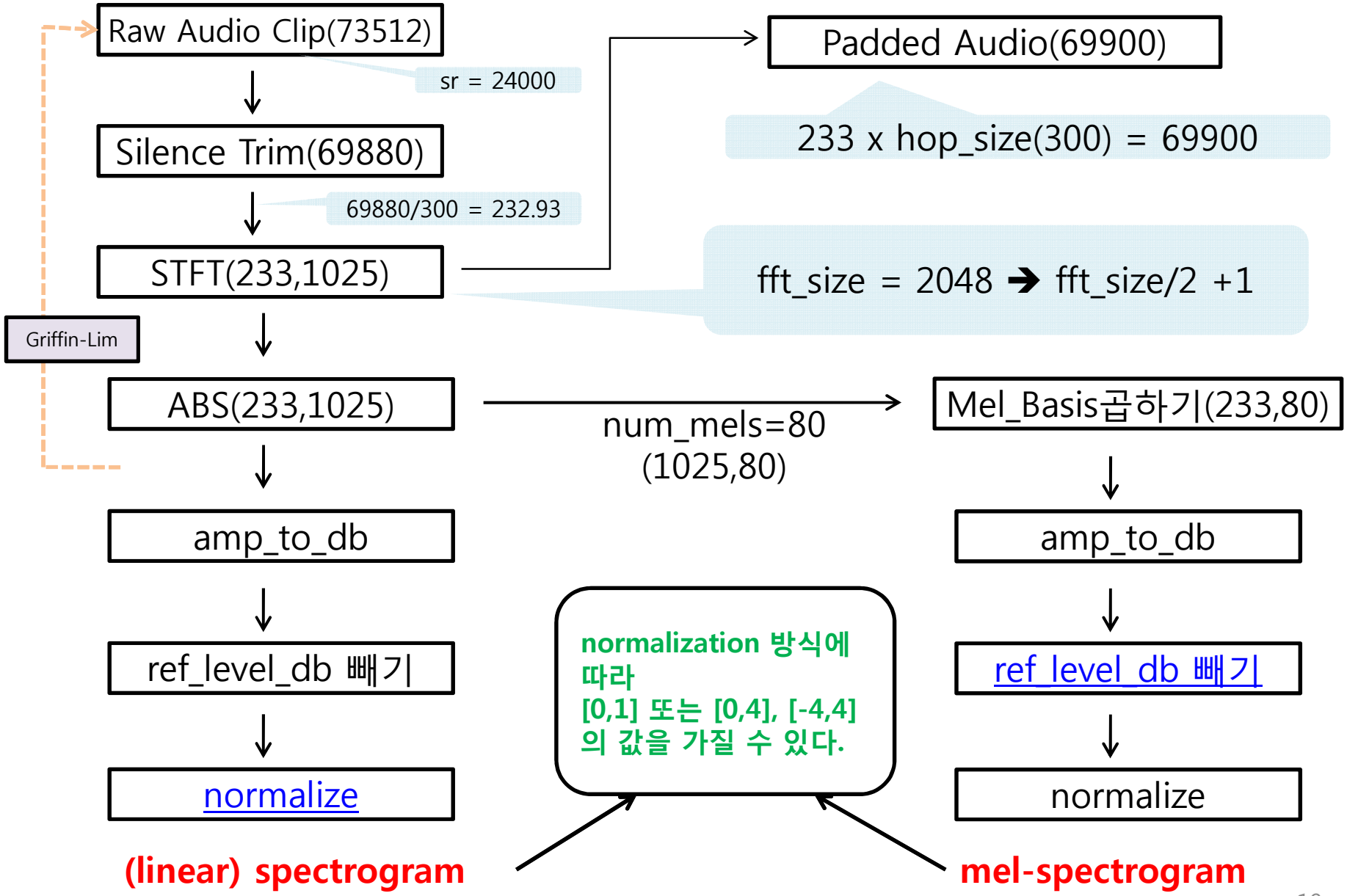

```
# STFT
import librosa
import matplotlib.pyplot as plt
N = 1000
T = 1.0 / N
x = np.linspace(0.0, N*T, N)    # 0~10 사이에 N(1000)개 sampling

y = np.sin(100.0 * 2.0*np.pi*x) + 0.5*np.sin(160.0 * 2.0*np.pi*x)
yf = librosa.stft(y,n_fft=2048)  # n_fft = 2048(default)
plt.plot(np.abs(yf))
plt.show()
```




$$y = \sin(100 \times 2\pi x) + 0.5\cos(160 \times 2\pi x)$$

n_fft가 output 크기 결정

# From Audio To Mel-spectrogram



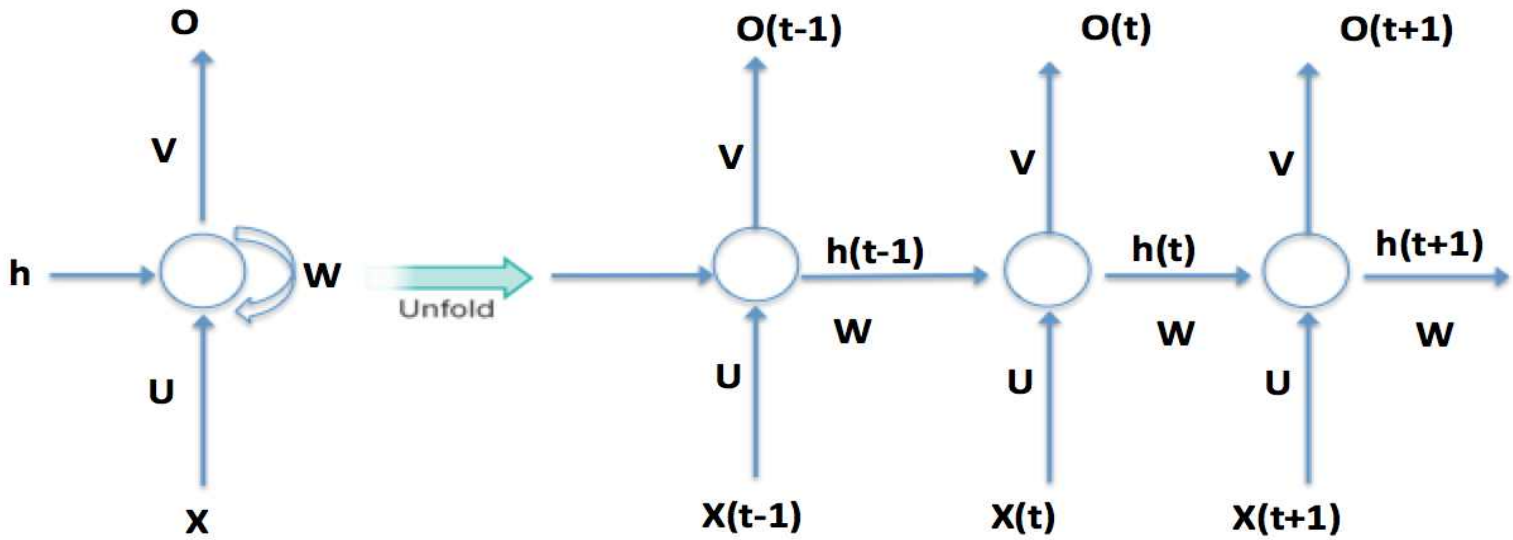
# Mini-Batch Data 생성

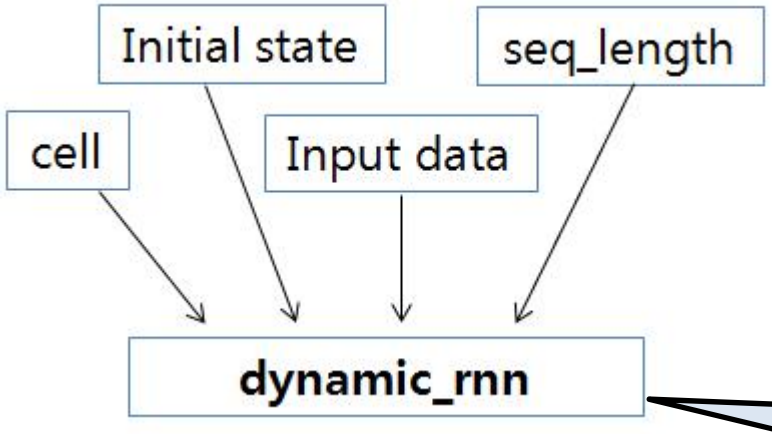
- (**audio**, **text**, **mel-spectrogram**, linear-spectrogram, **tokens**, …) 묶어 npz로 미리 만들어 놓는다.
- DataFeeder Class를 만들어, training 할 때 data를 공급한다.
  - Mini-Batch(N개. e.g. 32) data를 적당한 개수(M개. e.g. 32) 만큼 만들어 Queue에 쌓는 방식을 사용.
  - N x M 개의 data를 길이로 정렬 후, N개씩 나누어 공급한다

    ➔ **padding 최소화**
  - (input_data, input_length, mel_target, linear_target, speaker_id)
  - Speaker별로 feed되는 data의 비율이 동일하게 처리
  - hyper parameter가 바뀌면 data를 새로 만들어야 함. (eg. hop_size)

# 2. RNN, Attention

# Tensorflow BasicRNNCell/dynamic_rnn



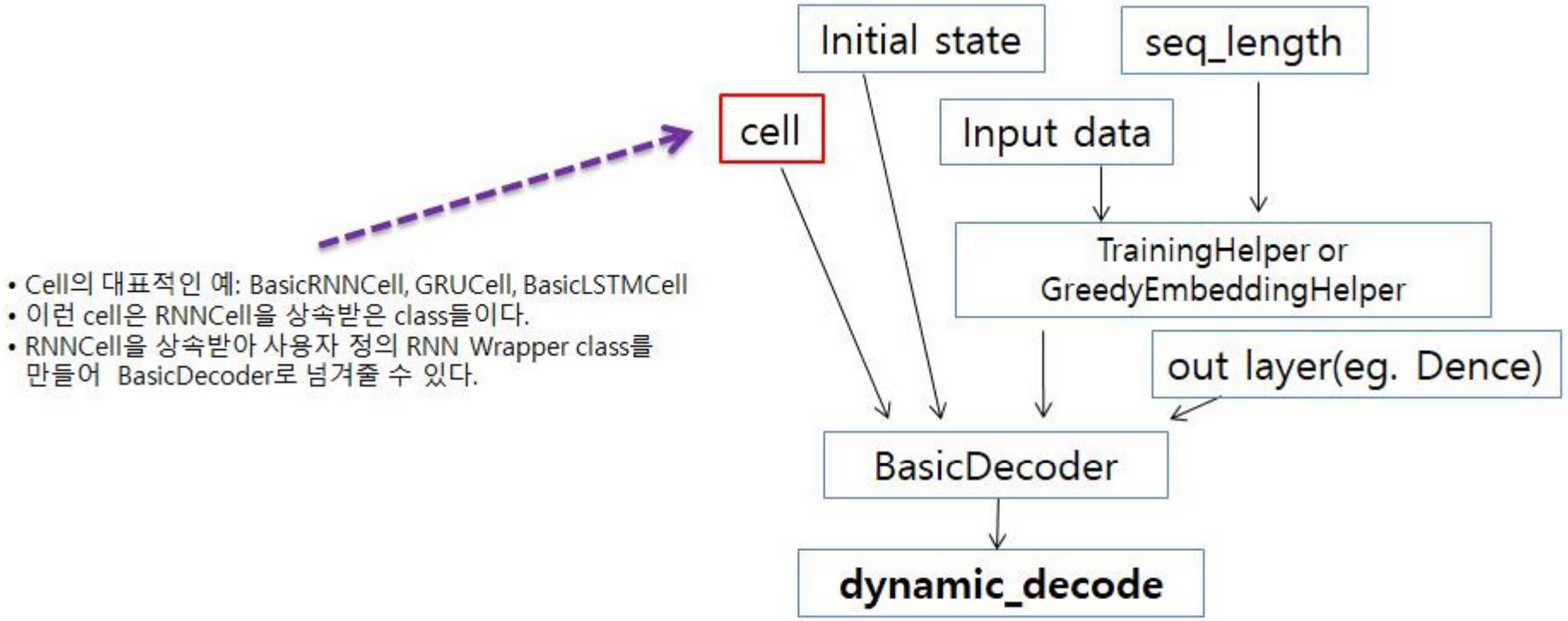

$$h_t = \tanh\left(x_t U + h_{t-1} W + b\right)$$
$$= \tanh\left([x_t | h_{t-1}] Y + b\right)$$

Tensorflow 내부에서는 U,W가 각각 잡히지 않고, 묶어서 Y 하나만 잡힌다.

dynamic_rnn은 teacher-forcing으로 구현은 가능하나, free-running(inference)에는 불편한 점이 많다.

