# 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

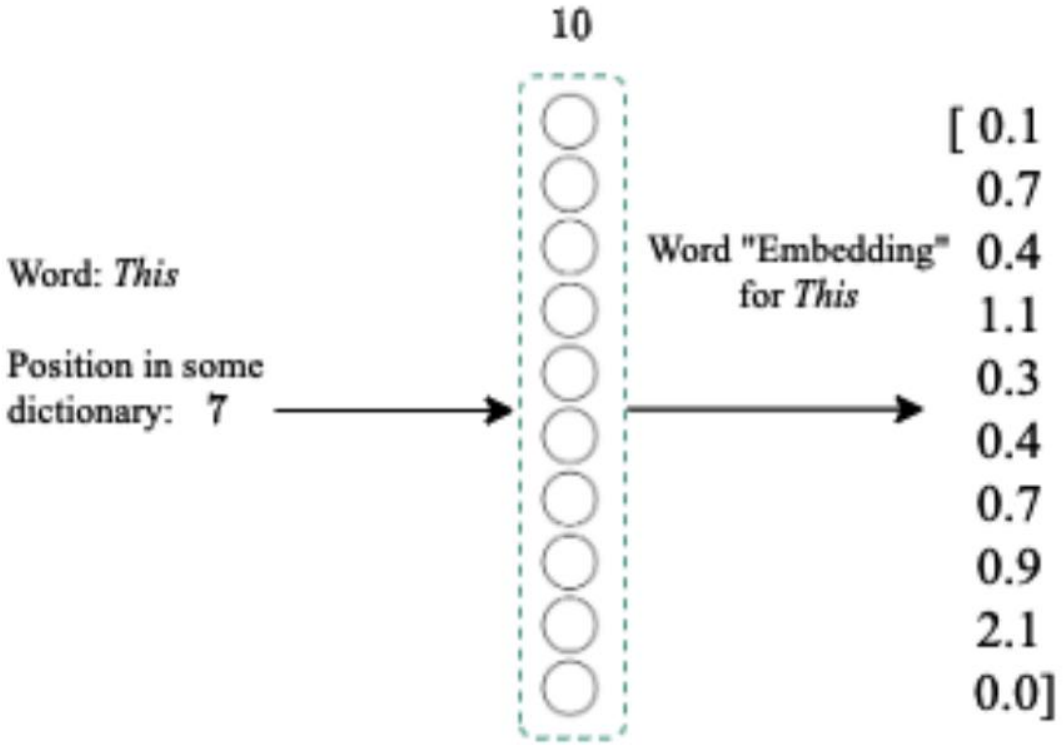
➢ 여기서는 간단한 방법으로 해보자.

embddng_dim = 10

- ✓ 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.

**This**

index : 7,
vector size : 10

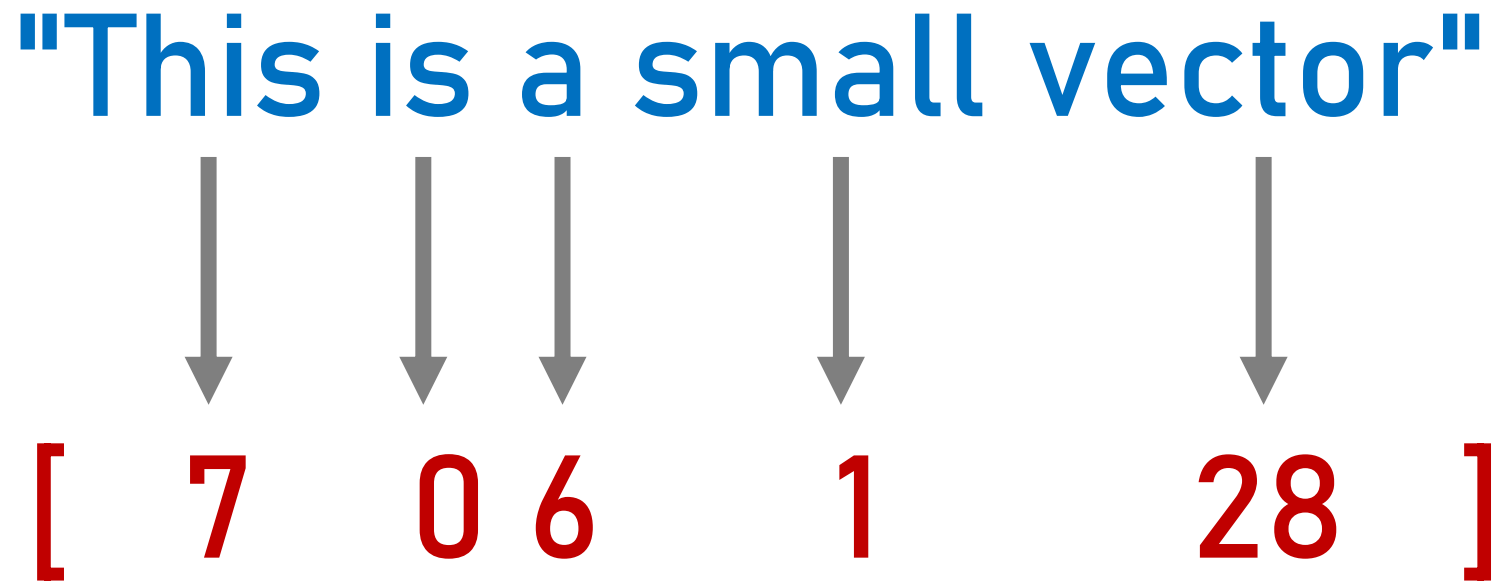


# Sequence of vectors

- 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.

