Lec12-1
Text Classification and Sentimental Analysis

Tokenizer, Embedding, LSTM, Keras

Colab 사용하기

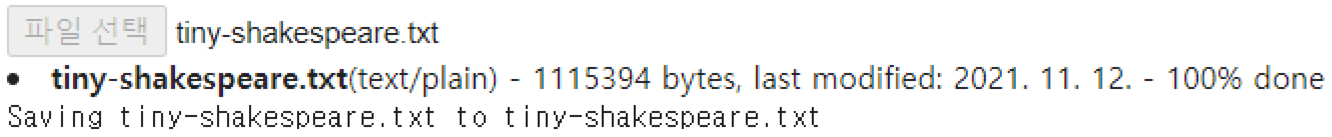
Simple RNN and LSTM

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

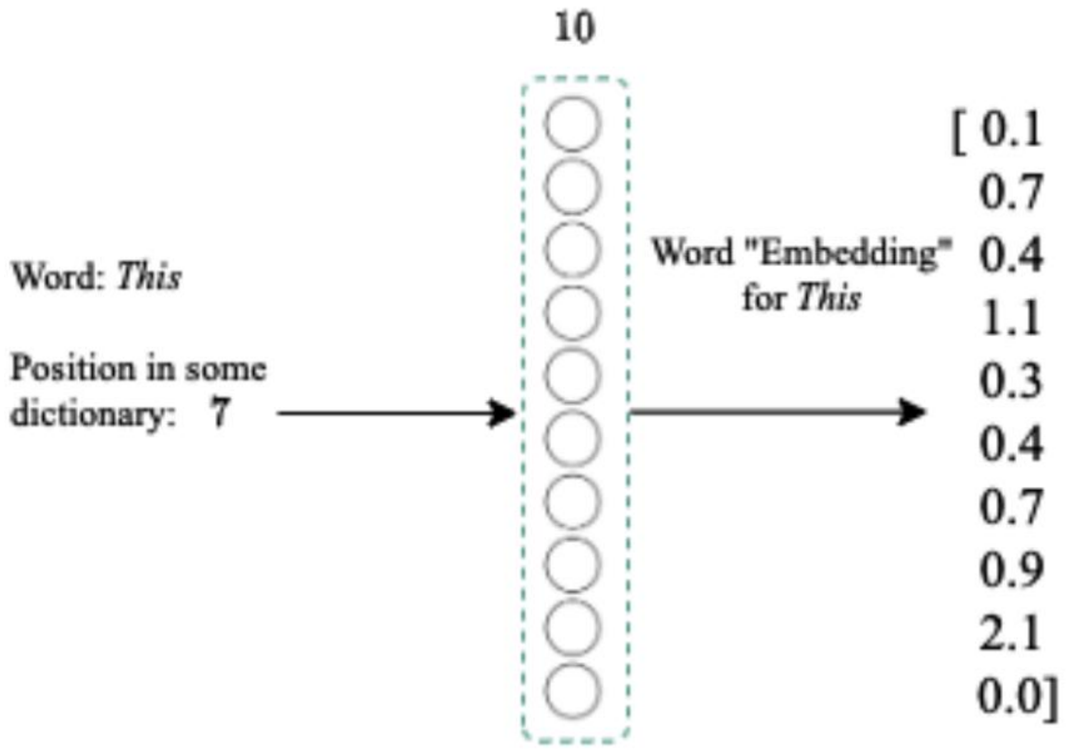
- 여기서는 간단한 방법으로 해보자.
  - ✓ 1) RNN에 계산에 사용할 문장의 길이를 정하자.
  - ✓ 2) 어휘(vocabulary)에 해당하는 단어 개수를 정하자.
    - 자주 사용되는 단어 순으로 ranking하자
  - ✓ 3) word-to-index 단어들을 인텍스로 바꾸자
  - ✓ 4) 단어 임베딩의 차원을 결정자.