Sample Code(C) 참조

# Tensorflow dynamic_decode




- Cell의 대표적인 예: BasicRNNCell, GRUCell, BasicLSTMCell
- 이런 cell은 RNNCell을 상속받은 class들이다.
- RNNCell을 상속받아 사용자 정의 RNN Wrapper class를 만들어 BasicDecoder로 넘겨줄 수 있다.

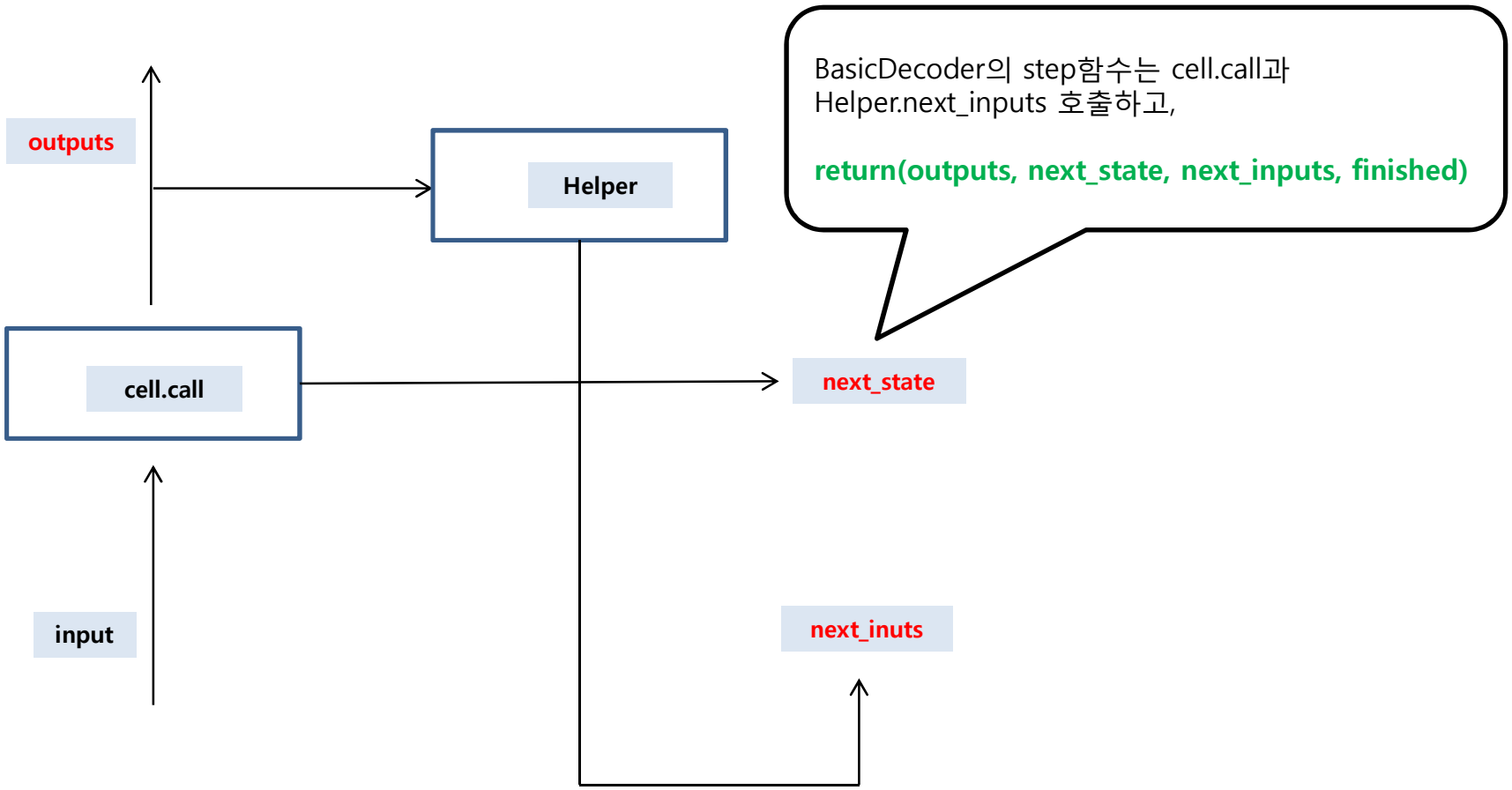
[Sample Code(A) 참조]

**[중요]**
cell, Helper, BasicDecoder를 customization할 수 있어야 Tensorflow에서 API가 제공되지 않더라도, 원하는 모듈을 구현할 수 있다.

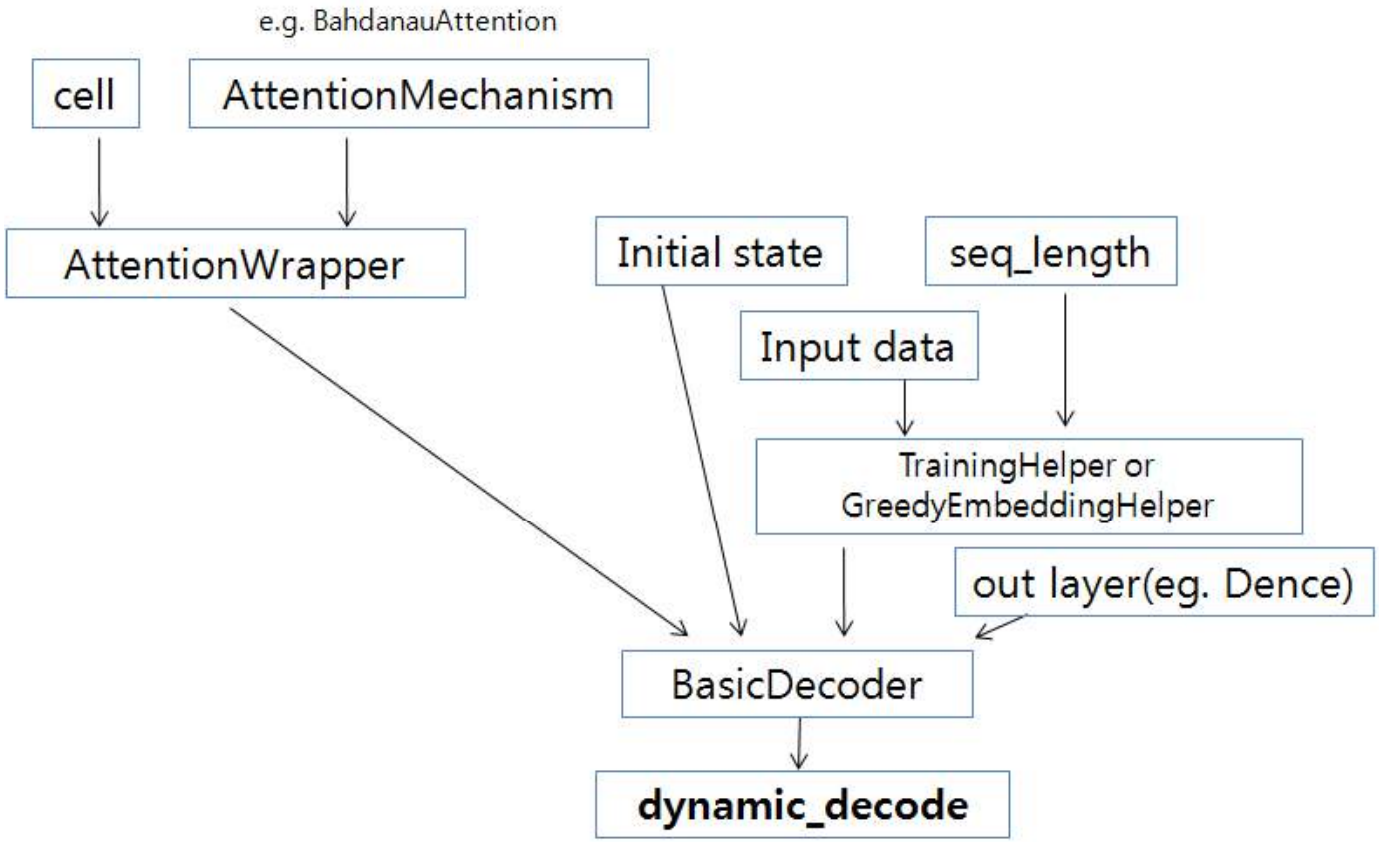
참고 자료: Customization Tutorial

# BasicDecoder

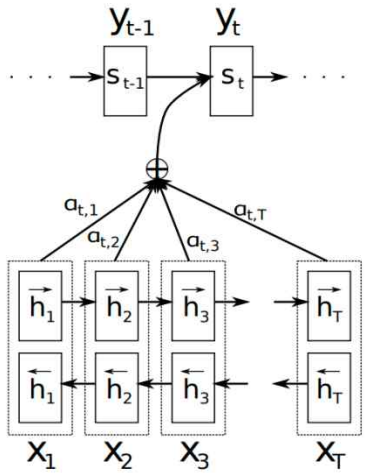
- BasicDecoder의 step함수는 cell, Helper를 결합하여 **(outputs, next_state, next_inputs, finished)**를 return한다.
- cell이나 Helper가 customization되어 있다면, BasicDecoder도 customization해야 한다.



# Tensorflow AttentionMechanism



# Attention




- score를 계산하는 과정에 encoder hidden state, decoder hidden sate 모두 반영된다.
- Bahdanau Attention과 Luong Attention은 score 계산방식에서만 차이가 있다.

용어:
- score
- alignment
- context
- attention

For decoder time step i

- $[h_1, h_2, \cdots, h_{T_e}]$ : encoder hidden state, $h_i \in R^{eh}$
- $s_i$ : decoder hidden state, $s_i \in R^{dh}$

$\rightarrow e_i = [e_{i1}, e_{i2}, \cdots, e_{iT_e}]$ : score

$\rightarrow$ softmax

$\rightarrow$ alignment(weight), $a_{ij} \in R$

$\rightarrow c_i = \sum_{j=1}^{T_e} a_{ij} h_j$ : *context*

$\rightarrow$ attention vector

# Bahdanau & Luong Attention

2: Luong
3: Bahdanau

$h_j$: encoder hidden state, $s_i$: decoder hidden state

$$score(s_i, h_j) = e_{ij} = \begin{cases} h_j s_i^T & \text{1. dot product attention} \\ (h_j Y_a) s_i^T & \text{2. general dot product attention} \\ v_a^T \tanh(Y_a [s_i | h_j]^T) & \text{3. concat(Bahdanau), additive attention} \\ = v_a^T \tanh\left([W_q^T | W_m^T] \begin{bmatrix} s_i^T \\ h_j^T \end{bmatrix}\right) & \\ = \tanh\left([s_i | h_j] \begin{bmatrix} W_q \\ W_m \end{bmatrix}\right) v_a & \\ = \tanh\left(s_i W_q + h_j W_m\right) v_a & \end{cases}$$

**_bahdanau_score**

$$e_i = [e_{i1}, \cdots, e_{iT_e}]$$

$$\alpha_i = softmax(e_i) = [\alpha_{i1}, \cdots, \alpha_{iT_e}] \leftarrow \text{alignment}$$

alignment를 AttentionMechanism에서 계산하고, 그 결과를 받아서 AttentionWrapper내에서 `_compute_attention`를 이용하여 최종 attention을 계산한다.

$$c_i = \sum_{j=1}^{T_e} \alpha_{ij} h_j \leftarrow \text{context}, N \times N_{eh}$$

# Tensorflow-AttentionMechanism

- Bahdanau Attention ➔ tf.contrib.seq2seq.BahdanauAttention
- Luong Attention ➔ tf.contrib.seq2seq.LuongAttention