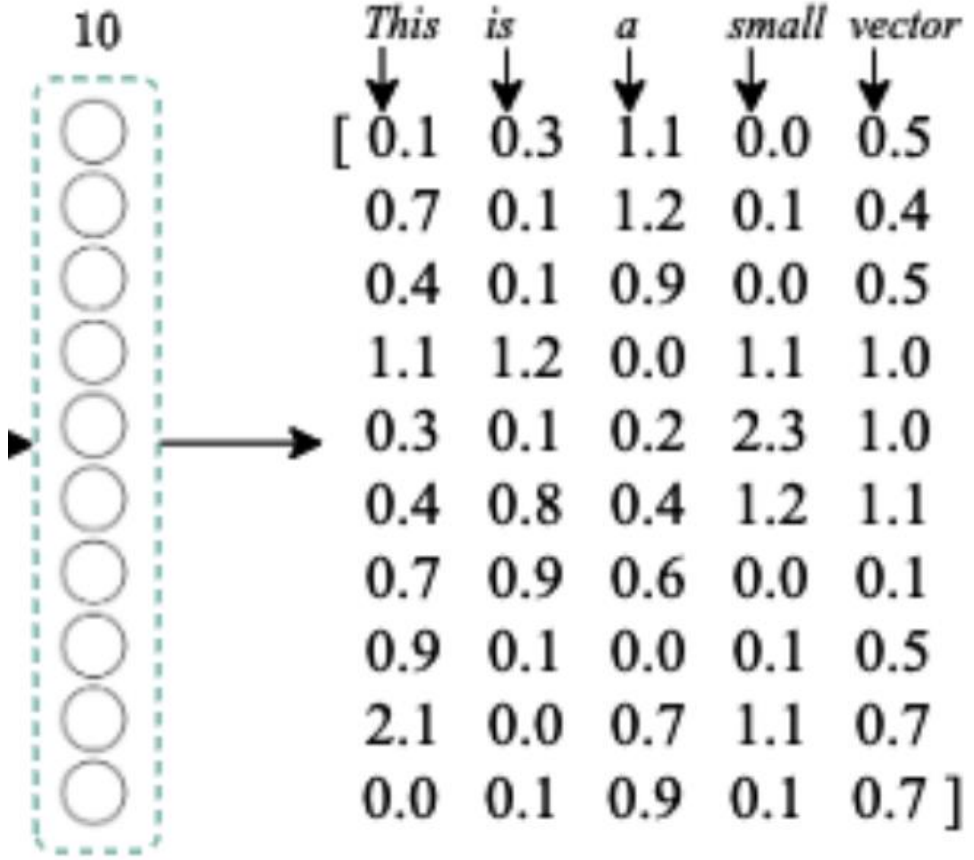
vocab_size = 30

seq_length = 5

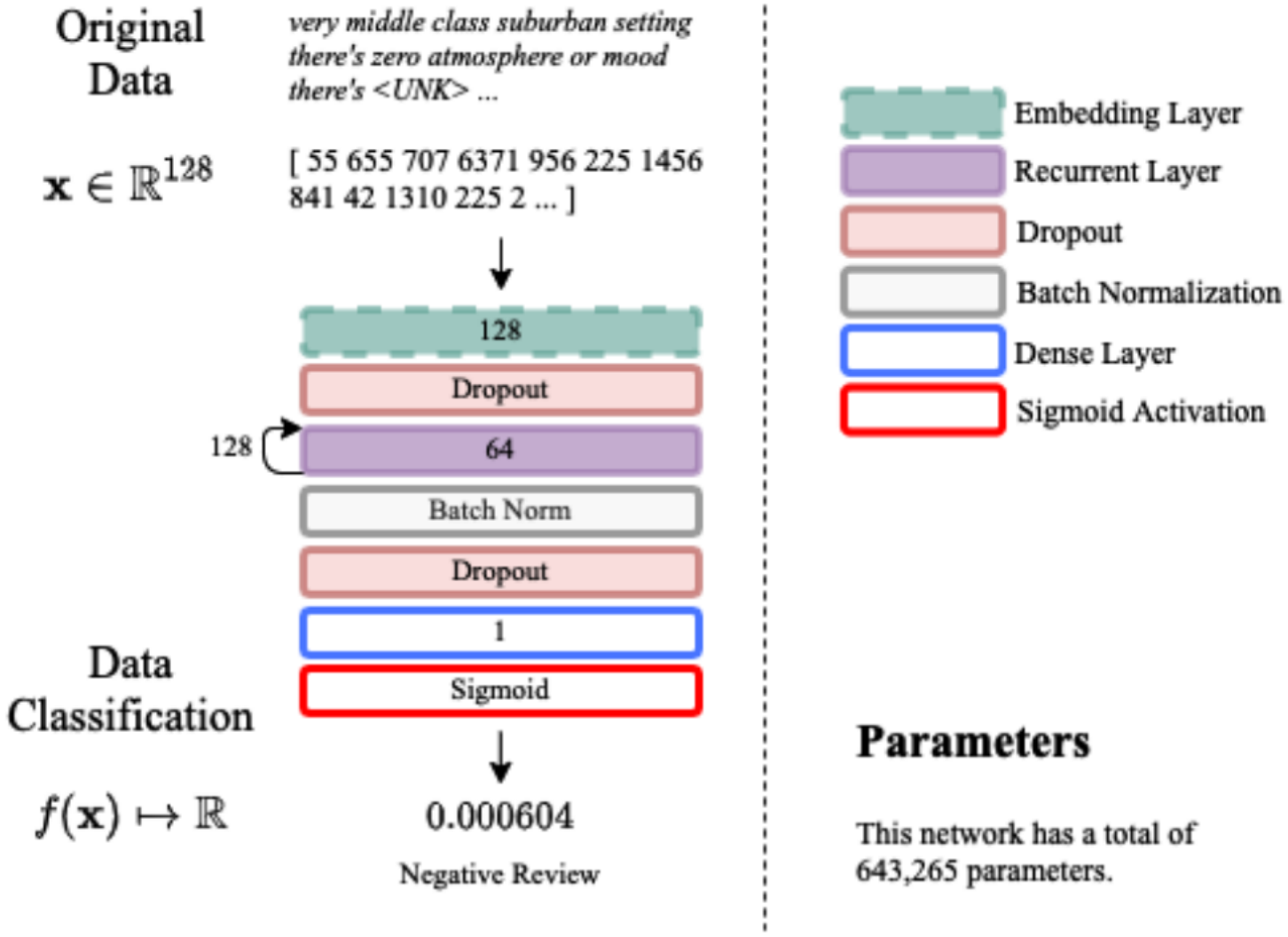


# Sequence of word embeddings

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.

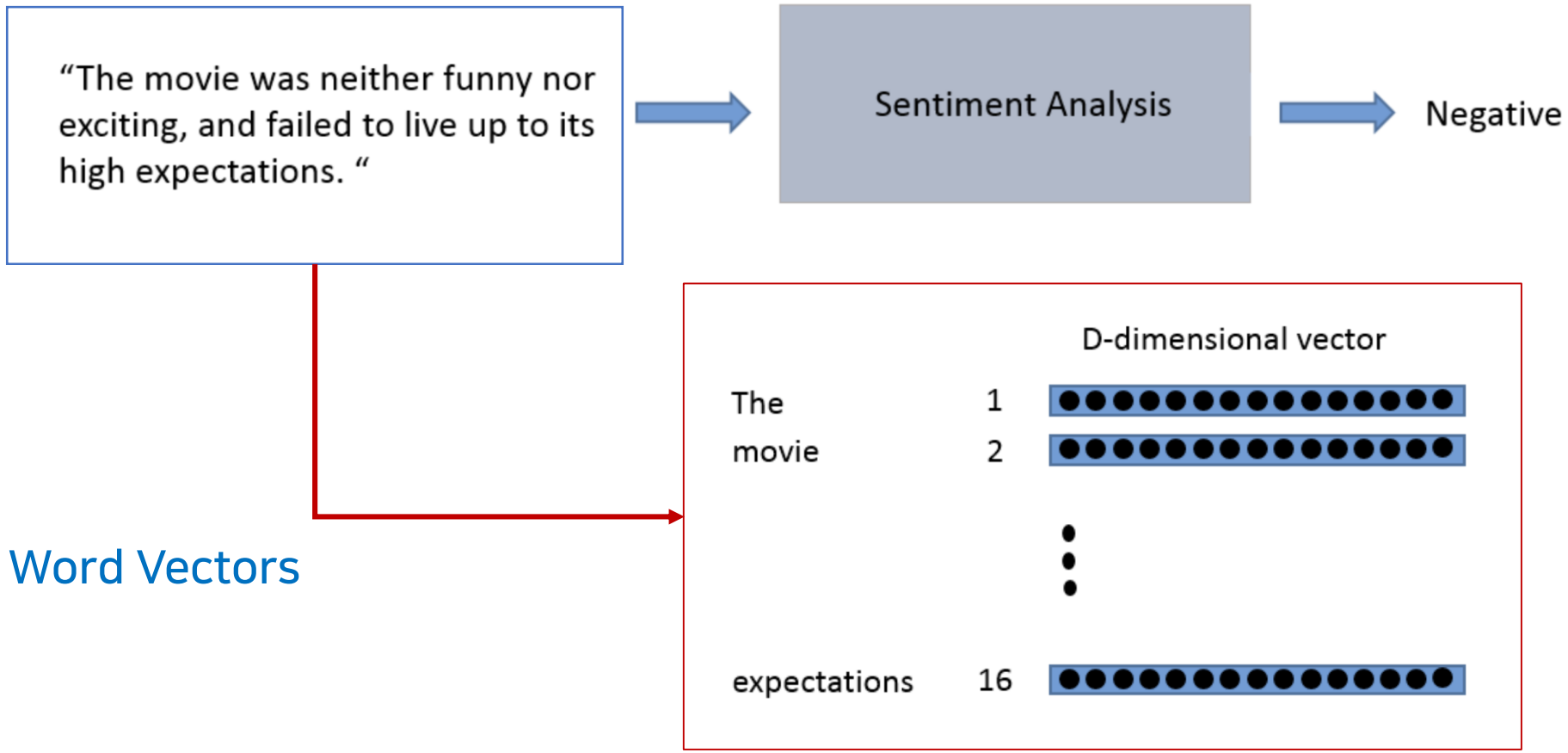
# Deep Learning for NLP

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


| | 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.

