Deep Learning
Deep Neural Network

Yoon Joong Kim,
Hanbat National University

Deep Learning

음성인식

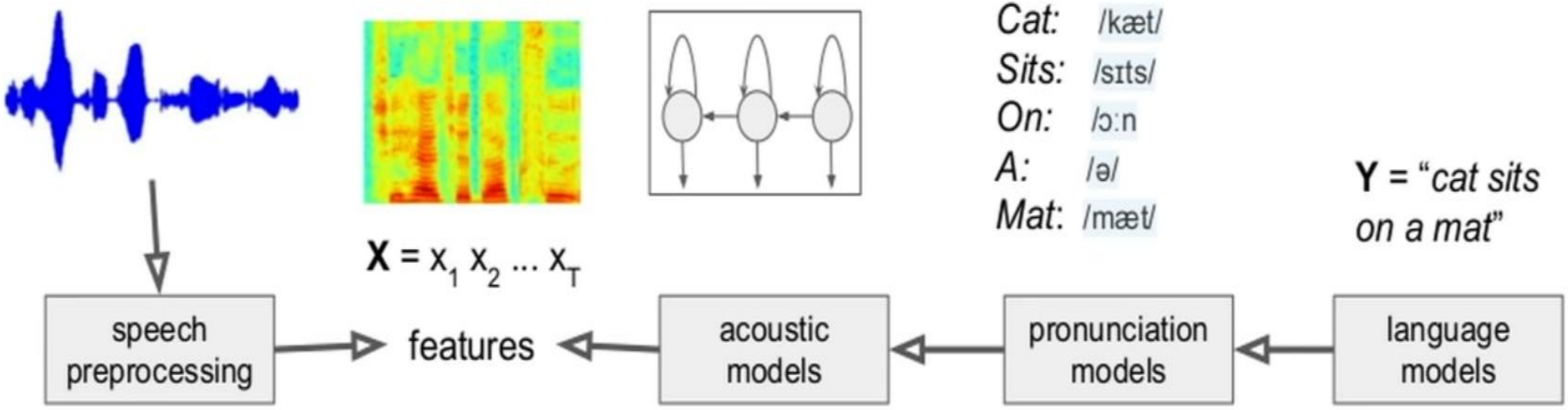
음성인식 개요
CTC **음성인식 모델**
음성인식을 위한 seq-to-seq, attention
CTC ASR **음성인식응용프로그램 개발**

Yoon Joong Kim
Department of Computer Engineering, Hanbat National University
yjkim@hanbat.ac.kr

Content

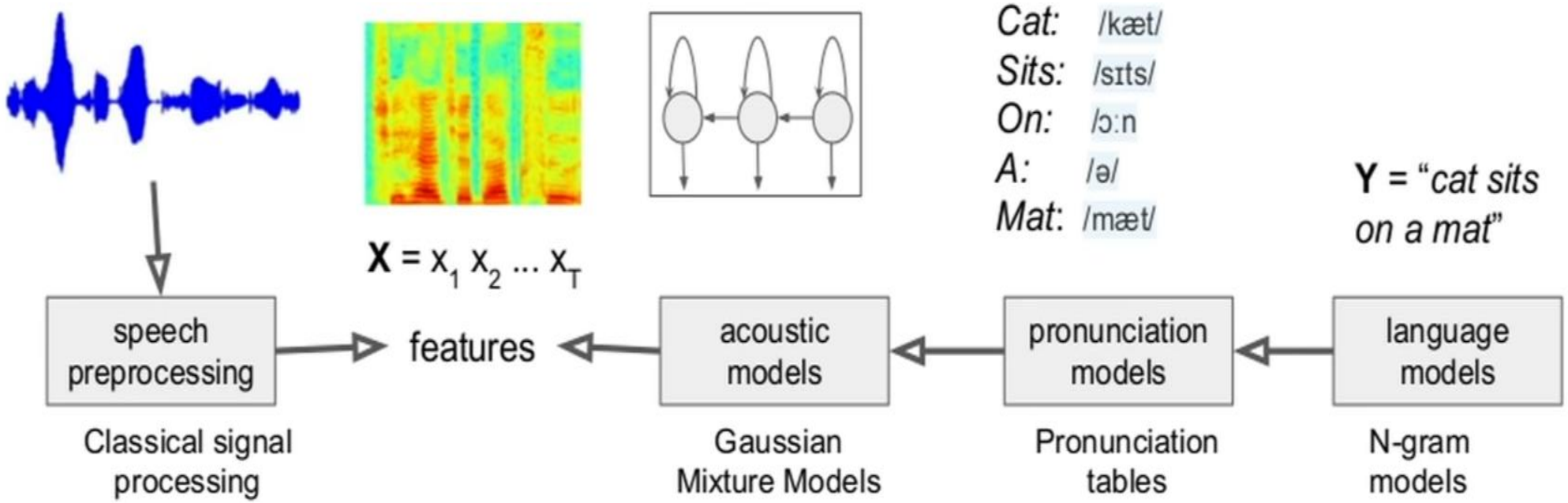
1. 음성인식개요

- 1.1 음성인식 – 기존방식(the classic way)
  - 음성인식 시스템 : 텍스트 시퀀스 $Y = y_1 y_2 \dots y_L$로부터 음향 특징열(audio features) $X = x_1 x_2 \dots x_T$로 매핑하기 위한 음성의 통계모델을 구축하고
  - 음성인식 : 입력 음성에 대하여 해당모델을 탐색하여 텍스트를 출력한다.



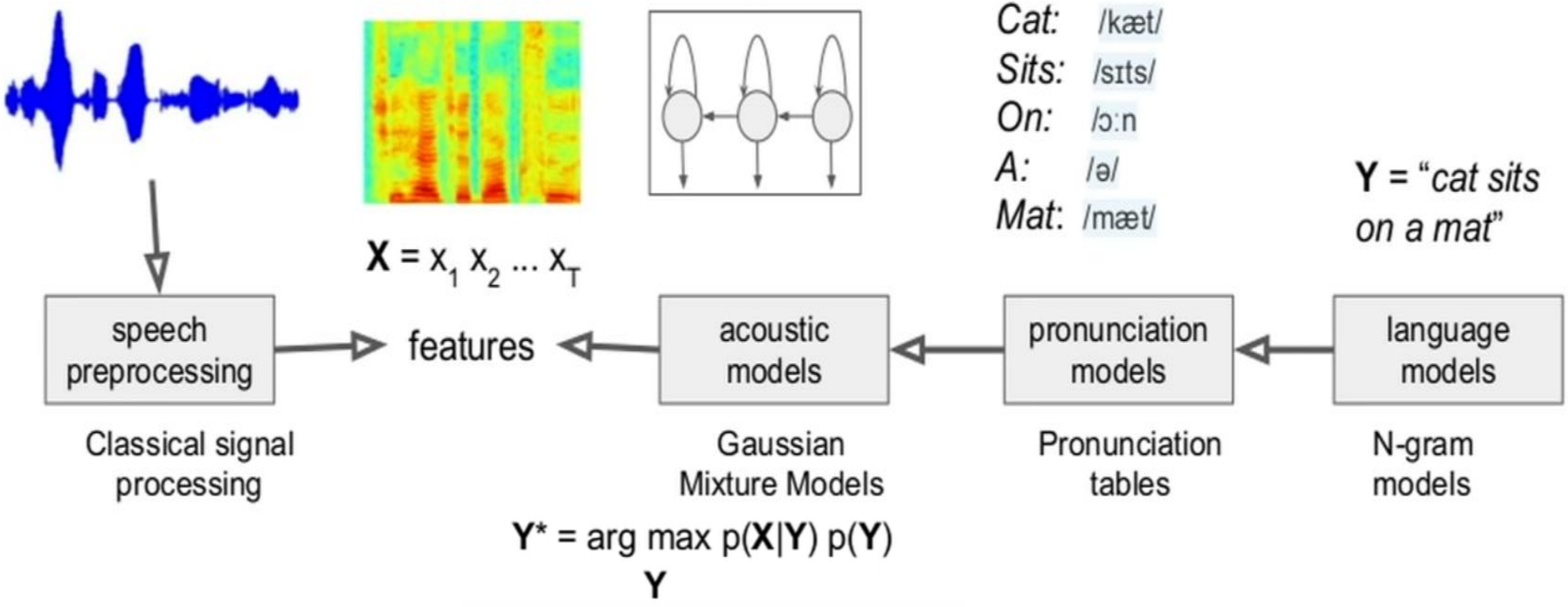
1.1 음성인식- 기존방식(cont.)

- 구성요소별로 제 각각의 모델을 사용하고 있다.



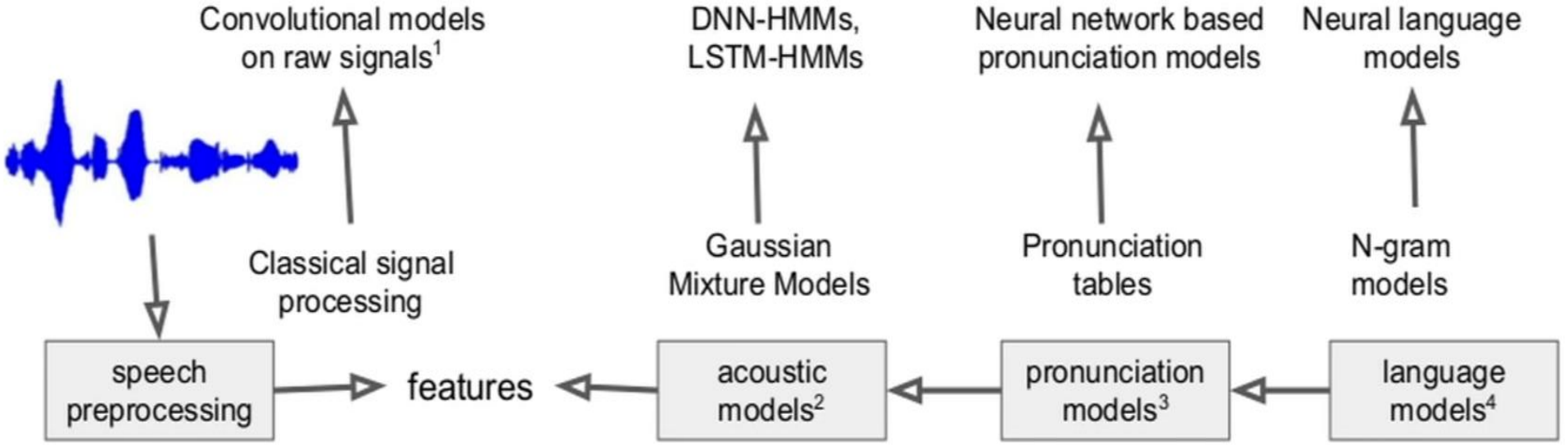
1.1 음성인식- 기존방식(cont.)

- 추론(인식)
  - 음향특징이 주어지면 이 음향특징을 가장 잘 만들어 낼 수 있는 모델을 탐색하여 텍스트 (문자열)를 출력한다.



1.2 신경망 음성인식

- 신경망 음성인식의 필요성
  - 각각의 구성요소들은 기존방식의 성능이 신경망보다 더 좋아 보인다.



1. Jaitly, Navdeep, and Geoffrey Hinton. "Learning a better representation of speech soundwaves using restricted boltzmann machines." *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, 2011.
2. Hinton, Geoffrey, et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." *IEEE Signal Processing Magazine* 29.6 (2012): 82-97.
3. Rao, Kanishka, et al. "Grapheme-to-phoneme conversion using long short-term memory recurrent neural networks." *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015.
4. Mikolov, Tomas, et al. "Recurrent neural network based language model." *Interspeech*. Vol. 2. 2010.

Stanford

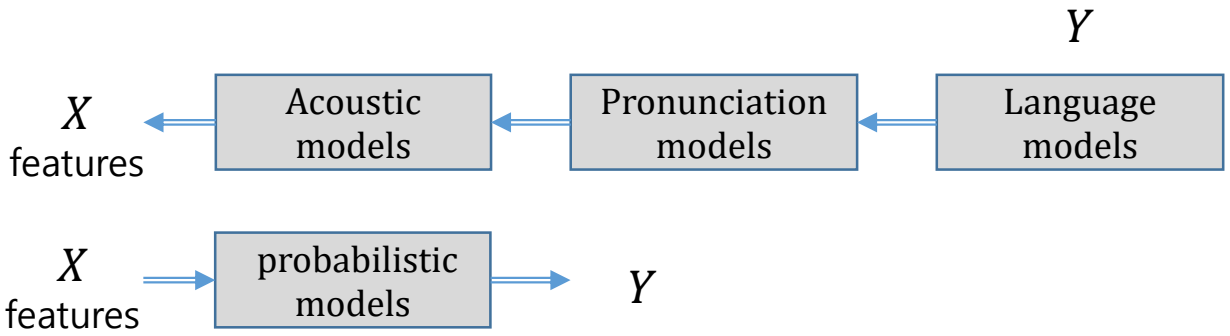
1.2 신경망 음성인식(cont.)