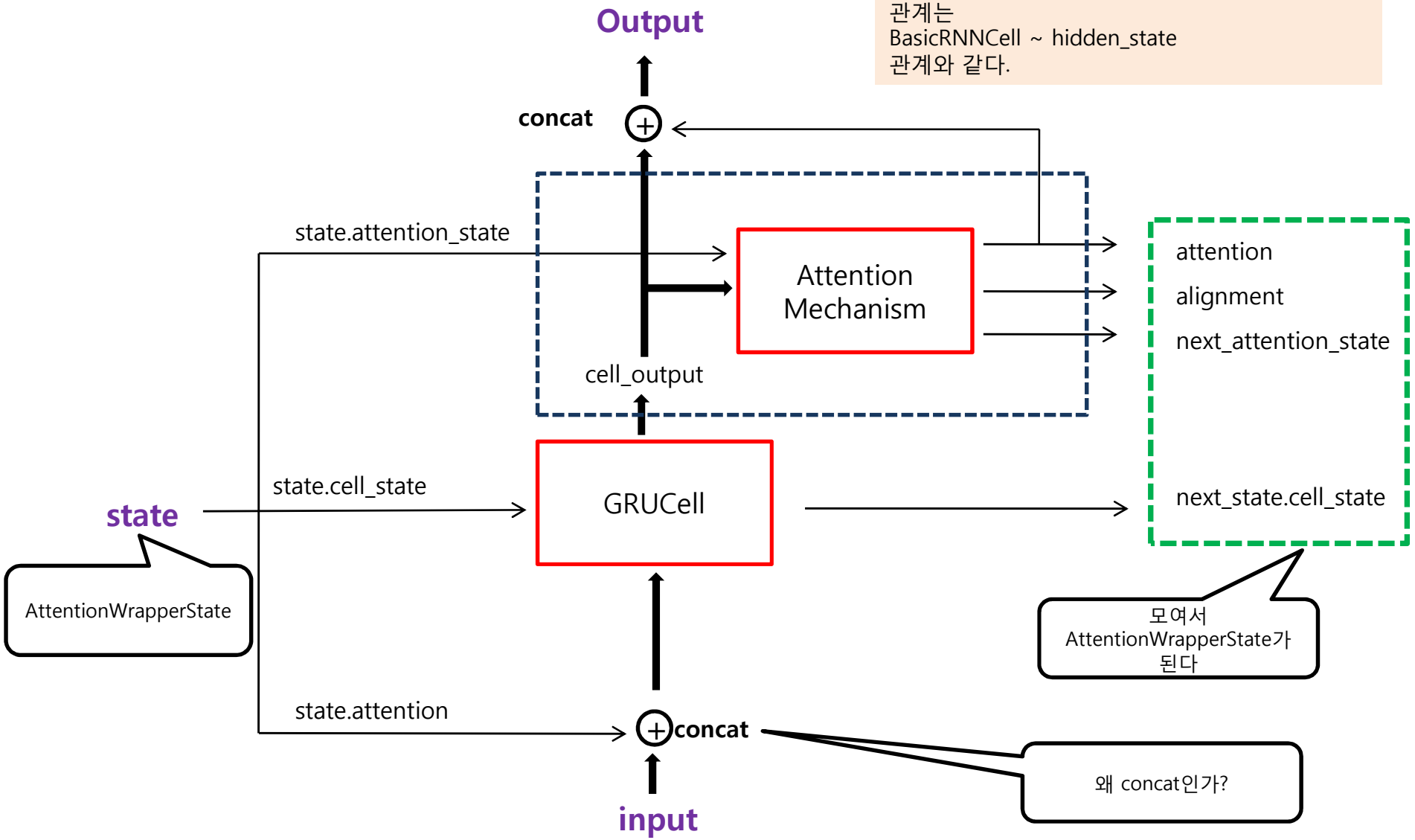
```
  attention_mechanism
= tf.contrib.seq2seq.BahdanauAttention(num_units=11,
                                       memory=encoder_outputs,
                                       memory_sequence_length=input_lengths)
```

## TIP

memory = encoder hidden state ➔ memory_layer ➔ key
query = decoder hidden state ➔ query_layer ➔ processed_query

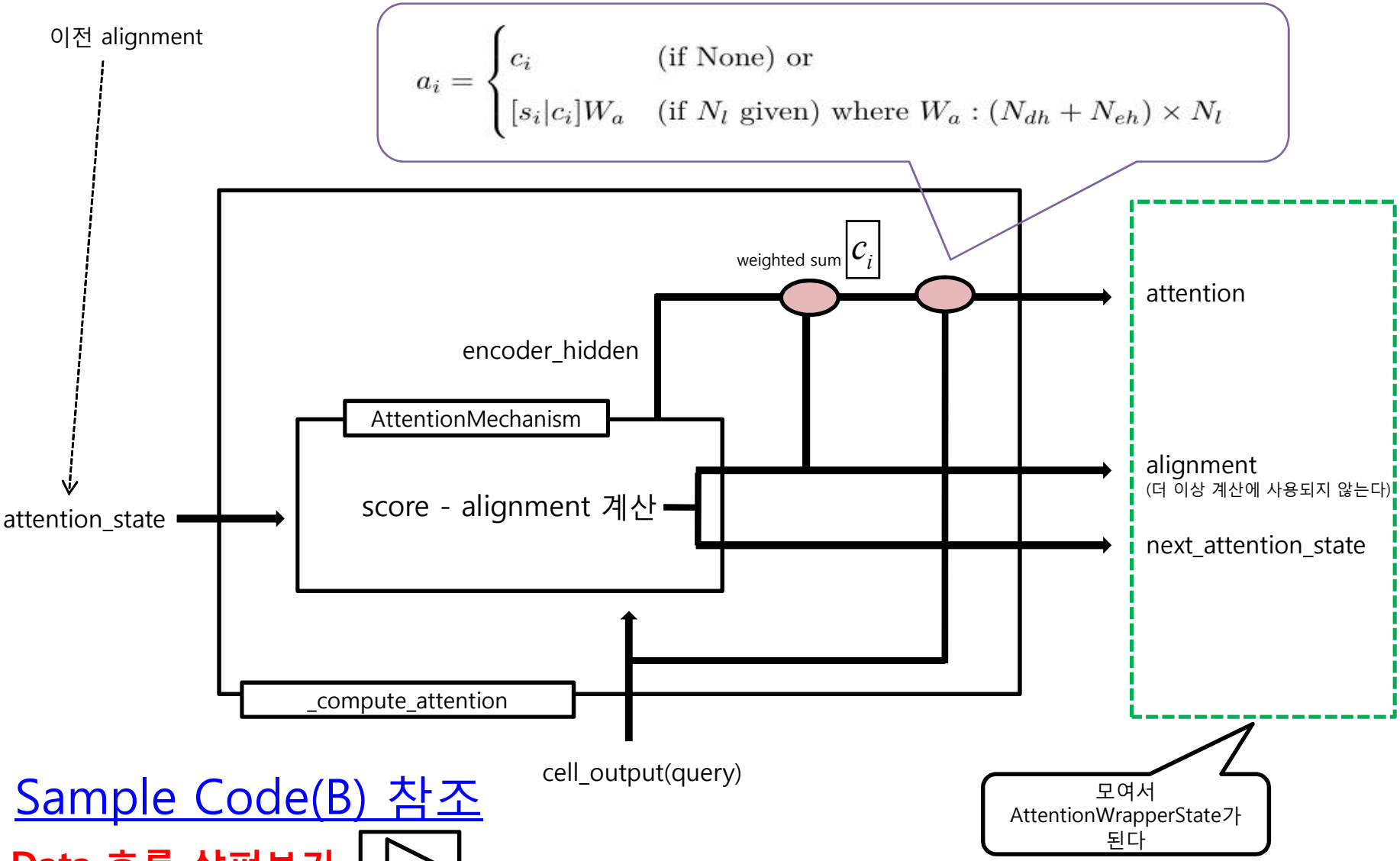
# AttentionWrapper & AttentionWrapperState

AttentionWrapper ~ AttentionWrapperState
관계는
BasicRNNCell ~ hidden_state
관계와 같다.



# AttentionWrapper

tf.contrib.seq2seq.AttentionWrapper(cell, attention_mechanism, attention_layer_size=13)




Sample Code(B) 참조

**Data 흐름 살펴보기** ▷

# AttentionWrapper

AttentionWrapper(cell,attention_mechanism)

★ self.__init__(**cell**, **attention_mechanism**)

★ self.call( input, state(AttentionWrapperState) )

cell은 BasicRNNCell, BasicLSTMCell같은 RNNCell이며, reture되는 cell_output과 next_cell_state는 같은 값이다.

```
cell_output, next_cell_state = cell(input,state.cell_state)
```

cell_state는 보통의 hidden_state

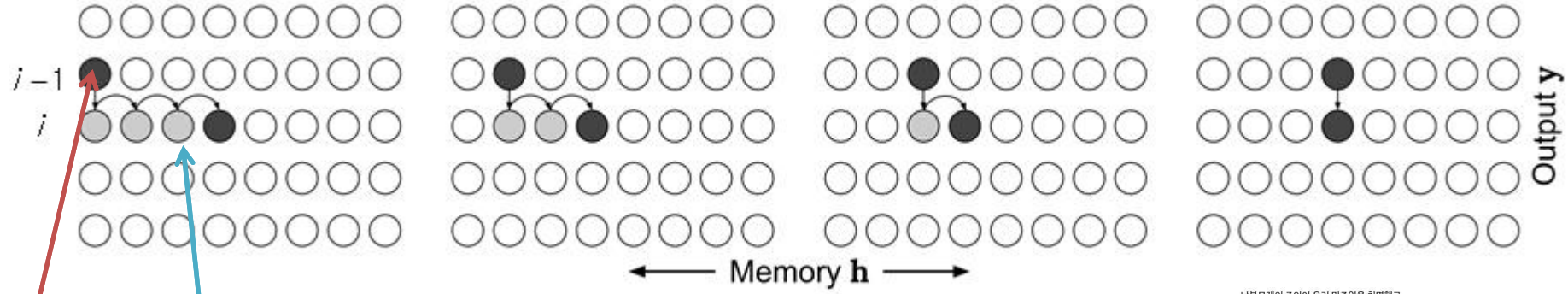
```
_compute_attention(attention_mechanism, cell_output,state.attention_state)
    alignments, next_attention_state = attention_mechanism(cell_output,state.attention_state)
    ...
    return attention, alignments, next_attention_state
...
return cell_output, next_state(AttentionWrapperState)
```

alignment와 next_attention_state는 같은 값이다.

```
next_state = AttentionWrapperState(
cell_state = next_cell_state,
attention = attention,
attention_state = next_attention_state,
alignments= alignments)
```

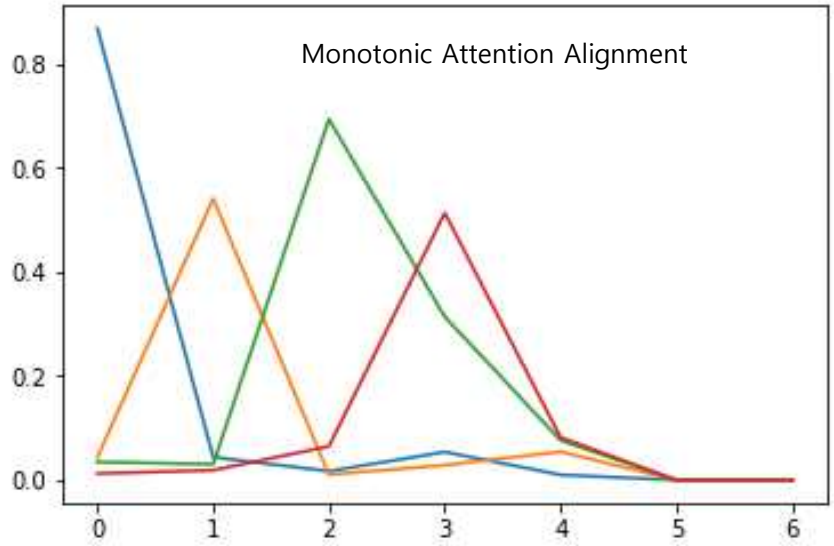
# MonotonicAttention

- tf.contrib.seq2seq. BahdanauMonotonicAttention
- tf.contrib.seq2seq. LuongMonotonicAttention