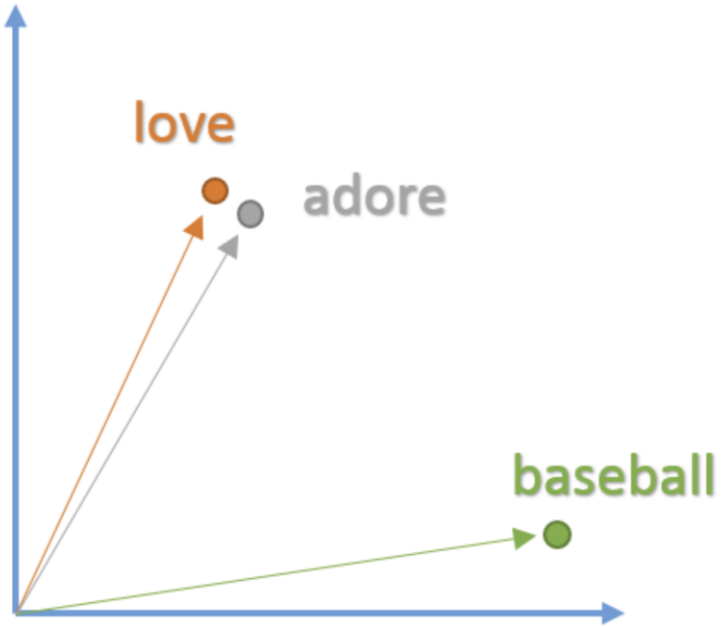



- Word Vectors

# Word embedding : Word2Vec

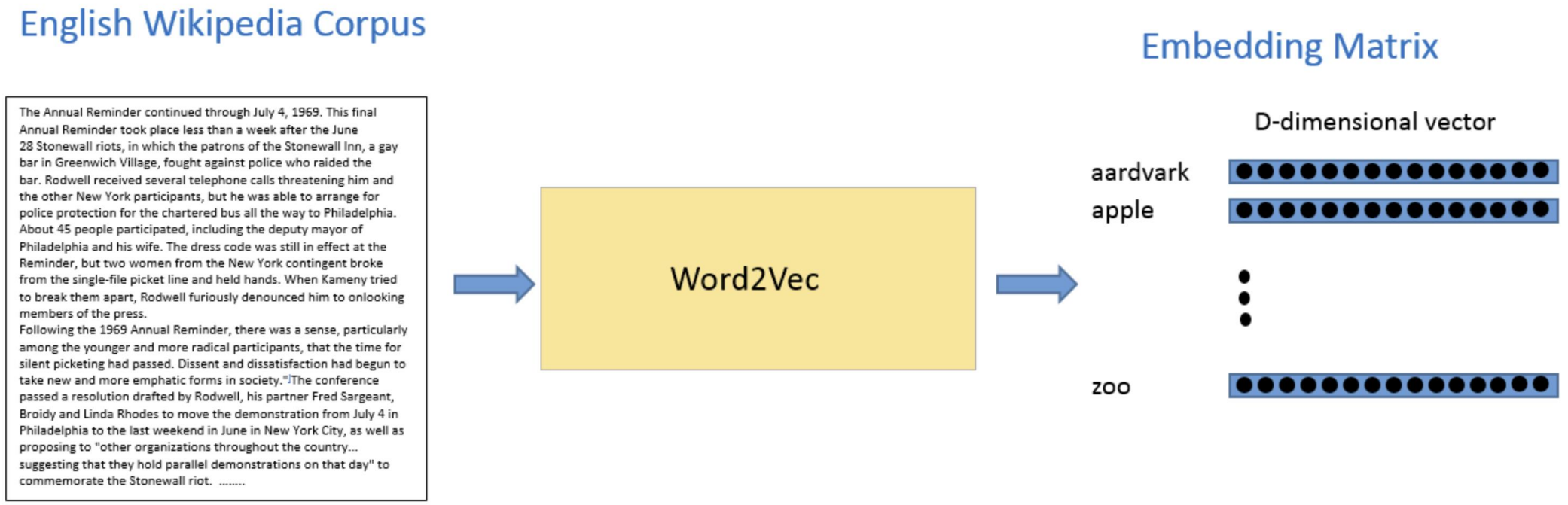
- The vector representation of a word is also known as a word embedding.
- Word2Vec :
  - ✓ the model creates word vectors by looking at the context with which words appear in sentences.

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.



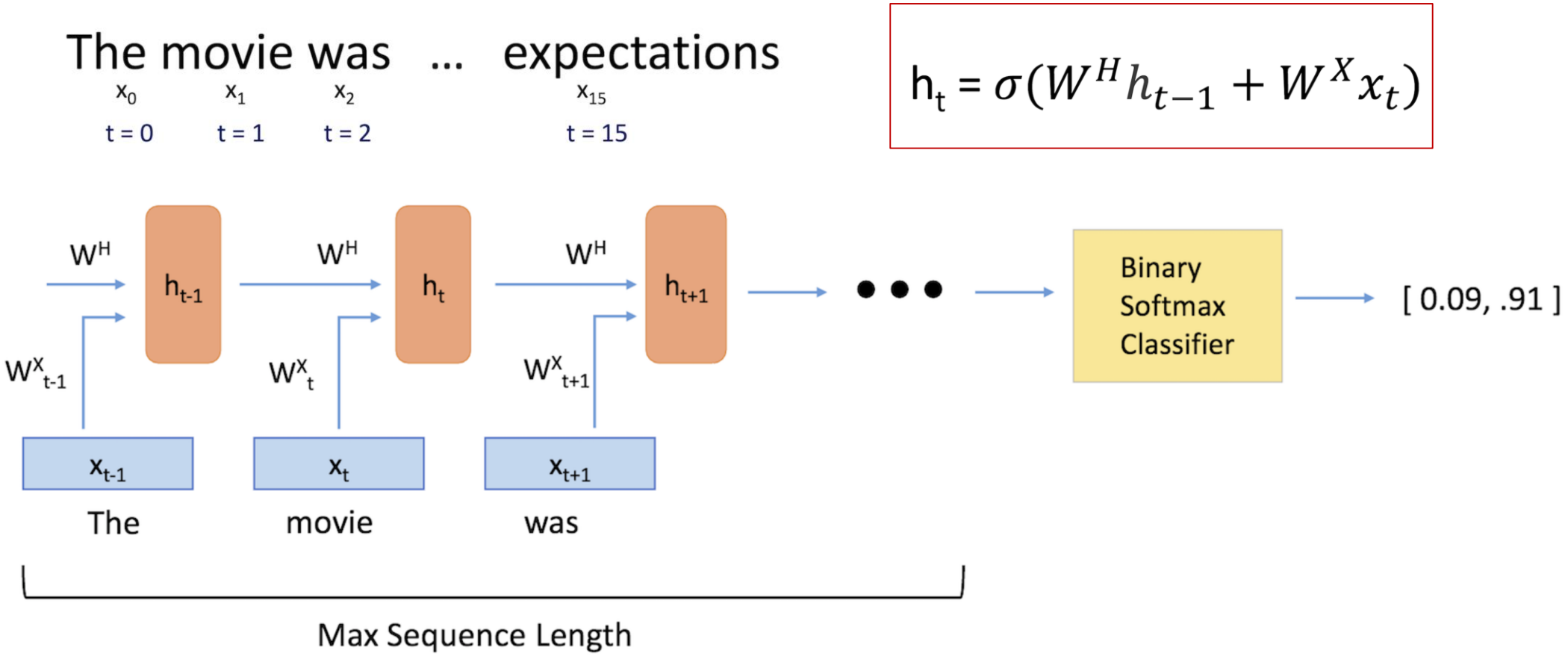
# Embedding matrix

- 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.