- 그러나
  - 구성요소들은 제 각각의 목적을 가지고 독립적으로 훈련된다.
  - 한 구성요소의 오류(error)은 다른 구성요소의 오류와 잘 어울리지 못한다.(Errors in one component may not behave with errors in another component)
  - 모든 요소들을 하나로 묶어서 훈련하기.(end-to-end model)
  - 기술
    - Connectionist Temporal Classification(CTC)
    - Sequence to sequence with attention
  - Amazing work was already done
    - Mozilla DeepSpeech (TensorFlow)
    - deepspeech.pytorch (PyTorch)
    - KerasDeepSpeech (Keras)

1.2 신경망 음성인식(cont.)

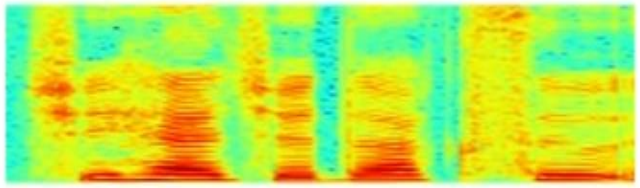
- End-to-end speech recognition as modeling task
  - Given audio $X = x_1 x_2 \dots x_T$ and corresponding output text $Y = y_1 y \dots y_L$ where $y \in \{a, b, c, \dots, z, ?, !, \dots\}$
  - Y is just a text sequence(transcript), X is the audio /processed spectrogram.
  - Perform speech recognition, by learning a probabilistic model p(Y|X)

$$Y^* = \underset{Y}{\operatorname{argmax}}\, p(Y|X)$$



2. Connectionist Temporal Classification(CTC)

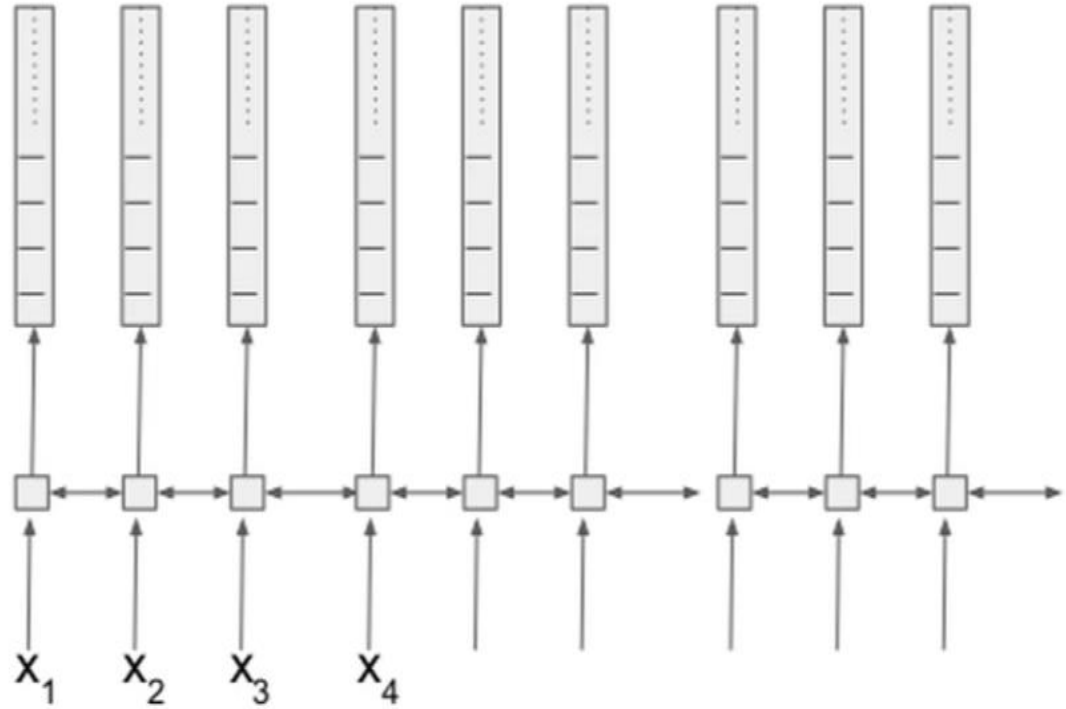
- CTC – a portable model p(Y|X), where
  - $X = x_1 x_2 \ldots x_T$,
  - $Y = y_1 y_2 \ldots y_L$,
  - $T \geq L$,
- has a specific structure that is suited for speech.



$$Y^* = \underset{Y}{\operatorname{argmax}}\, p(Y|X)$$

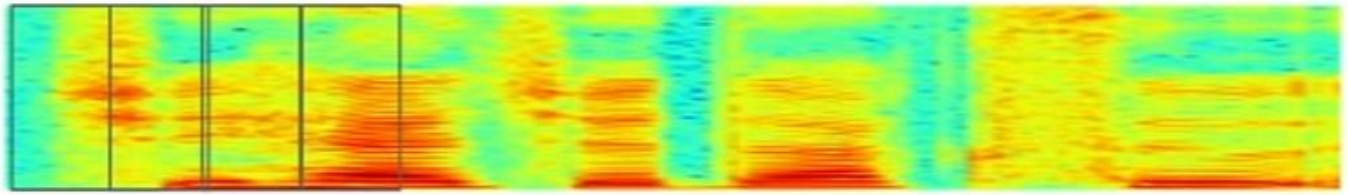
She is a cat

2. Connectionist Temporal Classification(Cont.)



Softmax over vocabulary {a, b, c, d, e, f, .. z, ?, ., !,..} and extra token <b>.

Softmax at step, t, gives a score s(k,t) = log Pr(k, t|**X**) to category k in the output at time t.

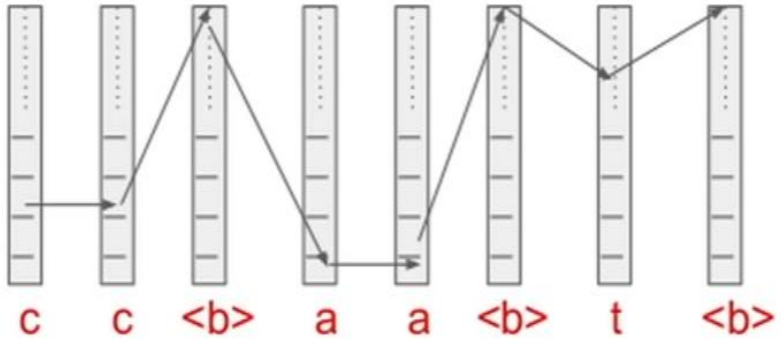


- Graves, Alex, and Navdeep Jaitly. "Towards End-To-End Speech Recognition with Recurrent Neural Networks." *ICML*. Vol. 14. 2014.
- Amodei, Dario, et al. "Deep speech 2: End-to-end speech recognition in english and mandarin." *arXiv preprint arXiv:1512.02595* (2015).
- H. Sak, A. Senior, K. Rao, O. Irsoy, A. Graves, F. Beaufays, and J. Schalkwyk, "Learning acoustic frame labeling for speech recognition with recurrent neural

2. Connectionist Temporal Classification(Cont.)

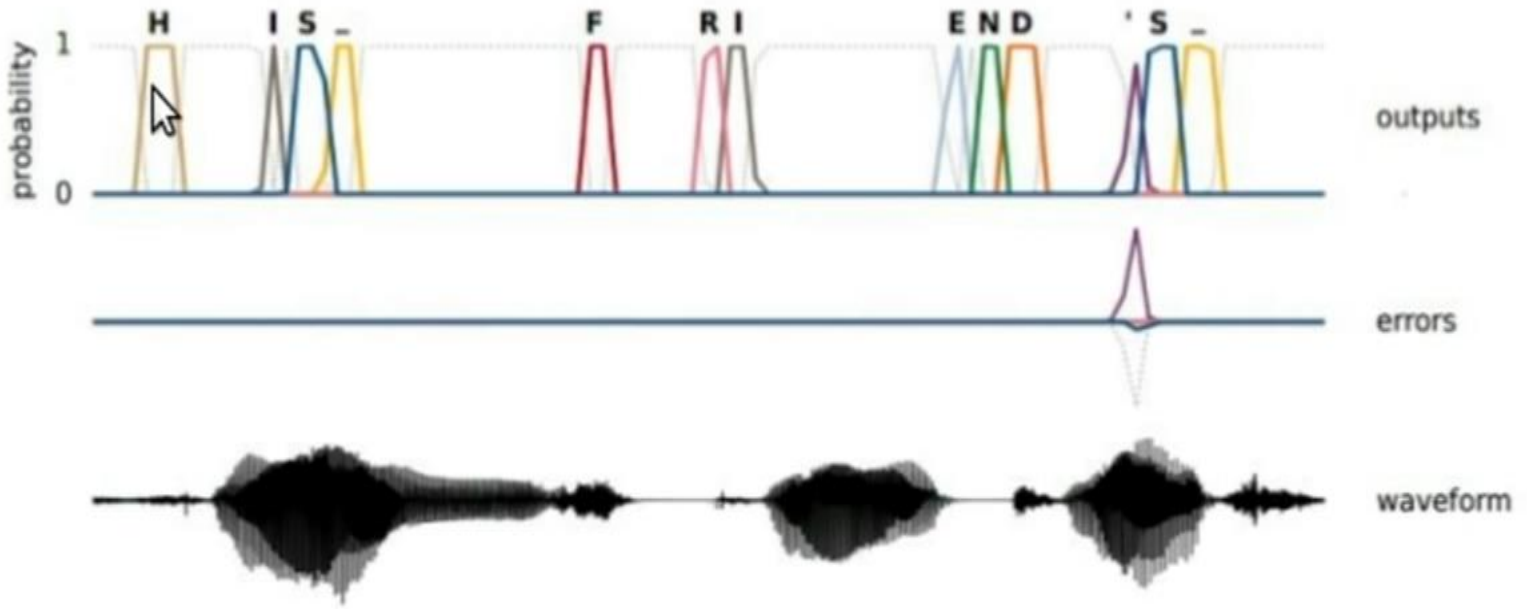
• CTC- How frame predictions map to output?

- Repeated tokens are deduplicated
  - cc <b> aa <b> t <b>
- Any original transcript, maps to all possible paths in the duplicated space:
  - cc<b>aa<b>t<b> maps to cat
  - cc<b><b>a<b>t<b> maps to cat
  - ccccc<b>aaaaa<b>ttttt<b> maps to cat
  - ccccc<b>aaaaa<b>ttttt<b> maps to cat
- The score (log probability) of any path is the sum of the scores of individual categories at the different time steps
- The probability of any transcript is the sum of probabilities of all paths that correspond to that transcript



*Because of dynamic programming, it is possible to compute both the log probability p(**Y**|**X**) and its gradient exactly! This gradient can be propagated to neural network whose parameters can then be adjusted by your favorite optimizer!*

2. Connectionist Temporal Classification(Cont.)



Model learns to make peaky predictions!

2. Connectionist Temporal Classification(Cont.)

- Some examples

target: TO ILLUSTRATE THE POINT A PROMINENT MIDDLE EAST ANALYST IN WASHINGTON RECOUNTS A CALL FROM ONE CAMPAIGN

output: TWO ALSTRAIT THE POINT A PROMINENT MIDILLE EAST ANALYST IM WASHINGTON RECOUNCACALL FROM ONE CAMPAIGN

2. Connectionist Temporal Classification(Cont.)

• Language Model과 CTC

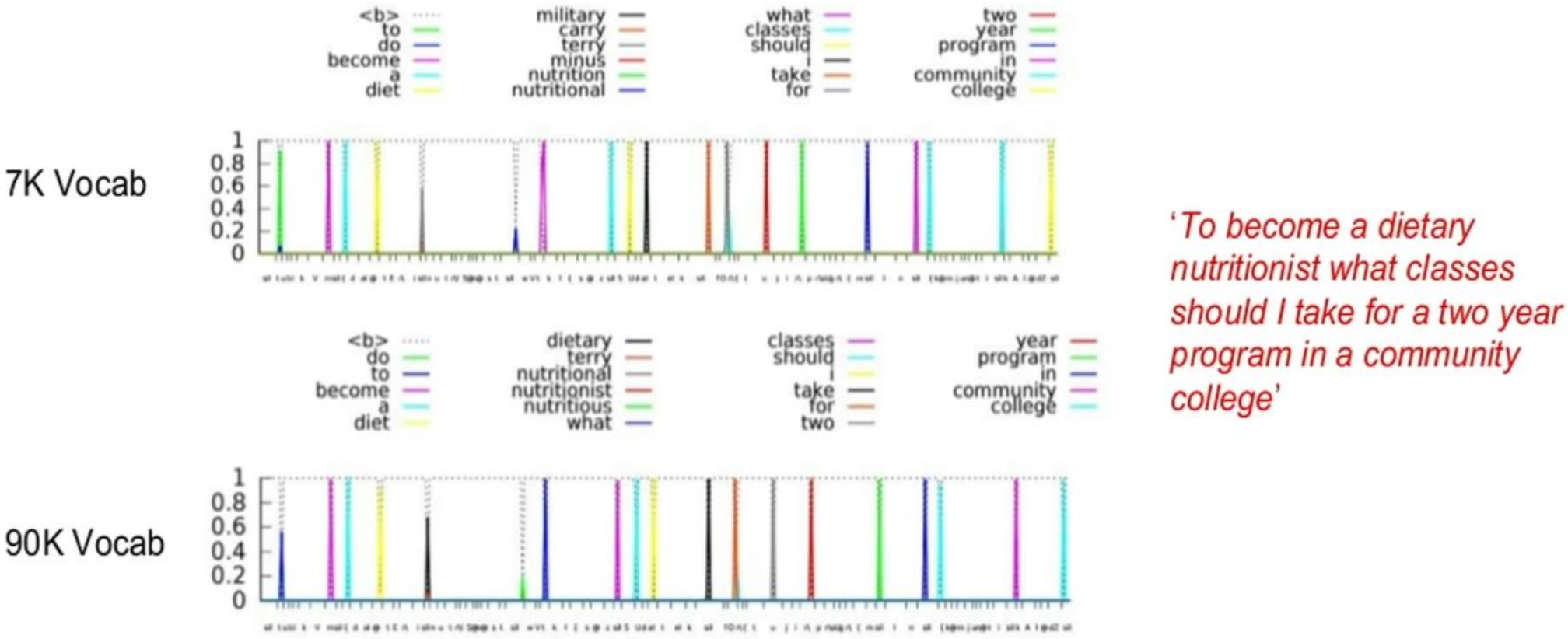
- Previous transcripts sounded correct, but clearly lacked the correct spelling and grammar
  - More training data can help, but eventually, a language model is required to fix these problems
  - With a simple language model rescoring, word error rate (WER) goes from 30.1% to 8.7%

- Google's CTC implementation fixes these problems by integrating a language model into CTC during training

H. Sak, A. Senior, K. Rao, O. Irsoy, A. Graves, F. Beaufays, and J. Schalkwyk, "Learning acoustic frame labeling for speech recognition with recurrent neural networks," in IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2015.