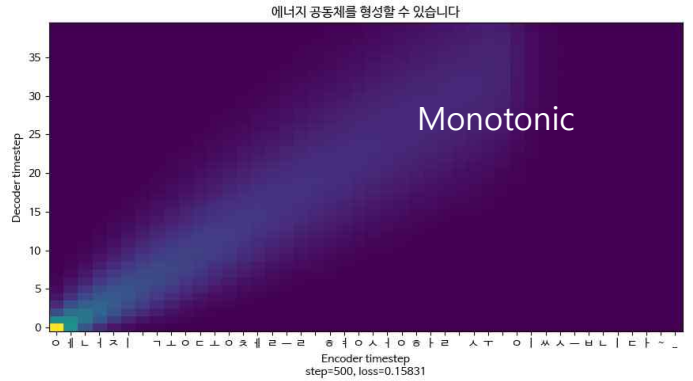



이전 alignment

score





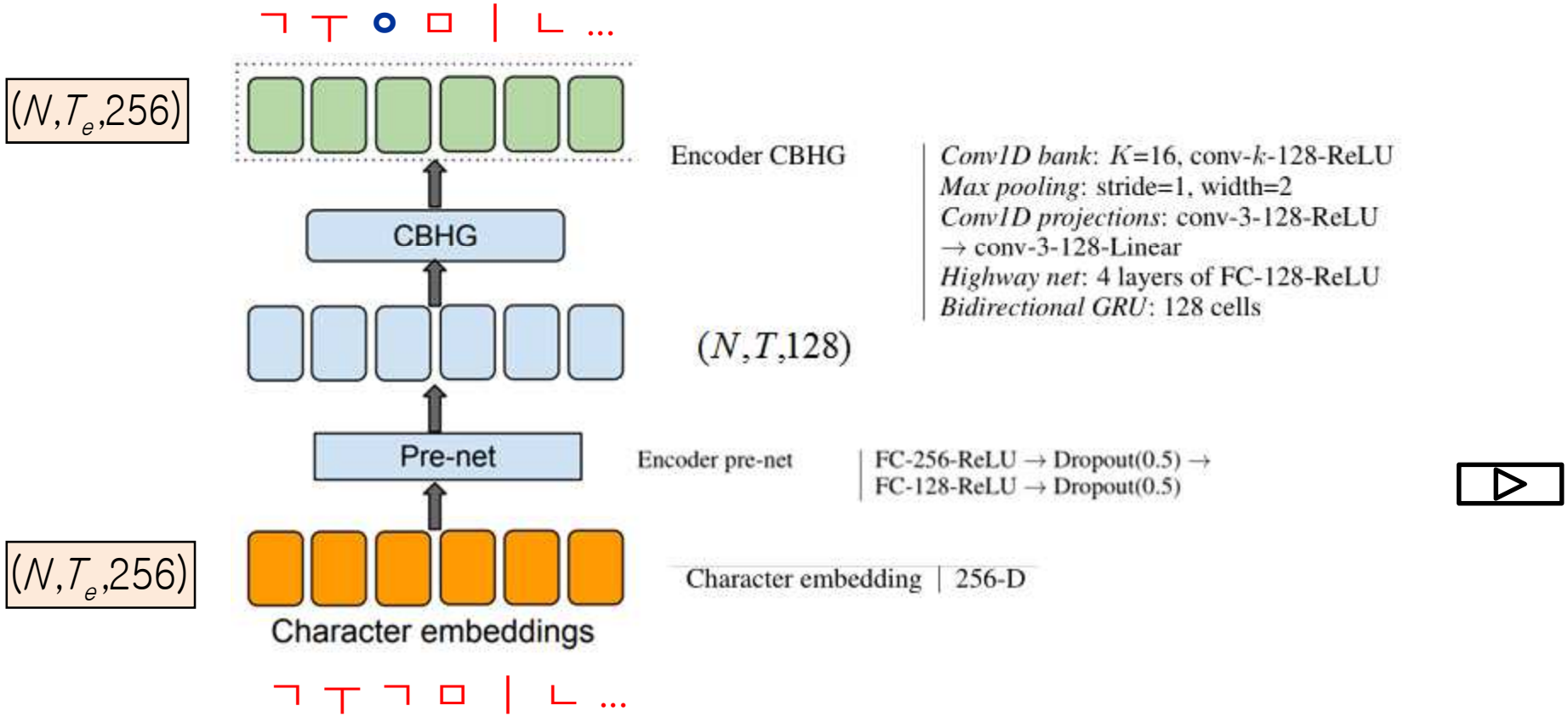



BahdanauMonotonicAttention(hp.attention_size, encoder_outputs,memory_sequence_length = input_lengths, normalize=True)

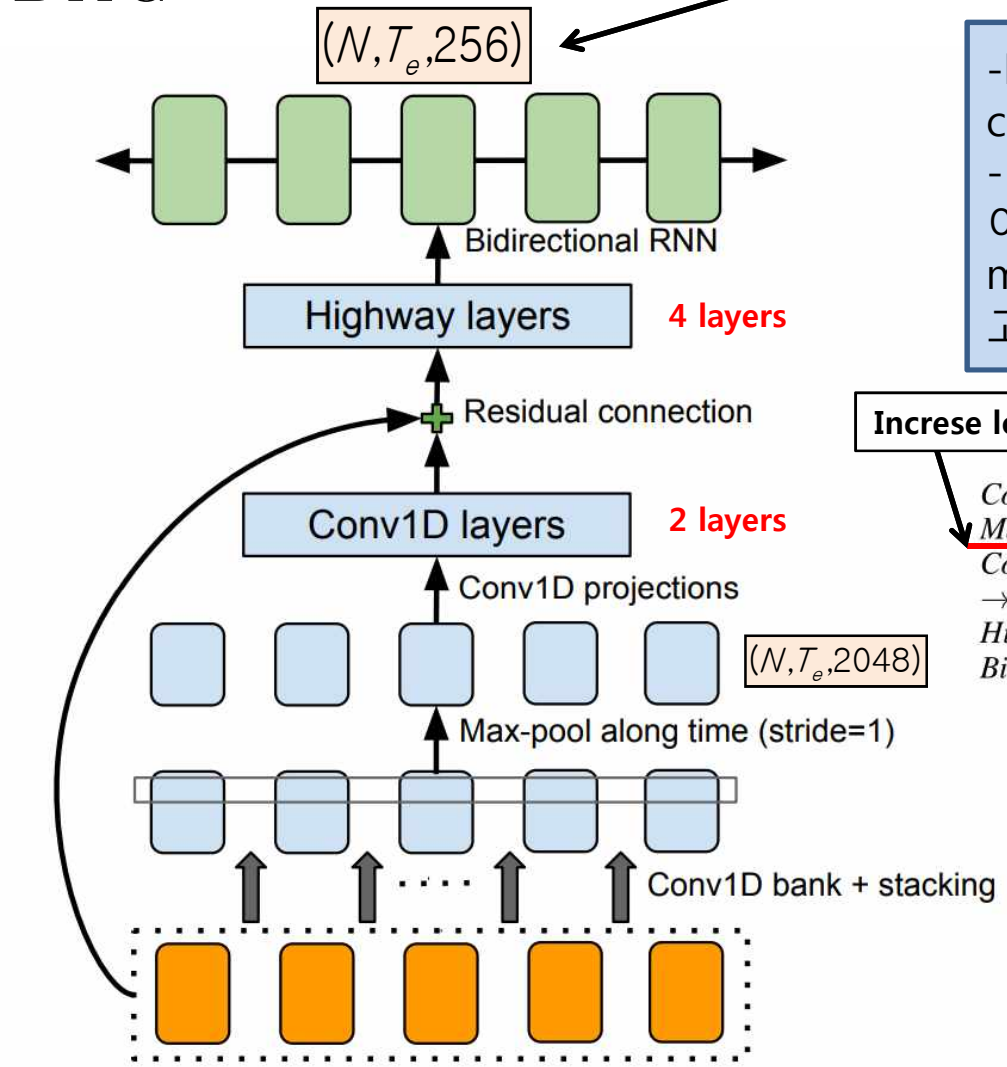
# 3. Tacotron 분석

# Encoder

'국민과 함께하는'
➔ ['ㄱ', 'ㅜ', 'ㄱ', 'ㅁ', 'ㅣ', 'ㄴ', 'ㄱ', 'ㅘ', ' ', 'ㅎ', 'ㅏ', 'ㅁ', 'ㄲ', 'ㅔ', 'ㅎ', 'ㅏ', 'ㄴ', 'ㅡ', 'ㄴ']
➔ [2, 34, 42, 8, 41, 45, 2, 30, 79, 20, 21, 57, 3, 26, 20, 21, 4, 39, 45]
➔ Character embedding



# CBHG



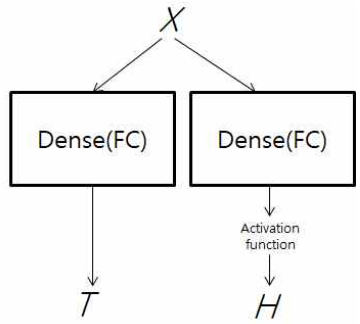

**Bidirectional RNN이므로, 128 x 2 = 256**

-**Batch normalization** is used for all convolutional layers.
- 논문에서는 CBHG가 overfitting을 줄이고, 보통의 RNN보다 mispronunciation을 적게 만들어 낸다고 하고 있다.

**Increse local invariances**

*Conv1D bank*: $K$=16, conv-$k$-128-ReLU
*Max pooling*: stride=1, width=2
*Conv1D projections*: conv-3-128-ReLU
→ conv-3-128-Linear
*Highway net*: 4 layers of FC-128-ReLU
*Bidirectional GRU*: 128 cells

**Kernel_size=3, filters=128**

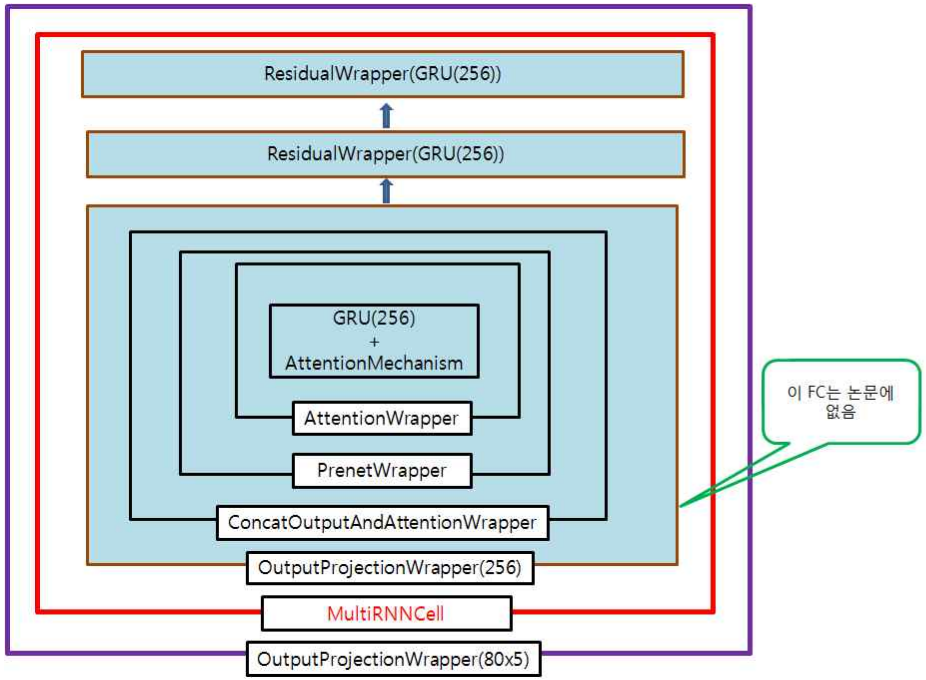


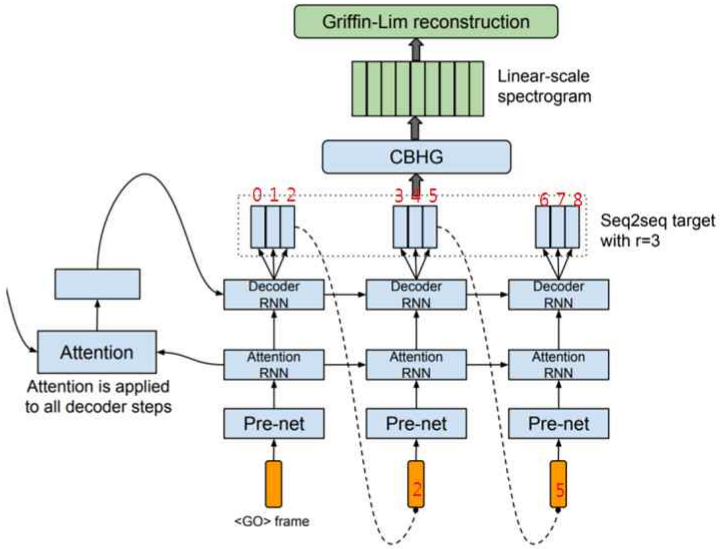

$\Rightarrow \sigma(T)H + (1-\sigma(T))X$

**Highway Net**

Convolution Bank Highway GRU

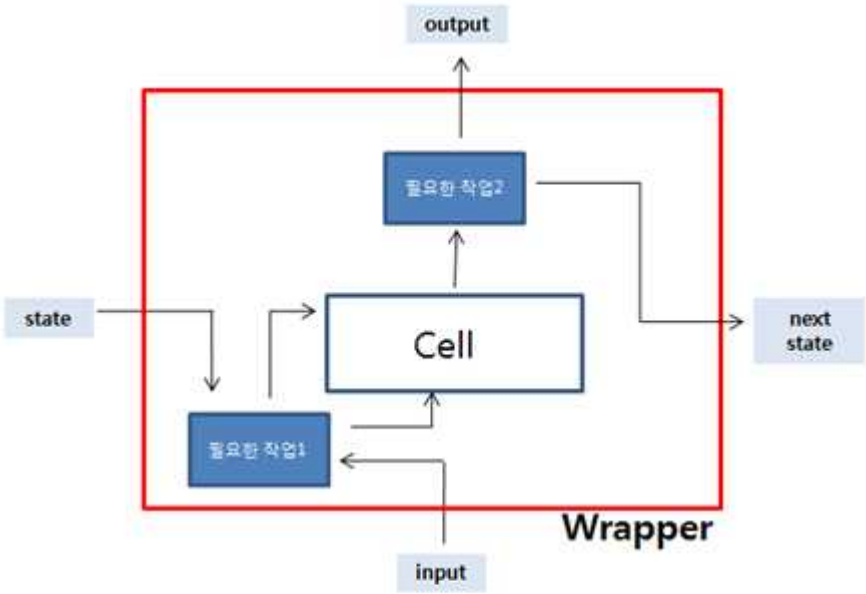
# Decoder




- ConcatOutputAndAttentionWrapper: [attention(256),output(256)] concat
  ➔ RNNWrapper Class를 만들 수 있어야 한다.