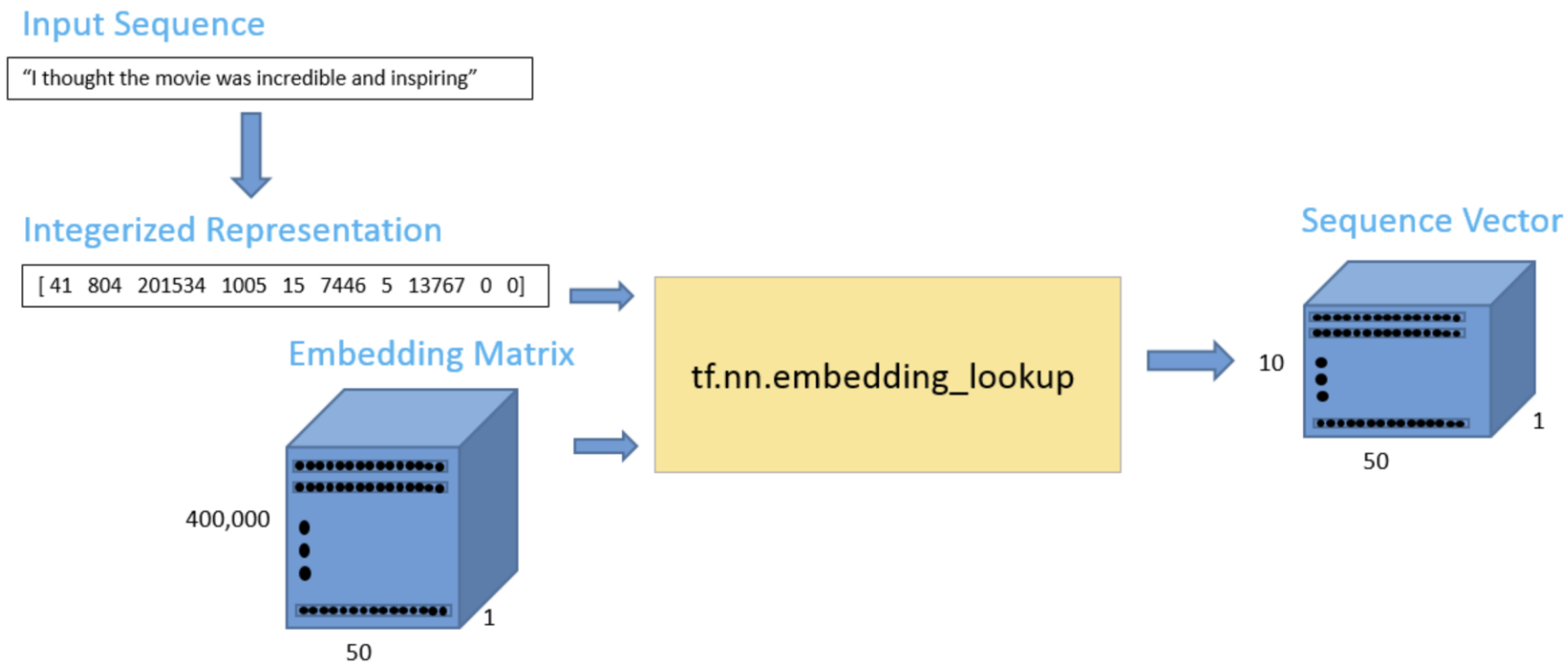
# RNN

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



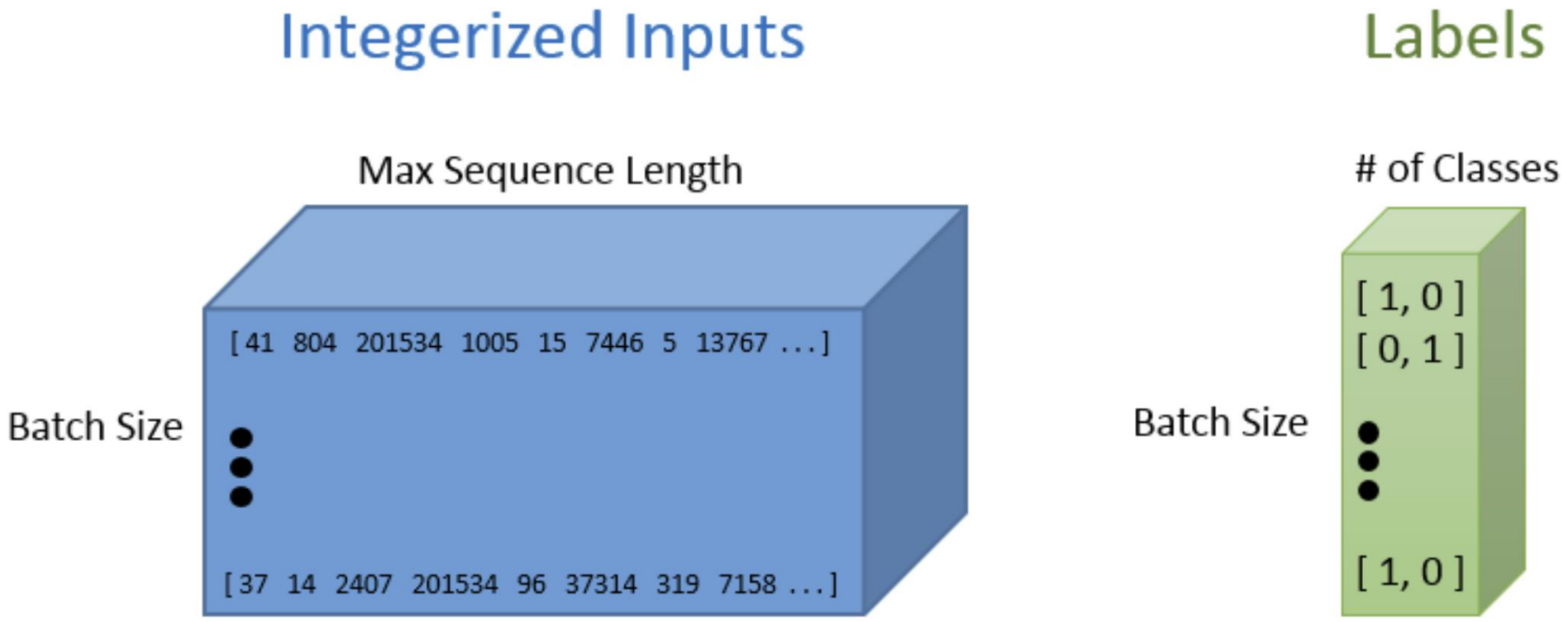
# The data pipeline : a simple example

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

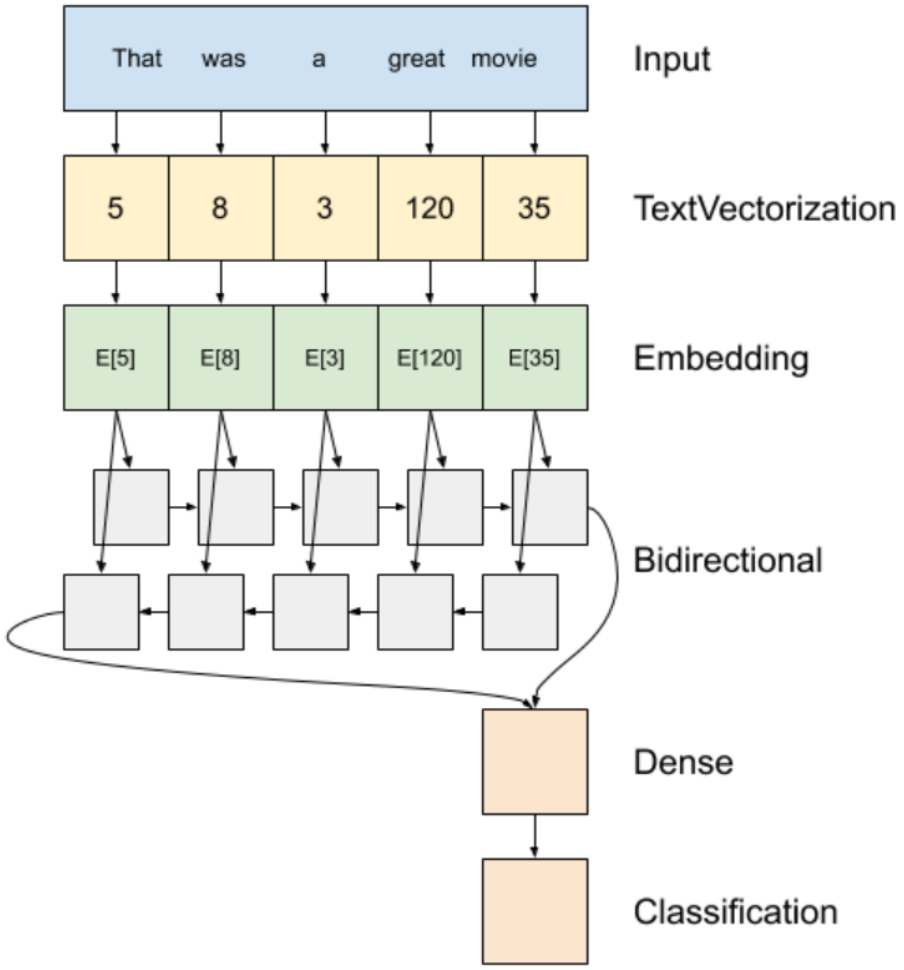



The 10 x 50 output should contain the 50 dimensional word vectors for each of the 10 words in the sequence.