2. Connectionist Temporal Classification(Cont.)

- CTC – with word targets

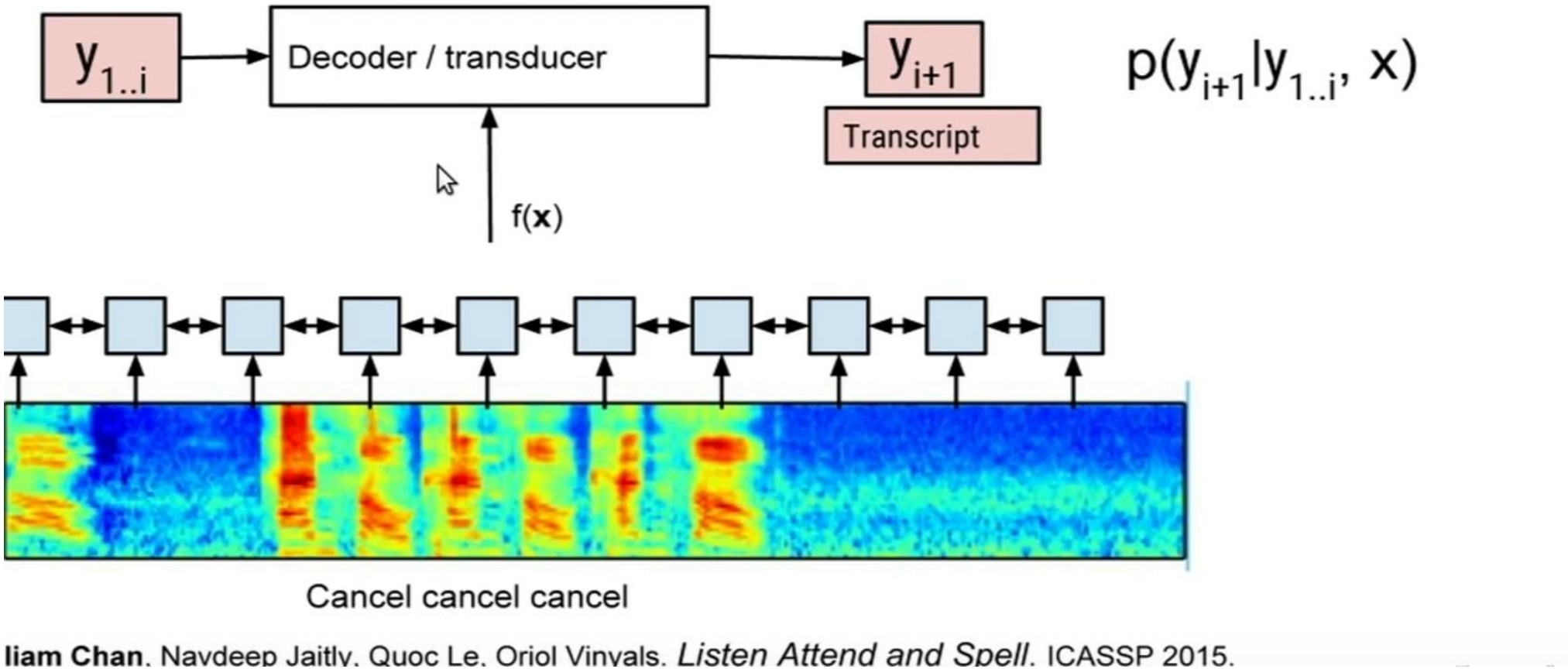


'To become a dietary nutritionist what classes should I take for a two year program in a community college'

H. Sak, A. Senior, K. Rao, O. Irsoy, A. Graves, F. Beaufays, and J. Schalkwyk, "Learning acoustic frame labeling for speech recognition with recurrent neural networks," in IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2015.

3. 음성인식, seq-to-seq와 attention

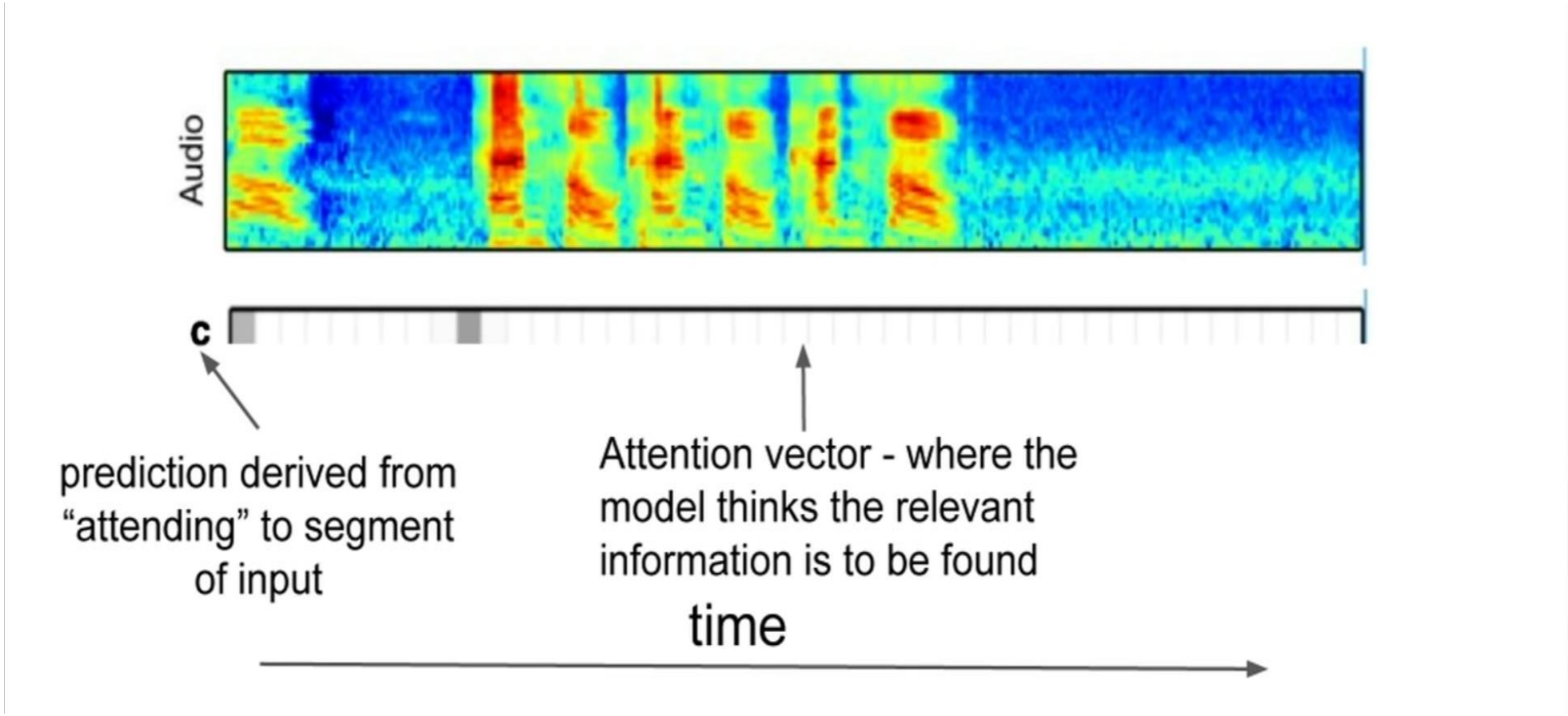
- 음성인식, attention and Sequence-to-sequence



liam Chan, Navdeep Jaitly, Quoc Le, Oriol Vinyals. *Listen Attend and Spell*. ICASSP 2015.

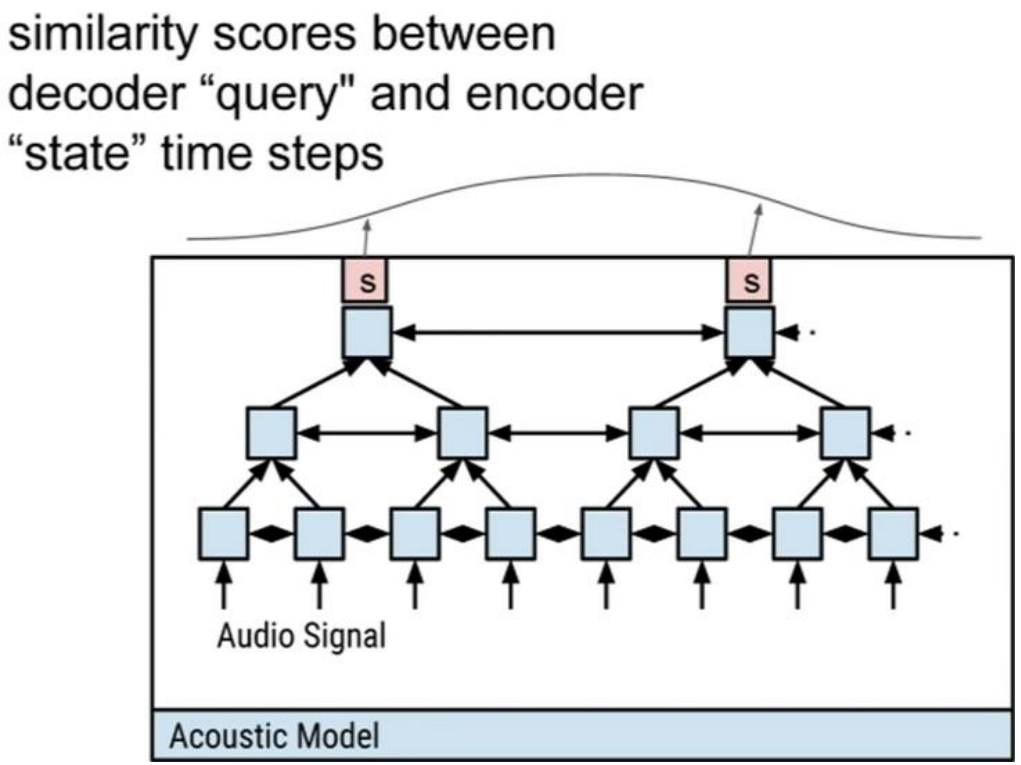
3. 음성인식, seq-to-seq와 attention(cont.)

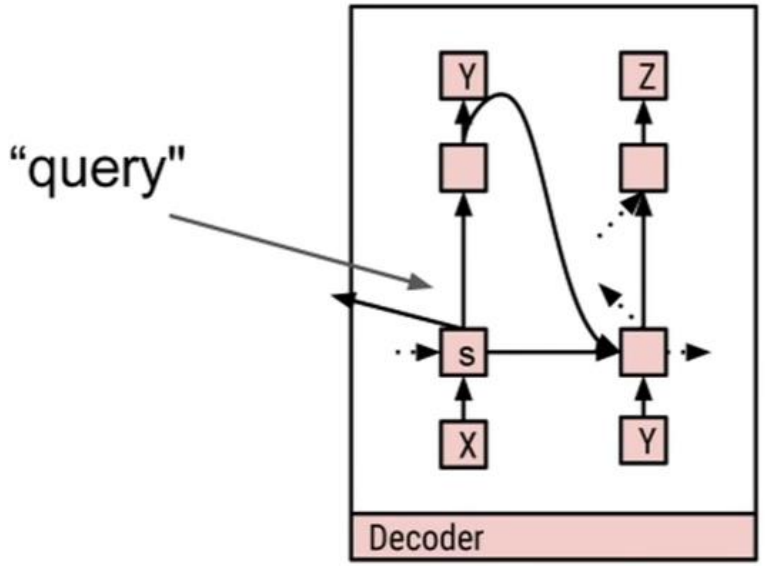
- Attention example



3. 음성인식, seq-to-seq와 attention(cont.)