# Tensorflow RNNCell-Wrapper 활용




```
class MyBasicRNNWrapper(RNNCell):
    # property(output_size, state_size) 2개와 call을 정의하면 된다.
    def __init__(self,cell,name=None):
        super(MyBasicRNNWrapper, self).__init__(name=name)
        self.cell = cell
    @property
    def output_size(self):
        return self.cell.output_size
    @property
    def state_size(self):
        return self.cell.state_size

    def call(self, inputs, state):
        # 필요한 작업1: inputs, state를 이용하여 필요한 작업을 수행하여 self.cell에 넘겨줄 새로은 inputs, state를 만든다.
        cell_output, next_state = self.cell(inputs,state)
    # 필요한 작업2: self.cell이 return한 cell_outpus, next_state를 가공하여 return 값을 만든다.
        return cell_output, next_state
```

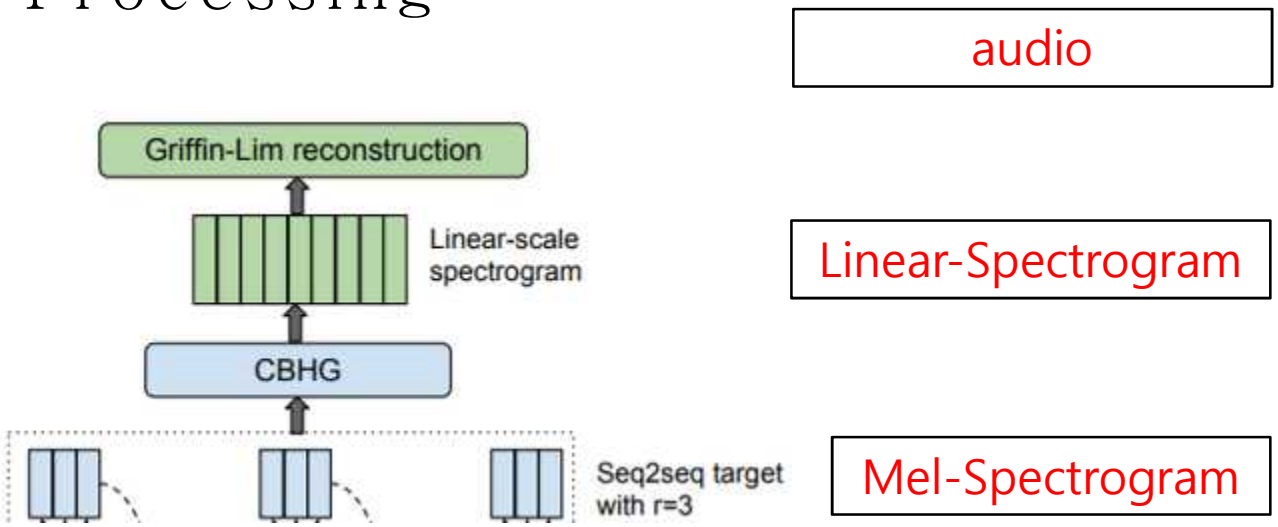
# Tensorflow RNNCell-Wrapper 활용

- Tensorflow에 구현되어 있는 RNNWrapper class
  - OutputProjectionWrapper
  - InputProjectionWrapper
  - ResidualWrapper

```
def call(self, inputs, state):
        cell_output, next_state = self.cell(inputs,state)
        cell_output = inputs + cell_output  # residual rnn
        return cell_output, next_state
```

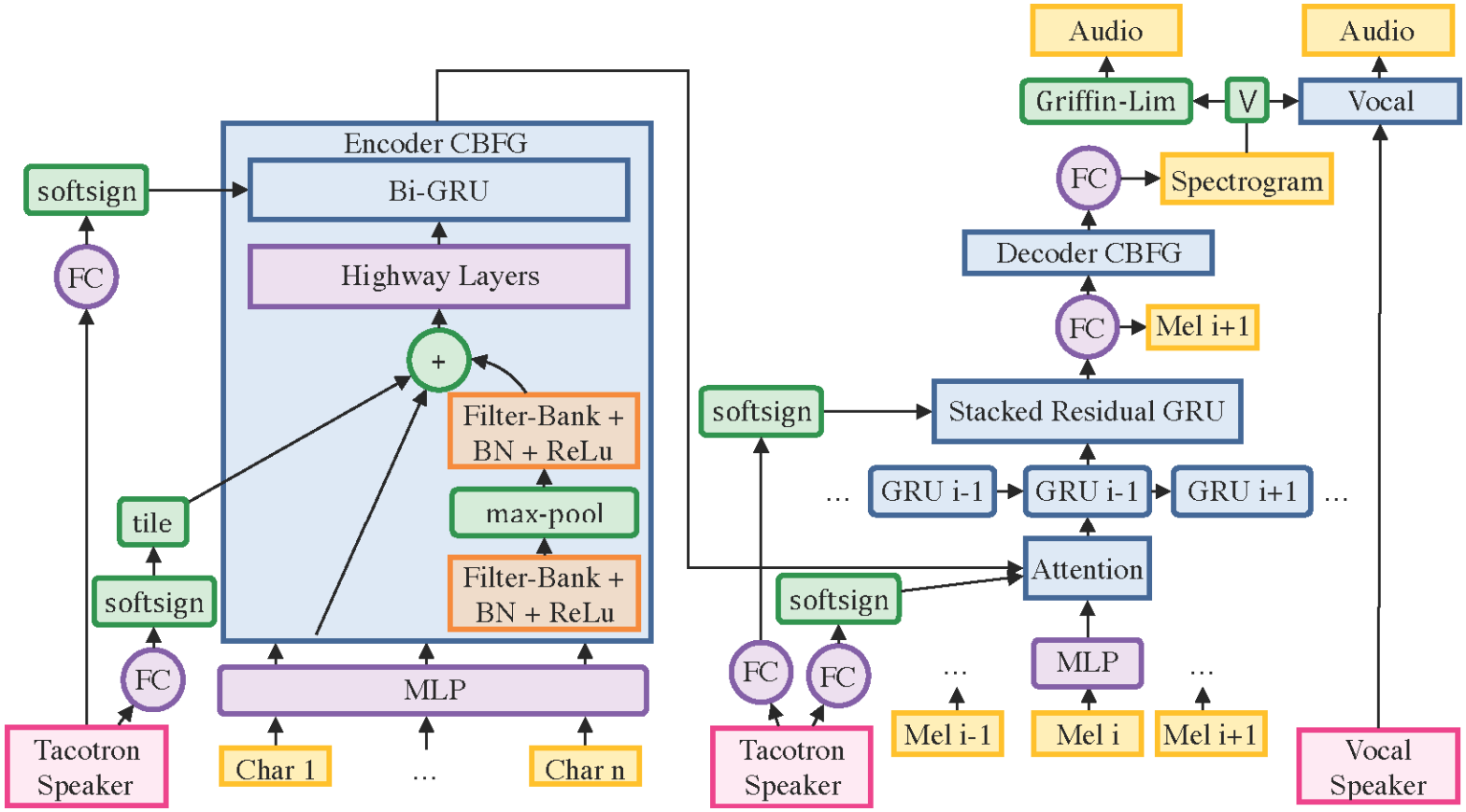
```
def call(self, inputs, state):
        fc_outputs = tf.layers.dense(inputs,units=5,name='myFC')  # FC layer
        cell_output, next_state = self.cell(fc_outputs,state)
        cell_output = inputs + cell_output  # residual rnn
        return cell_output, next_state
```

# Post-Processing




- Mel-Spectrogram을 CBHG layer에 넣어, Linear-Spectrogram을 만든다.
- 이 CBHG layer는 encoder에 있는 CBHG와 같은 layer지만, hyper parameter가 다르다.
- CBHG를 training할 때, 어떤 Mel-Spectrogram을 입력으로 사용할 것인가?
  - ➢ **이전 단계에서 만들어낸 Mel-Spectrogram** vs **Ground Truth Mel-Spectrogram**
- Tacotron 모델이 최종적으로 Linear-Spectrogram을 만들면, Griffin-Lim Algorithm을 이용해서 audio를 생성한다.
- Loss = | mel_output – mel_target | + | linear_output – linear_target |

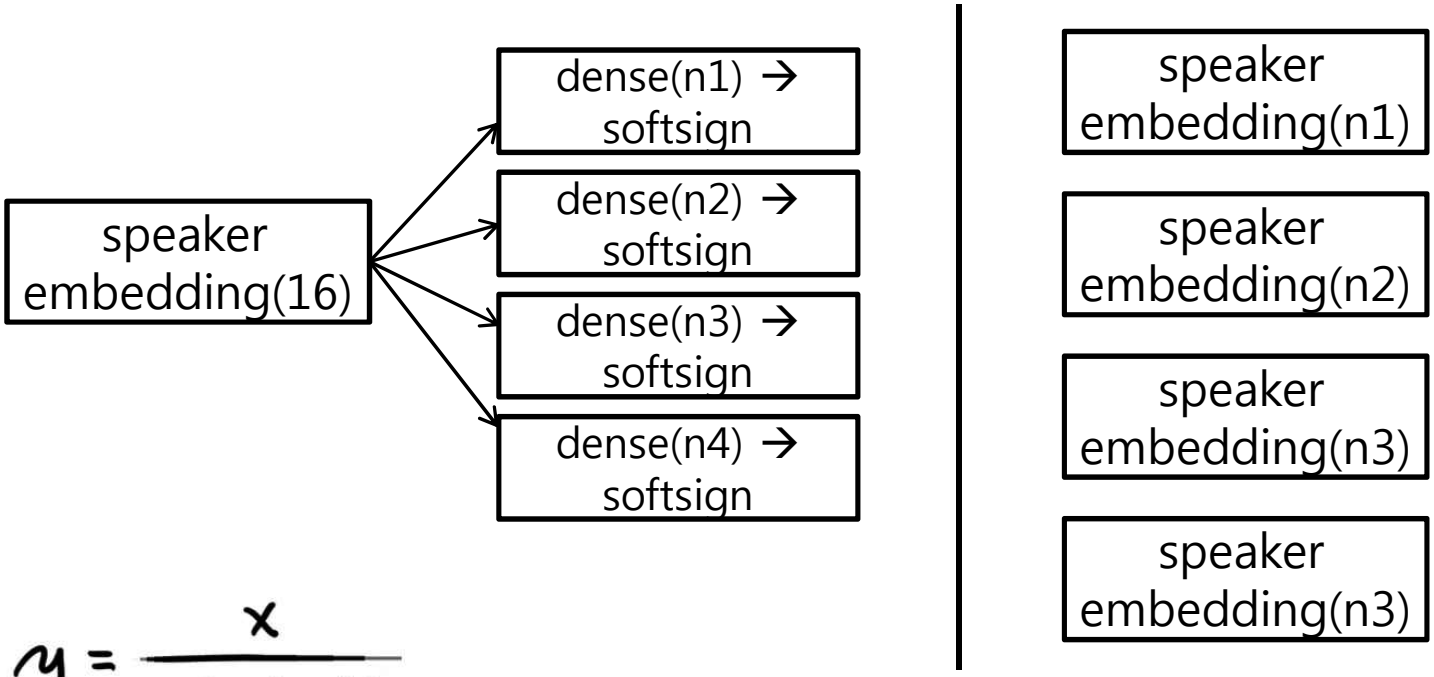
# Multi-Speaker Model로 확장(DeepVoice 2)




- speaker embedding vector를 network 중간-중간에 concat하는 방식의 모델도 있다.
- DeepVoice2 논문에는 Tacotron 모델을 Multi-Speaker 모델로 확장하는 방법을 제시하고 있다. 여기서는 speaker embedding vector를 RNN의 initial state로 넣어주는 방식을 사용한다.