# 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, 670, 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, 50, 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, 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
```

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]:
```
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]:
```
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
```

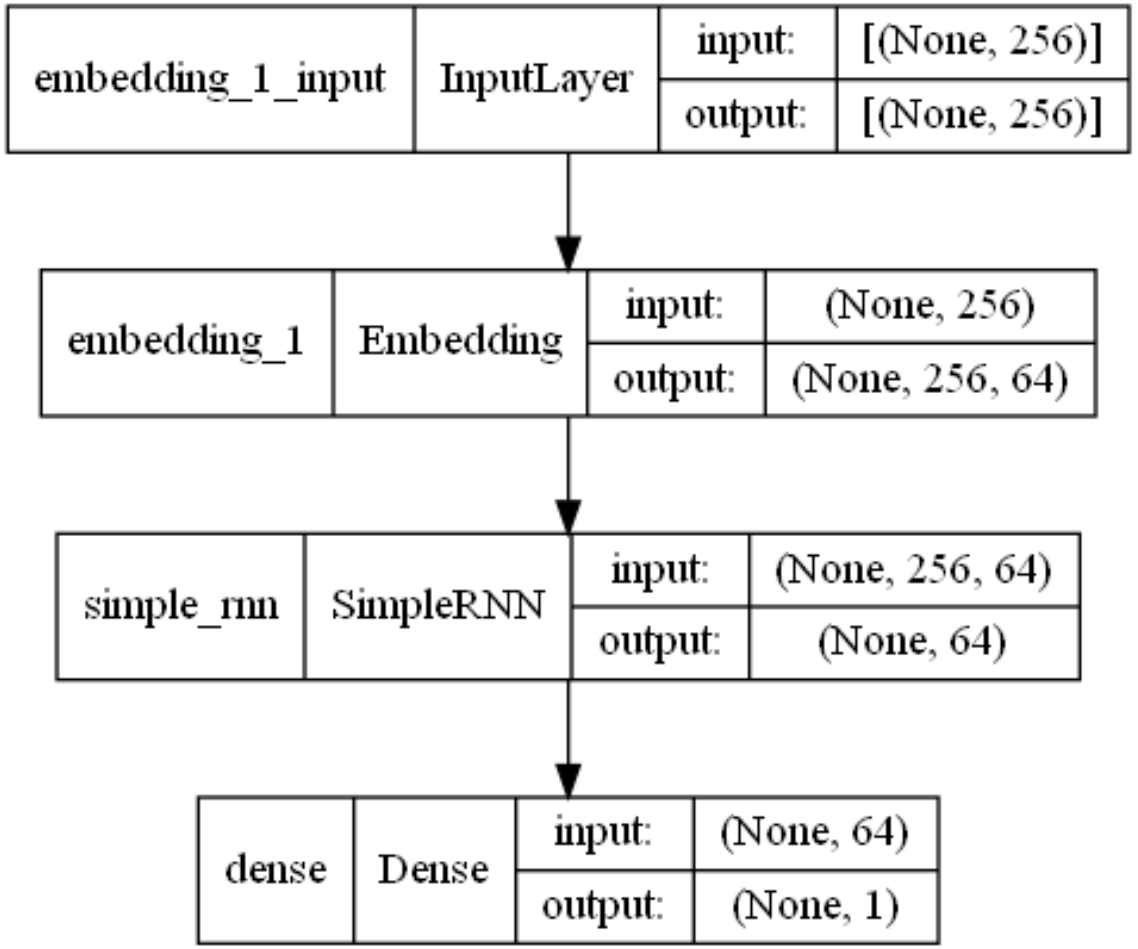
In [11]:
```
X_train = pad_sequences(X_train, maxlen=max_len, padding = 'pre')
X_test = pad_sequences(X_test, maxlen=max_len, padding = 'pre')
print('After X_train pad_sequences: ', len(X_train[0]))
print('After X_test pad_sequences: ', len(X_test[0]))
```

```
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
```

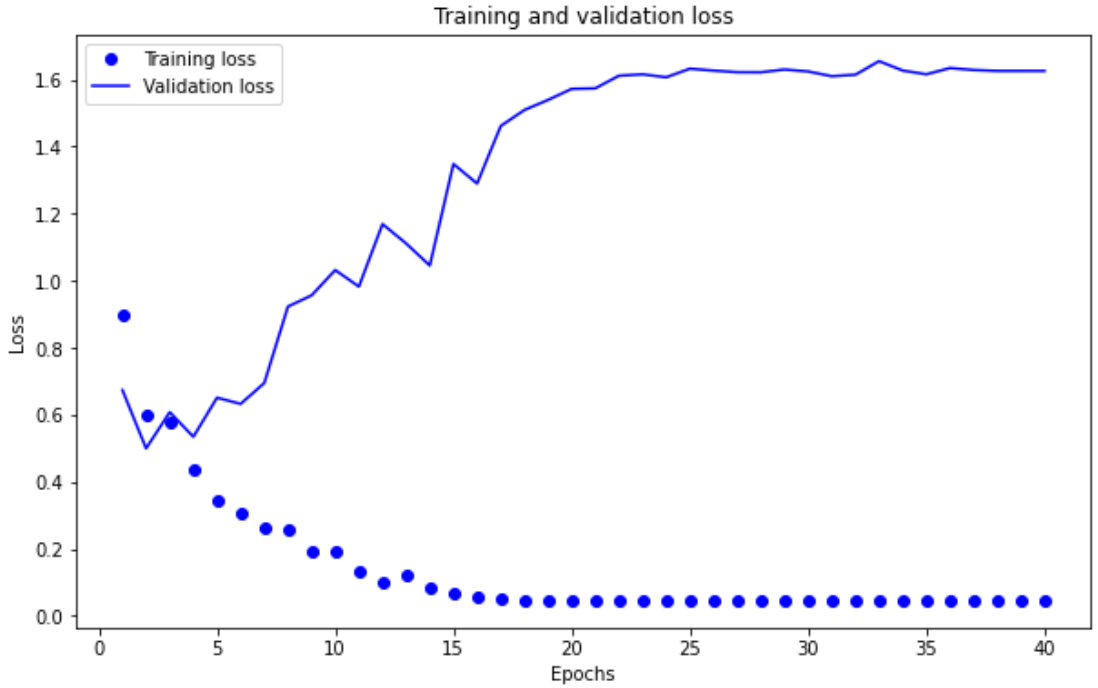
```
print("Test accuracy: {}".format(test_acurracy))
print("Test loss: {}".format(test_loss))
```