embddng_dim = 10

This

index : 7,

vector size : 10

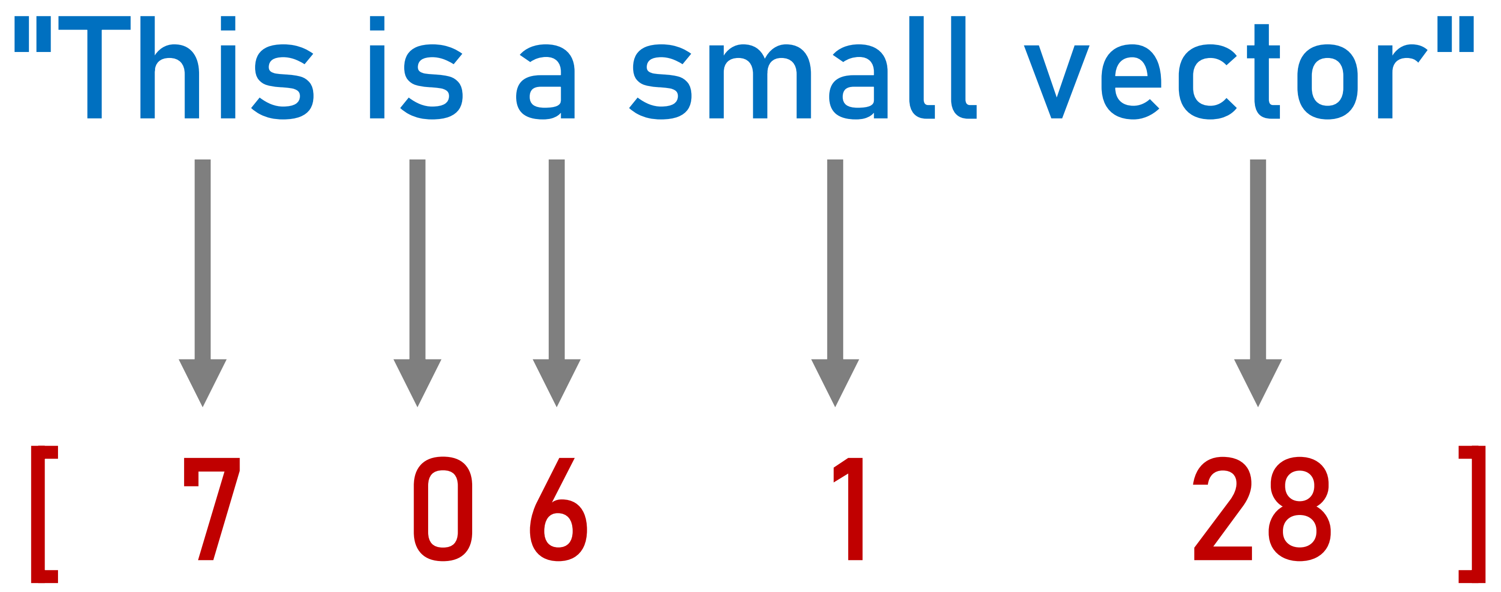


Sequence of vectors

vocab_size = 30

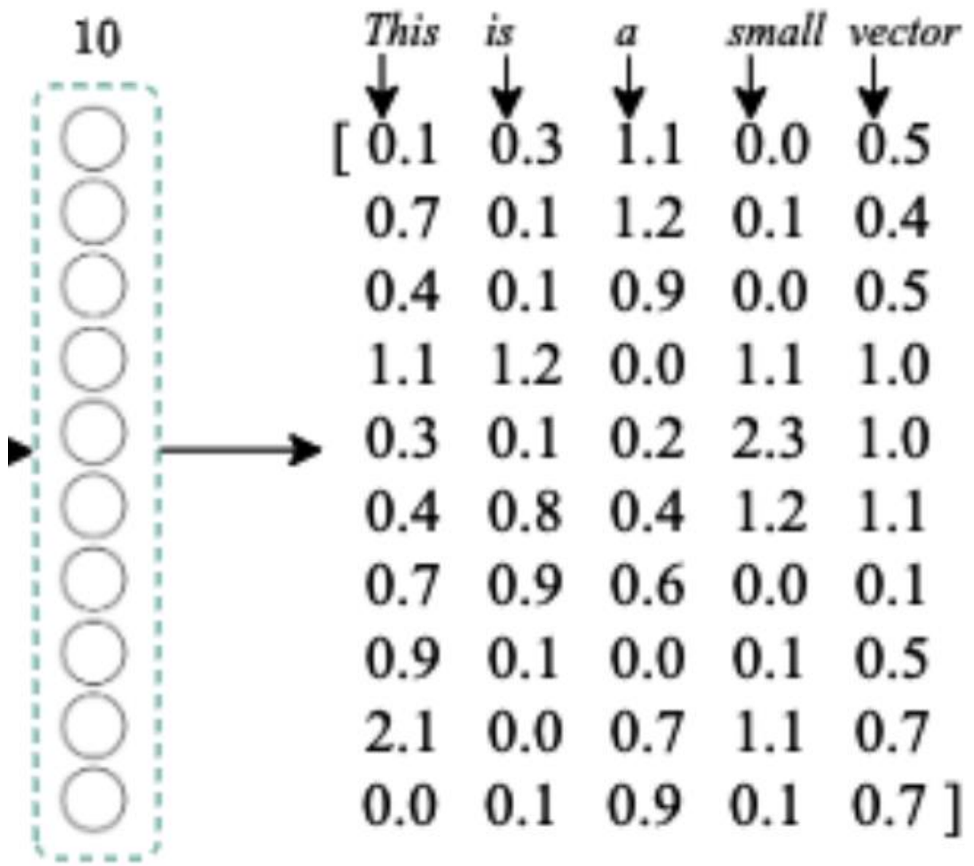
seq_length = 5

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



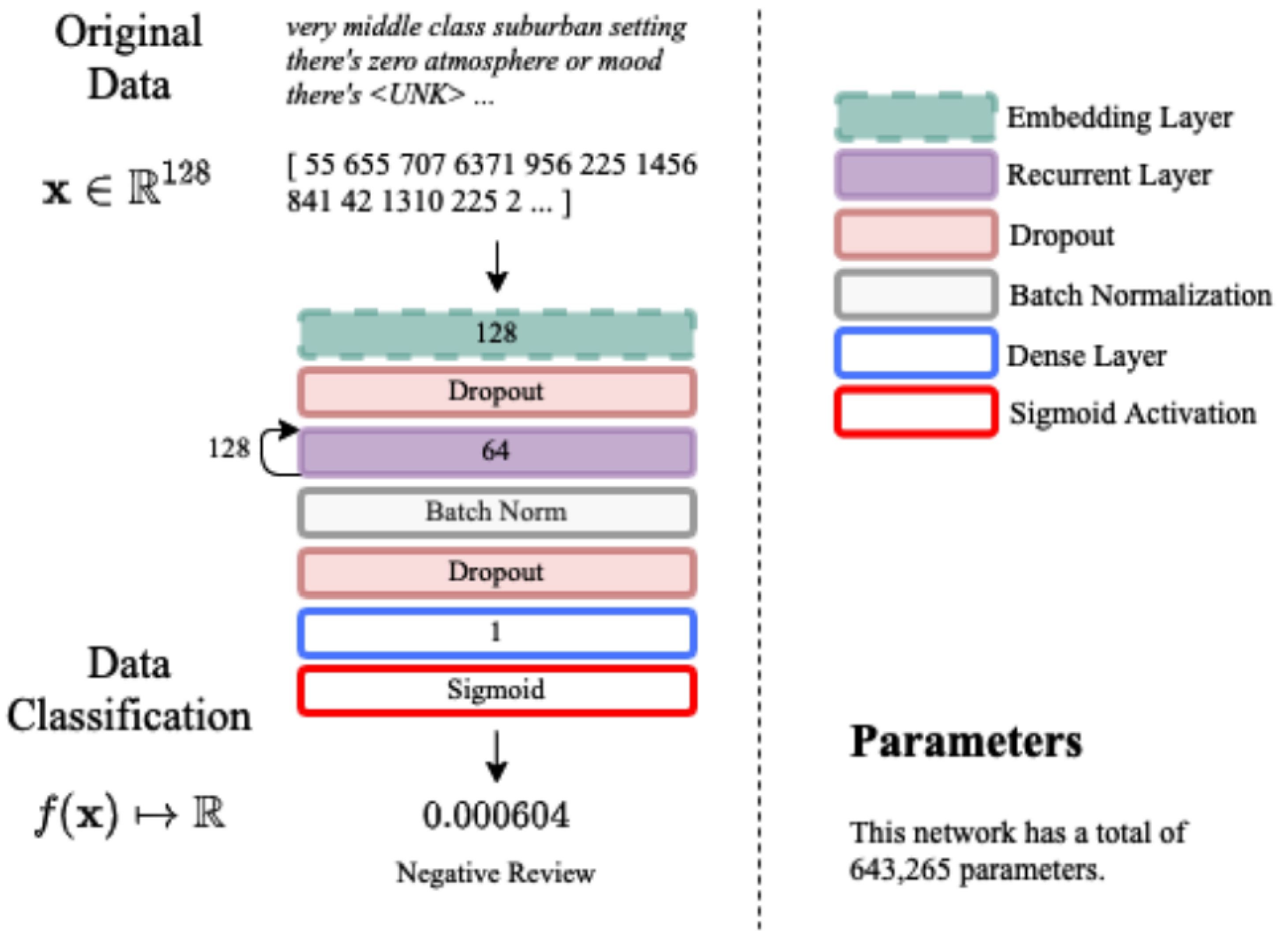
Sequence of word embeddings

vocab_size = 30
embddng_dim = 10
seqnc_lngth = 5



$$
\begin{array}{ccccc}
\downarrow \textit{This} & \downarrow \textit{is} & \downarrow \textit{a} & \downarrow \textit{small} & \downarrow \textit{vector} \\
[\,0.1 & 0.3 & 1.1 & 0.0 & 0.5 \\
0.7 & 0.1 & 1.2 & 0.1 & 0.4 \\
0.4 & 0.1 & 0.9 & 0.0 & 0.5 \\
1.1 & 1.2 & 0.0 & 1.1 & 1.0 \\
0.3 & 0.1 & 0.2 & 2.3 & 1.0 \\
0.4 & 0.8 & 0.4 & 1.2 & 1.1 \\
0.7 & 0.9 & 0.6 & 0.0 & 0.1 \\
0.9 & 0.1 & 0.0 & 0.1 & 0.5 \\
2.1 & 0.0 & 0.7 & 1.1 & 0.7 \\
0.0 & 0.1 & 0.9 & 0.1 & 0.7\,]
\end{array}
$$

An RNN architecture for the IMDB dataset

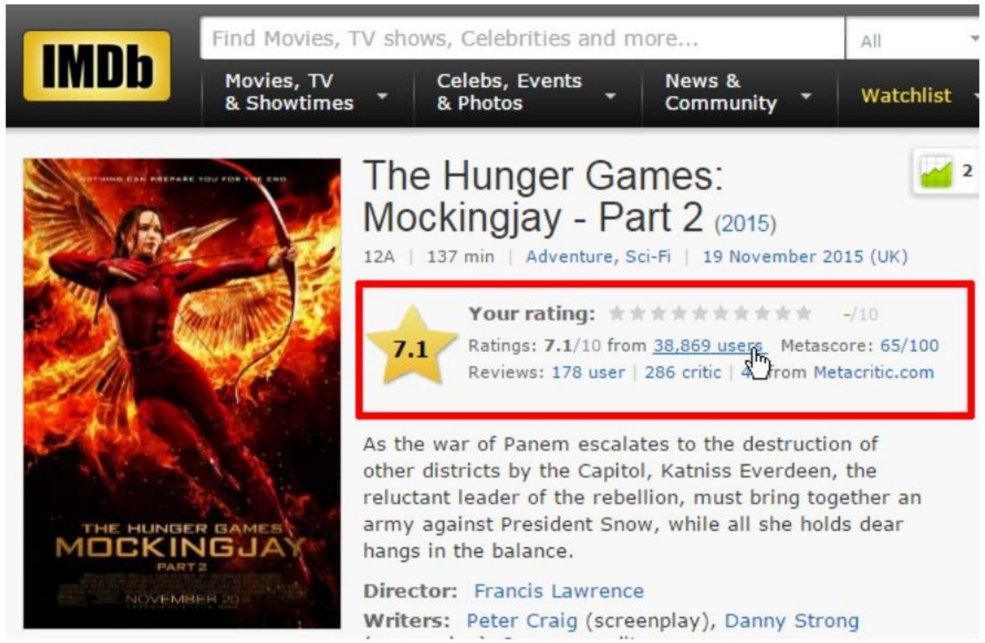


Parameters

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



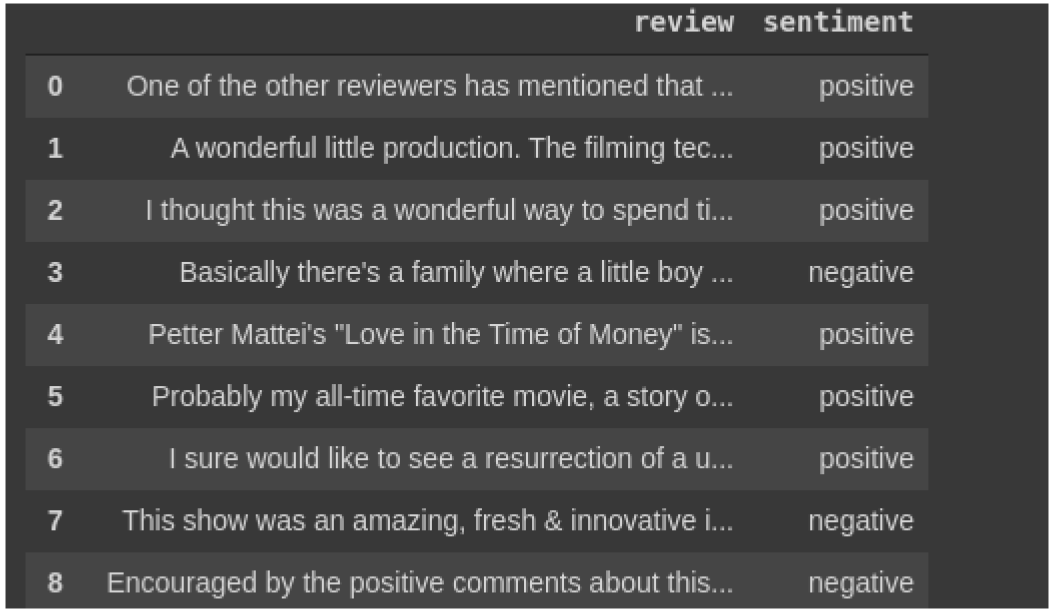
The Hunger Games: Mockingjay - Part 2 (2015)

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

Your rating: -/10
7.1 Ratings: 7.1/10 from 38,869 users Metascore: 65/100
Reviews: 178 user | 286 critic | 43 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.