# Speaker Embedding → Speaker별 data 불균형 해소

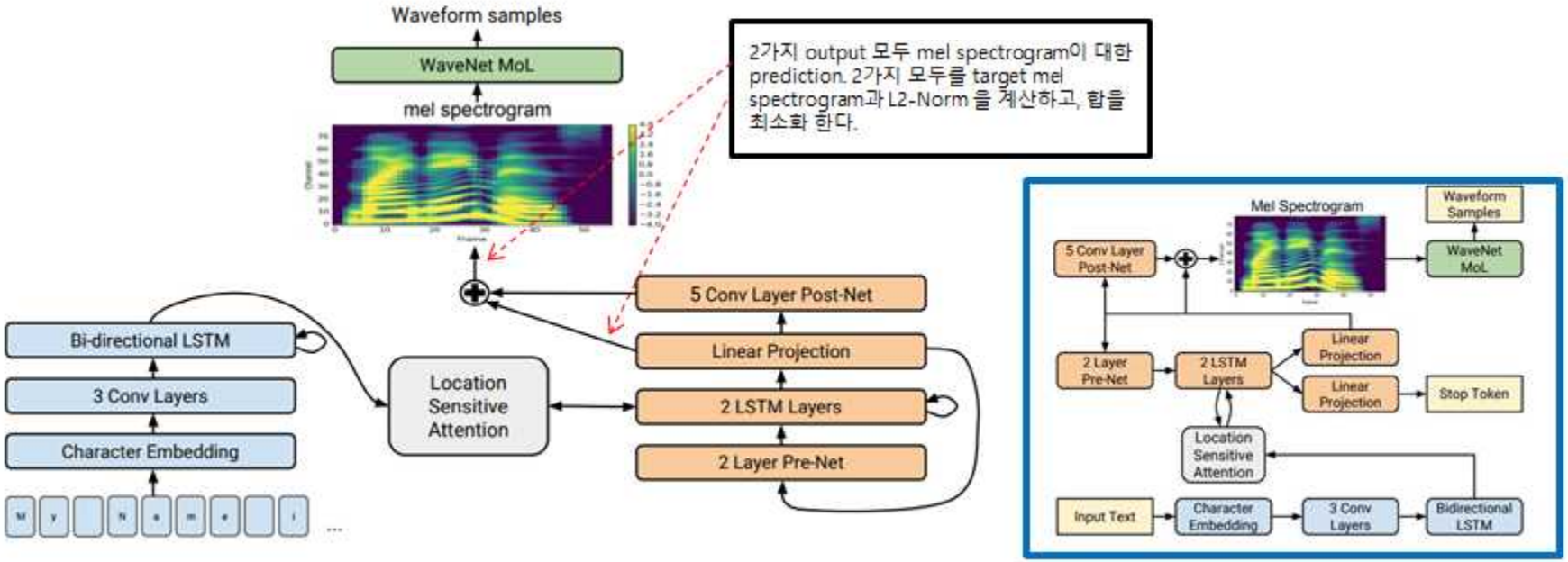
- 각 speaker별로 embedding vector(dim 16)을 먼저 만든 후, FC를 거치면서 필요한 크기로 만든 다음, activation function으로 softsign 함수를 적용한다.
- softsign함수는 tanh 대신 사용되었다고 보면 된다.
- 논문에서는 언급되지 않았지만, speaker embedding을 만들지 않고, initial state 자체를 embedding vector로 만들 수도 있다.




$$y = \frac{x}{(1+|x|)}$$

# Tacotron2

- 2017년 12월 발표
- CBHG 제외
- Location Sensitive Attention, Residual Layer 추가
- Stop Token 도입
- Vocoder로 Modified Wavenet사용: MoL
- L2 regularization
- 대표적인 구현 코드: [Rayhane Mama]



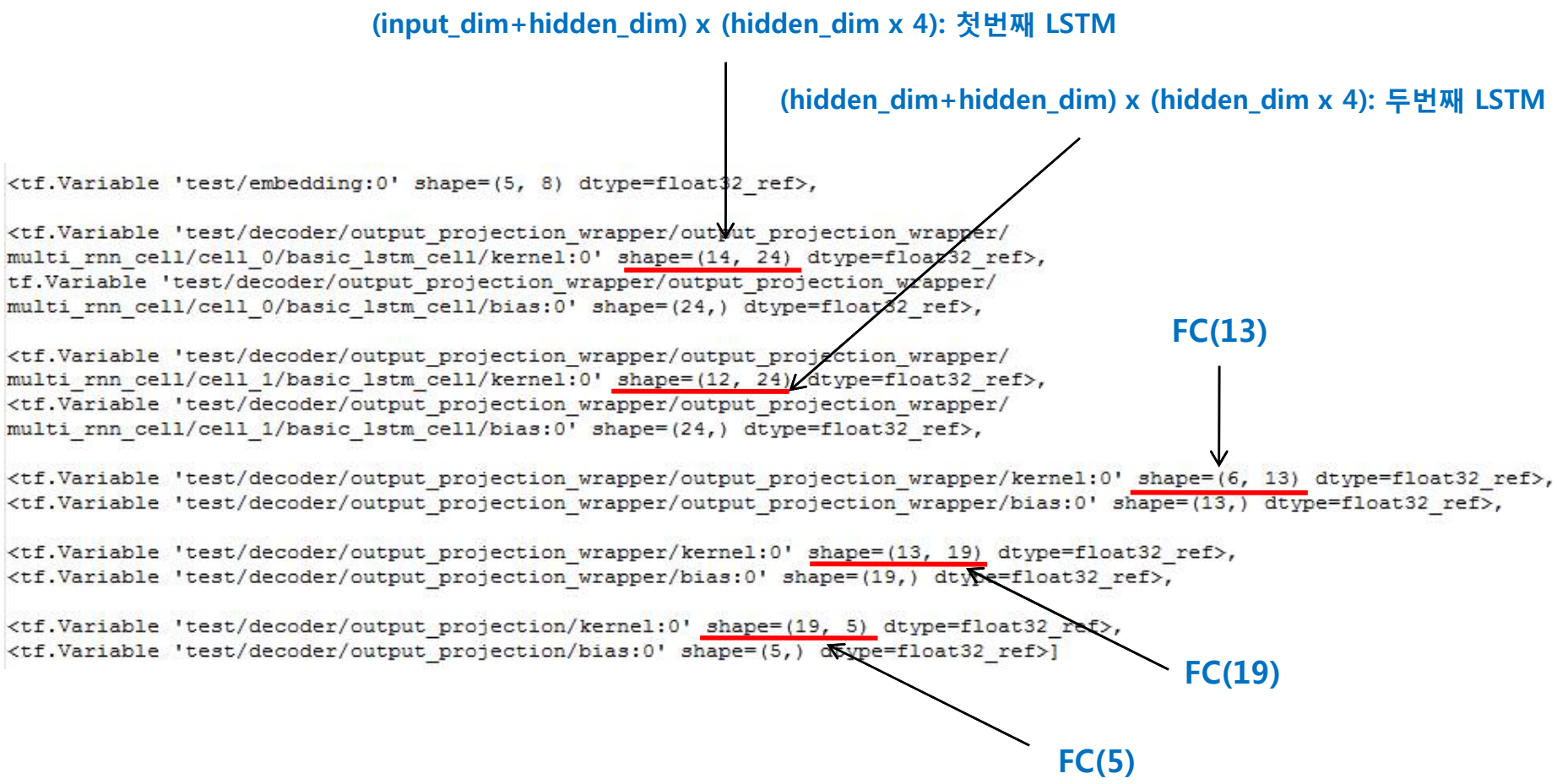
# 4. Appendix

# Sample Code(A)

```
def dynamic_decode_test():

    vocab_size = 5
    SOS_token = 0
    EOS_token = 4

    x_data = np.array([[SOS_token, 3, 1, 2, 3, 2],[SOS_token, 3, 1, 2, 3, 1],[SOS_token, 1, 3, 2, 2, 1]], dtype=np.int32)
    y_data = np.array([[1,2,0,3,2,EOS_token],[3,2,3,3,1,EOS_token],[3,1,1,2,0,EOS_token]],dtype=np.int32)
    print("data shape: ", x_data.shape)
    sess = tf.InteractiveSession()

    output_dim = vocab_size
    batch_size = len(x_data)
    hidden_dim =6
    num_layers = 2
    seq_length = x_data.shape[1]
    embedding_dim = 8
    state_tuple_mode = True
    init_state_flag = 0
    init = np.arange(vocab_size*embedding_dim).reshape(vocab_size,-1)

    train_mode = True
    with tf.variable_scope('test',reuse=tf.AUTO_REUSE) as scope:
        # Make rnn
        cells = []
        for _ in range(num_layers):
            #cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_dim)
            cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_dim,state_is_tuple=state_tuple_mode)
            cells.append(cell)
        cell = tf.contrib.rnn.MultiRNNCell(cells)
        #cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_dim)

        embedding = tf.get_variable("embedding", initializer=init.astype(np.float32),dtype = tf.float32)
        inputs = tf.nn.embedding_lookup(embedding, x_data) # batch_size  x seq_length x embedding_dim

        Y = tf.convert_to_tensor(y_data)
```

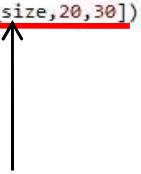
```
if True:
    cell = tf.contrib.rnn.OutputProjectionWrapper(cell,13)
    cell = tf.contrib.rnn.OutputProjectionWrapper(cell,19)

if init_state_flag==0:
     initial_state = cell.zero_state(batch_size, tf.float32) #(batch_size x hidden_dim) x layer 개수
else:
    if state_tuple_mode:
        h0 = tf.random_normal([batch_size,hidden_dim]) #h0 = tf.cast(np.random.randn(batch_size,hidden_dim),tf.float32)
        initial_state=(tf.contrib.rnn.LSTMStateTuple(tf.zeros_like(h0), h0),) + (tf.contrib.rnn.LSTMStateTuple(tf.zeros_like(h0), tf.zeros_like(h0)),)*(num_layers-1)

    else:
        h0 = tf.random_normal([batch_size,hidden_dim]) #h0 = tf.cast(np.random.randn(batch_size,hidden_dim),tf.float32)
        initial_state = (tf.concat((tf.zeros_like(h0),h0), axis=1),) + (tf.concat((tf.zeros_like(h0),tf.zeros_like(h0)), axis=1),) * (num_layers-1)
if train_mode:
    helper = tf.contrib.seq2seq.TrainingHelper(inputs, np.array([seq_length]*batch_size))
else:
    helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(embedding, start_tokens=tf.tile([SOS_token], [batch_size]), end_token=EOS_token)

output_layer = Dense(output_dim, name='output_projection')
decoder = tf.contrib.seq2seq.BasicDecoder(cell=cell,helper=helper,initial_state=initial_state,output_layer=output_layer)
# maximum_iterations를 설정하지 않으면, inference에서 EOS토큰을 만나지 못하면 무한 루프에 빠진다
# last_state는 num_layers 만큼 나온다.
outputs, last_state, last_sequence_lengths = tf.contrib.seq2seq.dynamic_decode(decoder=decoder,output_time_major=False,impute_finished=True,maximum_iterations=10)

weights = tf.ones(shape=[batch_size,seq_length])
loss =   tf.contrib.seq2seq.sequence_loss(logits=outputs.rnn_output, targets=Y, weights=weights)
```

- input_dim=8
- hidden_dim=6



# Sample Code(B)

```
def attention_test():
    # BasicRNNCell을 single로 쌓아 attention 적용
    vocab_size = 5
    SOS_token = 0
    EOS_token = 4

    x_data = np.array([[SOS_token, 3, 1, 2, 3, 2],[SOS_token, 3, 1, 2, 3, 1],[SOS_token, 1, 3, 2, 2, 1]], dtype=np.int32)
    y_data = np.array([[1,2,0,3,2,EOS_token],[3,2,3,3,1,EOS_token],[3,1,1,2,0,EOS_token]],dtype=np.int32)
    print("data shape: ", x_data.shape)
    sess = tf.InteractiveSession()

    output_dim = vocab_size
    batch_size = len(x_data)
    hidden_dim =6
    seq_length = x_data.shape[1]
    embedding_dim = 8

    init = np.arange(vocab_size*embedding_dim).reshape(vocab_size,-1)

    train_mode = True
    alignment_history_flag = True   # True이면 initial_state나 last state를 sess.run 하면 안됨. alignment_history가 function이기 때문에...
    with tf.variable_scope('test',reuse=tf.AUTO_REUSE) as scope:
        # Make rnn cell
        cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_dim)

        embedding = tf.get_variable("embedding", initializer=init.astype(np.float32),dtype = tf.float32)
        inputs = tf.nn.embedding_lookup(embedding, x_data) # batch_size  x seq_length x embedding_dim

        Y = tf.convert_to_tensor(y_data)

        #encoder_outputs = tf.ones([batch_size,20,30])
        encoder_outputs = tf.convert_to_tensor(np.random.normal(0,1,[batch_size,20,30]).astype(np.float32)) # 20: encoder sequence length, 30: encoder hidden dim

        input_lengths = [20]*batch_size
```



