

Lec12-1

Text Classification and Sentimental Analysis

Tokenizer, Embedding, LSTM, Keras

강의 자료 다운로드

<https://github.com/hongsukyi/RNN-NLP-2021-Fall>

Colab 사용하기

Simple RNN and LSTM

코랩에서 데이터 불러오기

- PC에 있는 데이터 파일을 불러오기
 - ✓ tiny_shakespeare.txt 파일이 있다면

```
# Colab에서 데이터 파일 읽기  
  
from google.colab import files  
myfile = files.upload()
```

파일 선택 tiny-shakespeare.txt
• **tiny-shakespeare.txt**(text/plain) - 1115394 bytes, last modified: 2021. 11. 12. - 100% done
Saving tiny-shakespeare.txt to tiny-shakespeare.txt

- ✓ 파일 선택 이후, 데이터 파일 읽기

```
data = open('./tiny-shakespeare.txt').read()
```

Simple RNN and Text

Simple RNN and LSTM

Embedding layers

- 한 문장이 올바른지 혹은 틀린지를 판별하는 RNN 모델을 만들어 보자
- 단어 임베딩(Word embedding)은 한 단어를 벡터로 변환하는 것이다.
 - ✓ 예, Word2Vec(2013), GloVe(2014)

"This is a small vector"

1) Word embedding 해보자

➤ 여기서는 간단한 방법으로 해보자.

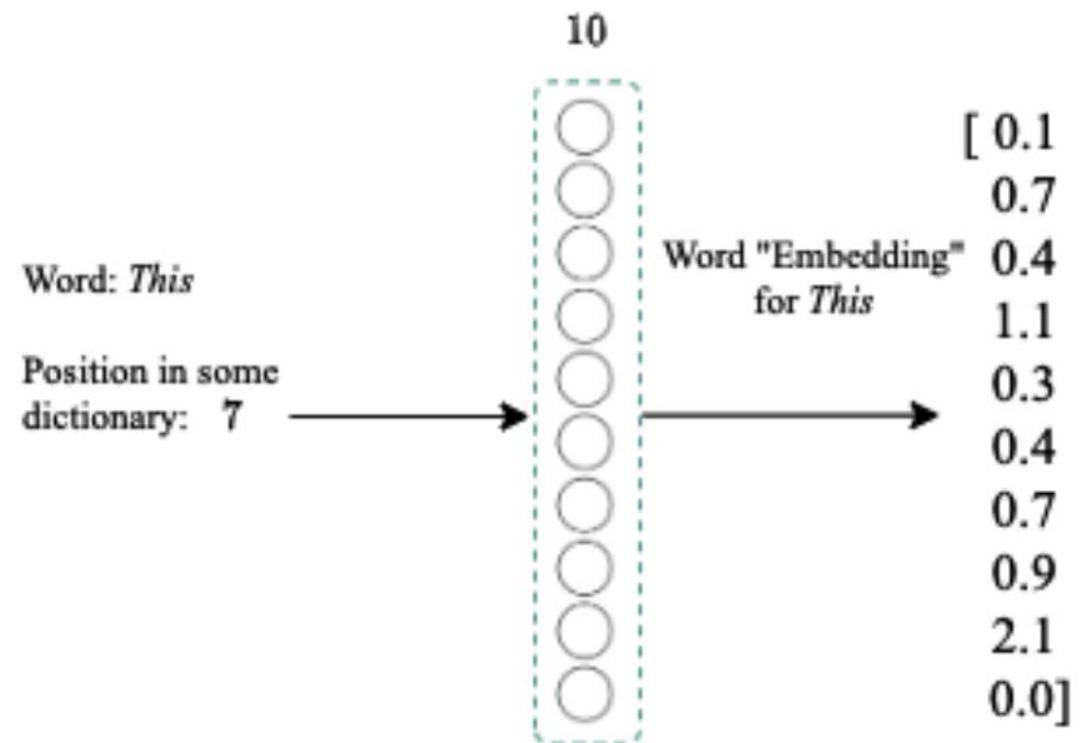
- ✓ 1) RNN에 계산에 사용할 문장의 길이를 정하자.
- ✓ 2) 어휘(vocabulary)에 해당하는 단어 개수를 정하자.
 - 자주 사용되는 단어 순으로 ranking하자
- ✓ 3) word-to-index 단어들을 인덱스로 바꾸자
- ✓ 4) 단어 임베딩의 차원을 결정자.

embedding_dim = 10

This

index : 7,

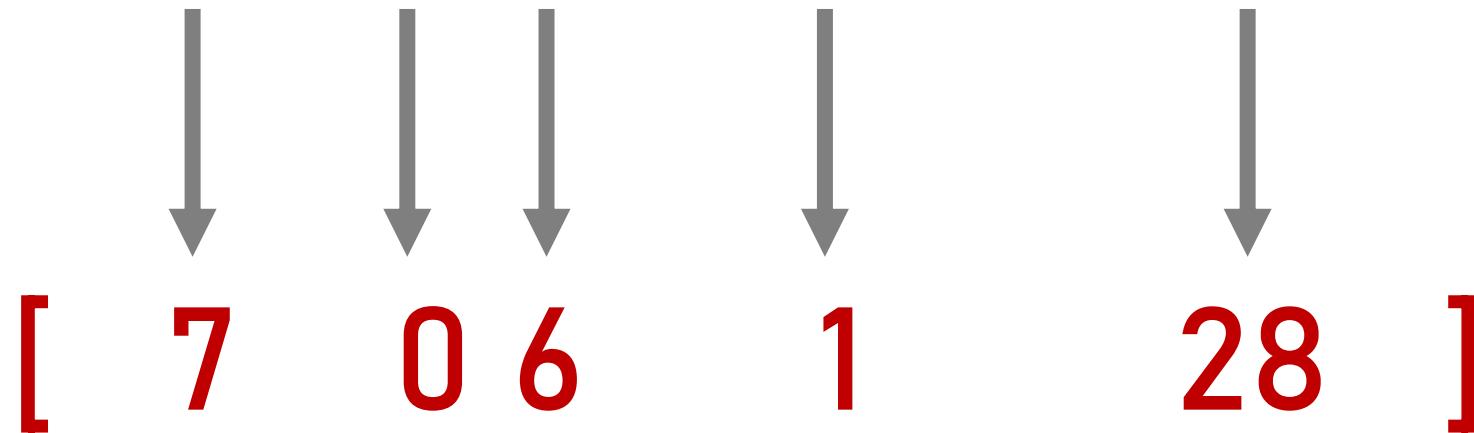
vector size : 10



Sequence of vectors

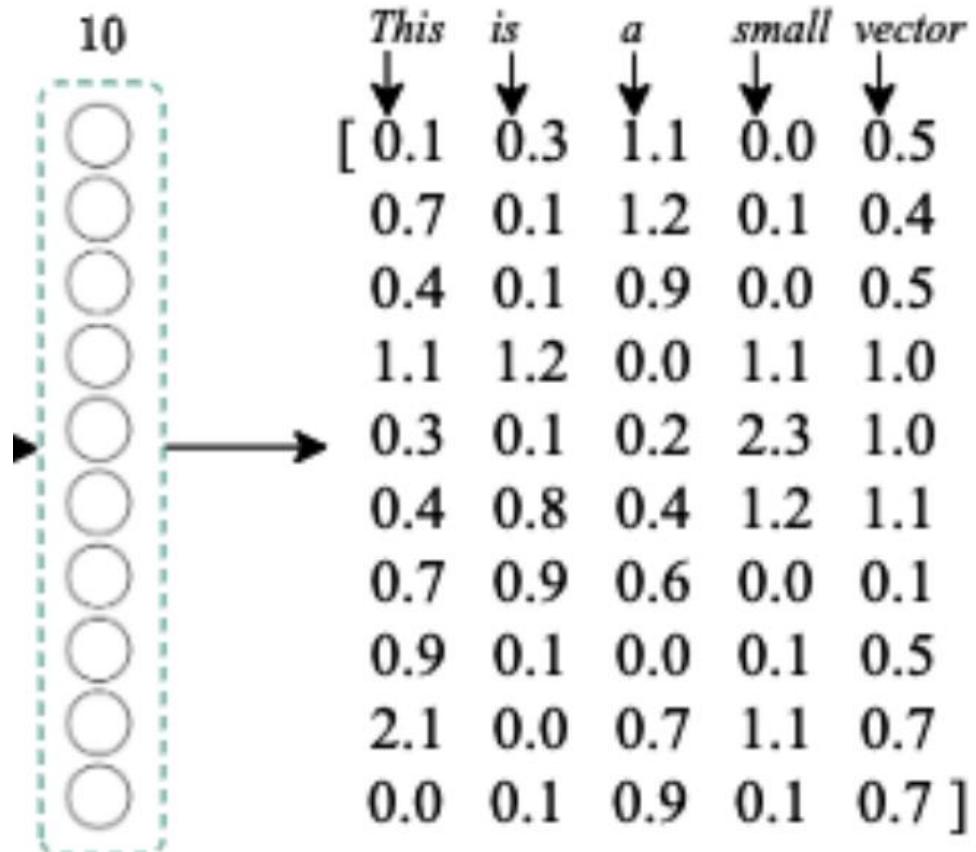
- 보통은 vocabulary 크기는 수 천 개이다. `vocab_size = 30`
- 보통은, 문장들은 5개 이상 단어로 구성된다. `seq_length = 5`
- 임베딩 차원은 작은 모델은 50 차원 정도이다. 좋은 모델은 500에서 1000개 차원도 있다.

"This is a small vector"