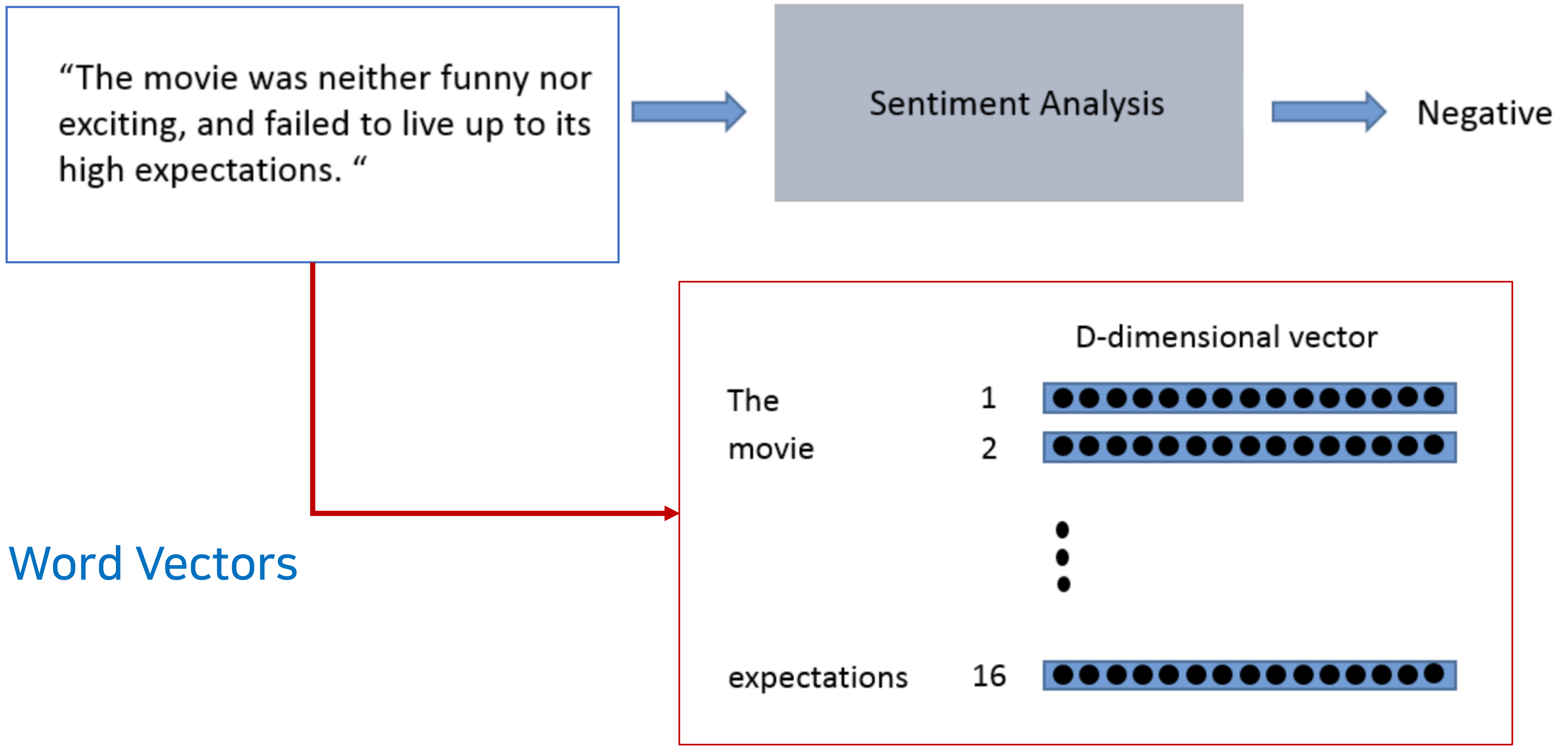
Director: Francis Lawrence
Writers: Peter Craig (screenplay), Danny Strong



	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.



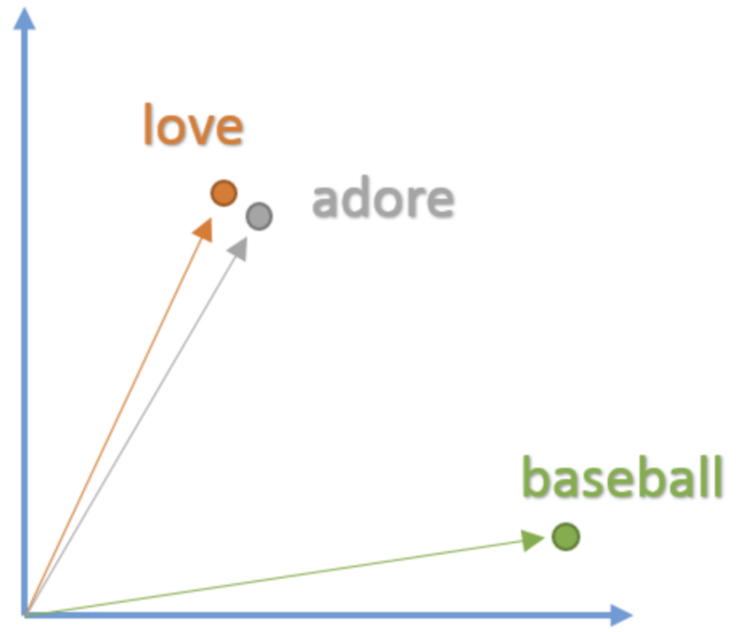
"The movie was neither funny nor exciting, and failed to live up to its high expectations. "
Sentiment Analysis
Negative
D-dimensional vector
The 1
movie 2
expectations 16

- Word Vectors

Word embedding : Word2Vec

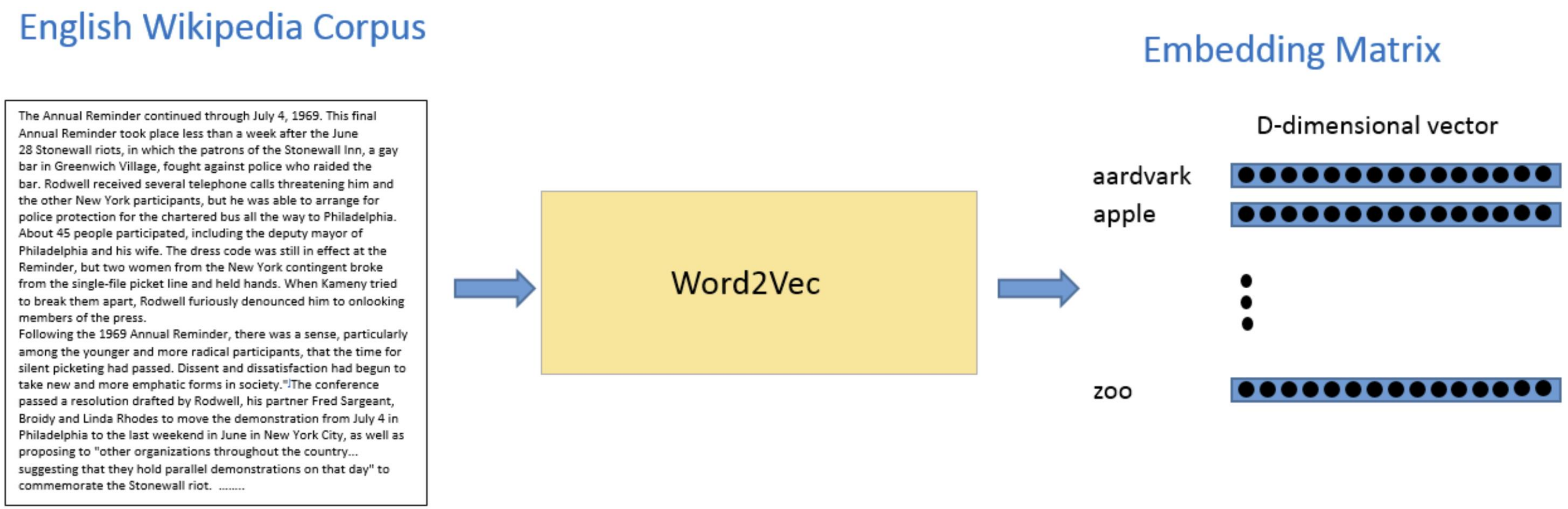
- The vector representation of a word is also known as a word embedding.
- Word2Vec :
  - ✓ the model creates word vectors by looking at the context with which words appear in sentences.

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

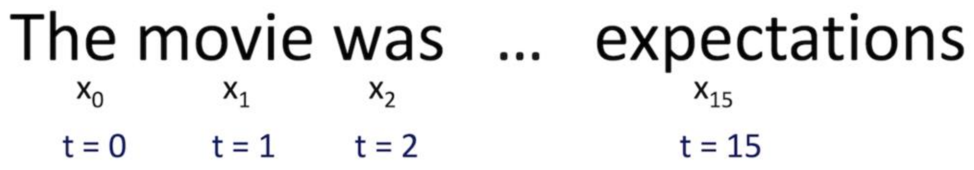
- ➢ The output of a Word2Vec model is called an embedding matrix
  - ✓ This embedding matrix will contain vectors for every distinct word in the training corpus.



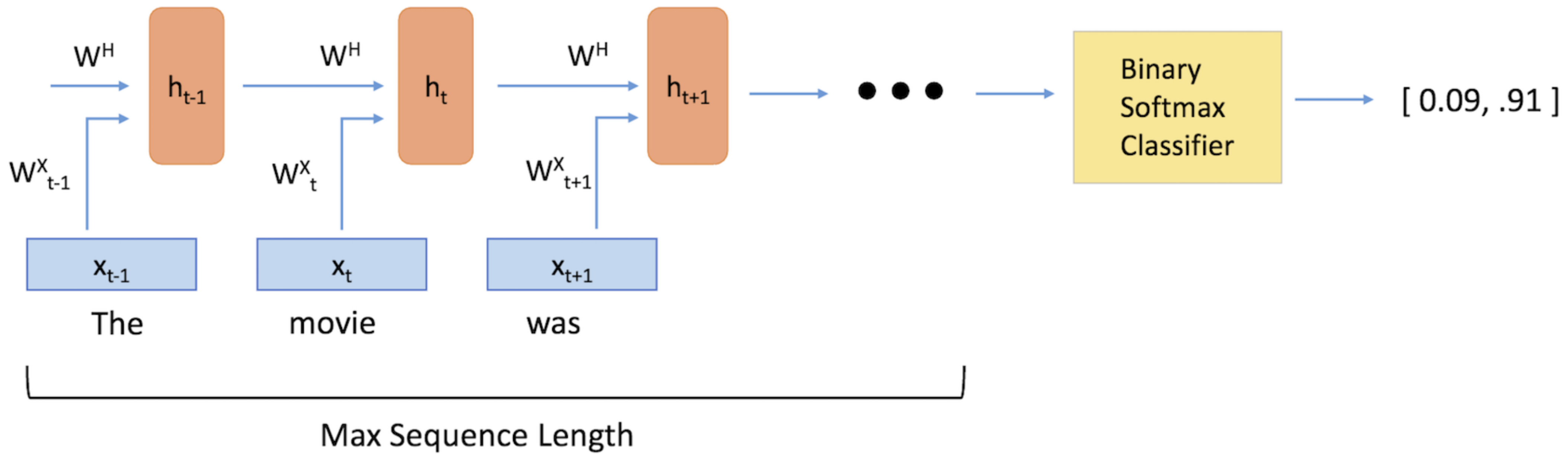
The Annual Reminder continued through July 4, 1969. This final Annual Reminder took place less than a week after the June 28 Stonewall riots, in which the patrons of the Stonewall Inn, a gay bar in Greenwich Village, fought against police who raided the bar. Rodwell received several telephone calls threatening 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."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. ....