**encoder_hidden_state: (N,encoder_length,encoder_hidden_dim)**

```python
# attention_mechanism  # num_units = Na = 11
attention_mechanism = tf.contrib.seq2seq.BahdanauAttention(num_units=11, memory=encoder_outputs,memory_sequence_length=input_lengths,normalize=False)
#attention_mechanism = tf.contrib.seq2seq.BahdanauMonotonicAttention(num_units=11, memory=encoder_outputs,memory_sequence_length=input_lengths)


# output_attention = True(default) ==> 이면 output으로 attention이 나가고, False이면 cell의 output이 나간다
# attention_layer_size = N_L
cell = tf.contrib.seq2seq.AttentionWrapper(cell, attention_mechanism, attention_layer_size=13,alignment_history=alignment_history_flag,output_attention=True)


initial_state = cell.zero_state(batch_size, tf.float32) #(batch_size x hidden_dim) x layer 개수

if train_mode:
    helper = tf.contrib.seq2seq.TrainingHelper(inputs, np.array([seq_length]*batch_size))
else:
    helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(embedding, start_tokens=tf.tile([SOS_token], [batch_size]), end_token=EOS_token)

output_layer = Dense(output_dim, name='output_projection')
decoder = tf.contrib.seq2seq.BasicDecoder(cell=cell,helper=helper,initial_state=initial_state,output_layer=output_layer)
# maximum_iterations를 설정하지 않으면, inference에서 EOS토큰을 만나지 못하면 무한 루프에 빠진다
outputs, last_state, last_sequence_lengths = tf.contrib.seq2seq.dynamic_decode(decoder=decoder,output_time_major=False,impute_finished=True,maximum_iterations=10)

weights = tf.ones(shape=[batch_size,seq_length])
loss =   tf.contrib.seq2seq.sequence_loss(logits=outputs.rnn_output, targets=Y, weights=weights)
```

(input_dim+decoder_hidden_dim+attention_layer_size) x (decoder_hidden_dim)

(encoder_hidden_dim x num_units) $W_m$

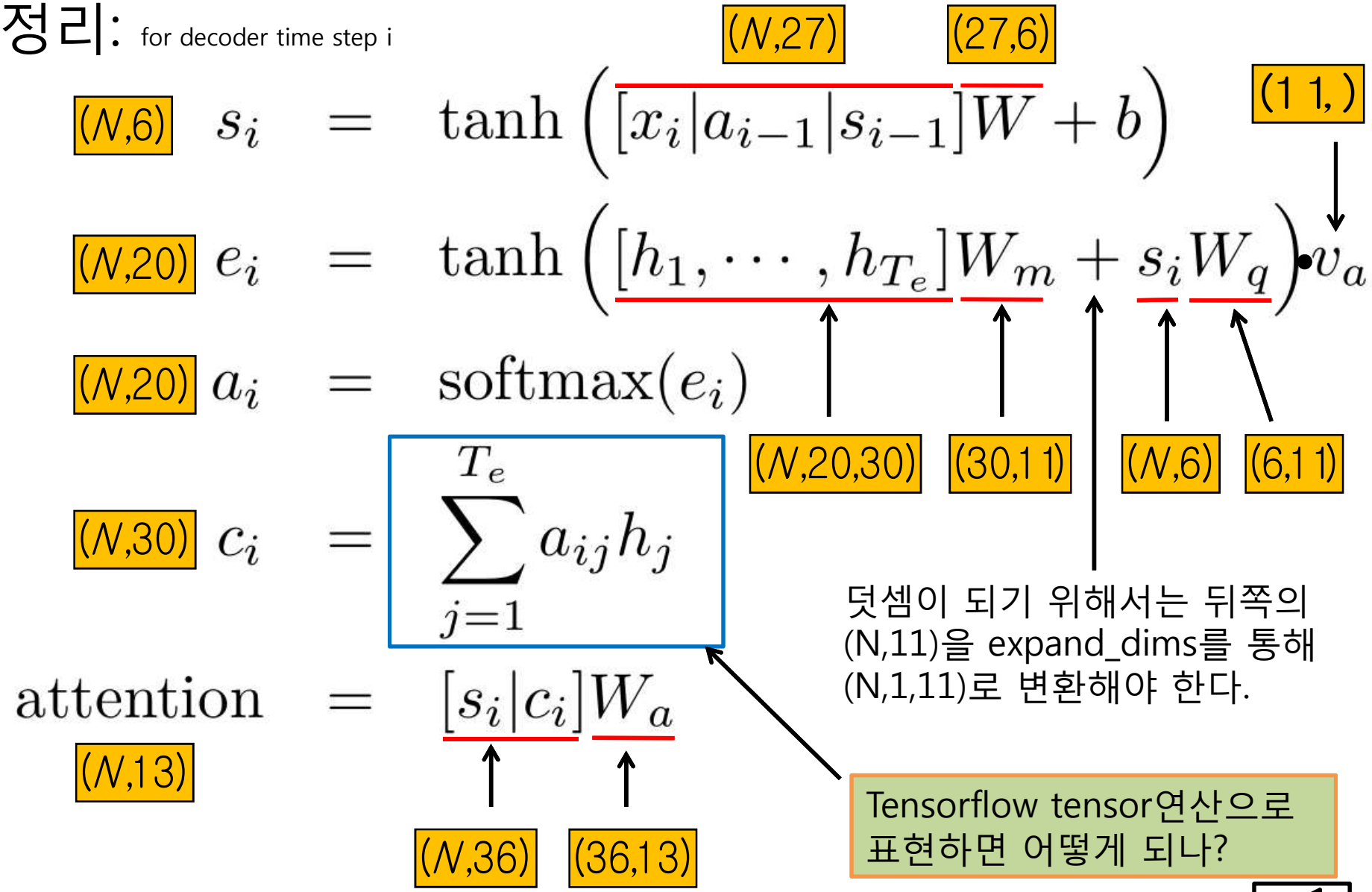
(decpcer_hidden_dim x num_units) $W_q$

```
<tf.Variable 'test/embedding:0' shape=(5, 8) dtype=float32_ref>,
<tf.Variable 'test/memory_layer/kernel:0' shape=(30, 11) dtype=float32_ref>,
<tf.Variable 'test/decoder/attention_wrapper/basic_rnn_cell/kernel:0' shape=(27, 6) dtype=float32_ref>,
<tf.Variable 'test/decoder/attention_wrapper/basic_rnn_cell/bias:0' shape=(6,) dtype=float32_ref>,
<tf.Variable 'test/decoder/attention_wrapper/bahdanau_attention/query_layer/kernel:0' shape=(6, 11) dtype=float32_ref>,
<tf.Variable 'test/decoder/attention_wrapper/bahdanau_attention/attention_v:0' shape=(11,) dtype=float32_ref>,
<tf.Variable 'test/decoder/attention_wrapper/attention_layer/kernel:0' shape=(36, 13) dtype=float32_ref>,
<tf.Variable 'test/decoder/output_projection/kernel:0' shape=(13, 5) dtype=float32_ref>,
<tf.Variable 'test/decoder/output_projection/bias:0' shape=(5,) dtype=float32_ref>]
```

$W_a$ $v_a$ (num_units)

(encoder_hidden_dim+decoder_hidden_dim) x attention_layer_size

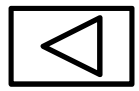
# 정리: for decoder time step i



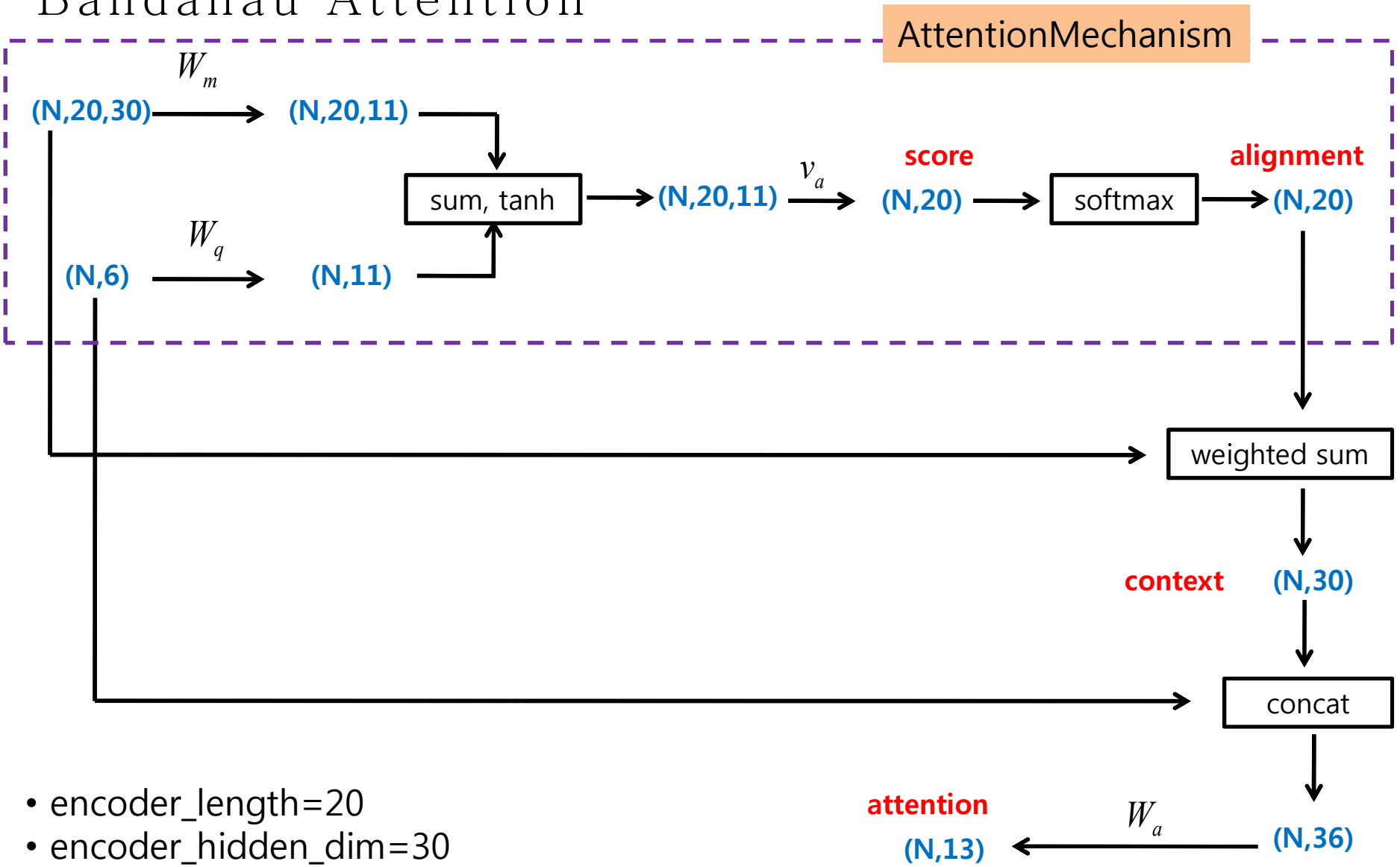
# Sample Code(C)

```
# Sample Code C
batch_size= 5
hidden_dim= 8

input = tf.placeholder(tf.float32,shape=[None,None,2])
cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_dim)
initial_state = cell.zero_state(batch_size, tf.float32)
outputs, states = tf.nn.dynamic_rnn(cell,input,initial_state=initial_state,dtype=tf.float32)
```

```
[<tf.Variable 'rnn/basic_rnn_cell/kernel:0' shape=(10, 8) dtype=float32_ref>,
 <tf.Variable 'rnn/basic_rnn_cell/bias:0' shape=(8,) dtype=float32_ref>]
```