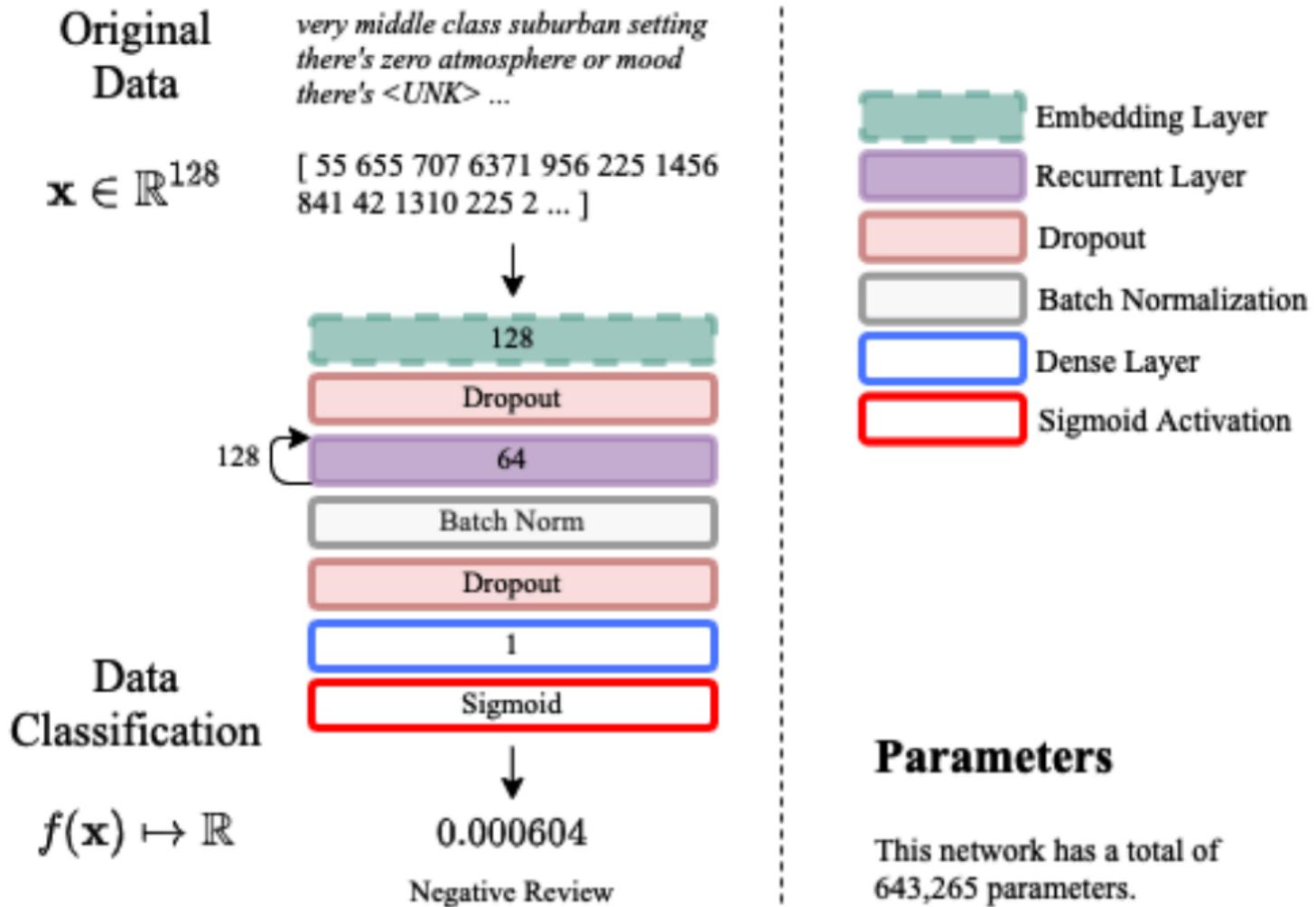


Sequence of word embeddings

vocab_size = 30
embddng_dim = 10
seqnc_lngth = 5



An RNN architecture for the IMDB dataset



Parameters

This network has a total of 643,265 parameters.

Deep Learning for NLP

- Sentiment Analysis : IMDB (Internet Movie Database)
 - ✓ Determining the emotional tone behind a piece of text

The Hunger Games: Mockingjay - Part 2 (2015)

12A | 137 min | Adventure, Sci-Fi | 19 November 2015 (UK)

Your rating: ★★★★★★★★★★ 7.1 /10

Ratings: 7.1/10 from 38,869 users Metascore: 65/100

Reviews: 178 user | 286 critic | from Metacritic.com

As the war of Panem escalates to the destruction of other districts by the Capitol, Katniss Everdeen, the reluctant leader of the rebellion, must bring together an army against President Snow, while all she holds dear hangs in the balance.

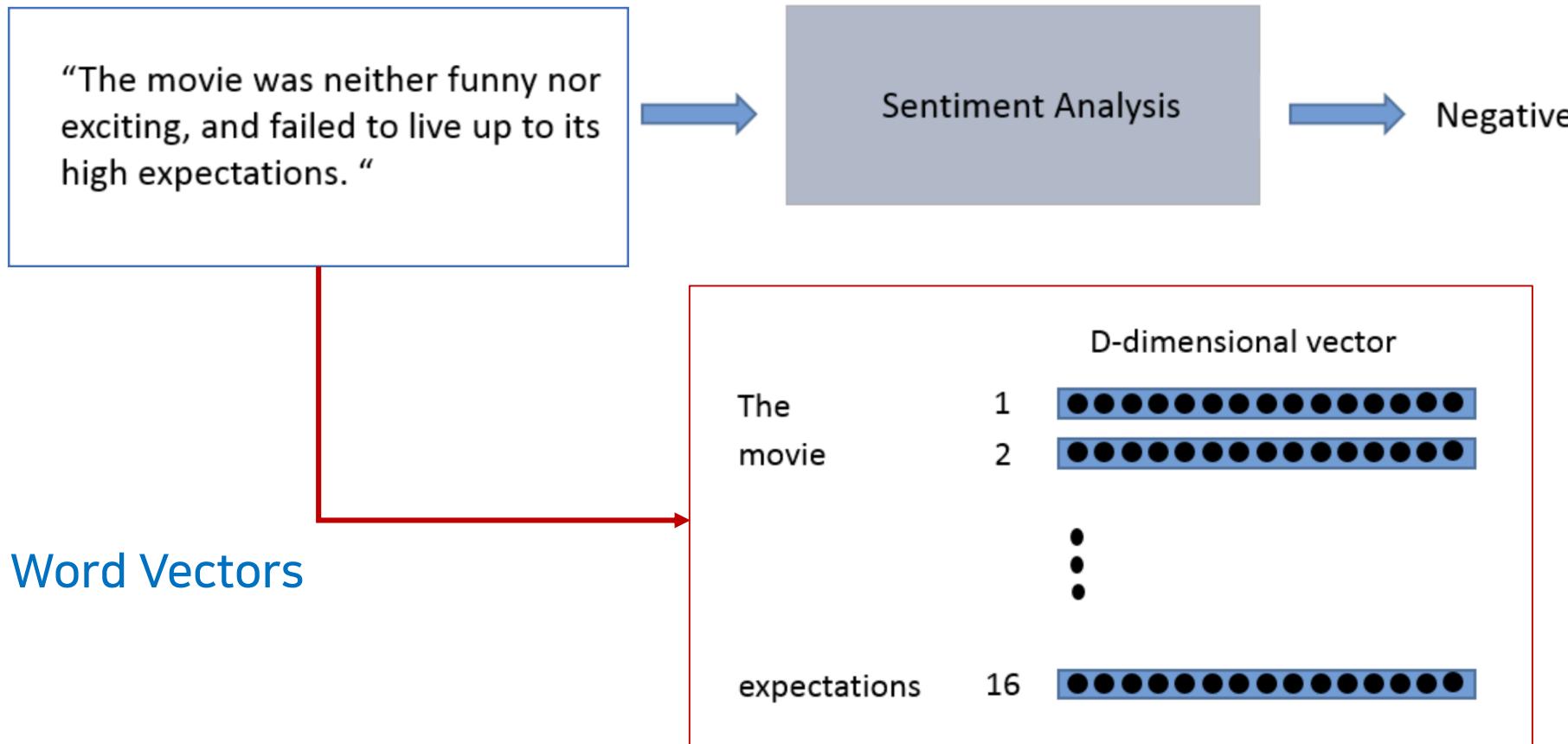
Director: Francis Lawrence

Writers: Peter Craig (screenplay), Danny Strong

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The filming tec...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative

We will need to convert each word in the sentence to a vector.

- When you think of NLP tasks, however, a data pipeline like this may come to mind.

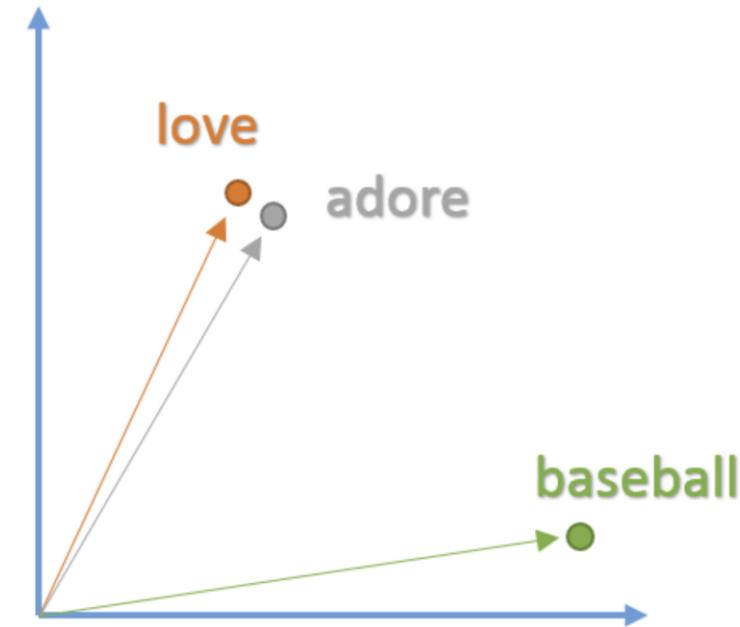


- Word Vectors

Word embedding : Word2Vec

- The vector representation of a word is also known as a word embedding.
- Word2Vec :
 - ✓ the model creates word vectors by looking at the context with which words appear in sentences.

I **love** taking long walks on the beach.
My friends told me that they **love** popcorn.
⋮
The relatives **adore** the baby's cute face.
I **adore** his sense of humor.

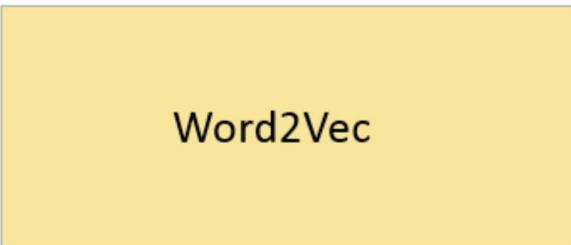


Embedding matrix

- The output of a Word2Vec model is called an embedding matrix
 - ✓ This embedding matrix will contain vectors for every distinct word in the training corpus.

English Wikipedia Corpus

The Annual Reminder continued through July 4, 1969. This final Annual Reminder took place less than a week after the June 28 Stonewall riots, in which the patrons of the Stonewall Inn, a gay bar in Greenwich Village, fought against police who raided the bar. Rodwell received several telephone calls threatening him and the other New York participants, but he was able to arrange for police protection for the chartered bus all the way to Philadelphia. About 45 people participated, including the deputy mayor of Philadelphia and his wife. The dress code was still in effect at the Reminder, but two women from the New York contingent broke from the single-file picket line and held hands. When Kameny tried to break them apart, Rodwell furiously denounced him to onlooking members of the press.
Following the 1969 Annual Reminder, there was a sense, particularly among the younger and more radical participants, that the time for silent picketing had passed. Dissent and dissatisfaction had begun to take new and more emphatic forms in society.¹⁰ The conference passed a resolution drafted by Rodwell, his partner Fred Sargeant, Broidy and Linda Rhodes to move the demonstration from July 4 in Philadelphia to the last weekend in June in New York City, as well as proposing to "other organizations throughout the country..." suggesting that they hold parallel demonstrations on that day" to commemorate the Stonewall riot.