RNN

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

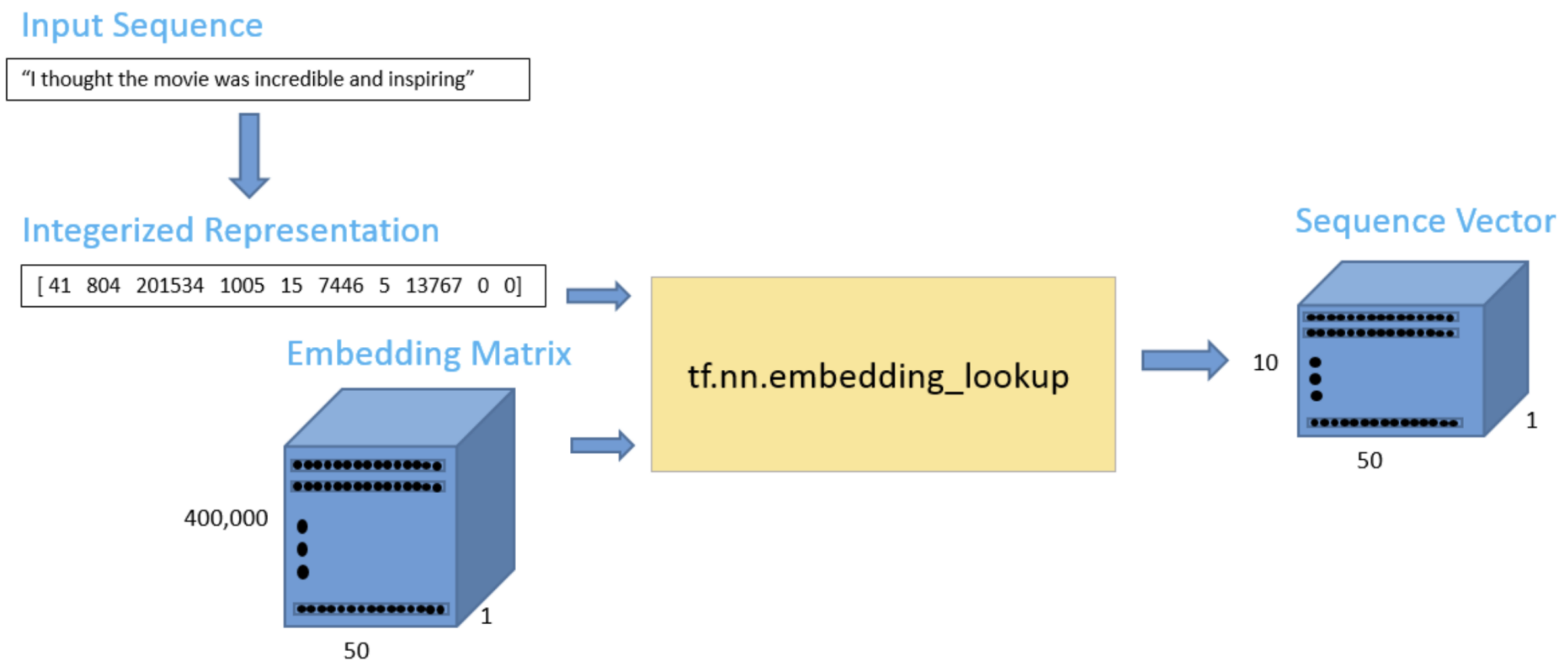


$$h_t = \sigma(W^H h_{t-1} + W^X x_t)$$



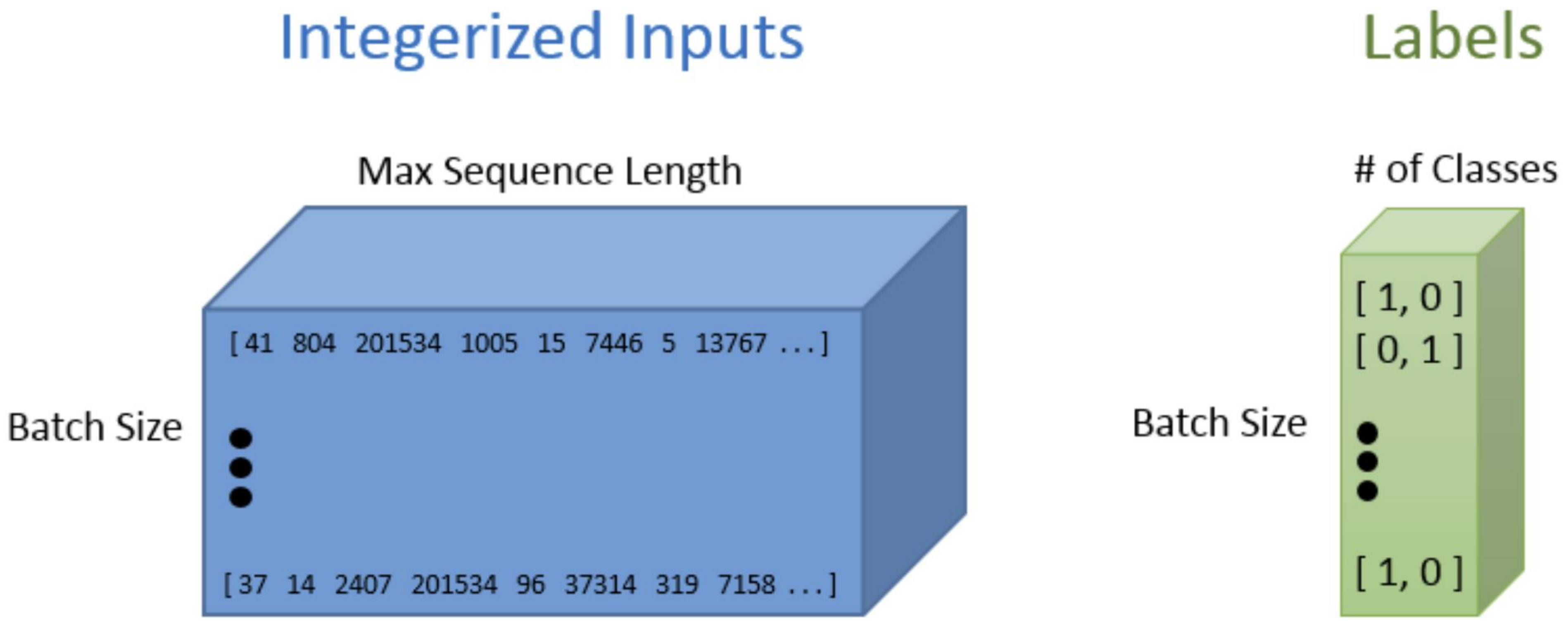
The data pipeline : a simple example

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

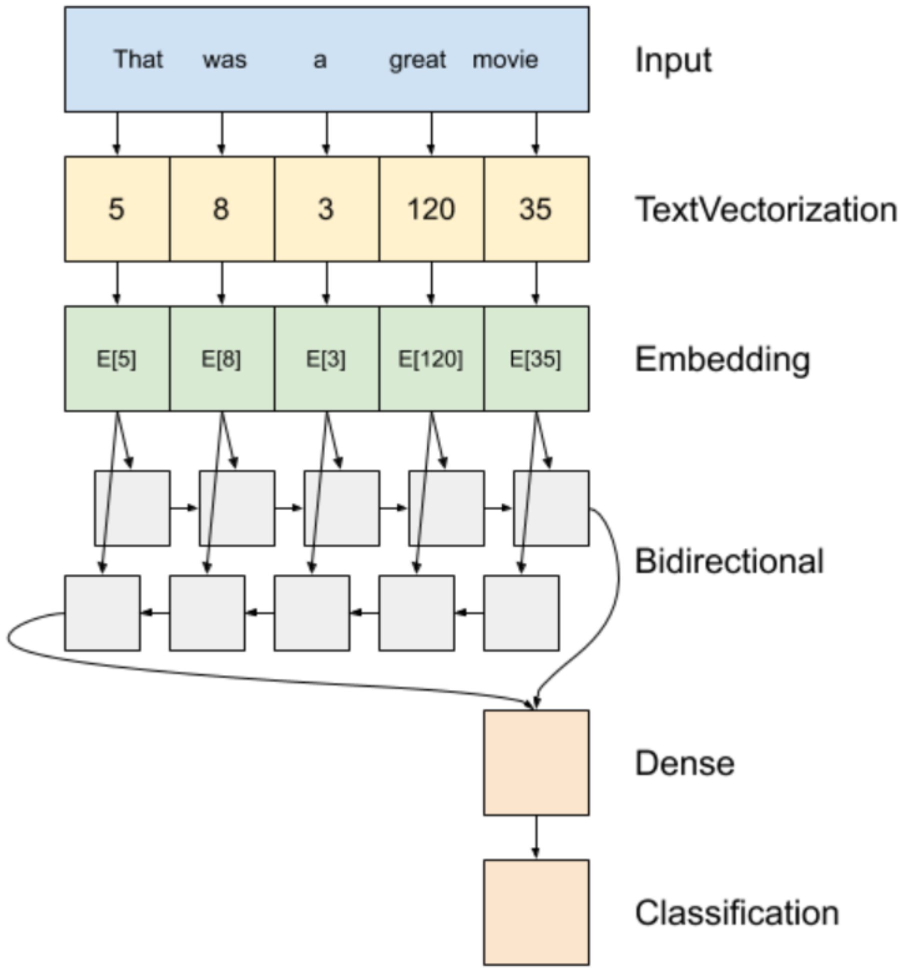


The 10 x 50 output should contain the 50 dimensional word vectors for each of the 10 words in the sequence.