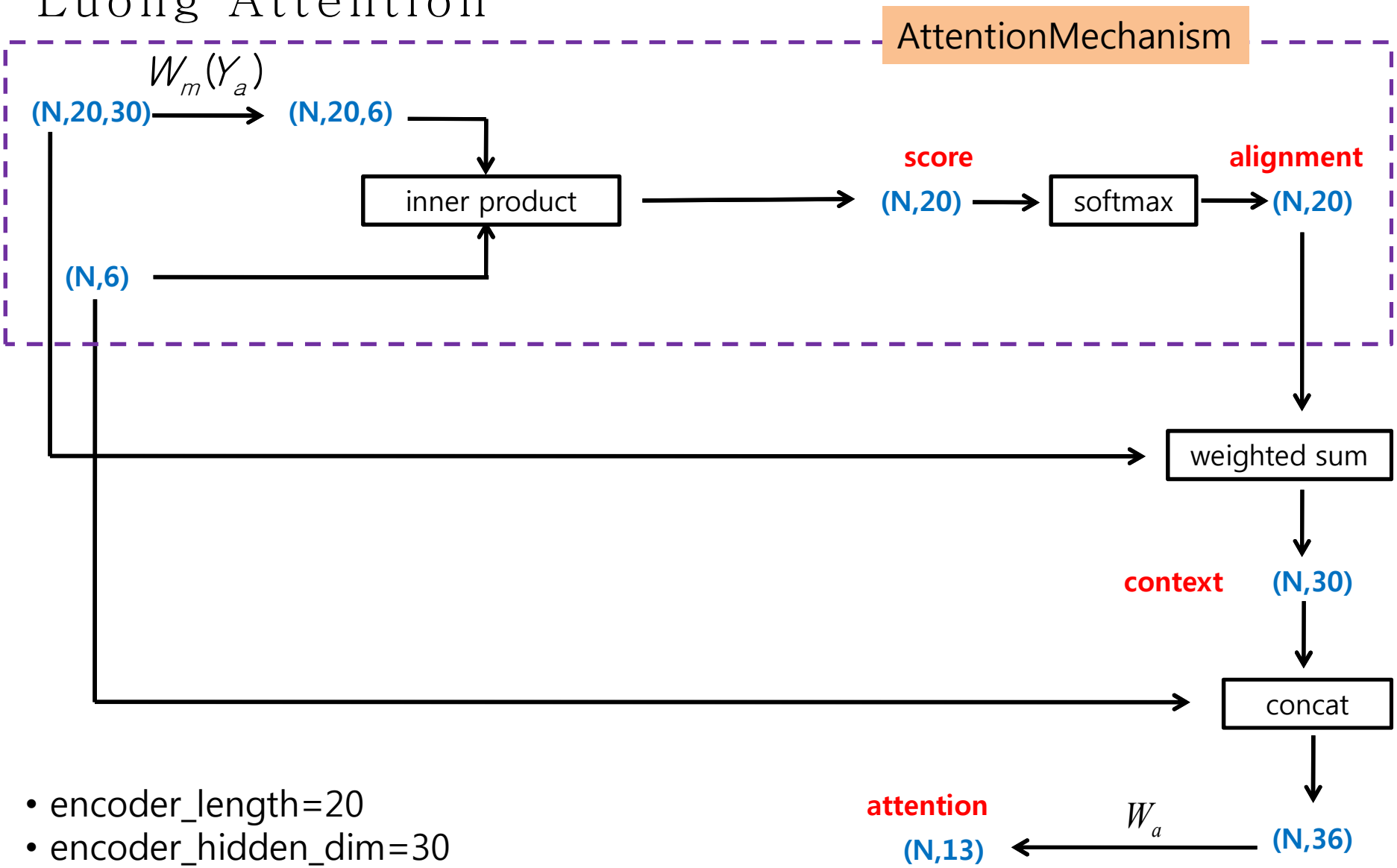


# Bahdanau Attention




- encoder_length=20
- encoder_hidden_dim=30
- decoder_hidden_dim=6

# Luong Attention




- encoder_length=20
- encoder_hidden_dim=30
- decoder_hidden_dim=6

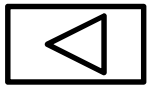
# Tensorflow-Dropout

## tf.layers.dropout

```
tf.layers.dropout(
    inputs,
    rate=0.5,
    noise_shape=None,
    seed=None,
    training=False,
    name=None
)
```

## tf.nn.dropout

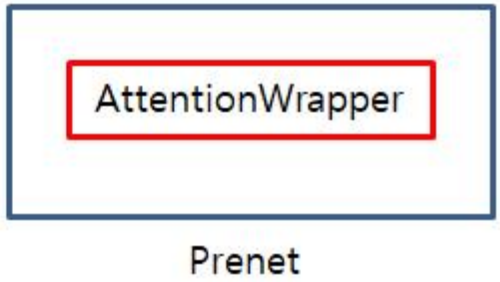
```
tf.nn.dropout(
    x,
    keep_prob,
    noise_shape=None,
    seed=None,
    name=None
)
```



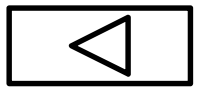
- keithito 코드에서는 2018년8월 31일 bug 수정

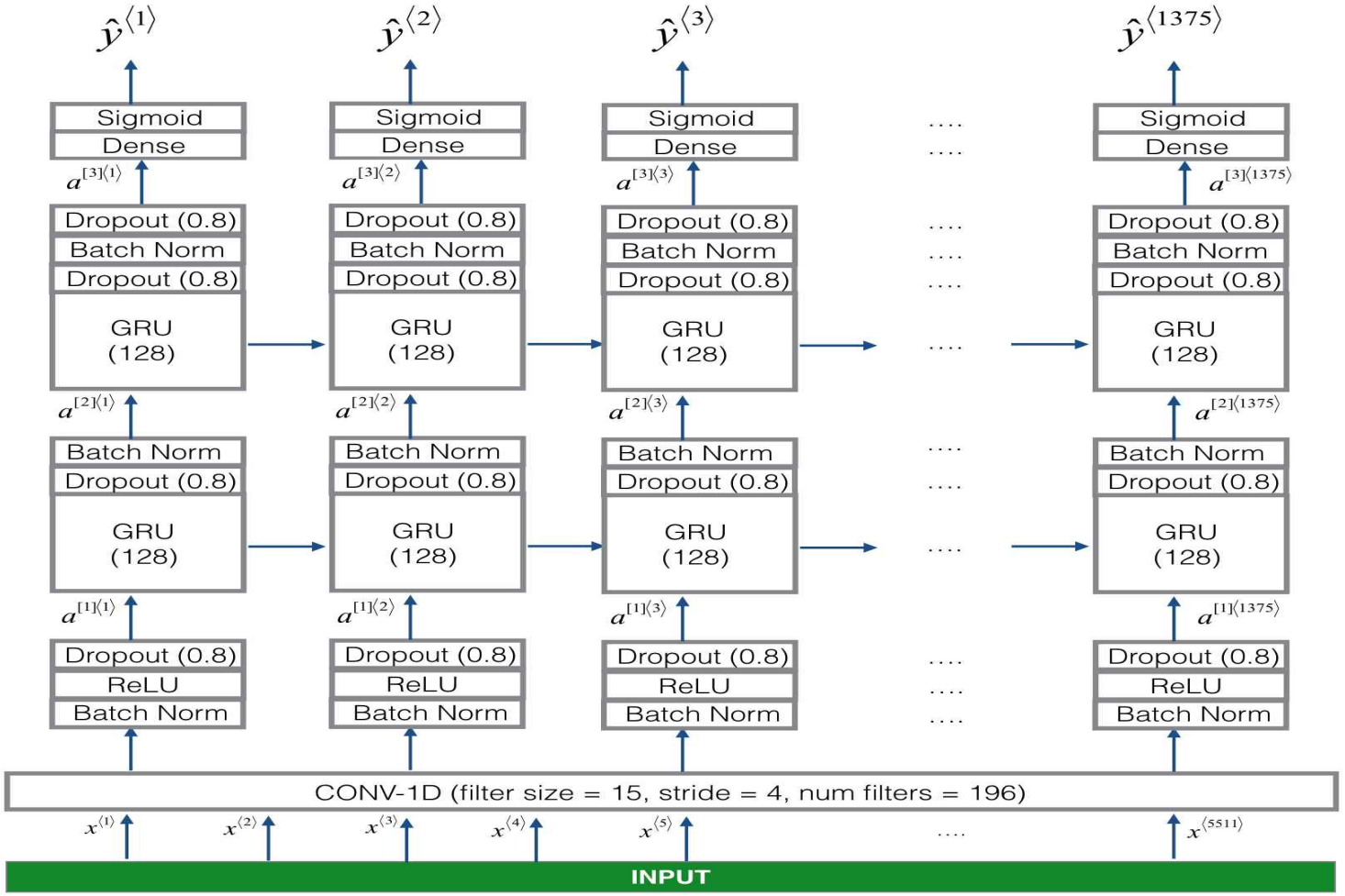
```
# Attention
attention_cell = AttentionWrapper( GRUCell(hp.attention_depth), BahdanauAttention(hp.attention_depth, encoder_outputs),
                                   alignment_history=True,output_attention=False)    # [N, T_in, attention_depth=256]

# Apply prenet before concatenation in AttentionWrapper.
attention_cell = DecoderPrenetWrapper(attention_cell, is_training, hp.prenet_depths)
```

Keith Ito




```
        dec_prenet_outputs = DecoderPrenetWrapper(   GRUCell(hp.attention_state_size), speaker_embed, is_training, hp.dec_prenet_sizes, hp.dropout_prob)

        # single: attention_size = 128
        if hp.attention_type == 'bah_mon':
            attention_mechanism = BahdanauMonotonicAttention(hp.attention_size, encoder_outputs,normalize=False)
        elif hp.attention_type == 'bah_mon_norm':
            attention_mechanism = BahdanauMonotonicAttention(hp.attention_size, encoder_outputs,normalize=True)
        elif hp.attention_type == 'bah_norm':
            attention_mechanism = BahdanauAttention(hp.attention_size, encoder_outputs, normalize=True)
        elif hp.attention_type == 'luong_scaled':
            attention_mechanism = LuongAttention( hp.attention_size, encoder_outputs, scale=True)
        elif hp.attention_type == 'luong':
            attention_mechanism = LuongAttention(hp.attention_size, encoder_outputs)
        elif hp.attention_type == 'bah':
            attention_mechanism = BahdanauAttention(hp.attention_size, encoder_outputs)
        elif hp.attention_type.startswith('ntm2'):
            shift_width = int(hp.attention_type.split('-')[-1])
            attention_mechanism = NTMAttention2( hp.attention_size, encoder_outputs, shift_width=shift_width)
        else:
            raise Exception(" [!] Unkown attention type: {}".format(hp.attention_type))


        # DecoderPrenetWrapper, attention_mechanism을 감싸는 AttentionWrapper를 만든다.
        attention_cell = AttentionWrapper(dec_prenet_outputs,attention_mechanism, self.is_manual_attention,self.manual_alignments,
                                          initial_cell_state=attention_rnn_init_state,alignment_history=True,output_attention=False)  # output_attention=
```


$\hat{y}^{\langle 1 \rangle}$
$\hat{y}^{\langle 2 \rangle}$
$\hat{y}^{\langle 3 \rangle}$
$\hat{y}^{\langle 1375 \rangle}$
Sigmoid
Dense
$a^{[3]\langle 1 \rangle}$
$a^{[3]\langle 2 \rangle}$
$a^{[3]\langle 3 \rangle}$
$a^{[3]\langle 1375 \rangle}$
Dropout (0.8)
Batch Norm
Dropout (0.8)
GRU
(128)
$a^{[2]\langle 1 \rangle}$
$a^{[2]\langle 2 \rangle}$
$a^{[2]\langle 3 \rangle}$
$a^{[2]\langle 1375 \rangle}$
Batch Norm
Dropout (0.8)
GRU
(128)
$a^{[1]\langle 1 \rangle}$
$a^{[1]\langle 2 \rangle}$
$a^{[1]\langle 3 \rangle}$
$a^{[1]\langle 1375 \rangle}$
Dropout (0.8)
ReLU
Batch Norm
CONV-1D (filter size = 15, stride = 4, num filters = 196)
$x^{\langle 1 \rangle}$
$x^{\langle 2 \rangle}$
$x^{\langle 3 \rangle}$
$x^{\langle 4 \rangle}$
$x^{\langle 5 \rangle}$
$x^{\langle 5511 \rangle}$
INPUT

# Tacotron Training 과정

| | |
|---|---|
| 9월 | - Single speaker 모델 시도. 왜 안되지?<br>- Attention Model로 MonotonicAttention적용. alignment합이 왜 1이 아닌가?<br>- librosa version 0.5.1 vs 0.6.1<br>- tensorflow 1.8로 변경<br>- dropout bug발견 ➔ keith ito 코드와 비교<br>- AttentionWrapper와 PrenetWrapper 순서 바로 잡음 |
| **10월** | - Padding에 Attention 가지 않도록<br>■ batch_size = 1로 변경?<br>■ Attention class로 customization시도 ➔ Tensorflow내에 이미 구현되어 있다는 것 발견<br>- 수작업으로 script 수정 ➔ Data 품앗이<br>- Tacotron2 모델의 stop token 적용, location sensitive attention, GMM attention 시도 |
| **11월** | - Mel Spectrogram 생성방식 수정 |