Embedding Matrix

D-dimensional vector

aardvark
apple



⋮

zoo



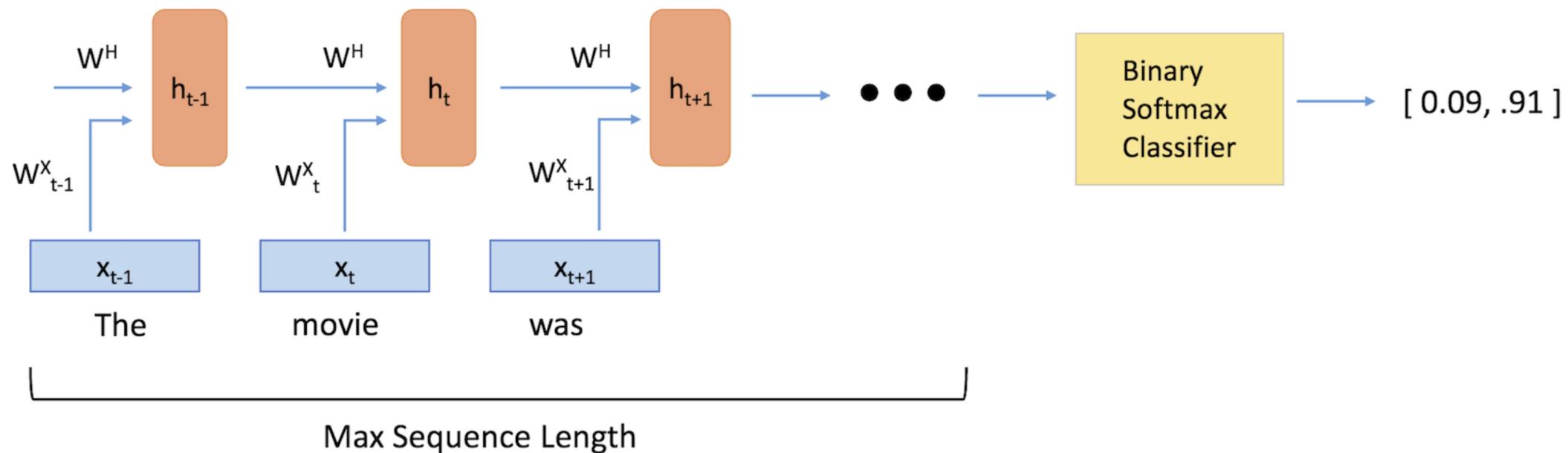
RNN

- In RNNs, each word in an input sequence will be associated with a specific time step.

The movie was ... expectations

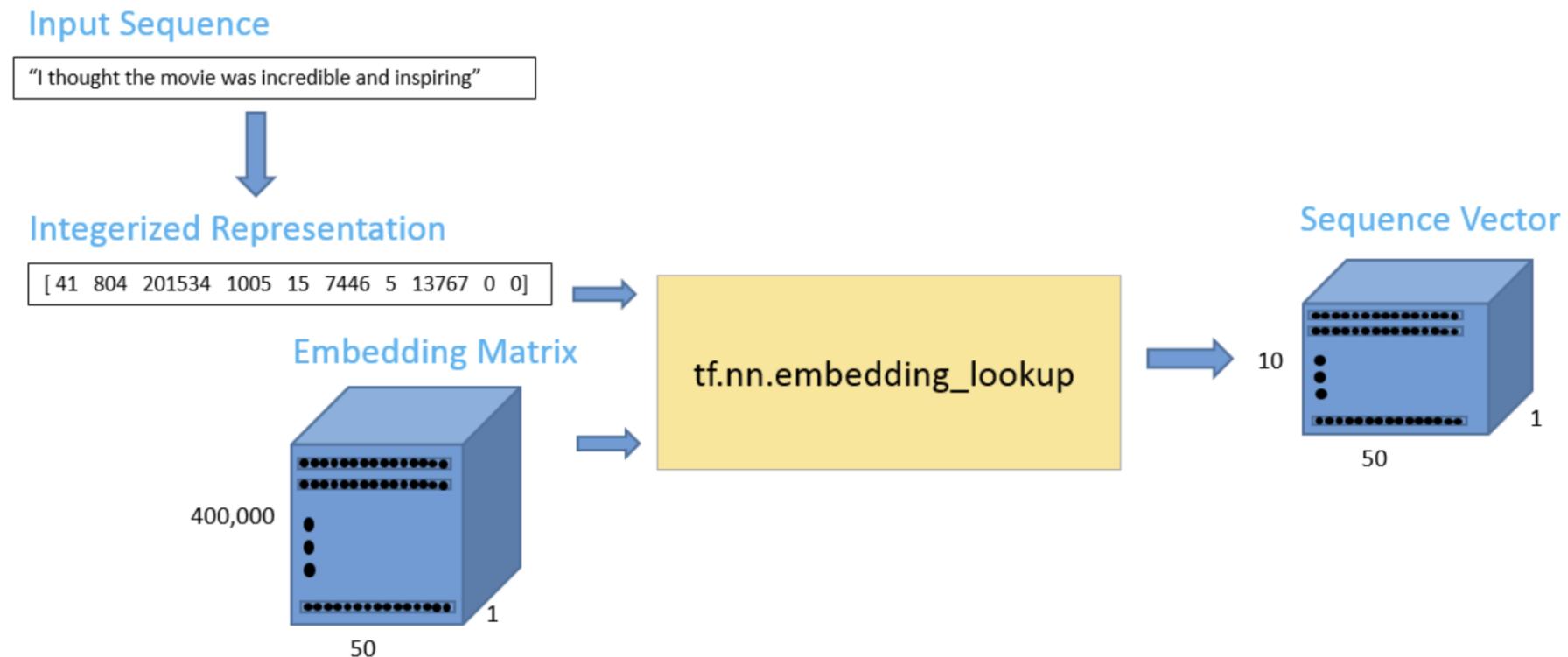
$x_0 \quad x_1 \quad x_2 \quad \dots \quad x_{15}$
t = 0 t = 1 t = 2 t = 15

$$h_t = \sigma(W^H h_{t-1} + W^X x_t)$$



The data pipeline : a simple example

The matrix contain 400,000 word vectors, each with a dimensionality of 50.



The 10 x 50 output should contain the 50 dimensional word vectors for each of the 10 words in the sequence.

Input and Output : Many-to-One

Integerized Inputs

Max Sequence Length

Batch Size

[41 804 201534 1005 15 7446 5 13767 ...]
⋮
[37 14 2407 201534 96 37314 319 7158 ...]

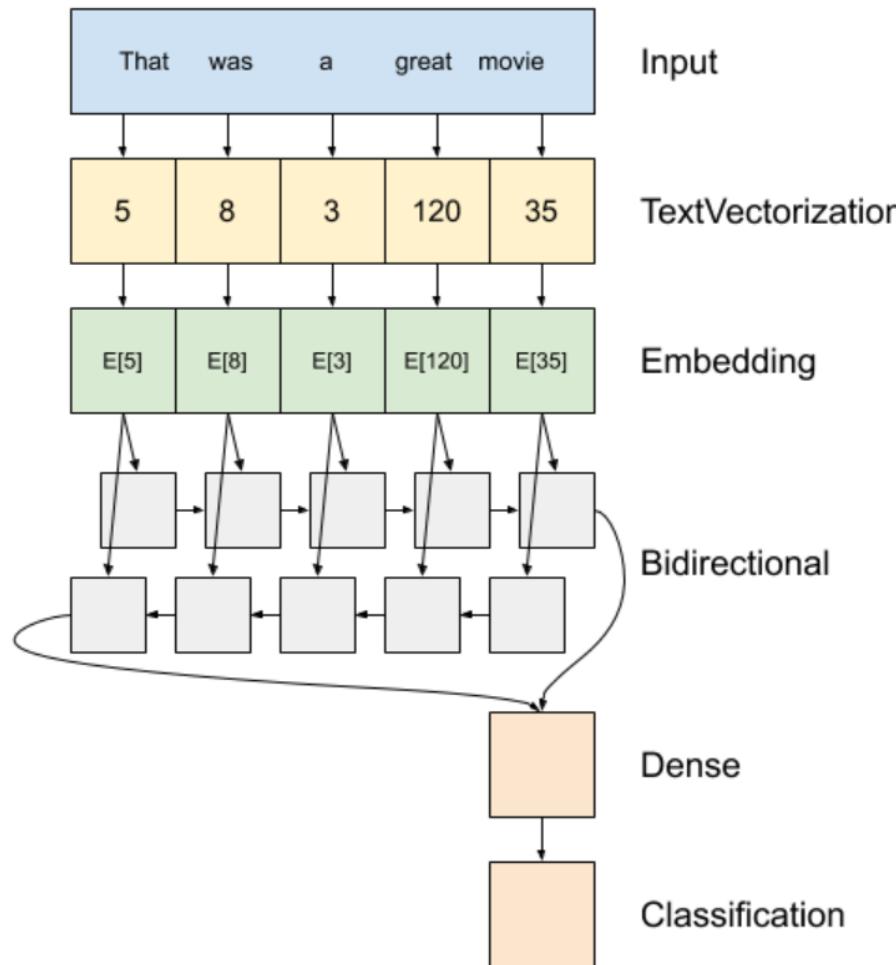
Labels

of Classes

Batch Size

[1, 0]
⋮
[0, 1]
⋮
[1, 0]

Bidirectional LSTM model architecture



Lab: Text Classification with IMDB

IMDB Text Classification

1. Setup

1. Setup

```
import tensorflow as tf

import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, GRU, Bidirectional
from tensorflow.keras.layers import Dropout, BatchNormalization, SimpleRNN
from tensorflow.keras.utils import to_categorical, plot_model
tf.__version__
]: '2.7.0'
```

1. Setup

```
: print("Version: ", tf.__version__)
print("Eager mode: ", tf.executing_eagerly())

print("GPU is", "available" if tf.config.list_physical_devices('GPU') else "NOT AVAILABLE")
```

Version: 2.7.0

Eager mode: True

GPU is available

2. Download the IMDB dataset

2. Download the IMDB dataset

- The reviews (sequences of words) have been converted to sequences of integers
- `vocal_size` ~ 10,000 most frequently occurring words.

```
vocal_size = 10000  
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocal_size)
```

3. Explore the data

3. Explore the data

- Each `y_train`, `y_test` is an integer value of either `0` (negative) or `1` (positive) review.

```
print("Training entries: {}, labels: {}".format(len(X_train), len(y_train)))
```

```
print("Test entries: {}, labels: {}".format(len(X_test), len(y_test)))
```

Training entries: 25000, labels: 25000

Test entries: 25000, labels: 25000

3. Explore the data

- 첫 샘플

```
print(X_train[0])
print('-----')
print(y_train[0])

[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 4
80, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 1
4, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386,
12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6,
22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400,
317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 4
80, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15,
16, 5345, 19, 178, 32]
-----
1
```

3. Explore the data

- Check the first data : Movie reviews may be different lengths.

```
In [7]: len(X_train[0]), len(X_train[1])
```

```
Out[7]: (218, 189)
```

(b) The 5 most frequently used words in the IMDB dataset

```
In [7]: imdb_get_word_index = {}

for key, value in imdb.get_word_index().items():
    imdb_get_word_index[value] = key

for i in range(1, 6):
    print('{}-th word which is used the most frequently = {}'.format(i, imdb_get_word_index[i]))
```

1-th word which is used the most frequently = the
2-th word which is used the most frequently = and
3-th word which is used the most frequently = a
4-th word which is used the most frequently = of
5-th word which is used the most frequently = to

(c) Convert the integers back to words

- Convert the integers back to words

```
: # A dictionary mapping words to an integer index
word_index = imdb.get_word_index()

# The first indices are reserved
word_index = {k:(v+3) for k,v in word_index.items()}
word_index["<PAD>"] = 0
word_index["<START>"] = 1
word_index["<UNK>"] = 2 # unknown
word_index["<UNUSED>"] = 3

reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
```

(c) Convert the integers back to words

```
def decode_review(text):
    return ' '.join([reverse_word_index.get(i, '?') for i in text])
```

- display the text for the first review:

```
decode_review(X_train[0])
```

1]: "<START> this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert <UNK> is an amazing actor and now the same being director <UNK> fat her came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty r emarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for <UNK> and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this definitely was also <UNK> to the t wo little boy's that played the <UNK> of norman and paul they were just brilliant children are often left out of the <UN K> list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised for what they have done don't you think the whole story was so lovely because it wa s true and was someone's life after all that was shared with us all"

4. Prepare the data

- pad the arrays so they all have the same length,

```
seq_len = 256  
print('Before pad_sequences: ', len(X_train[0]))
```

Before pad_sequences: 218

```
X_train = pad_sequences(X_train, maxlen=seq_len, padding = 'pre')  
X_test = pad_sequences(X_test, maxlen=seq_len, padding = 'pre')  
print('After X_train pad_sequences: ', len(X_train[0]))  
print('After X_test pad_sequences: ', len(X_test[0]))
```

After X_train pad_sequences: 256

After X_test pad_sequences: 256

5. Build the model

5. Build the model

- Construct a model using the Embedding layer
-

image.png

```
embedding_dim = 128
```

- The first layer is an Embedding layer.
- The resulting dimensions are: (batch, sequence, embedding, seq_len).