Input and Output : Many-to-One



Bidirectional LSTM model architecture



Lab:
Text Classification with IMDB

IMDB Text Classification

1. Setup

```
In [1]: 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, Dropout
        tf.__version__

Out[1]: '2.7.0'
```

2. Download the IMDB dataset

In [2]:
```
num_words = 10000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=num_words)
```

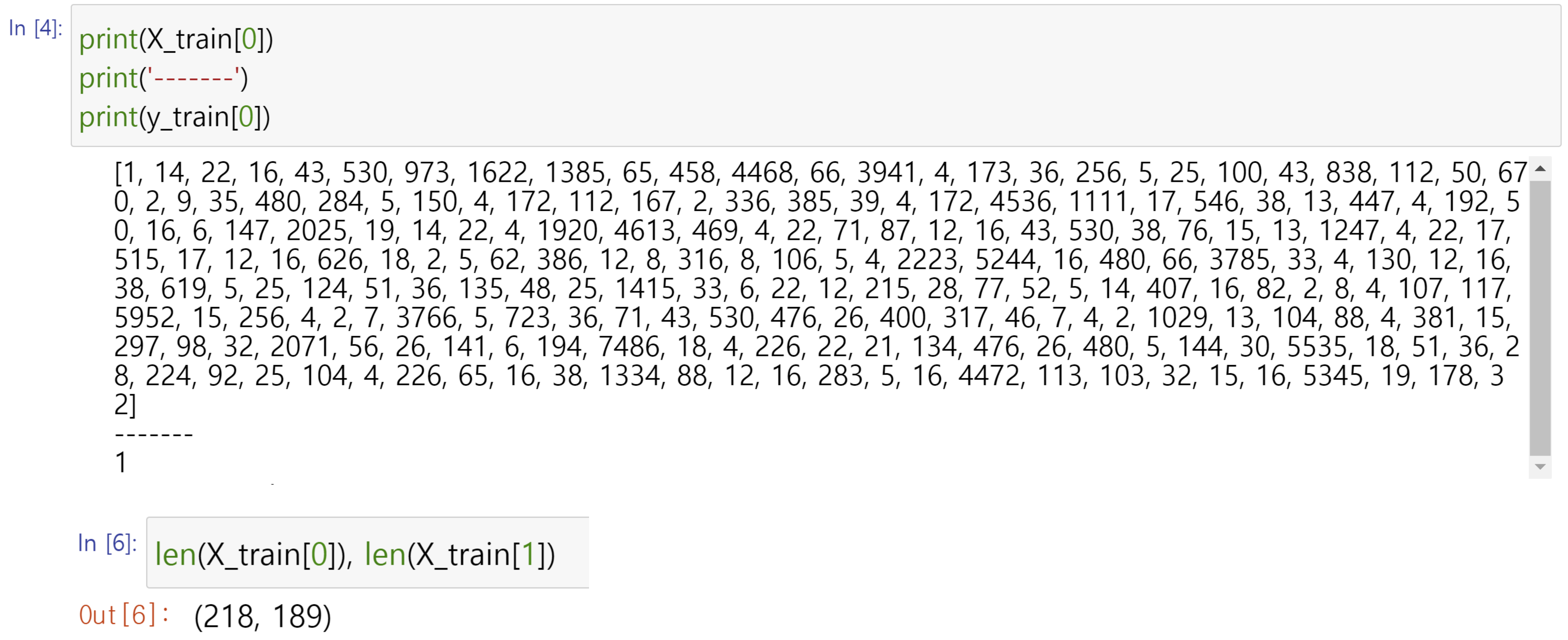
3. Explore the data

Each `label` is an integer value of either `0` (negative) or `1` (positive) review.

In [3]:
```
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



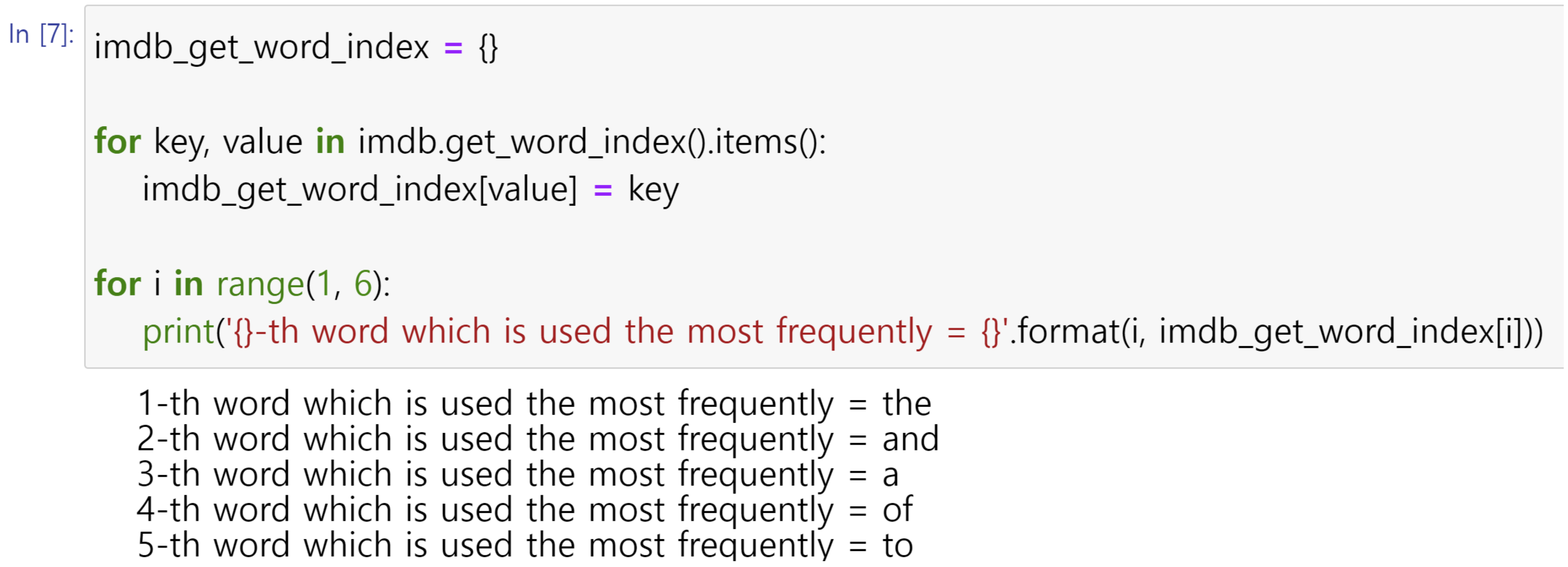
In [4]:
```
print(X_train[0])
print('-------')
print(y_train[0])
```

```
[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, 67
0, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 5
0, 16, 6, 147, 2025, 19, 14, 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, 480, 5, 144, 30, 5535, 18, 51, 36, 2
8, 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, 3
2]
-------
1
```

In [6]:
```
len(X_train[0]), len(X_train[1])
```

Out[6]:
```
(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

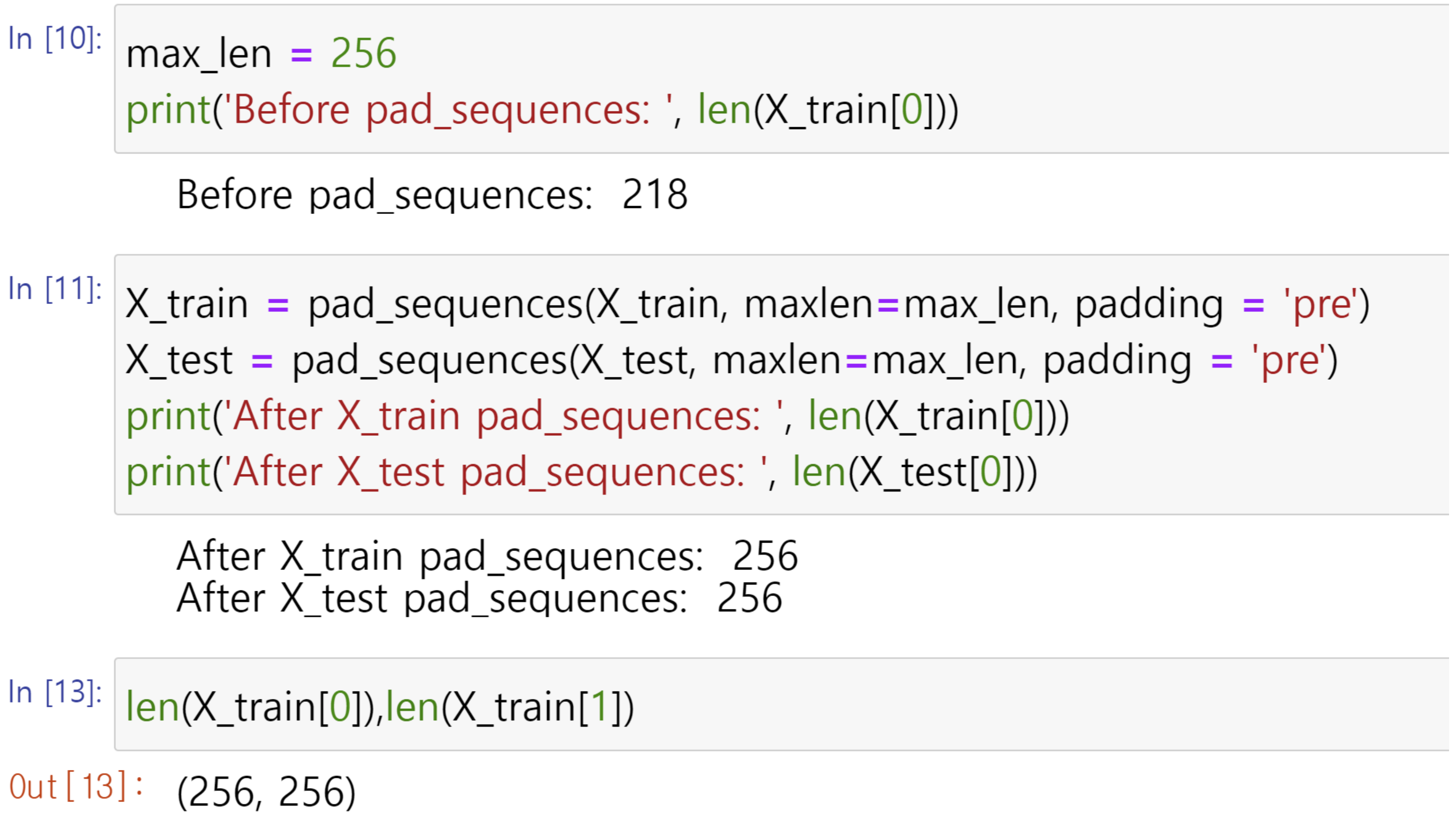
In [8]:
```python
def decode_review(text):
    return ' '.join([reverse_word_index.get(i, '?') for i in text])
```