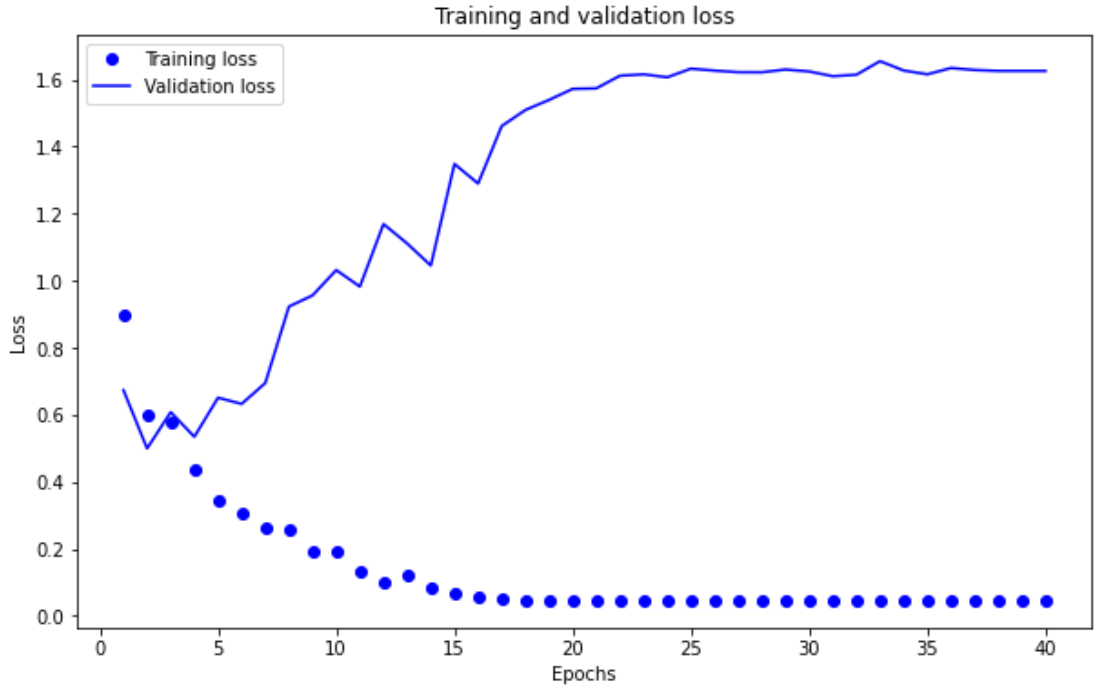
```
Test accuracy: 0.8281599879264832
Test loss: 1.7441462278366089
```

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

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

# Create a graph of accuracy

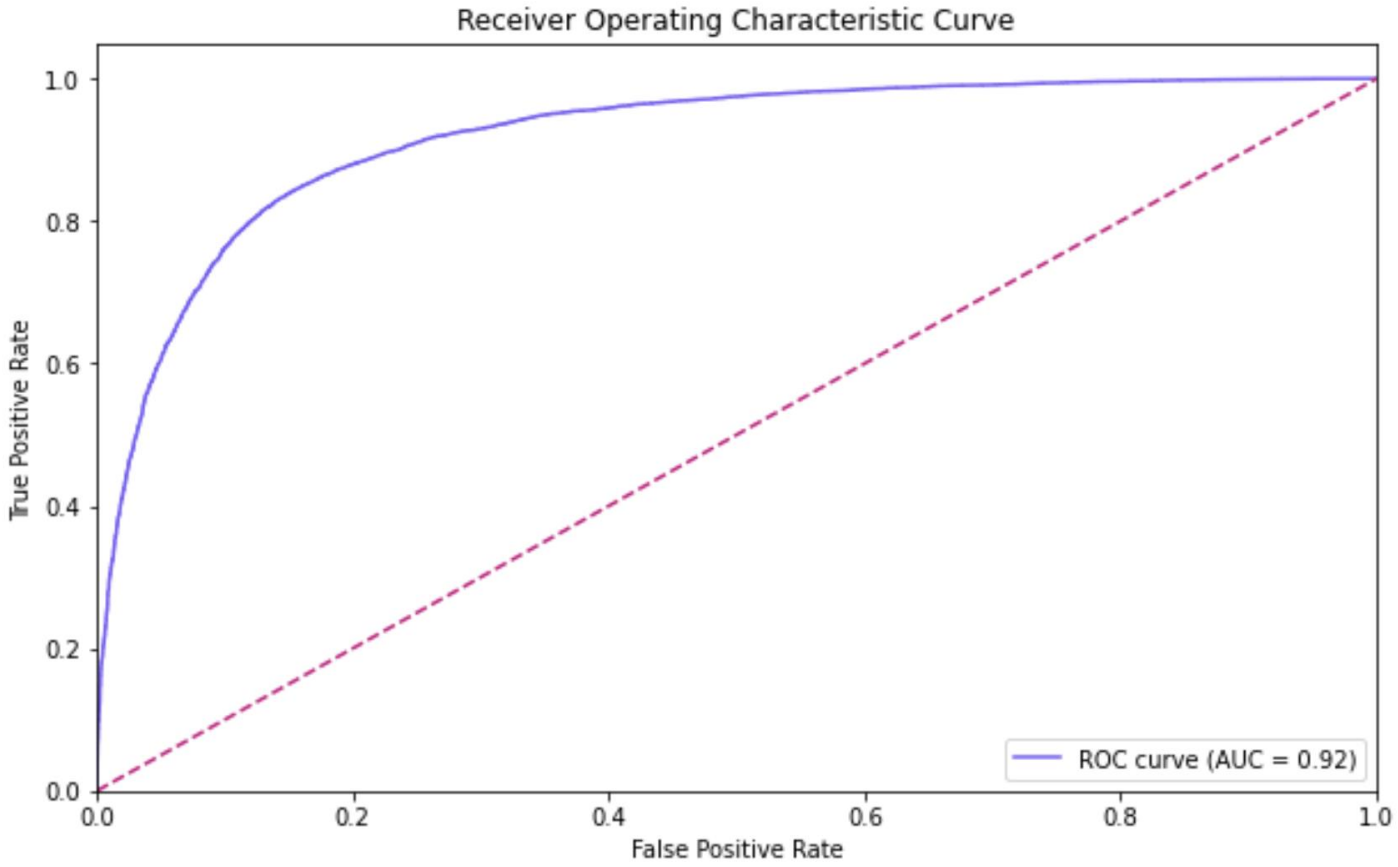


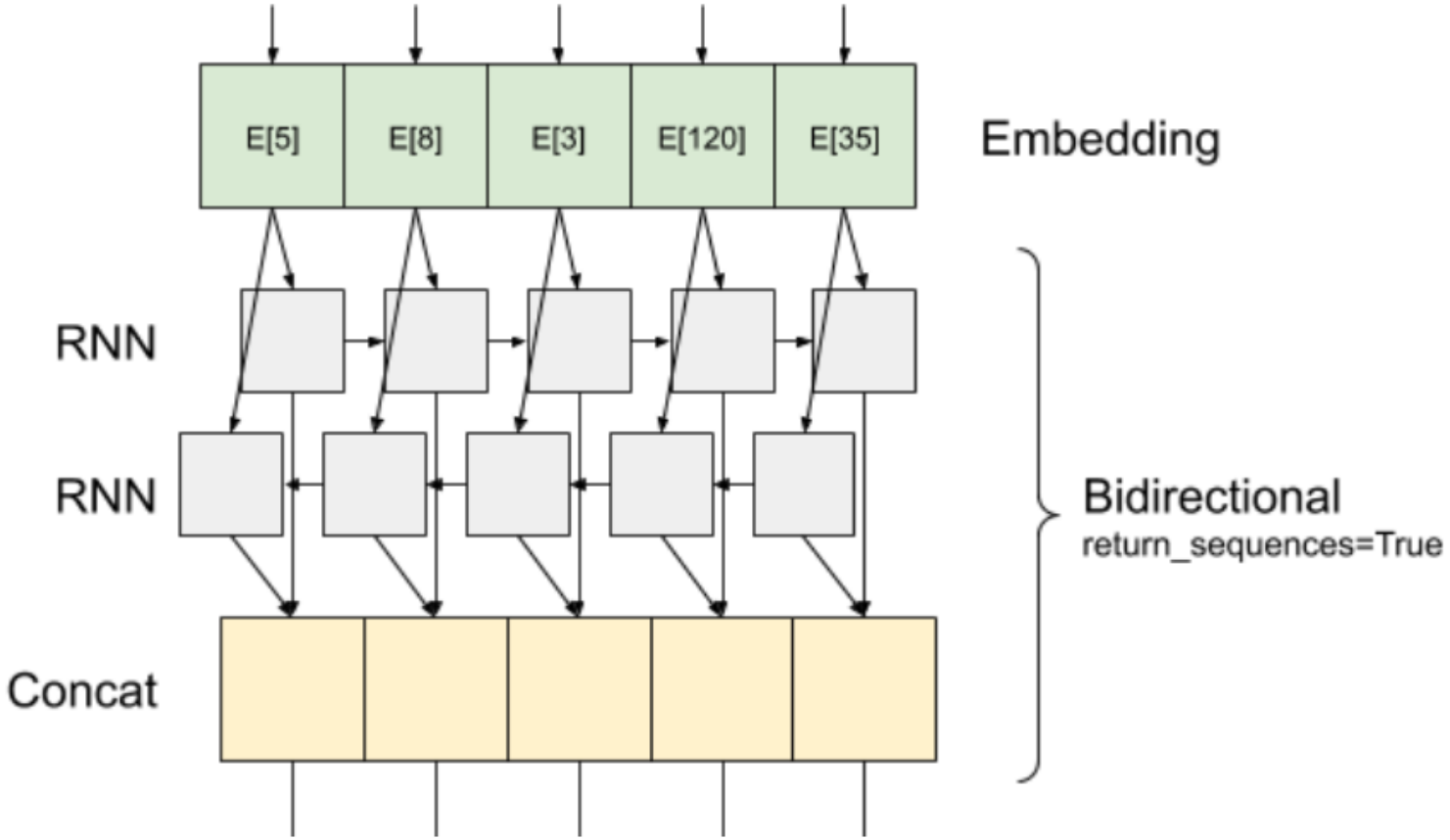
# 9. Model Evaluation : ROC Curve

```
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'))
```