• Encoder-decoder 와 attention mechanism

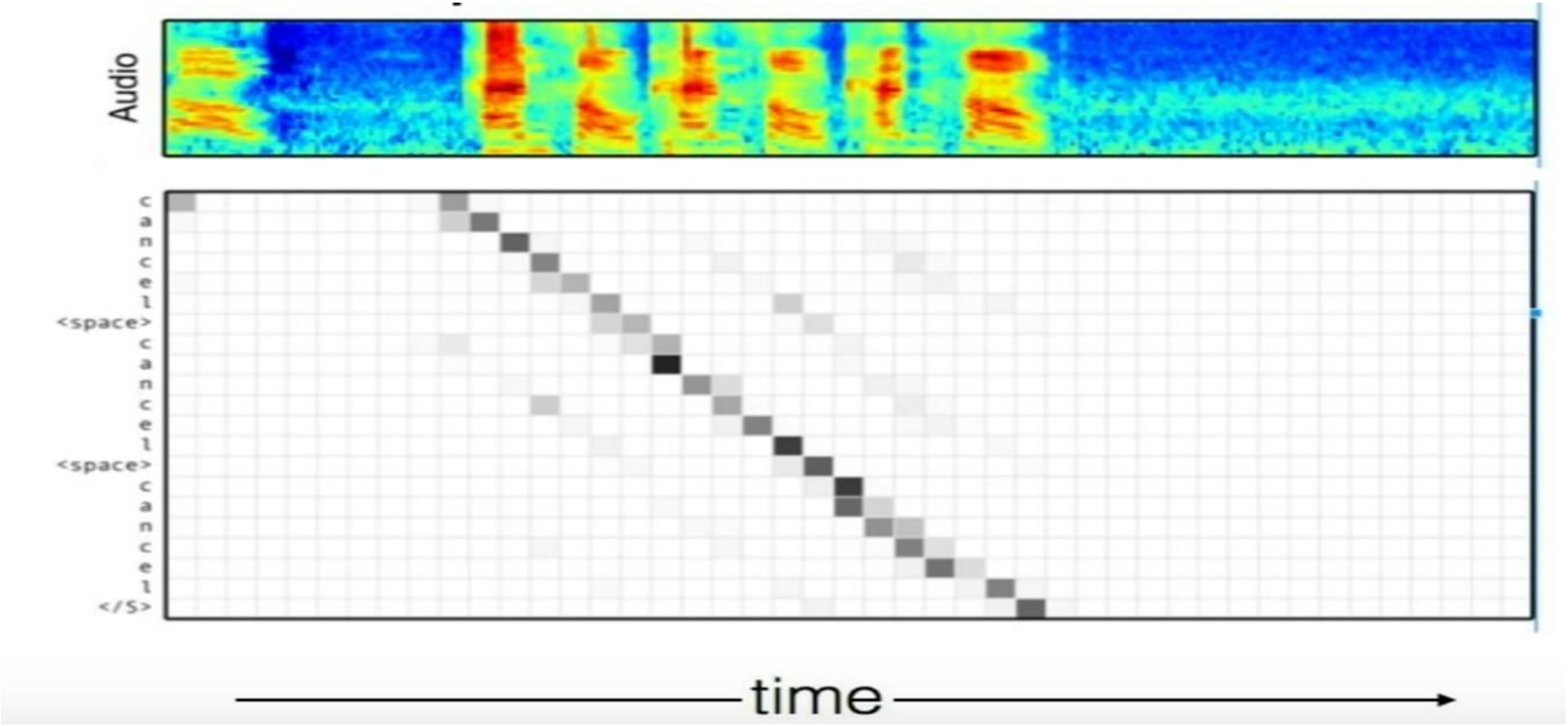




D. Bahdanau, K. Cho and Y. Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. ICLR 2015.

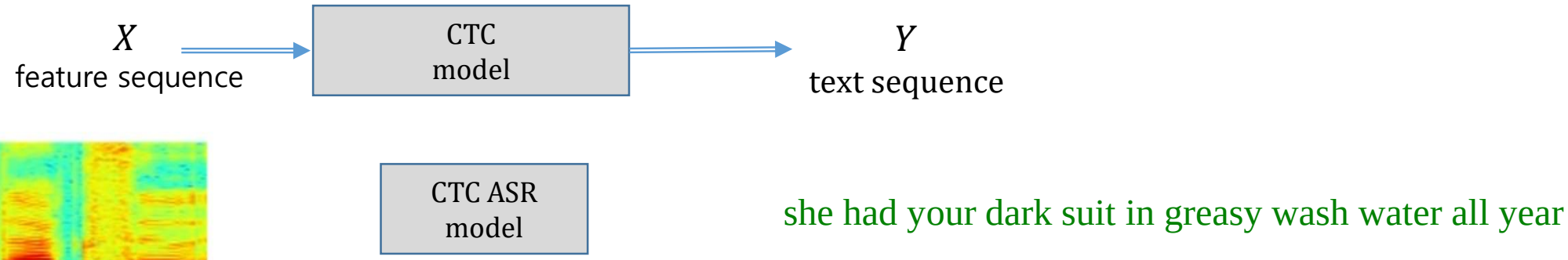
3. 음성인식, seq-to-seq와 attention(cont.)

- Attention example



4. CTC 음성인식 시스템

$$Y^* = \underset{Y}{\operatorname{argmax}}\, p(Y|X)$$



train dataset

LDC93S1.wav

target dataset

LDC93S1.txt

4.1 dataset 생성
4.2 CTC ASR model
4.3 훈련 및 검증
4.4 Full code

4.1 Dataset 생성

- TMIT dataset
  - TIMIT Acoustic-Phonetic Continuous Speech Corpus
    - https://catalog.ldc.upenn.edu/ldc93s1
    - https://catalog.ldc.upenn.edu/docs/LDC93S2/TIMIT.TXT
  - TIMIT
  - TIMIT is a corpus of phonemically and lexically transcribed speech of American English speakers of different sexes and dialects. Each transcribed element has been delineated in time.
  - TIMIT was designed to further acoustic-phonetic knowledge and automatic speech recognition systems. It was commissioned by DARPA and corpus design was a joint effort between the Massachusetts Institute of Technology, SRI International, and Texas Instruments (TI). The speech was recorded at TI, transcribed at MIT, and verified and prepared for publishing by the National Institute of Standards and Technology (NIST).[1] There is also a telephone bandwidth version called NTIMIT (Network TIMIT).
  - TIMIT and NTIMIT are not freely available — either membership of the Linguistic Data Consortium, or a monetary payment, is required for access to the dataset.

4.1 Dataset 생성(cont.)

- Training/test data
  - LDC93S1.wav
    - 16KHz,93594B,shape=(46797,)
  - LDC93S1.txt
    - '0 46797 She had your dark suit in greasy wash water all year.\n‘

4.1 Dataset 생성(cont.)

- Train dataset

```
import time
import scipy.io.wavfile as wav
import numpy as np
from python_speech_features import mfcc
from utils import sparse_tuple_from as sparse_tuple_from
```

```
import scipy.io.wavfile as wav
audio_filename = LDC93S1.wav’
target_filename = LDC93S1.txt'
fs, audio = wav.read(audio_filename)
inputs   = mfcc(audio, samplerate=fs)
```

```
Inputs (291,13)
array([[  7.07865742, -25.0219473 ,  -6.32589368, ...,  -4.32991717,    -3.98560931,  -1.78402763],
       [  7.16654419, -25.24814281,  -1.60271255, ...,   2.74616587,      -2.55490863,   2.81827641],
       ...,
       [  8.22333051, -21.97670506, -26.42023311, ...,  -8.50371043,     1.8066162 ,   2.84506726]])
```

```
train_inputs = np.reshape(inputs,(1,-1,13))
```

```
train_inputs (1,291,13)
array([[[  7.07865742, -25.0219473 ,  -6.32589368, ...,  -4.32991717,    -3.98560931,  -1.78402763],
        [  7.16654419, -25.24814281,  -1.60271255, ...,   2.74616587,      -2.55490863,   2.81827641],
        ...,
        [  8.22333051, -21.97670506, -26.42023311, ...,  -8.50371043,     1.8066162 ,   2.84506726]]])
```

```
train_inputs = (train_inputs - np.mean(train_inputs))/np.std(train_inputs)
```

```
train_inputs (1,291,13)
array([[[ 0.72197885, -1.22595829, -0.09143956, ...,  0.02968079,         0.05057417,  0.18417112],
        [ 0.72731201, -1.23968434,  0.19517373, ...,  0.45907348,         0.13739232,  0.46344931],
...,
        [ 0.79144019, -1.04116612, -1.31080943, ..., -0.2235944 ,         0.40205949,  0.46507504]]])
```

```
train_seq_len = [train_inputs.shape[1]]
```

```
[291]
```

4.1 Dataset 생성(cont.)

- Target dataset

```
alphabet=[' ']+[chr(c) for c in range(ord('a'),ord('z')+1)] #[' ' 'a' 'b' ...'z']
ch2code={ch:c for c,ch in enumerate(alphabet)}          #{' ': 0, 'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, ...}
code2ch={c:ch for c,ch in enumerate(alphabet)}

target_filename = 'LDC93S1.txt'
lines=open(targetfilename,'r').readlines()              #(1) ['0 46797 She had your dark suit in greasy wash water all year.\n']
original = ' '.join(lines[0].strip().lower().split(' ')[2:]).replace('.', '')    # 'she had your dark suit in greasy wash water all year'
targets  = [ch2code for c in original]                  # [19, 8, 5, 0, 8, 1, 4, ...18]
train_targets = sparse_tuple_from([targets])            #(array([[ 0,  0],[0,1],..[0,51]], dtype=int64),    #(52,2)
                                                          array([19,  8,  5,  0,...,1, 18],dtype=int64),     #(52,)
                                                          dense_shape=array([ 1, 52], dtype=int64))      #(2,)
# A validation dataset :
val_inputs, val_targets, val_seq_len = train_inputs, train_targets,train_seq_len
```

```
val_inputs
  array([[[ 0.72197885, -1.22595829, -0.09143956, ...,  0.02968079,        0.05057417, 0.18417112],
        ...,
        [ 0.79144019, -1.04116612, -1.31080943, ..., -0.2235944 ,        0.40205949,  0.46507504]]])

val_targets
   (array([[ 0,  0],[0,1],..[0,51]], dtype=int64),
    array([19,  8,  5,  0,...,1, 18],dtype=int64),
    dense_shape=array([ 1, 52], dtype=int64))
val_seq_len :   291
```

4.1 Dataset 생성(cont.)

```
train_inputs (0,291,13)
    array([[[ 0.72197885, -1.22595829, -0.09143956, ...,  0.02968079,         0.05057417,  0.18417112],
       ...,
            [ 0.79144019, -1.04116612, -1.31080943, ..., -0.2235944 ,         0.40205949,  0.46507504]]])
```

```
train_targers (3,)   (52,2),(52,),(2,)
    (   array([[ 0,  0],  [ 0,  1],   [ 0,  2],  …  [ 0, 51]], dtype=int64), array([19,  8,  5,  0,...,1, 18],dtype=int64),     dense_shape=array([ 1, 52], dtype=int64))
```

```
train_seq_len  (1,)
    [  291]
```

```
val_inputs (0,291,13)
    array([[[ 0.72197885, -1.22595829, -0.09143956, ...,  0.02968079,         0.05057417,  0.18417112],
            ...,
            [ 0.79144019, -1.04116612, -1.31080943, ..., -0.2235944 ,         0.40205949,  0.46507504]]])
```

```
val_targers (3,)   (52,2),(52,),(2,)
 (   array([[ 0,  0],  [ 0,  1],   [ 0,  2],  …  [ 0, 51]], dtype=int64), array([19,  8,  5,  0,...,1, 18],dtype=int64),     dense_shape=array([ 1, 52], dtype=int64))
```