5. Build the model

```
m1 = Sequential()  
  
m1.add(Embedding(vocal_size, embedding_dim, input_length = seq_len))  
m1.add(SimpleRNN(embedding_dim))  
m1.add(Dense(1, activation='sigmoid'))
```

5. Build the model

```
m2 = Sequential()  
  
m2.add(Embedding(vocal_size, embedding_dim, input_length = seq_len))  
m2.add(Bidirectional(LSTM(embedding_dim)))  
m2.add(Dropout(0.2))  
m2.add(Dense(embedding_dim, activation = 'relu'))  
m2.add(BatchNormalization())  
m2.add(Dropout(0.2))  
m2.add(Dense(1, activation='sigmoid'))
```

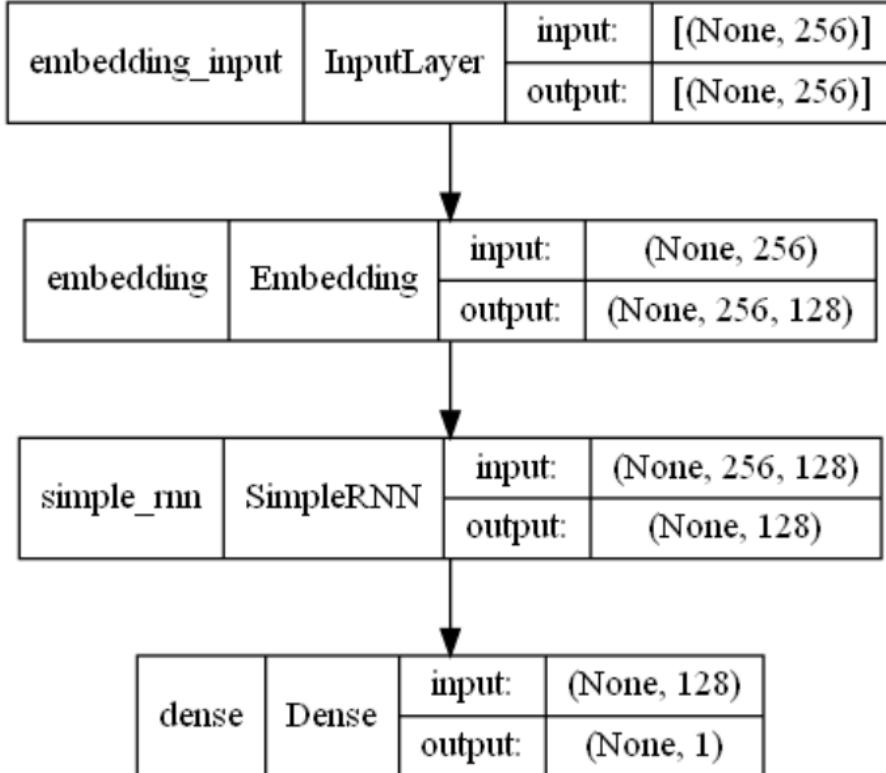
5. Build the model

```
m3 = Sequential()
m3.add(Embedding(input_dim = vocal_size, output_dim = embedding_dim, input_length = seq_len))
m3.add(Bidirectional(LSTM(embedding_dim, return_sequences = True )))
m3.add(Bidirectional(LSTM(embedding_dim)))
m3.add(Dense(embedding_dim, activation = 'relu'))
m3.add(Dropout(0.5))
m3.add(Dense(1, activation = 'sigmoid'))
```

5. Build the model

```
In [18]: model = m1; plot_model(model, show_shapes=True)
```

Out[18]:



Model compile

```
model = m1
```

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['acc'])
```

```
X_train.shape, y_train.shape
```

```
]: ((25000, 256), (25000,))
```

```
num_small = 10000
```

```
pX_train = X_train[num_small:]
py_train = y_train[num_small:]
```

```
x_val = X_train[:num_small]
y_val = y_train[:num_small]
```

```
print(x_val.shape, y_val.shape)
print(pX_train.shape, py_train.shape)
```

```
(10000, 256) (10000,)
(15000, 256) (15000,)
```

6. Train the model

6. Train the model

Colab: Elapsed 2933.763 seconds.

```
: import time
start = time.perf_counter()
|
history = model.fit(X_train, y_train, epochs=10, batch_size=64,
                     validation_data=(x_val, y_val),
                     verbose=1)

elapsed = time.perf_counter() - start
print('=====Elapsed %.3f seconds.' % elapsed)
```

6. Train the model

```
Epoch 4/10
391/391 [=====] - 295s 755ms/step - loss: 0.4769 - acc: 0.7690 - val_loss: 0.3726
- val_acc: 0.8418
Epoch 5/10
391/391 [=====] - 294s 753ms/step - loss: 0.4037 - acc: 0.8231 - val_loss: 0.3475
- val_acc: 0.8663
Epoch 6/10
391/391 [=====] - 291s 745ms/step - loss: 0.3452 - acc: 0.8565 - val_loss: 0.2745
- val_acc: 0.8902
Epoch 7/10
391/391 [=====] - 290s 743ms/step - loss: 0.2879 - acc: 0.8862 - val_loss: 0.2226
- val_acc: 0.9218
Epoch 8/10
391/391 [=====] - 296s 757ms/step - loss: 0.2408 - acc: 0.9102 - val_loss: 0.1882
- val_acc: 0.9374
Epoch 9/10
391/391 [=====] - 288s 737ms/step - loss: 0.2267 - acc: 0.9172 - val_loss: 0.1950
- val_acc: 0.9281
Epoch 10/10
391/391 [=====] - 293s 748ms/step - loss: 0.2421 - acc: 0.9055 - val_loss: 0.1627
- val_acc: 0.9477
=====
Elapsed 2933.763 seconds.
```

7. Evaluate the model

7. Evaluate the model

```
In [27]: X_test.shape, y_test.shape
```

```
Out[27]: ((25000, 256), (25000,))
```

```
In [28]: test_loss, test_accuracy = model.evaluate(X_test, y_test)
```

```
print("Test accuracy: {}".format(test_accuracy))
print("Test loss: {}".format(test_loss))
```

```
782/782 [=====] - 54s 69ms/step - loss: 0.5326 - acc: 0.7910
```

```
Test accuracy: 0.7910400032997131
```

```
Test loss: 0.5326084494590759
```

7. Evaluate the model

In [29]:

```
# Generate predictions  
predictions = model.predict(X_train[0:20])  
print(predictions)
```

```
[[0.95647436]  
 [0.09762806]  
 [0.09510067]  
 [0.9884554 ]  
 [0.08556186]  
 [0.03566658]  
 [0.9776307 ]  
 [0.02483661]  
 [0.9840465 ]  
 [0.05688852]  
 [0.9931352 ]  
 [0.09034158]  
 [0.14482702]  
 [0.01156923]  
 [0.01982516]  
 [0.03456545]  
 [0.9952172 ]  
 [0.6925166 ]  
 [0.05652772]  
 [0.9745355 ]]
```

7. Evaluate the model

```
In [30]: for i in range(2):
    INDEX_FROM=3 # word index offset
    word_to_id = imdb.get_word_index()
    word_to_id = {k:(v+INDEX_FROM) for k,v in word_to_id.items()}
    word_to_id["<PAD>"] = 0
    word_to_id["<START>"] = 1
    word_to_id["<UNK>"] = 2
    word_to_id["<UNUSED>"] = 3

    id_to_word = {value:key for key,value in word_to_id.items()}
    print('=====')
    print(f'Sample = {i} | Length = {len(X_test[i])}')
    print('=====')
    print(' '.join(id_to_word[id] for id in X_test[i] ))
```

7. Evaluate the model

Sample = 1 | Length = 256

a lot of patience because it focuses on mood and character development the plot is very simple and many of the scenes take place on the same set in frances <UNK> the sandy dennis character apartment but the film builds to a disturbing climax br br the characters create an atmosphere <UNK> with sexual tension and psychological <UNK> it's very interesting that robert altman directed this considering the style and structure of his other films still the trademark altman audio style is evident here and there i think what really makes this film work is the brilliant performance by sandy dennis it's definitely one of her darker characters but she plays it so perfectly and convincingly that it's scary michael burns does a good job as the mute young man regular altman player michael murphy has a small part the <UNK> moody set fits the content of the story very well in short this movie is a powerful study of loneliness sexual <UNK> and desperation be patient <UNK> up the atmosphere here and pay attention to the wonderfully written script br br i praise robert altman this is one of his many films that deals with unconventional fascinating subject matter this film is disturbing but it's sincere and it's sure to <UNK> a strong emotional response from the viewer if you want to see an unusual film some might even say bizarre this is worth the time br br unfortunately it's very difficult to find in video stores you may have to buy it off the internet

8. Create a graph of accuracy and loss over time

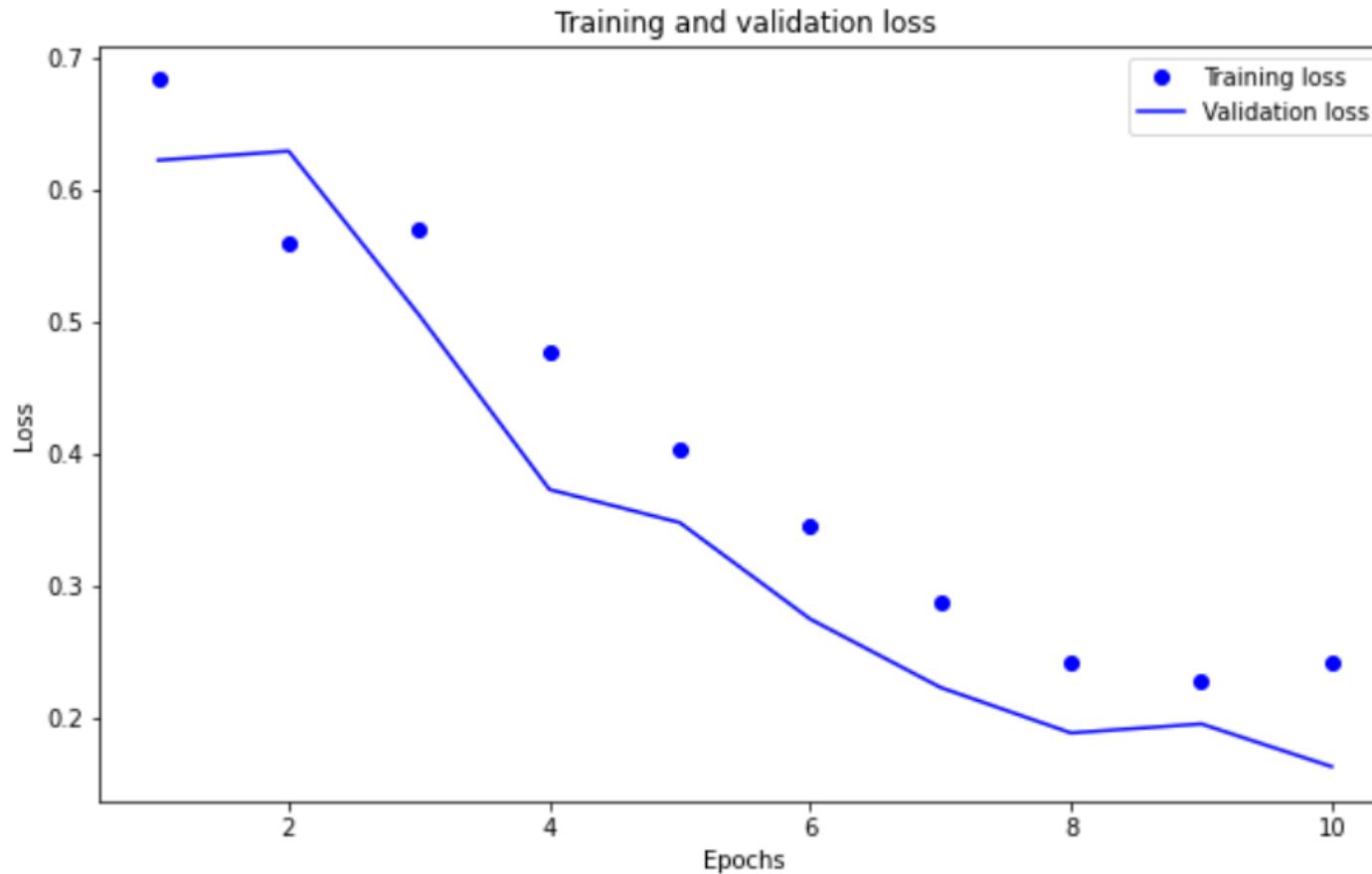
- `model.fit()` returns a `History` object that contains a dictionary with everything that happened during training:

```
In [31]: history_dict = history.history  
history_dict.keys()
```

```
Out[31]: dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

- There are four entries: one for each monitored metric during training and validation.
- We can use these to plot the training and validation loss for comparison, as well as the training and validation accuracy:

8. Create a graph of accuracy and loss over time



```
plt.figure(figsize=(10, 6))

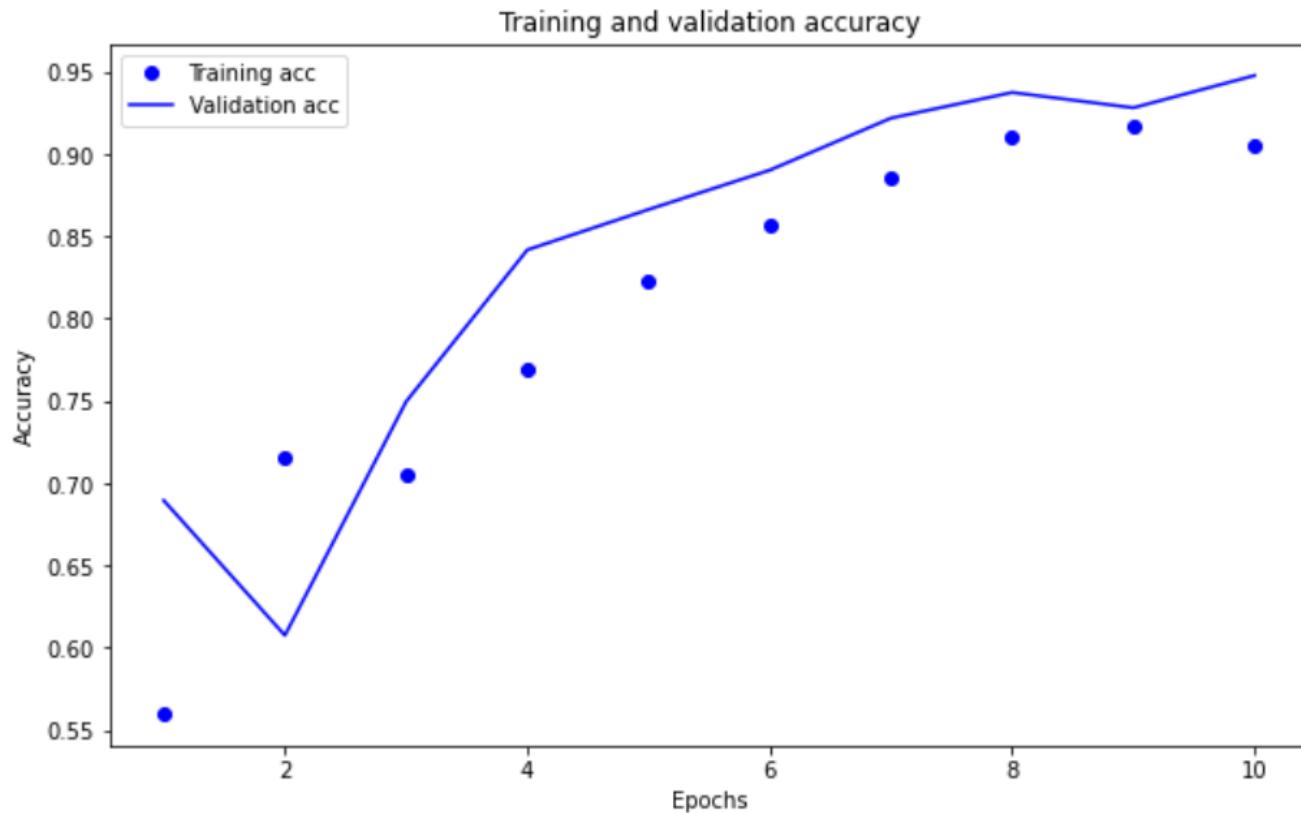
acc = history_dict['acc']
val_acc = history_dict['val_acc']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

8. Create a graph of accuracy and loss over time



```
#plt.clf() # clear figure
plt.figure(figsize=(10, 6))
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

9. Model Evaluation : ROC Curve

In [34]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np

y_hat = model.predict(X_test)

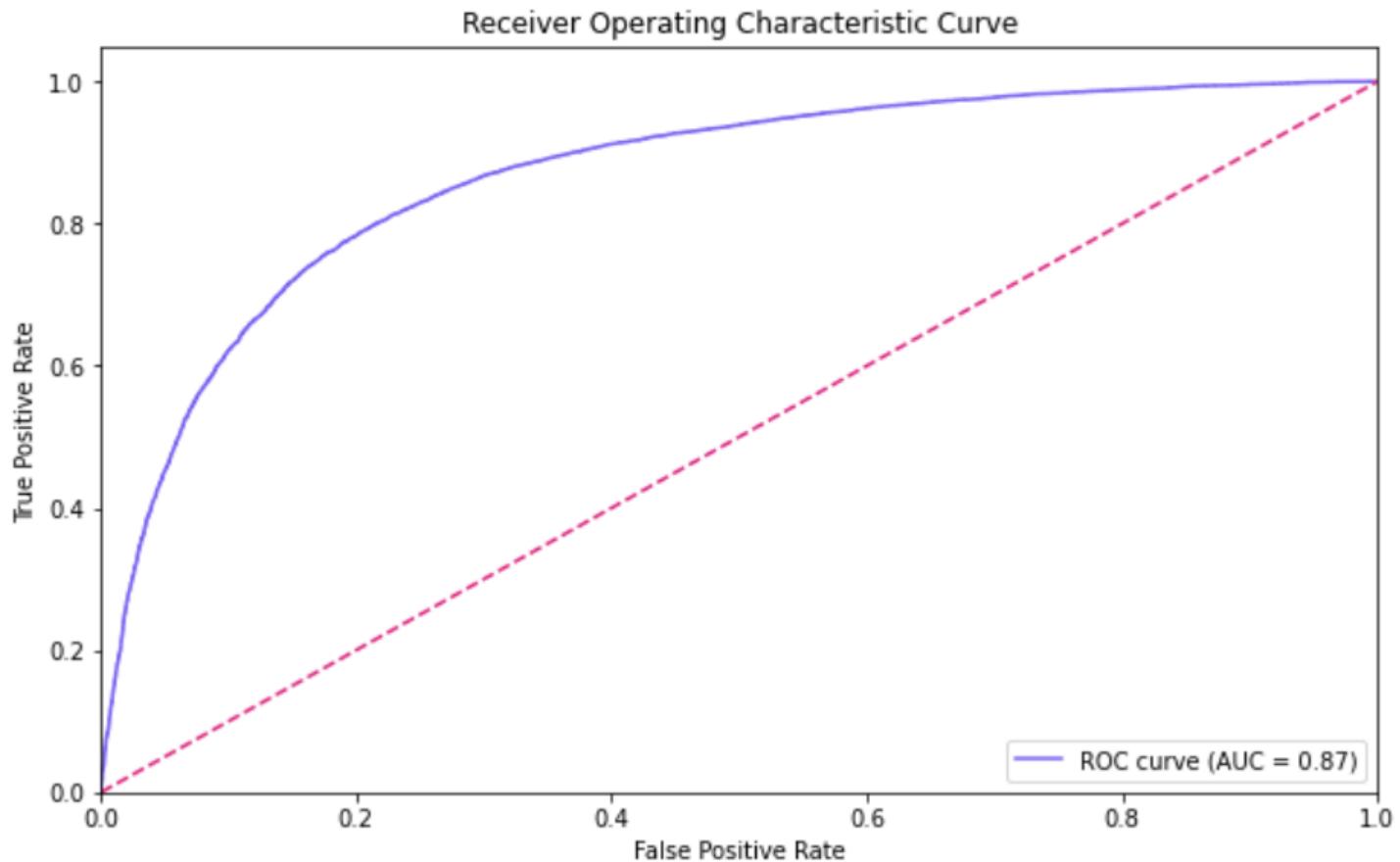
# Compute ROC curve and ROC area for each class
fpr, tpr, thresholds = roc_curve(y_test, y_hat)
roc_auc = auc(fpr, tpr)
```

9. Model Evaluation : ROC Curve

```
fig = plt.figure(figsize=(10,6))
plt.plot(fpr, tpr, color='#785ef0', label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='#dc267f', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve')
plt.legend(loc="lower right")

plt.show()
```

9. Model Evaluation : ROC Curve



9. Model Evaluation : ROC Curve

In [36]:

```
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print("Threshold value is:", optimal_threshold)
y_pred = np.where(y_hat>=optimal_threshold, 1, 0)
print('Balanced Accuracy Score:',balanced_accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