# Lec12-2
# Neural Machine Translation

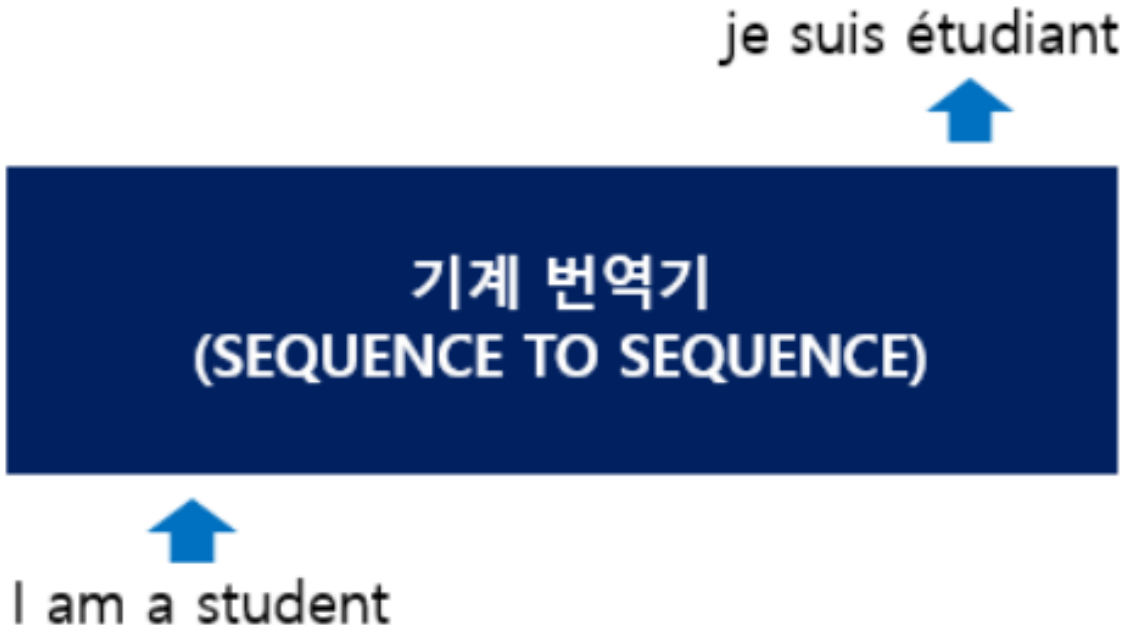
Sequence to Sequence, Seq2seq, Many-to-Many

# 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.

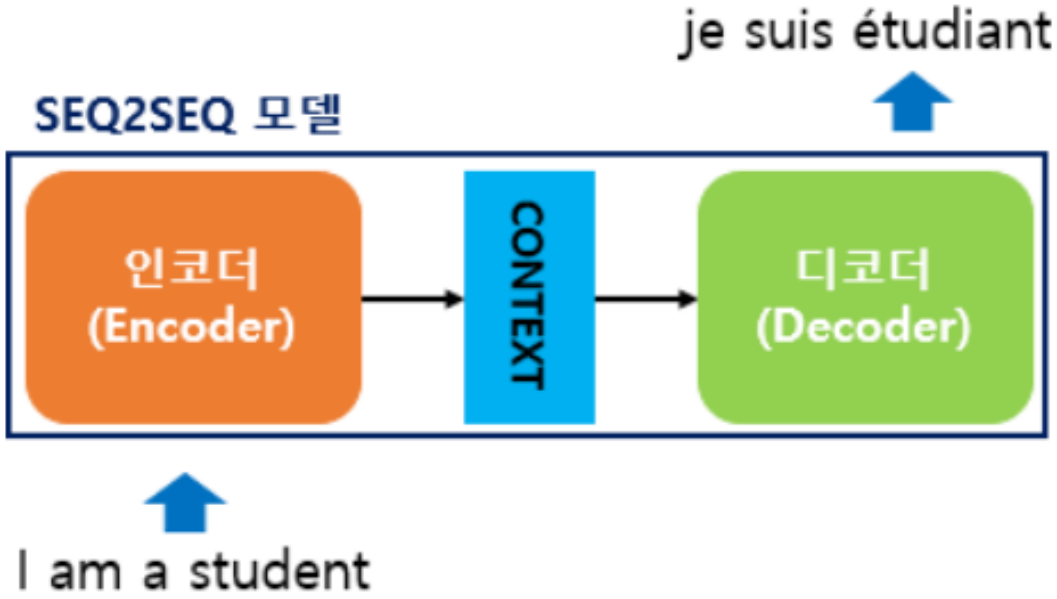
# 1) 시퀀스-투-시퀀스(Sequence-to-Sequence, seq2seq)

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



# seq2seq는 인코더와 디코더 구조

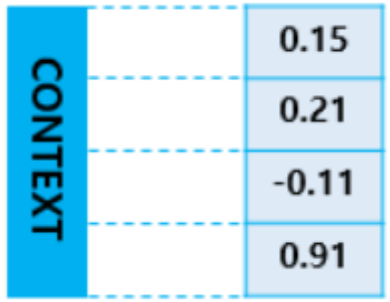
- 인코더는 단어들을 순차적으로 입력받고, 이 모든 단어 정보들을 압축해서 벡터를 만든다.
  - ✓ 입력 정보가 하나의 컨텍스트 벡터로 압축되면 인코더는 컨텍스트 벡터를 디코더로 전송합니다.
  - ✓ 디코더는 컨텍스트 벡터를 받아서 번역된 단어를 한 개씩 순차적으로 출력합니다.



# 컨텍스트 벡터(context vector)

➢ 컨텍스트 벡터의 간단한 예제

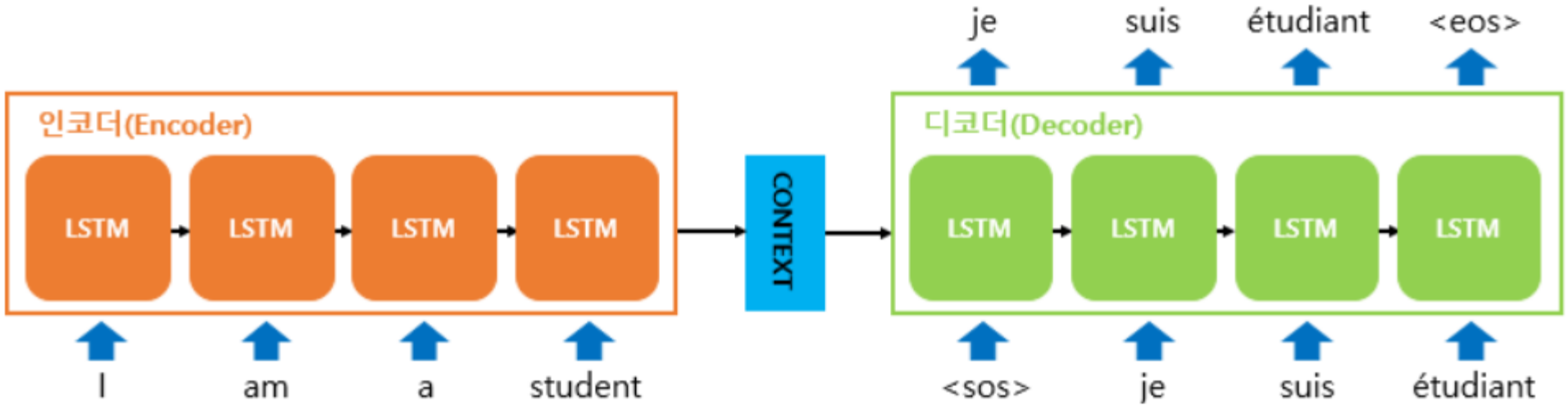
✓ 크기는 4로 표현하였다. 보통 수백 이상의 차원을 갖고있습니다.