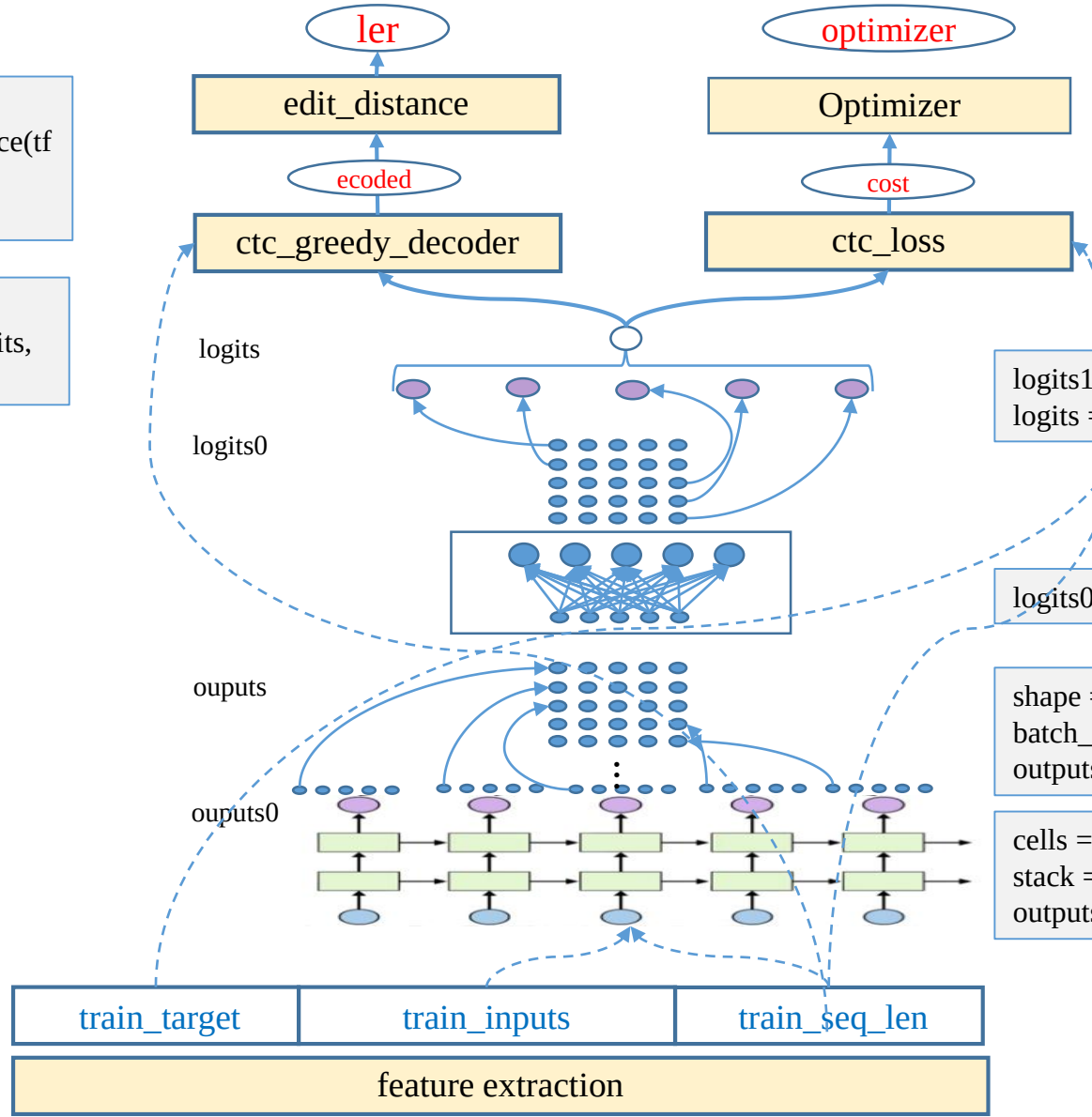
```
val_seq_len
    291
```

4.2 CTC ASR model

- Hyper parameters

```
# Some configs
alphabet=[' ']+[chr(c) for c in range(ord('a'),ord('z')+1)] #[' ' 'a' 'b' ...'z']
ch2code={ch:c for c,ch in enumerate(alphabet)}
code2ch={c:ch for c,ch in enumerate(alphabet)}
#{' ': 0, 'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, ...}

# Hyper-parameters
num_features  = 13              # feature mffc 차수
num_units     =50               # Number of units in the LSTM cell
num_classes   =len(alphabet)    # 27
num_layers    = 1               # lstm rnn layer

num_epochs    = 100             #
batch_size    = 1               #
initial_learning_rate = 1e-2
momentum      = 0.9             # training parameter
num_examples = 1                # max example number
num_batches_per_epoch = int(num_examples/batch_size) #
```

4.2 CTC ASR model




```
ler = tf.reduce_mean(tf.edit_distance(tf.cast(decoded[0], tf.int32), self.targets))
```

```
dcoded, log_prob = tf.nn.ctc_greedy_decoder(logits, seq_len)
```

```
optimizer = tf.train.MomentumOptimizer(initial_learning_rate, 0.9).minimize(cost)
```

```
loss = tf.nn.ctc_loss(targets, logits, seq_len)
cost= tf.reduce_mean(loss)
```

```
logits1 = tf.reshape(logits0, [batch_s, -1, num_classes])
logits = tf.transpose(logits1, (1, 0, 2))
```

```
logits0 = tf.layers.dense(outputs,num_classes,activation=None)
```

```
shape = tf.shape(inputs)
batch_s, max_timesteps = shape[0], shape[1]
outputs = tf.reshape(outputs0, [-1, num_hidden])
```

```
cells = [tf.nn.rnn_cell.LSTMCell(num_units)  for _ range(num_layers) ]
stack = tf.nn.rnn_cell.MultiRNNCell(cells)
outputs0, _ = tf.nn.dynamic_rnn(stack, inputs, seq_len, dtype=tf.float32)
```

Data flow in ctc model

```
num_examples=3; num_features=4; num_classes=8(8+1);  num_units  = 5
```

inputs:(3,) ((4,4) (6,4) (4,4))

```
[array([[-0.6876526 , -0.4449298 ,  1.049204  ,  0.45081407],
        [ 0.34290984,  0.38235345, -0.77697825,  1.3119483 ],
        [ 0.12359548, -0.4051013 ,  0.8457591 , -1.3446374 ],
        [-0.8168718 ,  0.15581544,  1.7109454 ,  0.68036443]],
       dtype=float32)
 array([[-0.32760054,  1.7909838 , -0.64140093, -0.47615314],
        [ 0.6111526 ,  1.4350855 ,  1.5237921 ,  0.4349823 ],
        [ 0.6082085 ,  0.15383737, -1.4190274 ,  1.0250797 ],
        [-0.9086777 ,  0.5417999 , -0.71691895,  0.00940917],
        [-0.09659185,  0.47538605, -0.41471446, -1.1005498 ],
        [-2.1820064 ,  1.7364386 ,  2.8513548 ,  0.8229001 ]],
       dtype=float32)
 array([[-0.67307454, -0.11945027,  0.9255997 ,  2.4126503 ],
        [ 0.7464948 ,  1.2977355 ,  0.8229865 ,  1.7041056 ],
        [-0.0643647 ,  0.64491385,  1.089884  ,  1.6388392 ],
        [-0.00347048, -0.7640366 ,  0.7786227 , -0.5764215 ]],
       dtype=float32)]
```

targets:(3,)

```
[array([5, 5],
dtype=int64)

array([2, 0],
dtype=int64)

 array([3],
dtype=int64)
```

logits0:(18, 8) = dense(outputs,num_classes=8)

```
[[-0.9280218  -1.8045325  -0.68495935  0.5426107  -1.8653798   4.110873  -1.7798609   2.2875605 ]
 [-0.14943197 -1.9516745   0.2661252   1.0902121  -1.9360601   1.4300921  -1.9269892   3.1968923 ]
 [-0.93730175 -1.793033   -0.51408404  0.7921333  -1.8442494   3.5639648  -1.743907    2.4544852 ]
 [-1.77
 [ 0.32
 [ 0.32
 [ 1.77
 [ 1.61
 [ 2.00
 [ 2.71
 [ 2.72
 [ 2.77
 [-0.73
 [-1.00
 [-1.44
 [-1.68
 [ 0.32
 [ 0.32
```

logits1:(3, 6, 8) = reshape(logits0, [b=3,-1, num_classes=8]

```
[[[-0.9280218  -1.8045325  -0.68495935  0.5426107  -1.8653798   4.110873  -1.7798609   2.2875605 ]
  [-0.14
  [-0.93
  [-1.77
  [ 0.32
  [ 0.32

 [[ 1.77
  [ 1.61
  [ 2.00
  [ 2.71
  [ 2.72
  [ 2.77

 [[-0.73
  [-1.00
  [-1.44
  [-1.68
  [ 0.32
  [ 0.32
```

logits:(6, 3, 8) =transpose(logits1, (1,0,2))

```
[[[-0.9280218  -1.8045325  -0.68495935  0.5426107  -1.8653798   4.110873  -1.7798609   2.2875605 ]
  [ 1.7714039  -1.6429461   1.7430081  -0.7886632  -1.6467034  -0.7623123 -1.6906673   3.0249548 ]
  [-0.7359055  -2.0762694   0.2817074   2.5182047  -2.0220516   0.52489704 -2.0259986   3.742854  ]]

 [[-0.14943197 -1.9516745   0.2661252   1.0902121  -1.9360601   1.4300921  -1.9269892   3.1968923 ]
  [ 1.6112201  -1.8933824   1.8283298   0.5572848  -1.8310149  -1.934924   -1.9494001   3.7281141 ]
  [-1.0007772  -2.2518601   0.232059    3.6622834  -2.1562378  -0.14092624 -2.2268133   4.1684256 ]]

 [[-0.93730175 -1.793033   -0.51408404  0.7921333  -1.8442494   3.5639648  -1.743907    2.4544852 ]
  [ 2.0065188  -1.6460155   2.0205228  -0.5262842  -1.6335787  -1.6081942  -1.7020456   3.1967413 ]
  [-1.4438432  -2.3001847   0.03063124  4.3065047  -2.1964982  -0.06661761 -2.2551336   4.288627  ]]

 [[-1.77378
  [ 2.71857
  [-1.68649

 [[ 0.32713
  [ 2.72935
  [ 0.32713

 [[ 0.32713
  [ 2.77553
  [ 0.32713
```

ouputs0:(3, 6, 5)=rnn(lstm(num_units=5),inputs,seq_len=[4,6,4]])

```
[[[ 0.22251584 -0.25205317  0.45003903  0.11799399  0.6036431 ]
  [ 0.43224868 -0.018
  [ 0.20574221 -0.360
  [ 0.58139867 -0.470
  [ 0.          0.
  [ 0.          0.

 [[-0.26760483  0.442
  [ 0.15492901  0.482
  [-0.27329874  0.393
  [-0.44105944  0.635
  [-0.33494404  0.714
  [-0.13126063  0.811

 [[ 0.5672871  -0.133
  [ 0.8167351  -0.094
  [ 0.90435195 -0.249
  [ 0.69125116 -0.467
  [ 0.          0.
  [ 0.          0.
```

ouputs:(18, 5) =reshape(output0, [-1.num_hidden=5])

```
[[ 0.22251584 -0.25205317  0.45003903  0.11799399  0.6036431 ]
 [ 0.43224868 -0.01831267  0.14647628 -0.02898286 -0.11386187]
 [ 0.20574221 -0.36022937  0.42626762  0.07788736  0.38363698]
 [ 0.58139867 -0.470982    0.66224843  0.23392873  0.8179126 ]
 [ 0.          0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.          0.        ]
 [-0.26760483  0.4420762  -0.44328085 -0.17935988 -0.37502208]
 [ 0.15492901  0.4820558  -0.35484442 -0.5261515  -0.75339603]
 [-0.27329874  0.3937921  -0.524049   -0.49579674 -0.57045555]
 [-0.44105944  0.63578165 -0.7666836  -0.4011293  -0.6473631 ]
 [-0.33494404  0.71433866 -0.7929684  -0.37143773 -0.39454642]
 [-0.13126063  0.8114905  -0.83039004 -0.62812567  0.18657665]
 [ 0.5672871  -0.13315794  0.47310093 -0.20868397 -0.6101549 ]
 [ 0.8167351  -0.0947417   0.6788795  -0.5986903  -0.8386651 ]
 [ 0.90435195 -0.24913134  0.8525947  -0.682692   -0.93658966]
 [ 0.69125116 -0.46754947  0.8683767  -0.43825883 -0.73035365]
 [ 0.          0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.          0.        ]]
```

Decoded,log_ropb=ctc_greedy_decoder(logits, seq_len=[4,6,4])

Decoded[0]

```
SparseTensorValue(
    indices=array(
      [[0, 0],
       [0, 1],
       [1, 0],
       [2, 0]], dtype=int64),
    values=array([5, 5, 0, 3], dtype=int64),
    dense_shape=array([3, 2], dtype=int64))
```

dense_decoded:=sparce_tensor_to_dense(Decoded[0],dfault_value=-1)

```
[[ 5  5]
 [ 0 -1]
 [ 3 -1]]
```

Train_ler =edit_distance(decoded[0],targets))

0.167

targets	Decoded[0]	
[[5, 5],	[[5, 5],	no error
[2, 0],	[0, -1],	1 deletion
[3]]	[3]]	no error