Now we can use the `decode_review` function to display the text for the first review:

In [9]:
```python
decode_review(X_train[0])
```

Out[9]:
```
"<START> this film was just brilliant casting location scenery story direction everyone's really suited the part t
hey played and you could just imagine being there robert <UNK> is an amazing actor and now the same bei
ng director <UNK> father came from the same scottish island as myself so i loved the fact there was a real c
onnection with this film the witty remarks throughout the film were great it was just brilliant so much that i b
ought 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 i
t must have been good and this definitely was also <UNK> to the two little boy's that played the <UNK> of
norman and paul they were just brilliant children are often left out of the <UNK> list i think because the star
s that play them all grown up are such a big profile for the whole film but these children are amazing and sh
ould be praised for what they have done don't you think the whole story was so lovely because it was true a
nd was someone's life after all that was shared with us all"
```

4. Prepare the data



```
In [10]:
max_len = 256
print('Before pad_sequences: ', len(X_train[0]))
```

```
Before pad_sequences:  218
```

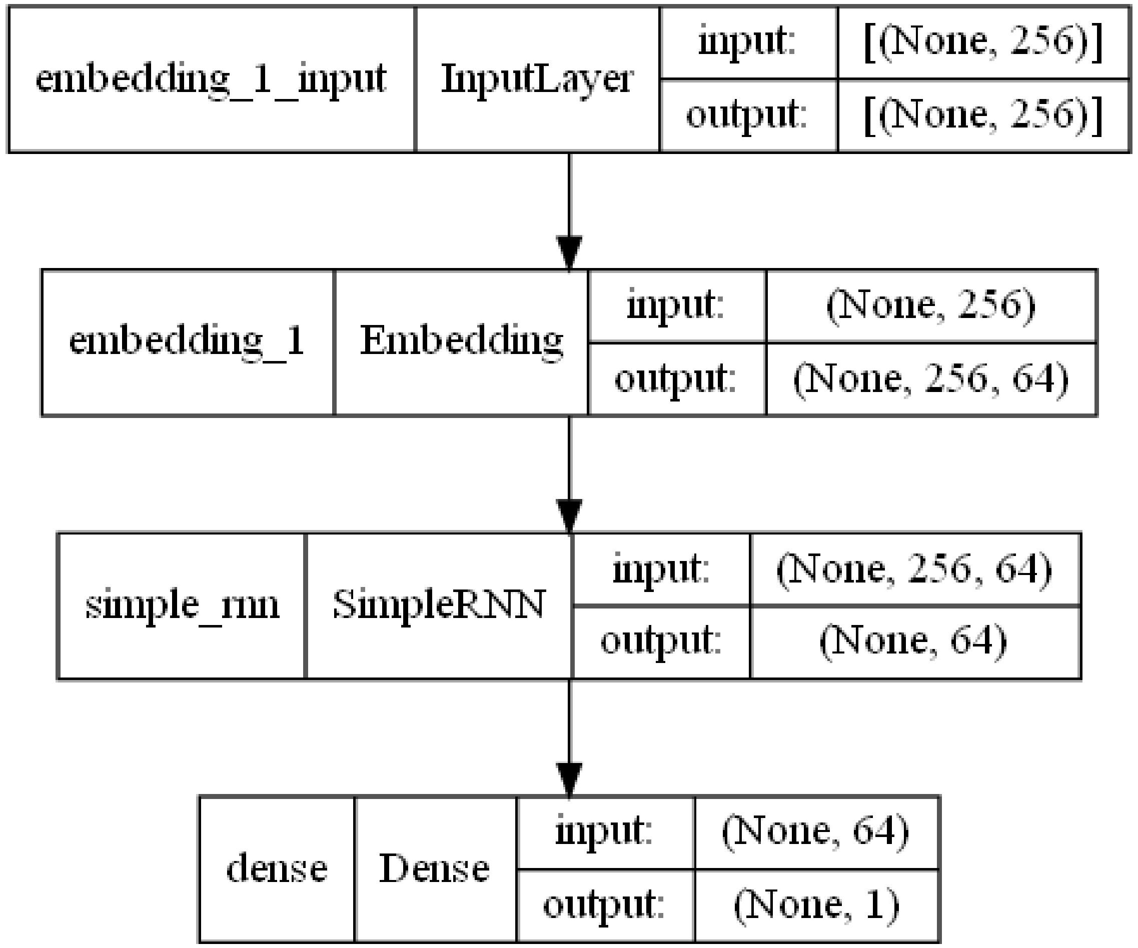
```
In [11]:
X_train = pad_sequences(X_train, maxlen=max_len, padding = 'pre')
X_test = pad_sequences(X_test, maxlen=max_len, padding = 'pre')
print('After X_train pad_sequences: ', len(X_train[0]))
print('After X_test pad_sequences: ', len(X_test[0]))
```

```
After X_train pad_sequences:  256
After X_test pad_sequences:  256
```

```
In [13]:
len(X_train[0]),len(X_train[1])
```

```
Out[13]: (256, 256)
```

5. Build the model



5. Build the model

In [14]:
```
nOut = 64
model = Sequential()

model.add(Embedding(input_dim = num_words, output_dim = nOut, input_length = max_len))
model.add(Bidirectional(LSTM(nOut)))
model.add(Dense(64, activation = 'relu'))
model.add(Dense(1))
```

(b) The layers are stacked sequentially to build the classifier:

- The first layer is an `Embedding` layer. which takes the integer-encoded vocabulary and looks up the embedding vector for each word-index.
- The resulting dimensions are: `(batch, sequence, embedding)`.

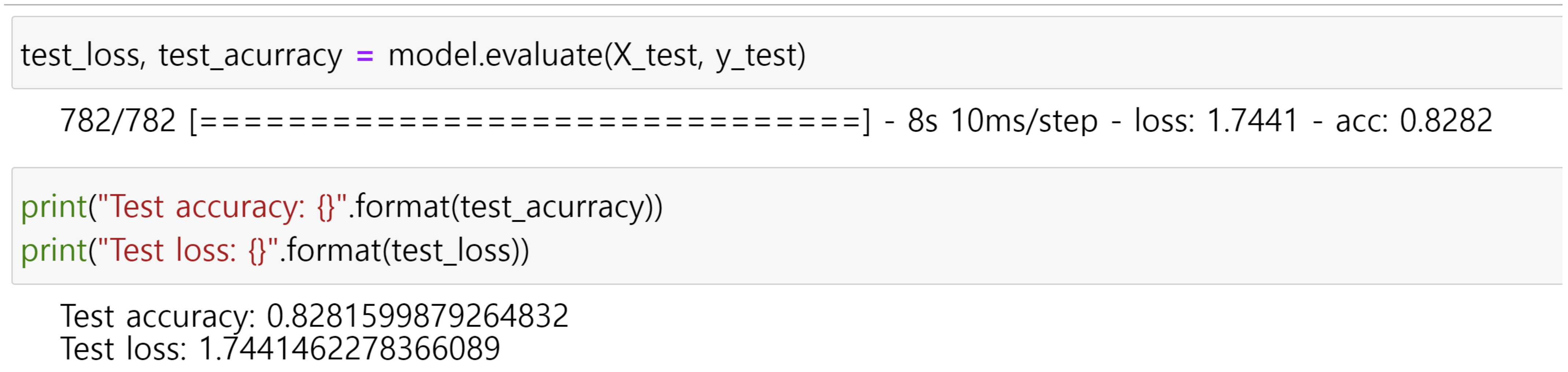
Model compile

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

```
n_value = 20000

partial_X_train = X_train[:n_value]
partial_y_train = y_train[:n_value]
```

```
history = model.fit(partial_X_train, partial_y_train,
                    epochs=40, batch_size=512,
                    validation_split = 0.2, verbose=1)
```