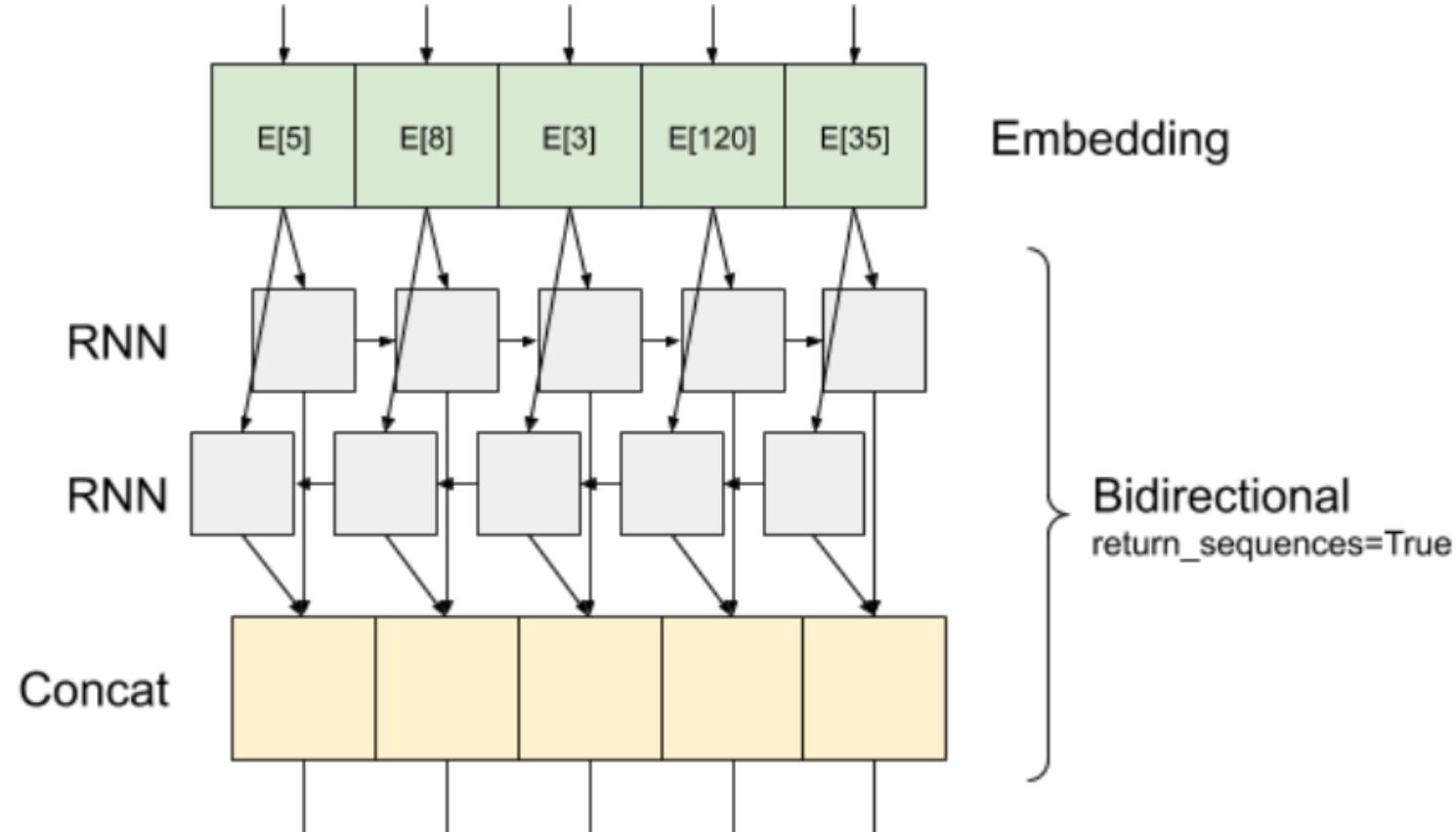
Threshold value is: 0.59686714

Balanced Accuracy Score: 0.79244

[[9972 2528]

[2661 9839]]

PART B Stack two or more LSTM layers



Model M3: Stack two or more LSTM layers

return_sequence : True or False

`return_sequence = False:`

- It returns only the last output for each input sequence (a 2D tensor of shape `(batch_size, output_features)`).

`return_sequence = True:`

- the full sequences of successive outputs for each timestep is returned (a 3D tensor of shape `(batch_size, timesteps, output_features)`).

Lec12-2

Neural Machine Translation

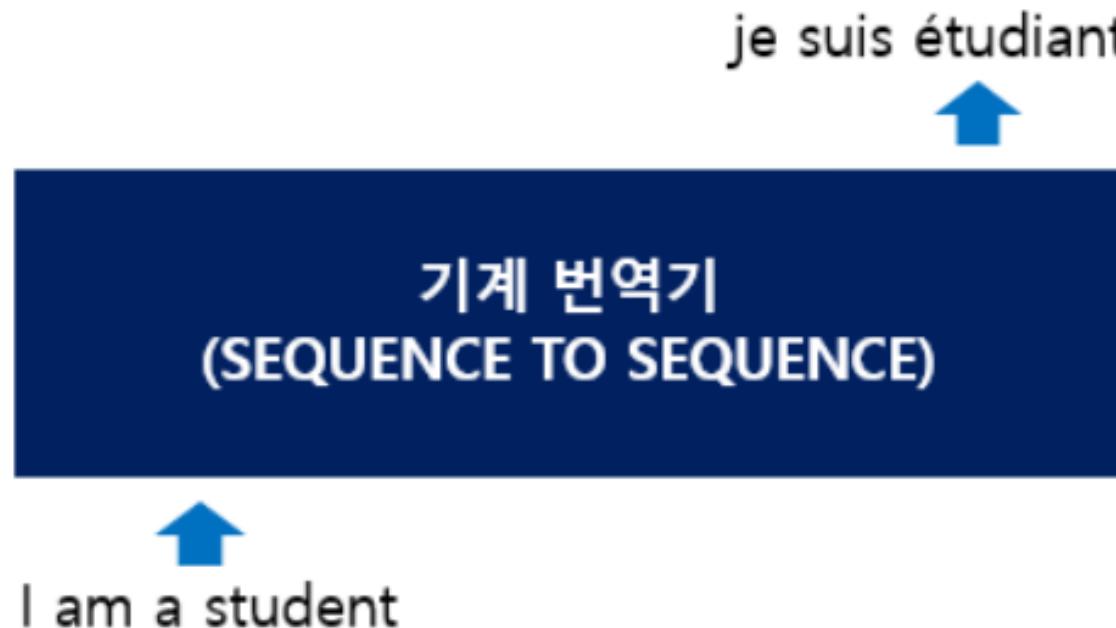
Sequence to Sequence, Seq2seq, Many-to-Many

1) 시퀀스-투-시퀀스(Sequence-to-Sequence, seq2seq)

- 기계 번역을 예제로 시퀀스-투-시퀀스를 설명합니다.
- 시퀀스-투-시퀀스(Sequence-to-Sequence)이란?
 - ✓ 입력 시퀀스로부터 다른 도메인의 시퀀스를 출력하는 모델이다.
 - ✓ 챗봇(Chatbot)과 기계 번역(Machine Translation)이 대표적인 예제이다.
 - ✓ 입력 시퀀스와 출력 시퀀스를 각각 질문과 대답으로 구성하면 챗봇으로 만들 수 있다.
 - ✓ 입력 시퀀스와 출력 시퀀스를 각각 입력 문장과 번역 문장으로 만들면 번역기로 만들 수 있습니다.

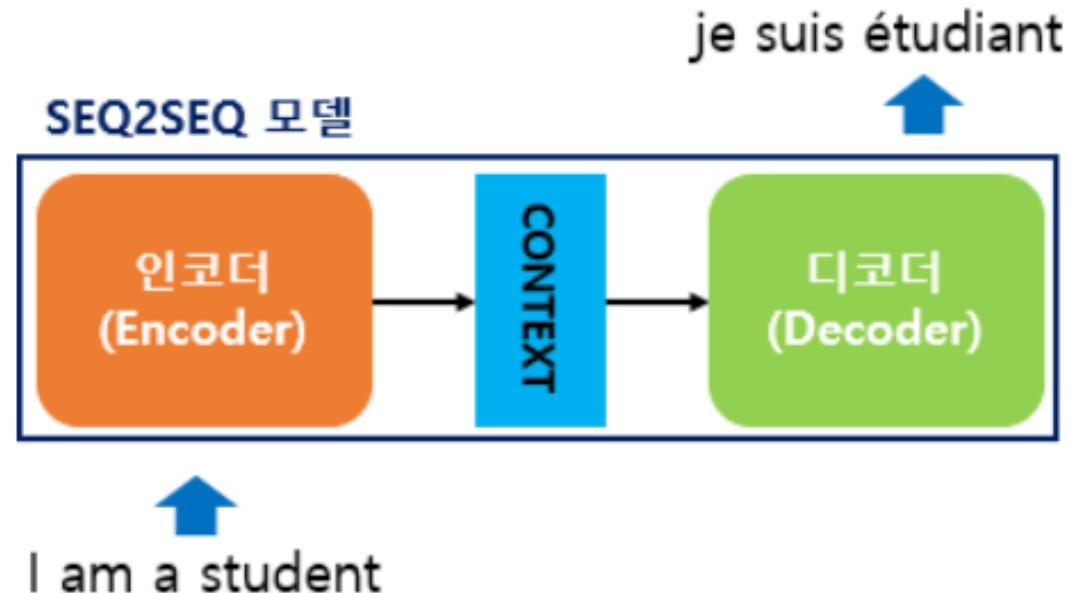
1) 시퀀스-투-시퀀스(Sequence-to-Sequence, seq2seq)

- 영어 문장을 프랑스 문장으로 번역하는 예제
- 'I am a student'라는 문장을 'je suis étudiant'라는 프랑스어로 출력



seq2seq는 인코더와 디코더 구조

- 인코더는 단어들을 순차적으로 입력받고, 이 모든 단어 정보들을 압축해서 벡터를 만든다.
 - ✓ 입력 정보가 하나의 컨텍스트 벡터로 압축되면 인코더는 컨텍스트 벡터를 디코더로 전송합니다.
 - ✓ 디코더는 컨텍스트 벡터를 받아서 번역된 단어를 한 개씩 순차적으로 출력합니다.



컨텍스트 벡터(context vector)

▶ 컨텍스트 벡터의 간단한 예제

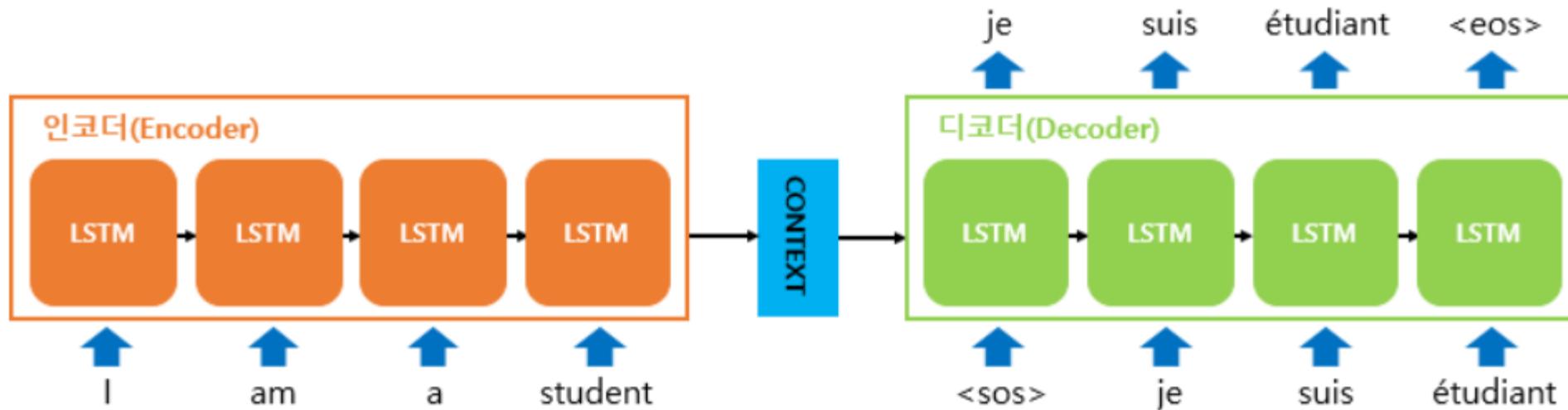
✓ 크기는 4로 표현하였다. 보통 수백 이상의 차원을 갖고있습니다.