4.3 훈련 및 검증

```
#train CTC ASR model
with tf.Session(graph=graph) as session:
    # Initializate the weights and biases
    tf.global_variables_initializer().run()
    #training
    for curr_epoch in range(num_epochs):
        train_cost = train_ler = 0
        start = time.time()

        #train one epoch
        for batch in range(num_batches_per_epoch):
            feed = {inputs  : train_inputs,
                    targets : train_targets,
                    seq_len : train_seq_len}
            batch_cost, _ = session.run([cost, optimizer], feed)
            train_cost  += batch_cost*batch_size
            train_ler   += session.run(ler, feed_dict=feed)*batch_size
        train_cost  /= num_examples
        train_ler   /= num_examples

        #validate model with vatation data
```

```
        #validate model with vatation data
        val_feed = {inputs  : val_inputs,
                    targets : val_targets,
                    seq_len : val_seq_len}
        val_cost, val_ler, predected = session.run([cost, ler,decoded[0]], feed_dict=va
l_feed)
        str_predicted=''.join([code2ch[c] for c in predected[1]])

        log = "Epoch {}/{}, train_cost = {:.3f}, train_ler = {:.3f}, val_cost = {:.3f}, v
al_ler = {:.3f}, time = {:.3f},  \t{}"
        print(log.format(curr_epoch+1, num_epochs, train_cost, train_ler,val_cost, va
l_ler, time.time() - start, str_predicted))

    # Decoding
    decoded_res = session.run(decoded[0], feed_dict=val_feed)
    str_decoded=''.join([code2ch[c] for c in decoded_res[1]])
    print('Original:\n%s' % original)
    print('Decoded:\n%s' % str_decoded)
```

4.3 훈련 및 검증

```
#train CTC ASR model
with tf.Session(graph=graph) as
    # Initializate the weights and
    tf.global_variables_initializer
    #training
    for curr_epoch in range(num_
        train_cost = train_ler = 0
        start = time.time()

        #train one epoch
        for batch in range(num_ba
            feed = {inputs  : train_in
                    targets : train_targe
                    seq_len : train_seq_
            batch_cost, _ = session.r
            train_cost  += batch_cos
            train_ler   += session.run
        train_cost  /= num_exampl
        train_ler   /= num_example

        #validate model with vatat
```

```
Epoch 1/100, train_cost = 744.466, train_ler = 1.000, val_cost = 165.979, val_ler = 1.000, time = 1.378,
Epoch 2/100, train_cost = 165.979, train_ler = 1.000, val_cost = 228.623, val_ler = 1.000, time = 0.633,
Epoch 3/100, train_cost = 228.623, train_ler = 0.808, val_cost = 371.199, val_ler = 0.808, time = 0.590,
Epoch 4/100, train_cost = 371.199, train_ler = 0.981, val_cost = 256.641, val_ler = 0.981, time = 0.846,
Epoch 5/100, train_cost = 256.641, train_ler = 0.808, val_cost = 288.962, val_ler = 0.808, time = 0.678,
Epoch 6/100, train_cost = 288.962, train_ler = 0.846, val_cost = 238.940, val_ler = 0.846, time = 0.869,
Epoch 7/100, train_cost = 238.940, train_ler = 0.808, val_cost = 166.978, val_ler = 0.808, time = 0.610,
Epoch 8/100, train_cost = 166.978, train_ler = 0.750, val_cost = 142.904, val_ler = 0.750, time = 0.570,        a a aaa aa aaa aa
Epoch 9/100, train_cost = 142.904, train_ler = 0.827, val_cost = 155.726, val_ler = 0.827, time = 0.617,       aaaaairaaaaaa
Epoch 10/100, train_cost = 155.726, train_ler = 0.750, val_cost = 157.334, val_ler = 0.750, time = 0.744,      aaraaaasiraraaararar
Epoch 11/100, train_cost = 157.334, train_ler = 0.885, val_cost = 150.976, val_ler = 0.885, time = 0.775,      rrrirrrr
Epoch 12/100, train_cost = 150.976, train_ler = 0.942, val_cost = 152.543, val_ler = 0.942, time = 0.733,      siy
Epoch 13/100, train_cost = 152.543, train_ler = 0.904, val_cost = 145.110, val_ler = 0.904, time = 0.475,          y
Epoch 14/100, train_cost = 145.110, train_ler = 0.827, val_cost = 136.122, val_ler = 0.827, time = 0.550,
Epoch 15/100, train_cost = 136.122, train_ler = 0.808, val_cost = 136.542, val_ler = 0.808, time = 0.540,
Epoch 16/100, train_cost = 136.542, train_ler = 0.808, val_cost = 138.777, val_ler = 0.808, time = 0.561,
Epoch 17/100, train_cost = 138.777, train_ler = 0.808, val_cost = 139.396, val_ler = 0.808, time = 0.799,
Epoch 18/100, train_cost = 139.396, train_ler = 0.808, val_cost = 138.521, val_ler = 0.808, time = 0.644,
Epoch 19/100, train_cost = 138.521, train_ler = 0.808, val_cost = 136.033, val_ler = 0.808, time = 0.588,
Epoch 20/100, train_cost = 136.033, train_ler = 0.808, val_cost = 132.465, val_ler = 0.808, time = 0.632,
Epoch 21/100, train_cost = 132.465, train_ler = 0.827, val_cost = 129.796, val_ler = 0.827, time = 0.714,
Epoch 22/100, train_cost = 129.796, train_ler = 0.808, val_cost = 129.092, val_ler = 0.808, time = 0.586,         a aaaa
Epoch 23/100, train_cost = 129.092, train_ler = 0.788, val_cost = 128.889, val_ler = 0.788, time = 0.597,       a    aaae
Epoch 24/100, train_cost = 128.889, train_ler = 0.769, val_cost = 127.843, val_ler = 0.769, time = 0.578,      a a a a  aaaae
Epoch 25/100, train_cost = 127.843, train_ler = 0.750, val_cost = 126.115, val_ler = 0.750, time = 0.784,      a a  i reaaae
Epoch 26/100, train_cost = 126.115, train_ler = 0.769, val_cost = 124.098, val_ler = 0.769, time = 0.601,       a i eaaale
Epoch 27/100, train_cost = 124.098, train_ler = 0.808, val_cost = 121.595, val_ler = 0.808, time = 0.651,       i ersale
Epoch 28/100, train_cost = 121.595, train_ler = 0.788, val_cost = 118.714, val_ler = 0.788, time = 0.663,       i ers le
Epoch 29/100, train_cost = 118.714, train_ler = 0.769, val_cost = 116.286, val_ler = 0.769, time = 0.495,       i e   le
Epoch 30/100, train_cost = 116.286, train_ler = 0.750, val_cost = 114.797, val_ler = 0.750, time = 0.537,        i e    le
Epoch 31/100, train_cost = 114.797, train_ler = 0.750, val_cost = 113.348, val_ler = 0.750, time = 0.591,        i  e    le
Epoch 32/100, train_cost = 113.348, train_ler = 0.673, val_cost = 110.794, val_ler = 0.673, time = 0.524,        i rea a   l lee
```

4.3 훈련 및 검증

```
#train CTC ASR model
with tf.Session(graph=grap
    # Initializate the weights
    tf.global_variables_initi
    #training
    for curr_epoch in range(
        train_cost = train_ler
        start = time.time()

        #train one epoch
        for batch in range(nur
            feed = {inputs  : tra
                    targets : train_
                    seq_len : train
            batch_cost, _ = sess
            train_cost  += batch
            train_ler   += sessio
        train_cost  /= num_ex
        train_ler   /= num_exa

        #validate model with
```

```
Epoch 35/100, train_cost = 104.926, train_ler = 0.731, val_cost = 102.946, val_ler = 0.731, time = 0.665,          in reaal lee
Epoch 36/100, train_cost = 102.946, train_ler = 0.712, val_cost = 101.142, val_ler = 0.712, time = 0.590,        h h  in geaal lee
Epoch 37/100, train_cost = 101.142, train_ler = 0.731, val_cost = 99.054, val_ler = 0.731, time = 0.597,         h h in geaal ee
Epoch 38/100, train_cost = 99.054, train_ler = 0.654, val_cost = 96.705, val_ler = 0.654, time = 0.534,         h h duin geaaral ee
Epoch 39/100, train_cost = 96.705, train_ler = 0.615, val_cost = 94.281, val_ler = 0.615, time = 0.470,         h h uri in greaaral ee
Epoch 40/100, train_cost = 94.281, train_ler = 0.558, val_cost = 91.941, val_ler = 0.558, time = 0.567,         h h ur   it in greaaral le
Epoch 41/100, train_cost = 91.941, train_ler = 0.519, val_cost = 89.791, val_ler = 0.519, time = 0.642,         h h u r  it in grea ar al le
Epoch 42/100, train_cost = 89.791, train_ler = 0.519, val_cost = 87.701, val_ler = 0.519, time = 0.573,         h h u r  it in greas r al le
Epoch 43/100, train_cost = 87.701, train_ler = 0.481, val_cost = 85.443, val_ler = 0.481, time = 0.539,         h h u r uit in greasa r al le
Epoch 44/100, train_cost = 85.443, train_ler = 0.481, val_cost = 83.428, val_ler = 0.481, time = 0.727,         h y ur r it in grea ar al le
Epoch 45/100, train_cost = 83.428, train_ler = 0.558, val_cost = 81.449, val_ler = 0.558, time = 0.555,         h ya u rr it in gea aralle
Epoch 46/100, train_cost = 81.449, train_ler = 0.558, val_cost = 79.338, val_ler = 0.558, time = 0.524,       h h a u rrit in geaar alle
Epoch 47/100, train_cost = 79.338, train_ler = 0.558, val_cost = 77.681, val_ler = 0.558, time = 0.503,       h h ha u rrit in geaar alle
Epoch 48/100, train_cost = 77.681, train_ler = 0.481, val_cost = 75.378, val_ler = 0.481, time = 0.605,       h h a u rr it in greahar alleae
Epoch 49/100, train_cost = 75.378, train_ler = 0.442, val_cost = 73.609, val_ler = 0.442, time = 0.585,       h h a u rur uit in greahar allear
Epoch 50/100, train_cost = 73.609, train_ler = 0.423, val_cost = 71.568, val_ler = 0.423, time = 0.567,       h h a u rur uit in greah ar allear
Epoch 51/100, train_cost = 71.568, train_ler = 0.423, val_cost = 69.560, val_ler = 0.423, time = 0.557,       ha h a u rur uit in greah ar al ear
Epoch 52/100, train_cost = 69.560, train_ler = 0.404, val_cost = 67.621, val_ler = 0.404, time = 0.669,       ha ha u rur uit in greah ar al ear
Epoch 53/100, train_cost = 67.621, train_ler = 0.404, val_cost = 65.513, val_ler = 0.404, time = 0.622,       ha ha u rur uit in greah ar al  ear
Epoch 54/100, train_cost = 65.513, train_ler = 0.404, val_cost = 63.296, val_ler = 0.404, time = 0.611,       ha ha u rur uit in grea ahar al  er
Epoch 55/100, train_cost = 63.296, train_ler = 0.404, val_cost = 61.109, val_ler = 0.404, time = 0.585,       ha ha urur uit in grea ahar al  er
Epoch 56/100, train_cost = 61.109, train_ler = 0.385, val_cost = 58.970, val_ler = 0.385, time = 0.689,       he ha u r ur uit in grea ahar al  er
Epoch 57/100, train_cost = 58.970, train_ler = 0.346, val_cost = 56.845, val_ler = 0.346, time = 0.607,       he ha u r ur uit in grea ah ar al  yer
Epoch 58/100, train_cost = 56.845, train_ler = 0.327, val_cost = 54.569, val_ler = 0.327, time = 0.572,       he ha yu r ur uit in grea ah ar al yer
Epoch 59/100, train_cost = 54.569, train_ler = 0.308, val_cost = 52.474, val_ler = 0.308, time = 0.682,       he ha yu r ur uit in grea wah ar al yer
Epoch 60/100, train_cost = 52.474, train_ler = 0.288, val_cost = 50.418, val_ler = 0.288, time = 0.729,       he ha yur ur uit in grea ywah ar al yer
Epoch 61/100, train_cost = 50.418, train_ler = 0.269, val_cost = 47.974, val_ler = 0.269, time = 0.809,       he had yur ur uit in grea ah ar al year
Epoch 62/100, train_cost = 47.974, train_ler = 0.288, val_cost = 45.937, val_ler = 0.288, time = 0.605,       he had yu r r u uit in grea wah ar al year
Epoch 63/100, train_cost = 45.937, train_ler = 0.231, val_cost = 44.037, val_ler = 0.231, time = 0.520,       he had yur dr suit in greayah ar al year
Epoch 64/100, train_cost = 44.037, train_ler = 0.212, val_cost = 41.557, val_ler = 0.212, time = 0.587,       he had yur dr suit in greay ah ar al year
Epoch 65/100, train_cost = 41.557, train_ler = 0.192, val_cost = 40.180, val_ler = 0.192, time = 0.514,       he had yur dr suit in grea wah war al year
Epoch 66/100, train_cost = 40.180, train_ler = 0.173, val_cost = 38.231, val_ler = 0.173, time = 0.588,       he had yur dr suit in greay ah watar al year
Epoch 67/100, train_cost = 38.231, train_ler = 0.173, val_cost = 35.957, val_ler = 0.173, time = 0.503,       he had yur drk suit in greay ah war al year
Epoch 68/100, train_cost = 35.957, train_ler = 0.135, val_cost = 36.546, val_ler = 0.135, time = 0.529,       he had yur drk suit in greas wsh watar al year
Epoch 69/100, train_cost = 36.546, train_ler = 0.154, val_cost = 31.979, val_ler = 0.154, time = 0.704,       he had yur drk uit in greay wah watr al year
Epoch 70/100, train_cost = 31.979, train_ler = 0.154, val_cost = 30.318, val_ler = 0.154, time = 0.633,       he had yur drk suit in greay ahwater al year
Epoch 71/100, train_cost = 30.318, train_ler = 0.077, val_cost = 28.125, val_ler = 0.077, time = 0.521,       he had yur dark suit in greay wash water al year
```