7. Evaluate the model

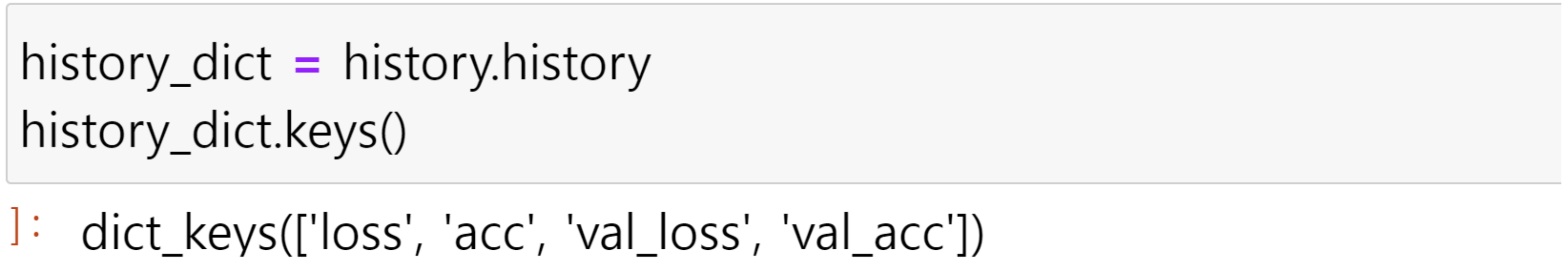


```
test_loss, test_acurracy = model.evaluate(X_test, y_test)
```

```
782/782 [==============================] - 8s 10ms/step - loss: 1.7441 - acc: 0.8282
```

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

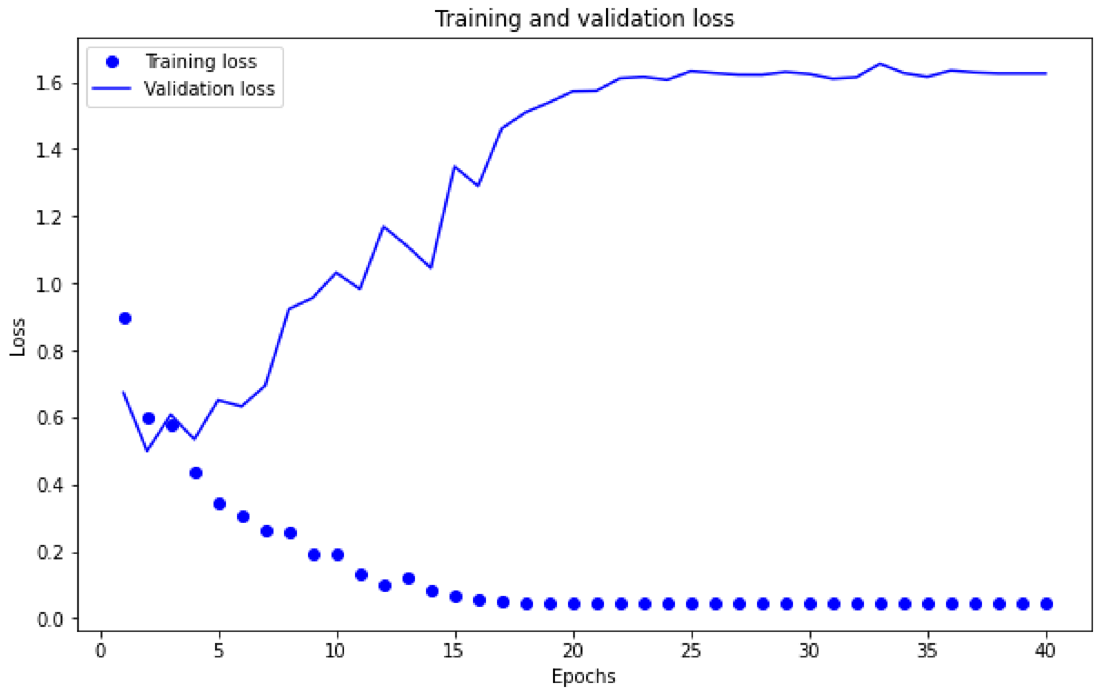
```
Test accuracy: 0.8281599879264832
Test loss: 1.7441462278366089
```

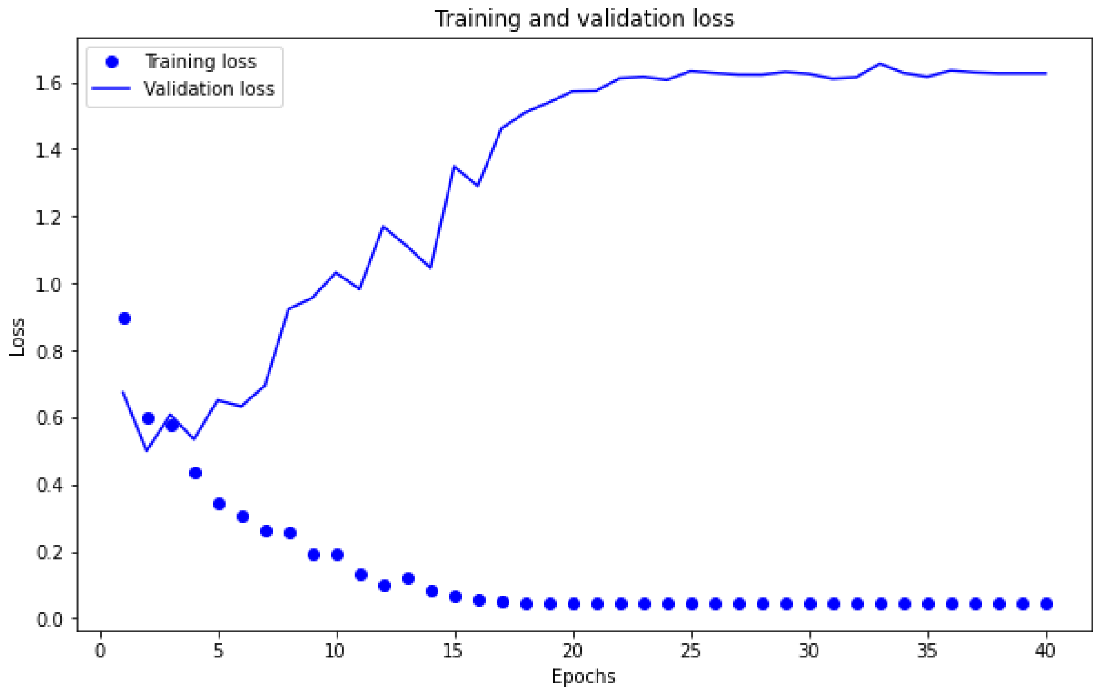


```
history_dict = history.history
history_dict.keys()
```