CONTEXT	0.15
	0.21
	-0.11
	0.91

인코더와 디코더는 RNN 아키텍처이다.

➤ 컨텍스트 벡터(context vector)

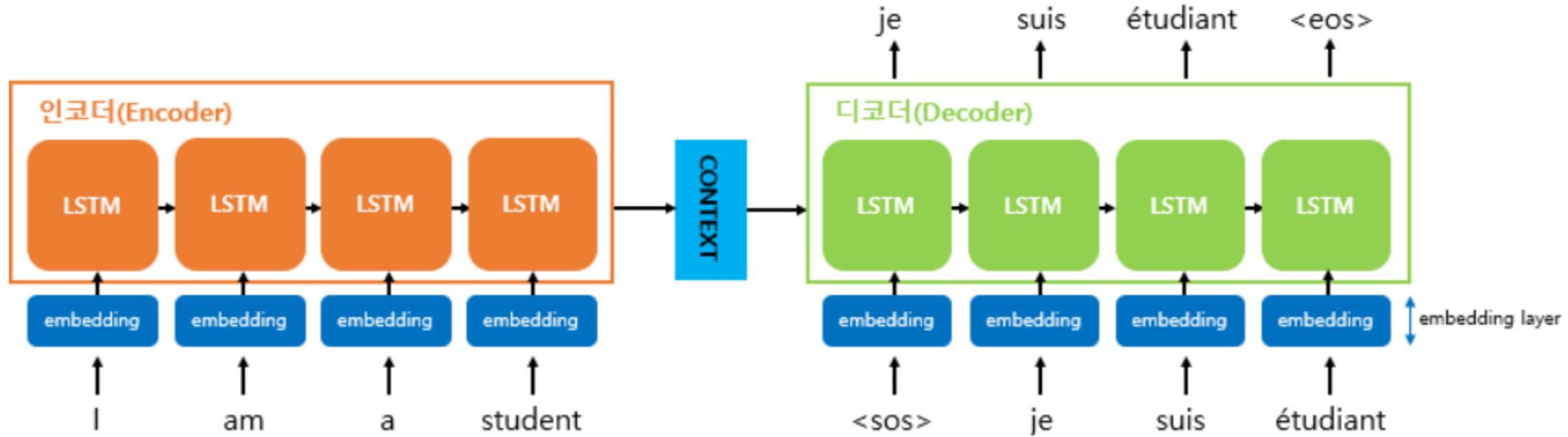
- ✓ 인코더는 문장의 모든 단어들을 순차적으로 입력받은 뒤에 모든 단어 정보들을 압축한다.
- ✓ 디코더는 컨텍스트 벡터를 받아서 번역된 단어를 한 개씩 순차적으로 출력합니다.
- ✓ 컨텍스트 벡터는 디코더 RNN 셀의 첫번째 은닉 상태로 사용됩니다.



디코더는 초기 입력으로 문장의 시작을 의미하는 심볼 <sos>가 들어갑니다. 디코더는 <sos>가 입력되면, 다음에 등장할 확률이 높은 단어를 예측합니다. 첫번째 시점(time step)의 디코더 RNN 셀은 다음에 등장할 단어로 je를 예측하였습니다

인코더와 디코더

- 단어들은 워드 임베딩 벡터이다.

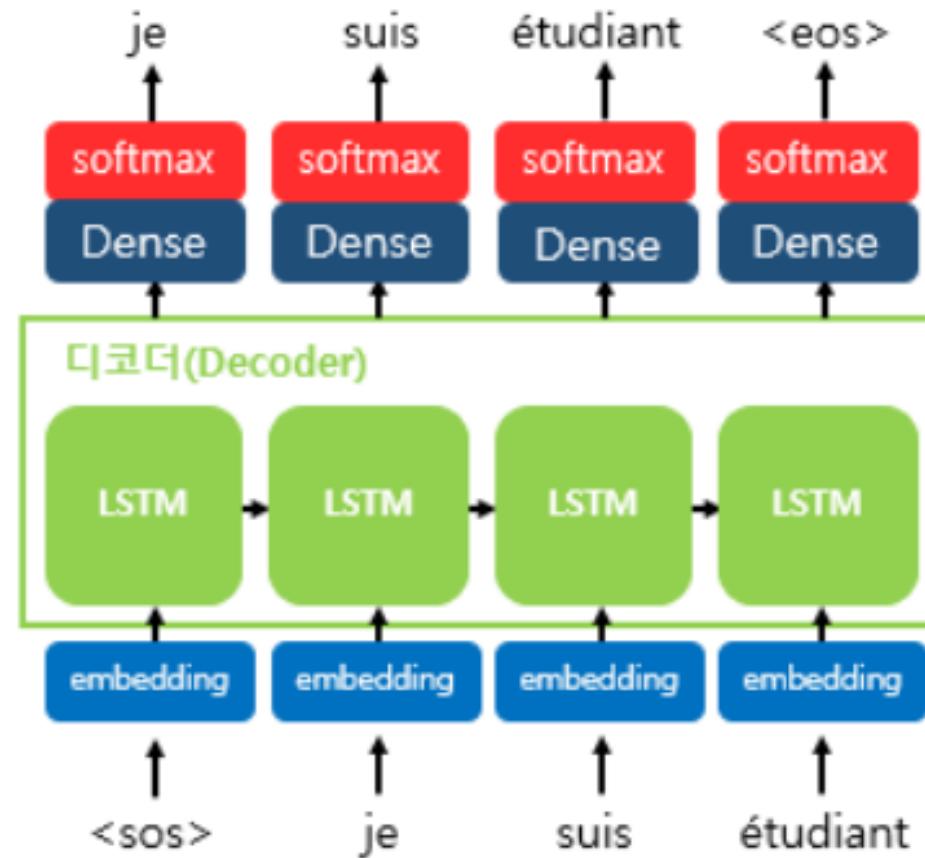


인코더와 디코더

- 단어에 대해서 임베딩 층 (Embedding layer)
 - ✓ 예를 들어 I, am, a, student라는 단어들에 대한 임베딩 벡터 모습

-	0.157
	-0.25
	0.478
	-0.78
am	0.78
	0.29
	-0.96
	0.52
a	0.75
	-0.81
	0.96
	0.12
student	0.88
	-0.17
	0.29
	0.48

디코더

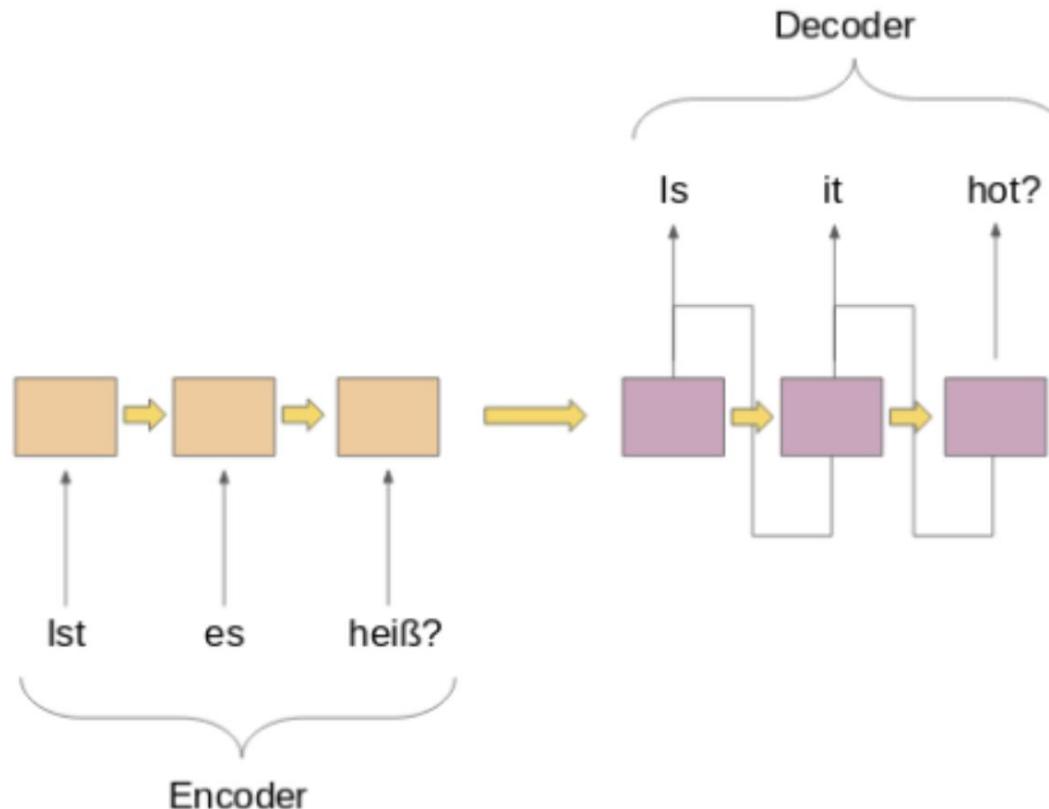


Lab12-2

Character-Level Neural Machine Translation

Simple RNN and LSTM

Introduction to Sequence-to-Sequence (Seq2Seq) Modeling



In [2]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding, Bidirectional,
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import load_model
from tensorflow.keras import optimizers
import matplotlib.pyplot as plt
```

3. Read Data

In [5]:

```
data = read_text("kor.txt")
kor_eng = to_lines(data)
kor_eng = np.array(kor_eng)
print(len(kor_eng))
kor_eng.shape
```

3729

Out [5]: (3729, 3)

4. Text Pre-Processing

(a) Text Cleaning

- Let's take a look at our data, then we will decide which pre-processing steps to adopt.

kor_eng

인 것 같아.'

'CC-BY 2.0 (France) Attribution: tatoeba.org #953635 (CK) & #8384140 (Eunhee)']

['If someone who doesn't know your background says that you sound like a native speaker, it means they probably noticed something about your speaking that made them realize you weren't a native speaker. In other words, you don't really sound like a native speaker.]

'만일 네 사정도 잘 모르는 사람이 원어민 같다고 말한다면 그건 그 사람이 네가 원어민이 아니라고 깨닫게 해주는 뭔가를 네 말 속에서 캐치해 낸 것이겠지. 다르게 표현하자면 넌 원어민처럼 말하지 않아.'

'CC-BY 2.0 (France) Attribution: tatoeba.org #953936 (CK) & #8813370 (Eunhee)']

['Doubtless there exists in this world precisely the right woman for any given man to marry and vice versa; but when you consider that a human being has the opportunity of being acquainted with only a few hundred people, and out of the few hundred that there are but a dozen or less whom he knows intimately, and out of the dozen, one or two friends at most, it will easily be seen, when we remember the number of millions who inhabit this world, that probably, since the earth was created, the right man has never yet met the right woman.]

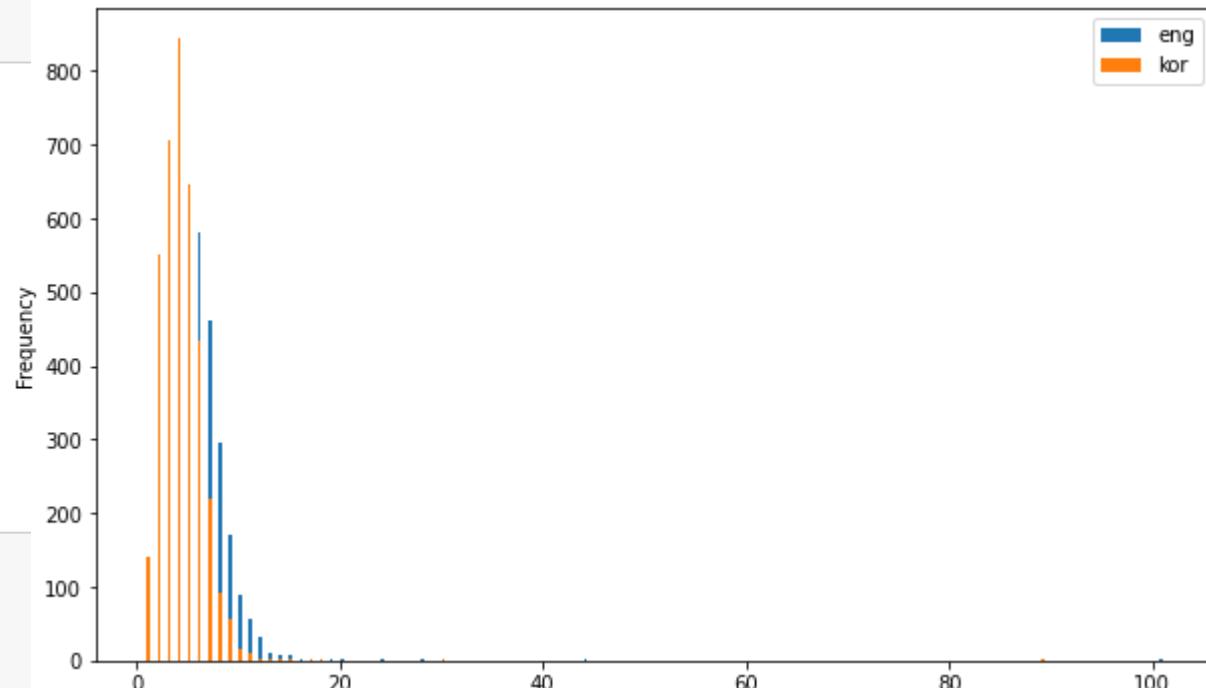
'의심의 여지 없이 세상에는 어떤 남자이든 정확히 딱 알맞는 여자와 결혼하거나 그 반대의 상황이 존재하지. 그런데 인간이 수백 명의 사람만 알고 지내는 사이가 될 기회를 갖는다고 생각해 보면, 또 그 수백 명 중 열여 명쯤 이하만 잘 알 수 있고, 그리고 나서 그 열여 명 중에 한두 명만 친구가 될 수 있다면, 그리고 또 만일 으리가 이 세상에 살고 이느 스웨덴 며이 사라드마 기어하고 있다며 따 마느 나자느 친구가 새거나 아래로 따

(b) Text to Sequence Conversion

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3729 entries, 0 to 3728
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
 ---  -- 
 0   eng     3729 non-null   int64  
 1   kor     3729 non-null   int64  
 dtypes: int64(2)
 memory usage: 58.4 KB
```

```
df.plot.hist(bins = 300,figsize=(10, 6))
plt.show()
```



Tokenizer

- The maximum length of the Korean sentences is 15 and that of the English is 20.

```
In [15]: # function to build a tokenizer
def tokenization(lines):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer
```

```
In [16]: # prepare english tokenizer
eng_tokenizer = tokenization(kor_eng[:, 0])
eng_vocab_size = len(eng_tokenizer.word_index) + 1

eng_length = 20

print('English Vocabulary Size: %d' % eng_vocab_size)
```

English Vocabulary Size: 2561

encode and pad sequences

```
In [18]: # encode and pad sequences
def encode_sequences(tokenizer, length, lines):
    # integer encode sequences
    seq = tokenizer.texts_to_sequences(lines)
    # pad sequences with 0 values
    seq = pad_sequences(seq, maxlen=length, padding='post')
return seq
```

5. Model Building



Model Architecture

We will now split the data into train and test set for model training and evaluation, respectively.

```
In [20]: from sklearn.model_selection import train_test_split  
train, test = train_test_split(kor_eng, test_size=0.2, random_state = 12)
```

Build NMT model

```
# build NMT model
def build_model(in_vocab, out_vocab, in_timesteps, out_timesteps, units):
    model = Sequential()
    model.add(Embedding(in_vocab, units, input_length=in_timesteps, mask_zero=True))
    model.add(LSTM(units))
    model.add(RepeatVector(out_timesteps))
    model.add(LSTM(units, return_sequences=True))
    model.add(Dense(out_vocab, activation='softmax'))
    return model
```

Model and Compile

We are using RMSprop optimizer in this model as it is usually a good choice for recurrent neural networks.

```
model = build_model(kor_vocab_size, eng_vocab_size, kor_length, eng_length, 64)
rms = optimizers.RMSprop(learning_rate=0.001)
model.compile(optimizer=rms, loss='sparse_categorical_crossentropy',metrics=['acc'])
```

loss='sparse_categorical_crossentropy': because it allows us to use the target sequence as it is instead of one hot encoded format.

- One hot encoding the target sequences with such a huge vocabulary might consume our system's entire memory.
- We will train it for 30 epochs and with a batch size of 64.

Train the model

```
history = model.fit(trainX, trainY.reshape(trainY.shape[0], trainY.shape[1], 1),
                     epochs=30, batch_size=64,
                     validation_split = 0.2, verbose=1)

38/38 [=====] - 1s 22ms/step - loss: 1.5617 - acc: 0.7490 - val_loss:
1.8888 - val_acc: 0.7428
Epoch 26/30
38/38 [=====] - 1s 21ms/step - loss: 1.5605 - acc: 0.7492 - val_loss:
1.8925 - val_acc: 0.7409
Epoch 27/30
38/38 [=====] - 1s 21ms/step - loss: 1.5530 - acc: 0.7498 - val_loss:
1.8945 - val_acc: 0.7414
Epoch 28/30
38/38 [=====] - 1s 21ms/step - loss: 1.5457 - acc: 0.7499 - val_loss:
1.8934 - val_acc: 0.7433
Epoch 29/30
38/38 [=====] - 1s 21ms/step - loss: 1.5390 - acc: 0.7504 - val_loss:
1.9095 - val_acc: 0.7435
Epoch 30/30
38/38 [=====] - 1s 21ms/step - loss: 1.5280 - acc: 0.7511 - val_loss:
1.9169 - val_acc: 0.7364
```

Plot history

Let's compare the training loss and the validation loss.

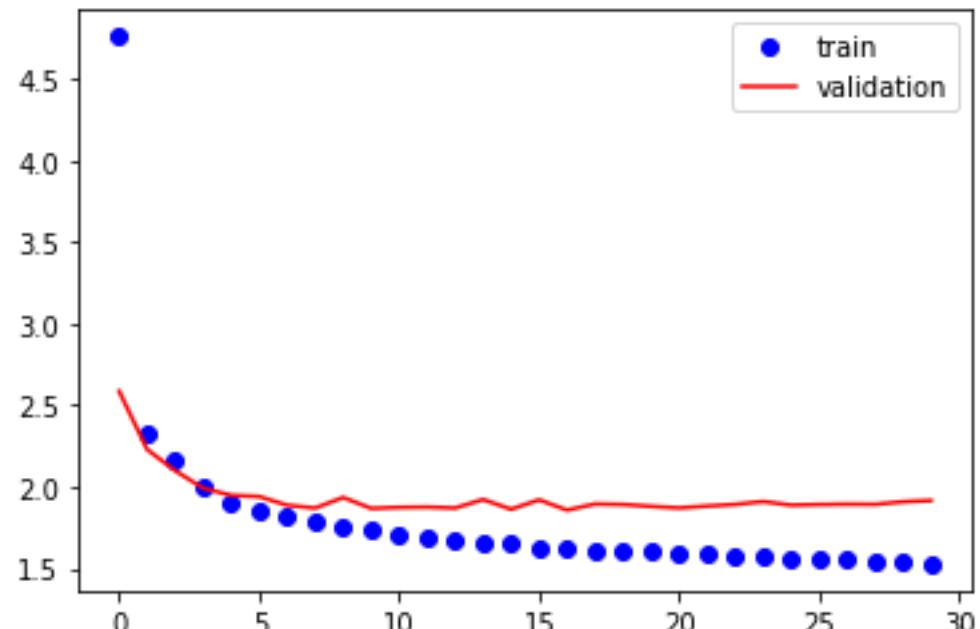
```
In [28]: history_dict = history.history  
history_dict.keys()
```

```
Out [28]: dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

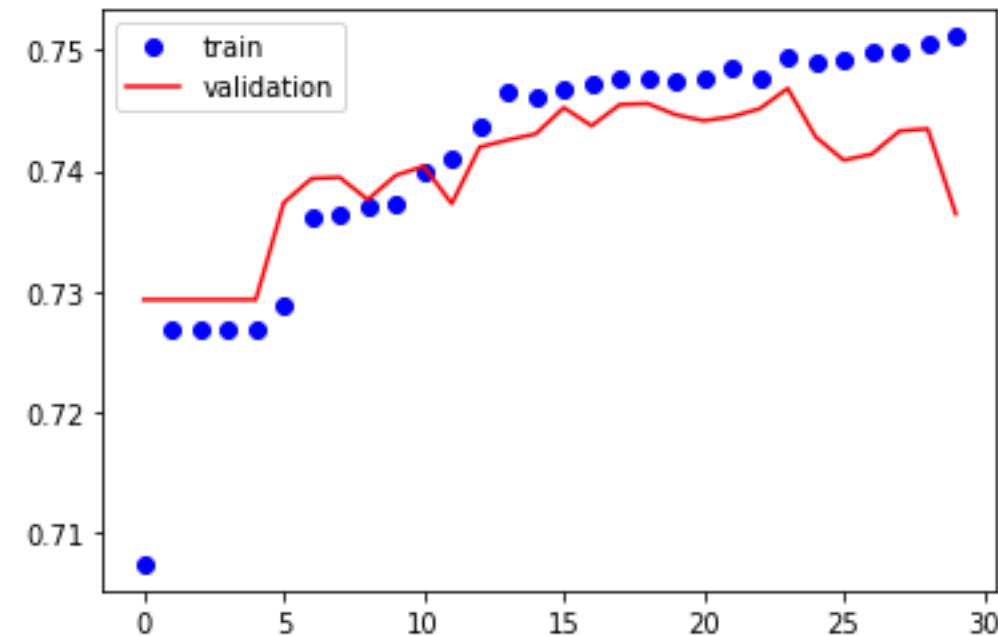
```
In [30]: plt.plot(history.history['loss'],'bo')  
plt.plot(history.history['val_loss'],'r')  
plt.legend(['train','validation'])  
plt.show()
```

Loss and Accuracy

Loss



Accuracy



Thank You!