4.3 훈련 및

```
#train CTC ASR model
with tf.Session(graph=graph) as ses
        #validate model with vatatio
        val_feed = {inputs  : val_inp
                    targets : val_targets,
                    seq_len : val_seq_len
        val_cost, val_ler, predected =
l_feed)
        str_predicted=''.join([code2c

        log = "Epoch {}/{}, train_co
al_ler = {:.3f}, time = {:.3f},  \t{}
        print(log.format(curr_epoch+
l_ler, time.time() - start, str_predi

    # Decoding
    decoded_res = session.run(deco
    str_decoded=''.join([code2ch[c
    print('Original:\n%s' % original
    print('Decoded:\n%s' % str_dec
```

```
Epoch 61/100, train_cost = 50.418, train_ler = 0.269, val_cost = 47.974, val_ler = 0.269, time = 0.809,         he had yur ur uit in grea ah ar al year
Epoch 62/100, train_cost = 47.974, train_ler = 0.288, val_cost = 45.937, val_ler = 0.288, time = 0.605,         he had yu r r u uit in grea wah ar al year
Epoch 63/100, train_cost = 45.937, train_ler = 0.231, val_cost = 44.037, val_ler = 0.231, time = 0.520,         he had yur dr suit in greayah ar al year
Epoch 64/100, train_cost = 44.037, train_ler = 0.212, val_cost = 41.557, val_ler = 0.212, time = 0.587,         he had yur dr suit in greay ah ar al year
Epoch 65/100, train_cost = 41.557, train_ler = 0.192, val_cost = 40.180, val_ler = 0.192, time = 0.514,         he had yur dr suit in grea wah war al year
Epoch 66/100, train_cost = 40.180, train_ler = 0.173, val_cost = 38.231, val_ler = 0.173, time = 0.588,         he had yur dr suit in greay ah watar al year
Epoch 67/100, train_cost = 38.231, train_ler = 0.173, val_cost = 35.957, val_ler = 0.173, time = 0.503,         he had yur drk suit in greay ah war al year
Epoch 68/100, train_cost = 35.957, train_ler = 0.135, val_cost = 36.546, val_ler = 0.135, time = 0.529,         he had yur drk suit in greas wsh watar al year
Epoch 69/100, train_cost = 36.546, train_ler = 0.154, val_cost = 31.979, val_ler = 0.154, time = 0.704,         he had yur drk uit in greay wah watr al year
Epoch 70/100, train_cost = 31.979, train_ler = 0.154, val_cost = 30.318, val_ler = 0.154, time = 0.633,         he had yur drk suit in greay ahwater al year
Epoch 71/100, train_cost = 30.318, train_ler = 0.077, val_cost = 28.125, val_ler = 0.077, time = 0.521,         he had yur dark suit in greay wash water al year
Epoch 72/100, train_cost = 28.125, train_ler = 0.077, val_cost = 26.318, val_ler = 0.077, time = 0.652,         she had yur drk suit in greay wash water al year
Epoch 73/100, train_cost = 26.318, train_ler = 0.058, val_cost = 25.031, val_ler = 0.058, time = 0.601,         she had yur dark suit in greay wash water al year
Epoch 74/100, train_cost = 25.031, train_ler = 0.058, val_cost = 24.135, val_ler = 0.058, time = 0.840,         she had your drk suit in greay wash water al year
Epoch 75/100, train_cost = 24.135, train_ler = 0.038, val_cost = 21.997, val_ler = 0.038, time = 0.680,         she had your dark suit in greay wash water al year
Epoch 76/100, train_cost = 21.997, train_ler = 0.038, val_cost = 20.913, val_ler = 0.038, time = 0.613,         she had your dark suit in greay wash water al year
Epoch 77/100, train_cost = 20.913, train_ler = 0.038, val_cost = 19.550, val_ler = 0.038, time = 0.701,         she had your drk suit in greasy wash water al year
Epoch 78/100, train_cost = 19.550, train_ler = 0.019, val_cost = 18.095, val_ler = 0.019, time = 0.577,         she had your dark suit in greasy wash water al year
Epoch 79/100, train_cost = 18.095, train_ler = 0.038, val_cost = 16.746, val_ler = 0.038, time = 0.698,         she had your dark suit in greay wash water al year
Epoch 80/100, train_cost = 16.746, train_ler = 0.038, val_cost = 15.992, val_ler = 0.038, time = 0.586,         she had your dark suit in greay wash water al year
Epoch 81/100, train_cost = 15.992, train_ler = 0.019, val_cost = 14.615, val_ler = 0.019, time = 0.539,         she had your dark suit in greasy wash water al year
Epoch 82/100, train_cost = 14.615, train_ler = 0.019, val_cost = 13.934, val_ler = 0.019, time = 0.734,         she had your dark suit in greasy wash water al year
Epoch 83/100, train_cost = 13.934, train_ler = 0.019, val_cost = 12.856, val_ler = 0.019, time = 0.606,         she had your dark suit in greasy wash water al year
Epoch 84/100, train_cost = 12.856, train_ler = 0.000, val_cost = 12.267, val_ler = 0.000, time = 0.581,         she had your dark suit in greasy wash water all year
Epoch 85/100, train_cost = 12.267, train_ler = 0.000, val_cost = 11.387, val_ler = 0.000, time = 0.668,         she had your dark suit in greasy wash water all year
Epoch 86/100, train_cost = 11.387, train_ler = 0.000, val_cost = 10.687, val_ler = 0.000, time = 0.662,         she had your dark suit in greasy wash water all year
Epoch 87/100, train_cost = 10.687, train_ler = 0.000, val_cost = 10.279, val_ler = 0.000, time = 0.552,         she had your dark suit in greasy wash water all year
Epoch 88/100, train_cost = 10.279, train_ler = 0.000, val_cost = 9.491, val_ler = 0.000, time = 0.852,         she had your dark suit in greasy wash water all year
Epoch 89/100, train_cost = 9.491, train_ler = 0.000, val_cost = 9.062, val_ler = 0.000, time = 0.632,    she had your dark suit in greasy wash water all year
Epoch 90/100, train_cost = 9.062, train_ler = 0.000, val_cost = 8.509, val_ler = 0.000, time = 0.669,    she had your dark suit in greasy wash water all year
Epoch 91/100, train_cost = 8.509, train_ler = 0.000, val_cost = 8.000, val_ler = 0.000, time = 0.519,    she had your dark suit in greasy wash water all year
Epoch 92/100, train_cost = 8.000, train_ler = 0.000, val_cost = 7.632, val_ler = 0.000, time = 0.667,    she had your dark suit in greasy wash water all year
Epoch 93/100, train_cost = 7.632, train_ler = 0.000, val_cost = 7.154, val_ler = 0.000, time = 0.534,    she had your dark suit in greasy wash water all year
Epoch 94/100, train_cost = 7.154, train_ler = 0.000, val_cost = 6.807, val_ler = 0.000, time = 0.564,    she had your dark suit in greasy wash water all year
Epoch 95/100, train_cost = 6.807, train_ler = 0.000, val_cost = 6.437, val_ler = 0.000, time = 0.517,    she had your dark suit in greasy wash water all year
Epoch 96/100, train_cost = 6.437, train_ler = 0.000, val_cost = 6.128, val_ler = 0.000, time = 0.663,    she had your dark suit in greasy wash water all year
Epoch 97/100, train_cost = 6.128, train_ler = 0.000, val_cost = 5.828, val_ler = 0.000, time = 0.594,    she had your dark suit in greasy wash water all year
Epoch 98/100, train_cost = 5.828, train_ler = 0.000, val_cost = 5.558, val_ler = 0.000, time = 0.580,    she had your dark suit in greasy wash water all year
Epoch 99/100, train_cost = 5.558, train_ler = 0.000, val_cost = 5.303, val_ler = 0.000, time = 0.507,    she had your dark suit in greasy wash water all year
Epoch 100/100, train_cost = 5.303, train_ler = 0.000, val_cost = 5.042, val_ler = 0.000, time = 0.505,         she had your dark suit in greasy wash water all year
Original:
she had your dark suit in greasy wash water all year
Decoded:
she had your dark suit in greasy wash water all year
```

4.4. Full code

- Package and hyper parameters

```
import time
import scipy.io.wavfile as wav
import numpy as np
from python_speech_features import mfcc
from utils import sparse_tuple_from as sparse_tuple_from

import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
from tensorflow.python.client import device_lib
device_lib.list_local_devices() # GPU 활성화
gpus = tf.config.experimental.list_physical_devices('GPU')
print(f'GPUs {gpus}')
```

```
# Some configs
alphabet=[' ']+[chr(c) for c in range(ord('a'),ord('z')+1)] #[' ' 'a' 'b' ...'z']
ch2code={ch:c for c,ch in enumerate(alphabet)}
code2ch={c:ch for c,ch in enumerate(alphabet)}
#{' ': 0, 'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, ...}

# Hyper-parameters
num_features = 13               # mffc 차수
num_units=50                    # Number of units in the LSTM cell
num_classes=len(alphabet)       #
num_layers = 1                  # lstm rnn layer

num_epochs = 100                #
batch_size = 1                  #
initial_learning_rate = 1e-2
momentum = 0.9                  # training
num_examples = 1                # max example number
num_batches_per_epoch = int(num_examples/batch_size)
```

4.4. Full code

- Dataset 생성

```
# Loading the audio data and create train dataset
audio_filename = 'LDC93S1.wav'
fs, audio = wav.read(audio_filename)                    #16000, 42797
inputs = mfcc(audio, samplerate=fs)                     #(291,13)
train_inputs = np.reshape(inputs,(1,-1,13))             #(1,291,13)
train_inputs = (train_inputs - np.mean(train_inputs))/np.std(train_inputs) #정규화
train_seq_len = [train_inputs.shape[1]]                 #[291]


# Loading the text script and and create encoded targets dataset
target_filename ='LDC93S1.txt'
lines    = open(target_filename, 'r').readlines()       # '0 46797 She had your dark suit in greasy wash water all year.\n'
original = ' '.join(lines[0].strip().lower().split(' ')[2:]).replace('.', '')
                                                        # 'she had your dark suit in greasy wash water all year'
targets  = [ch2code[c] for c in original]               #[19, 8, 5, 0, 8, 1, 4, ...18]

# Creating sparse representation to feed the placeholder
train_targets = sparse_tuple_from([targets])
                                                        #(array([[ 0,  0],[0,1],..[0,51]], dtype=int64),      #(52,2)
                                                        # array([19,  8,  5,  0,...,1, 18],dtype=int64),      #(52,)
                                                        # dense_shape=array([ 1, 52], dtype=int64))     #(2,)
# create validation dataset
val_inputs, val_targets, val_seq_len = train_inputs, train_targets, train_seq_len
```

4.4. Full code

- CTC ASR Model

```
#create CTC ASR model
graph = tf.Graph()
with graph.as_default():
    # e.g: log filter bank or MFCC features
    # Has size [batch_size, max_stepsize, num_features], but the
    # batch_size and max_stepsize can vary along each step
    inputs = tf.placeholder(tf.float32, [None, None, num_features])
                                    #(1,291,13)
    # Here we use sparse_placeholder that will generate a
    # SparseTensor required by ctc_loss op.
    targets = tf.sparse_placeholder(tf.int32)   #(1,(52,2),(52,))
    # 1d array of size [batch_size]
    seq_len = tf.placeholder(tf.int32, [None])  #(1,) [291]

    # Defining the LSTM RNN
    cells = []
    for _ in range(num_layers):
        cell = tf.nn.rnn_cell.LSTMCell(num_units)
        cells.append(cell)
    stack = tf.nn.rnn_cell.MultiRNNCell(cells)
            # The second output is the last state and we will no use that
    outputs, _ = tf.nn.dynamic_rnn(stack, inputs, seq_len, dtype=tf.float32)
                                    #(1,291,13)=>(1,291,50)

    shape = tf.shape(inputs)
    batch_s, max_timesteps = shape[0], shape[1]     #1,291

    # Reshaping to apply the same weights over the timesteps
    outputs = tf.reshape(outputs, [-1, num_hidden])             #(1,291,50)=>(291,50)
    logits = tf.layers.dense(outputs,num_classes,activation=None) #(291,28)

    # Reshaping back to the original shape
    logits = tf.reshape(logits, [batch_s, -1, num_classes])     #(1,291,28)

    # Time major
    logits = tf.transpose(logits, (1, 0, 2))                    #(291,1,28)

    loss = tf.nn.ctc_loss(targets, logits, seq_len)             #[14.62]
    cost = tf.reduce_mean(loss)                                 #14.62

    optimizer = tf.train.MomentumOptimizer(
                initial_learning_rate,0.9).minimize(cost)

    # Option 2: tf.nn.ctc_beam_search_decoder
    decoded, log_prob = tf.nn.ctc_greedy_decoder(logits, seq_len)

    # Inaccuracy: label error rate
    ler = tf.reduce_mean(tf.edit_distance(tf.cast(decoded[0], tf.int32),
                                          targets))
```