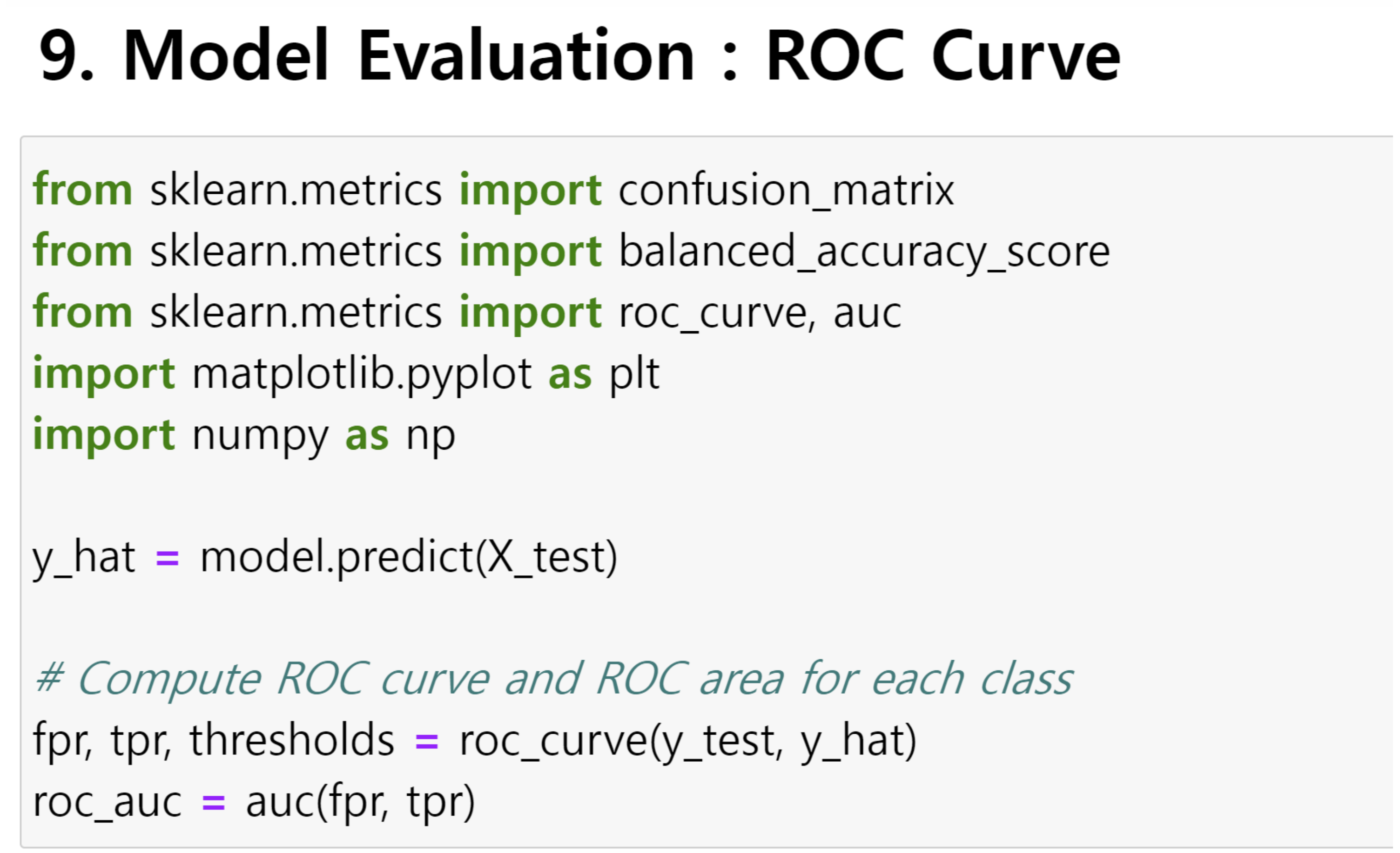
```
]:  dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

Create a graph of accuracy





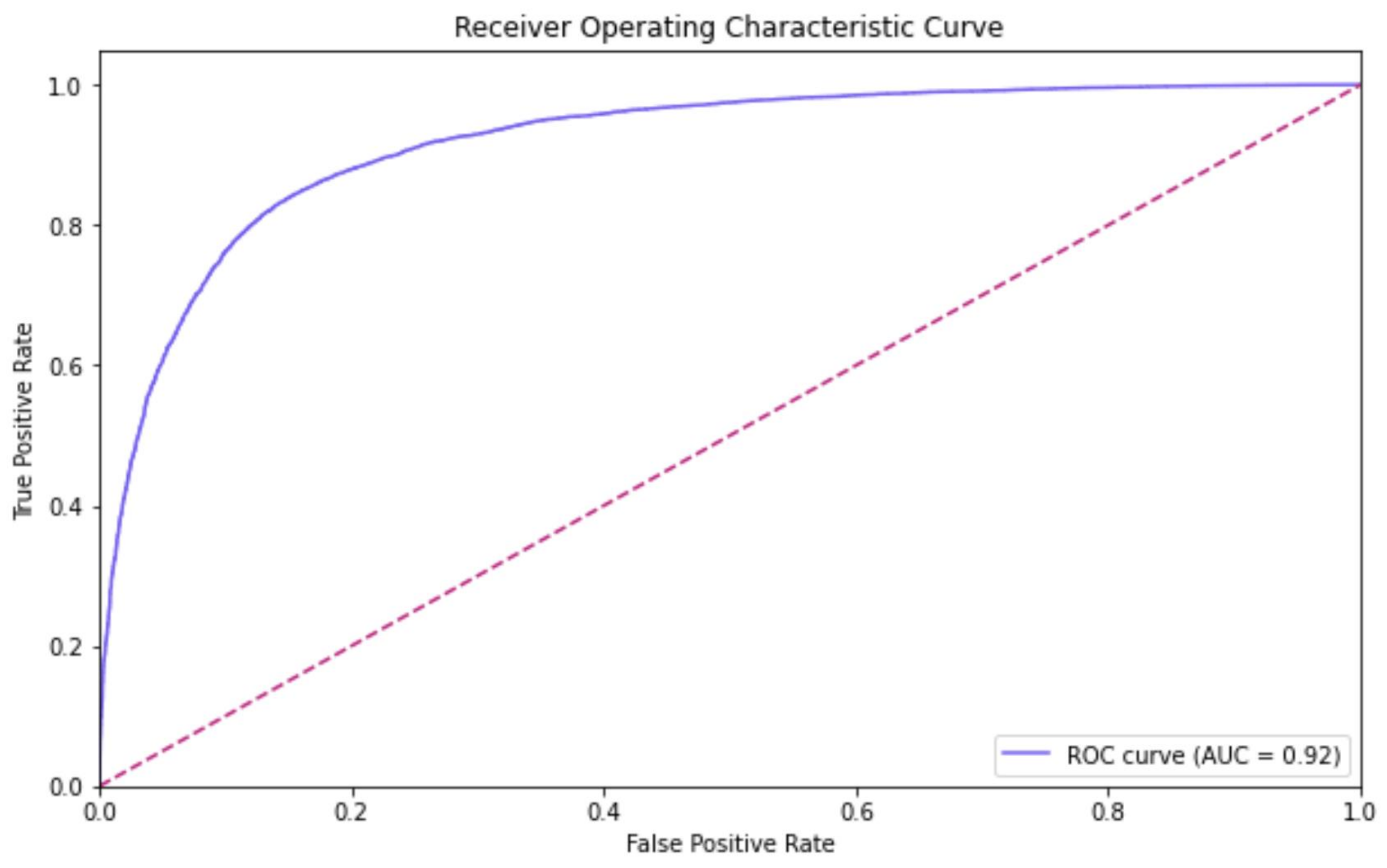
ROC AUC



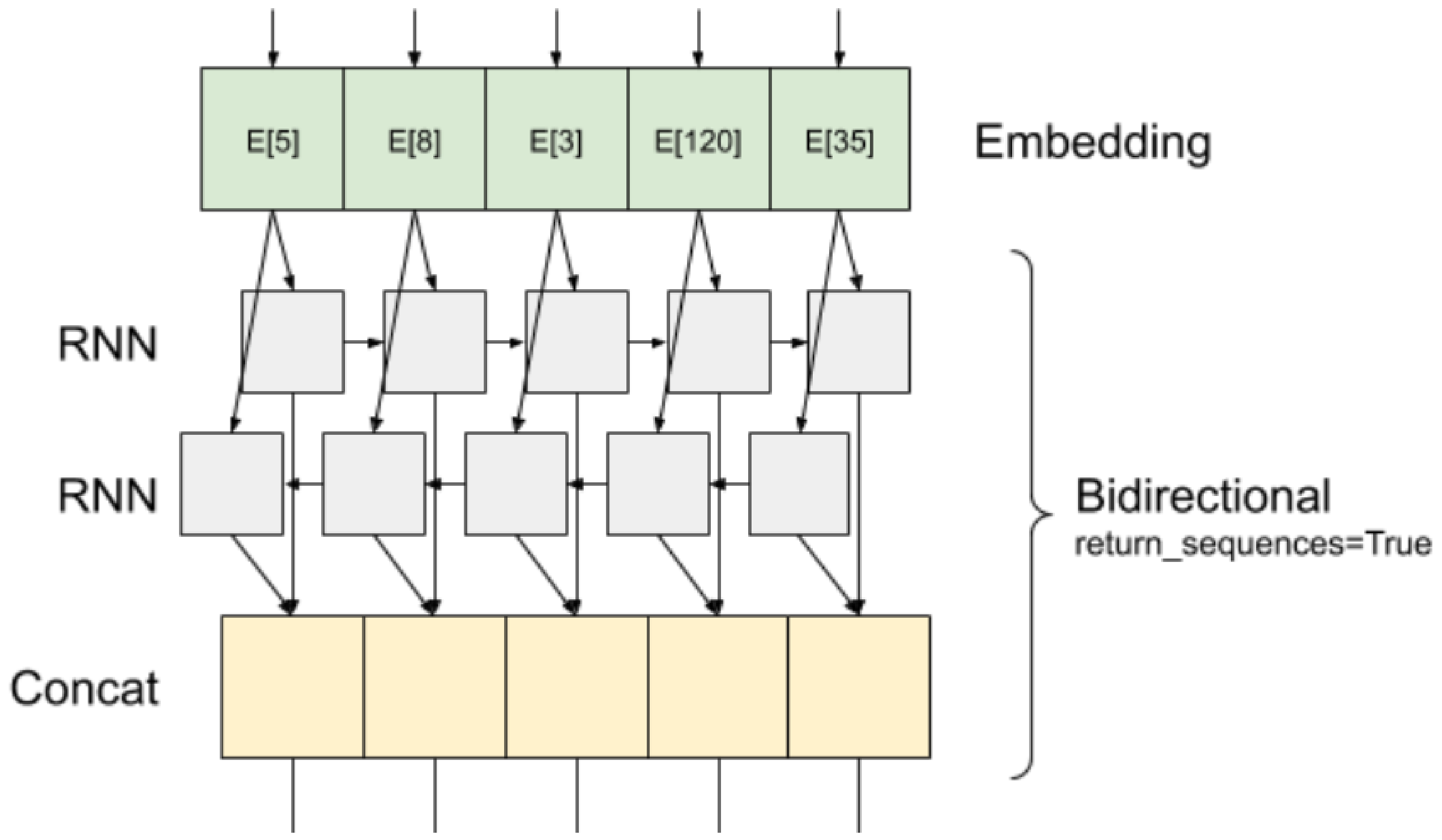
9. Model Evaluation : ROC Curve

```
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)
```

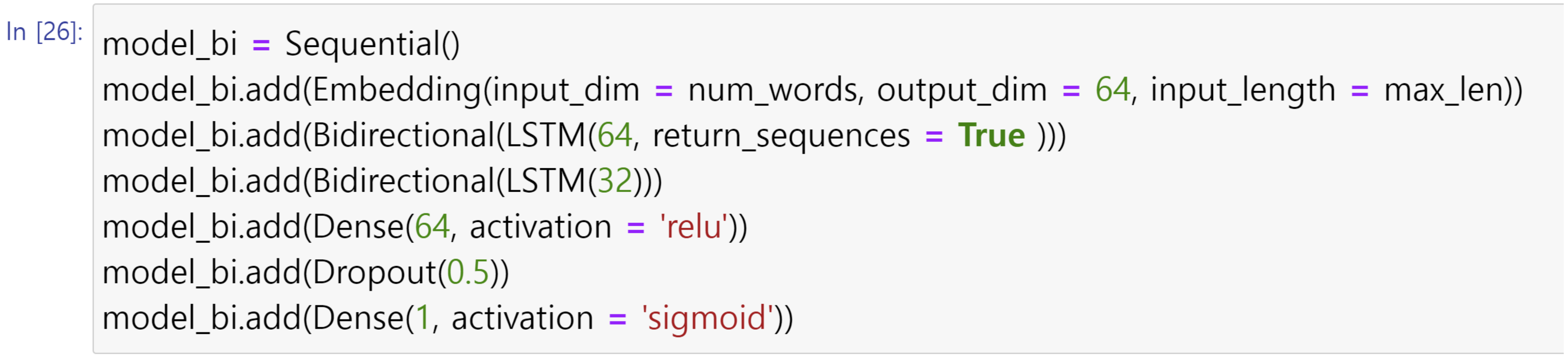
ROC Curve (AUC = 0.92)



9. Stack two or more LSTM layers



Model



```
In [26]:
model_bi = Sequential()
model_bi.add(Embedding(input_dim = num_words, output_dim = 64, input_length = max_len))
model_bi.add(Bidirectional(LSTM(64, return_sequences = True )))
model_bi.add(Bidirectional(LSTM(32)))
model_bi.add(Dense(64, activation = 'relu'))
model_bi.add(Dropout(0.5))
model_bi.add(Dense(1, activation = 'sigmoid'))
```