4.4. Full code

- Train and validate

```
#train CTC ASR model
with tf.Session(graph=graph) as session:
    # Initializate the weights and biases
    tf.global_variables_initializer().run()
    #training
    for curr_epoch in range(num_epochs):
        train_cost = train_ler = 0
        start = time.time()

        #train one epoch
        for batch in range(num_batches_per_epoch):
            feed = {inputs  : train_inputs,
                    targets : train_targets,
                    seq_len : train_seq_len}
            batch_cost, _ = session.run([cost, optimizer], feed)
            train_cost  += batch_cost*batch_size
            train_ler   += session.run(ler, feed_dict=feed)*batch_size
        train_cost  /= num_examples
        train_ler   /= num_examples

        #validate model with vatation data
```

```
        #validate model with vatation data
        val_feed = {inputs  : val_inputs,
                    targets : val_targets,
                    seq_len : val_seq_len}
        val_cost, val_ler, predected = session.run([cost, ler,decoded[0]], feed_dict=va
l_feed)
        str_predicted=''.join([code2ch[c] for c in predected[1]])

        log = "Epoch {}/{}, train_cost = {:.3f}, train_ler = {:.3f}, val_cost = {:.3f}, v
al_ler = {:.3f}, time = {:.3f},  \t{}"
        print(log.format(curr_epoch+1, num_epochs, train_cost, train_ler,val_cost, va
l_ler, time.time() - start, str_predicted))

    # Decoding
    decoded_res = session.run(decoded[0], feed_dict=val_feed)
    str_decoded=''.join([code2ch[c] for c in decoded_res[1]])
    print('Original:\n%s' % original)
    print('Decoded:\n%s' % str_decoded)
```

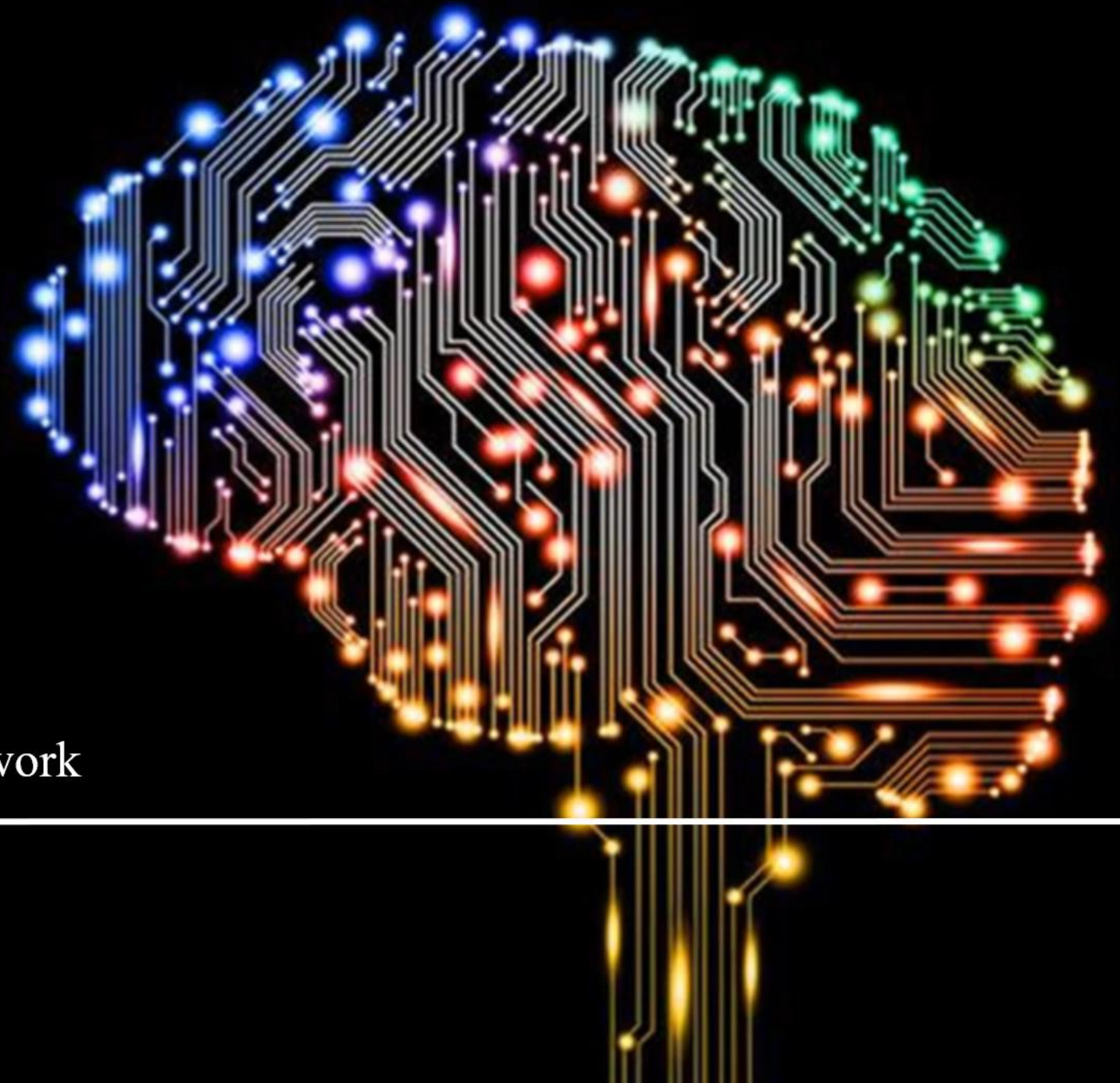
4.4. Full code

```python
import time
import scipy.io.wavfile as wav
import numpy as np
from python_speech_features import mfcc
#from utils import sparse_tuple_from as sparse_tuple_from

import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

from tensorflow.python.client import device_lib
device_lib.list_local_devices() # GPU 활성화
gpus = tf.config.experimental.list_physical_devices('GPU')
print(f'GPUs {gpus}')

def sparse_tuple_from(sequences, dtype=np.int32):
    """Create a sparse representention of x.
    Args:
        sequences: a list of lists of type dtype where each element is a sequence
    Returns:
        A tuple with (indices, values, shape)
    """
    indices = []
    values = []

    for n, seq in enumerate(sequences):
        indices.extend(zip([n]*len(seq), range(len(seq))))
        values.extend(seq)

    indices = np.asarray(indices, dtype=np.int64)
    values = np.asarray(values, dtype=dtype)
    shape = np.asarray([len(sequences), np.asarray(indices).max(0)[1]+1], dtype=np.int64)

    return indices, values, shape

# Some configs
alphabet=[' ']+[chr(c) for c in range(ord('a'),ord('z')+1)]
                #[' ' 'a' 'b' ...'z']
ch2code={ch:c for c,ch in enumerate(alphabet)}
code2ch={c:ch for c,ch in enumerate(alphabet)}
#{' ': 0, 'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, ...

# Hyper-parameters
num_features = 13   #mffc 차수
num_units=50        # Number of units in the LSTM cell
num_classes=len(alphabet) #27
num_hidden = 50     #50  num_units of lstm cell
num_layers = 1      #lstm rnn layer
num_epochs = 100    #200
batch_size = 1      #
initial_learning_rate = 1e-2
momentum = 0.9      # training

num_examples = 1    #max example number
num_batches_per_epoch = int(num_examples/batch_size)

# Loading the audio data and create train dataset
audio_filename = 'LDC93S1.wav'
fs, audio = wav.read(audio_filename)        #16000, 42797
inputs = mfcc(audio, samplerate=fs)         #(291,13)
train_inputs = np.reshape(inputs,(1,-1,13)) #(1,291,13)
train_inputs = (train_inputs - np.mean(train_inputs))/np.std(train_inputs) #정규화
train_seq_len = [train_inputs.shape[1]]     #[291]

# Loading the text script and and create encoded targets dataset
target_filename ='LDC93S1.txt'  #62
lines    = open(target_filename, 'r').readlines()
                # '0 46797 She had your dark suit in greasy wash water all year.\n'
original = ' '.join(lines[0].strip().lower().split(' ')[2:]).replace('.', '')
                # 'she had your dark suit in greasy wash water all year'
targets  = [ch2code[c] for c in original]
                #[19, 8, 5, 0, 8, 1, 4, ...18]
# Creating sparse representation to feed the placeholder
train_targets = sparse_tuple_from([targets])
                #(array([[ 0,  0],[0,1],..[0,51]], dtype=int64), #(52,2)
                # array([19,  8,  5,  0,...,1, 18],dtype=int64), #(52,)
                # dense_shape=array([ 1, 52], dtype=int64))      #(2,)

# create validation dataset
val_inputs, val_targets, val_seq_len = train_inputs, train_targets, \
                                       train_seq_len

#create CTC ASR model
graph = tf.Graph()
with graph.as_default():
    # e.g: log filter bank or MFCC features
    # Has size [batch_size, max_stepsize, num_features], but the
    # batch_size and max_stepsize can vary along each step
    inputs = tf.placeholder(tf.float32, [None, None, num_features])
                                           #(1,291,13)
```

```
    # Here we use sparse_placeholder that will generate a
    # SparseTensor required by ctc_loss op.
    targets = tf.sparse_placeholder(tf.int32)   #(1,(52,2),(52,))
    # 1d array of size [batch_size]
    seq_len = tf.placeholder(tf.int32, [None])  #(1,) [291]

    # Defining the LSTM RNN
    cells = []
    for _ in range(num_layers):
        cell = tf.nn.rnn_cell.LSTMCell(num_units)  # Or LSTMCell(num_units)
        cells.append(cell)
    stack = tf.nn.rnn_cell.MultiRNNCell(cells)#..contrib.rnn.MultiRNNCell(cells)
    # The second output is the last state and we will no use that
    outputs, _ = tf.nn.dynamic_rnn(stack, inputs, seq_len, dtype=tf.float32) #(1,291,13)=>(

    shape = tf.shape(inputs)
    batch_s, max_timesteps = shape[0], shape[1]     #1,291

    # Reshaping to apply the same weights over the timesteps
    outputs = tf.reshape(outputs, [-1, num_hidden])  #(1,291,50)=>(291,50)
    logits = tf.layers.dense(outputs,num_classes,activation=None) #(291,28)

    # Reshaping back to the original shape
    logits = tf.reshape(logits, [batch_s, -1, num_classes])     #(1,291,28)

    # Time major
    logits = tf.transpose(logits, (1, 0, 2))        #(291,1,28)

    loss = tf.nn.ctc_loss(targets, logits, seq_len) #[14.62]
    cost = tf.reduce_mean(loss)                     #14.62

    optimizer = tf.train.MomentumOptimizer(initial_learning_rate,
                                           0.9).minimize(cost)

    # Option 2: tf.nn.ctc_beam_search_decoder
    decoded, log_prob = tf.nn.ctc_greedy_decoder(logits, seq_len)

    # Inaccuracy: label error rate
    ler = tf.reduce_mean(tf.edit_distance(tf.cast(decoded[0], tf.int32),
                                          targets))
```

```
#train CTC ASR model
with tf.Session(graph=graph) as session:
    # Initializate the weights and biases
    tf.global_variables_initializer().run()
    #training
    for curr_epoch in range(num_epochs):
        train_cost = train_ler = 0
        start = time.time()

        #train one epoch
        for batch in range(num_batches_per_epoch):
            feed = {inputs  : train_inputs,
                    targets : train_targets,
                    seq_len : train_seq_len}
            batch_cost, _ = session.run([cost, optimizer], feed)
            train_cost  += batch_cost*batch_size
            train_ler   += session.run(ler, feed_dict=feed)*batch_size
        train_cost  /= num_examples
        train_ler   /= num_examples

        #validate model with vatation data
        val_feed = {inputs  : val_inputs,
                    targets : val_targets,
                    seq_len : val_seq_len}
        val_cost, val_ler, predected = session.run([cost, ler,decoded[0]], feed_dict=val_feed)
        str_predicted=''.join([code2ch[c] for c in predected[1]])

        log = "Epoch {}/{}, train_cost = {:.3f}, train_ler = {:.3f}, val_cost = {:.3f}, val_ler = {:.3f}, time = {:.3f},  \t{}"
        print(log.format(curr_epoch+1, num_epochs, train_cost, train_ler,val_cost, val_ler, time.time() - start, str_predicted))

    # Decoding
    decoded_res = session.run(decoded[0], feed_dict=val_feed)
    str_decoded=''.join([code2ch[c] for c in decoded_res[1]])
    print('Original:\n%s' % original)
    print('Decoded:\n%s' % str_decoded)
```

The End



Deep Learning
Deep Neural Network

Yoon Joong Kim,
Hanbat National University