# 인코더와 디코더는 RNN 아키텍처이다.

## ➢ 컨텍스트 벡터(context vector)

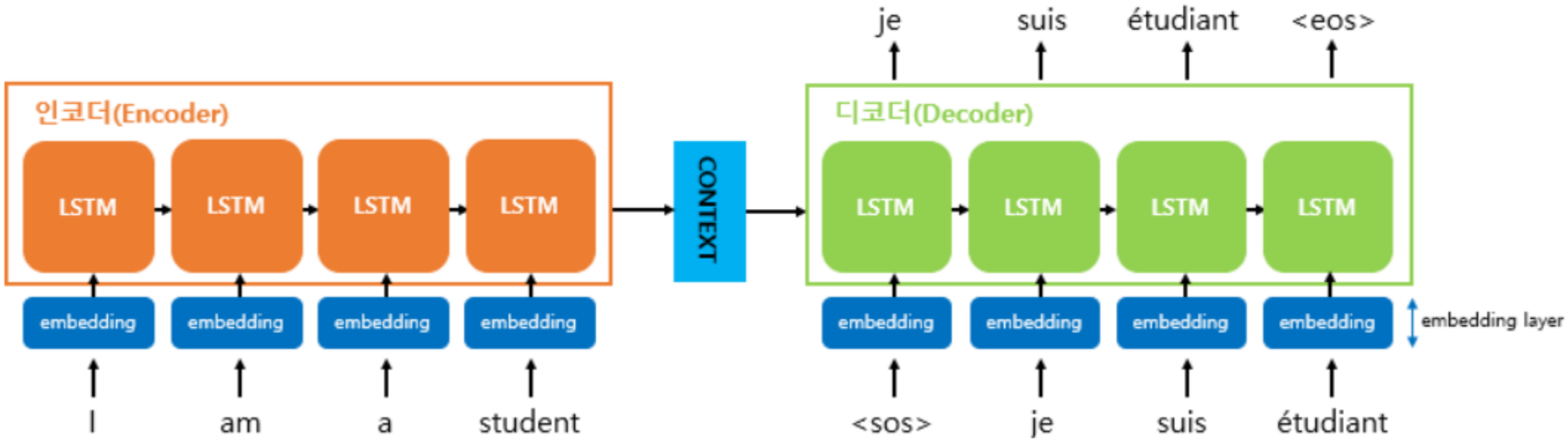
- ✓ 인코더는 문장의 모든 단어들을 순차적으로 입력받은 뒤에  모든 단어 정보들을 압축한다.
- ✓ 디코더는 컨텍스트 벡터를 받아서 번역된 단어를 한 개씩 순차적으로 출력합니다.
- ✓ 컨텍스트 벡터는 디코더 RNN 셀의 첫번째 은닉 상태로 사용됩니다.




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

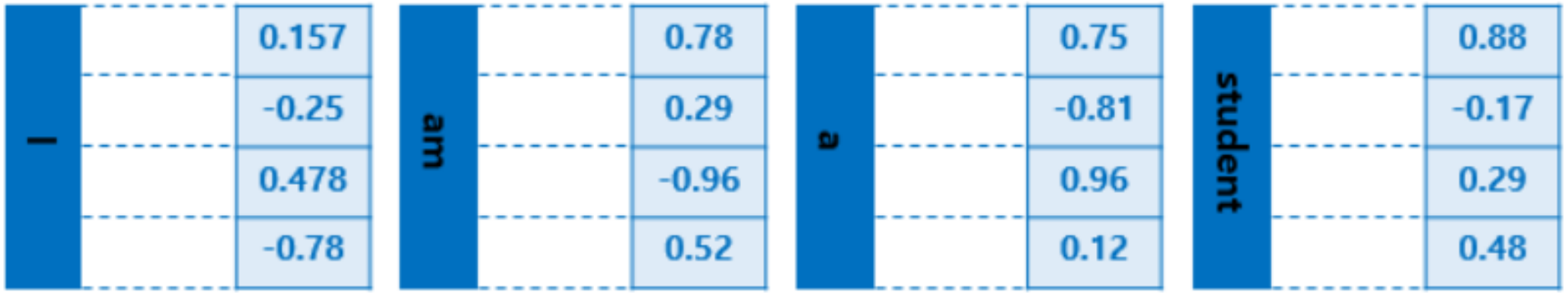
➢ 단어들은 워드 임베딩 벡터이다.



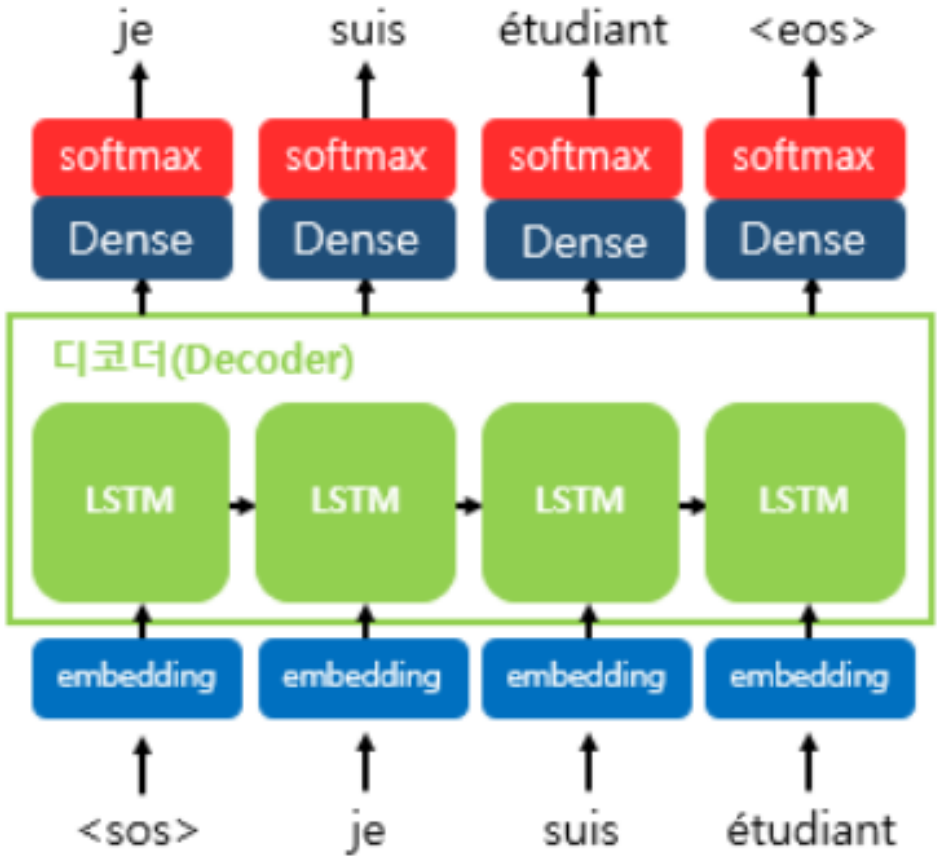
# 인코더와 디코더

## ➢ 단어에 대해서 임베딩 층 (Embedding layer)

✓ 예를 들어 I, am, a, student라는 단어들에 대한 임베딩 벡터 모습



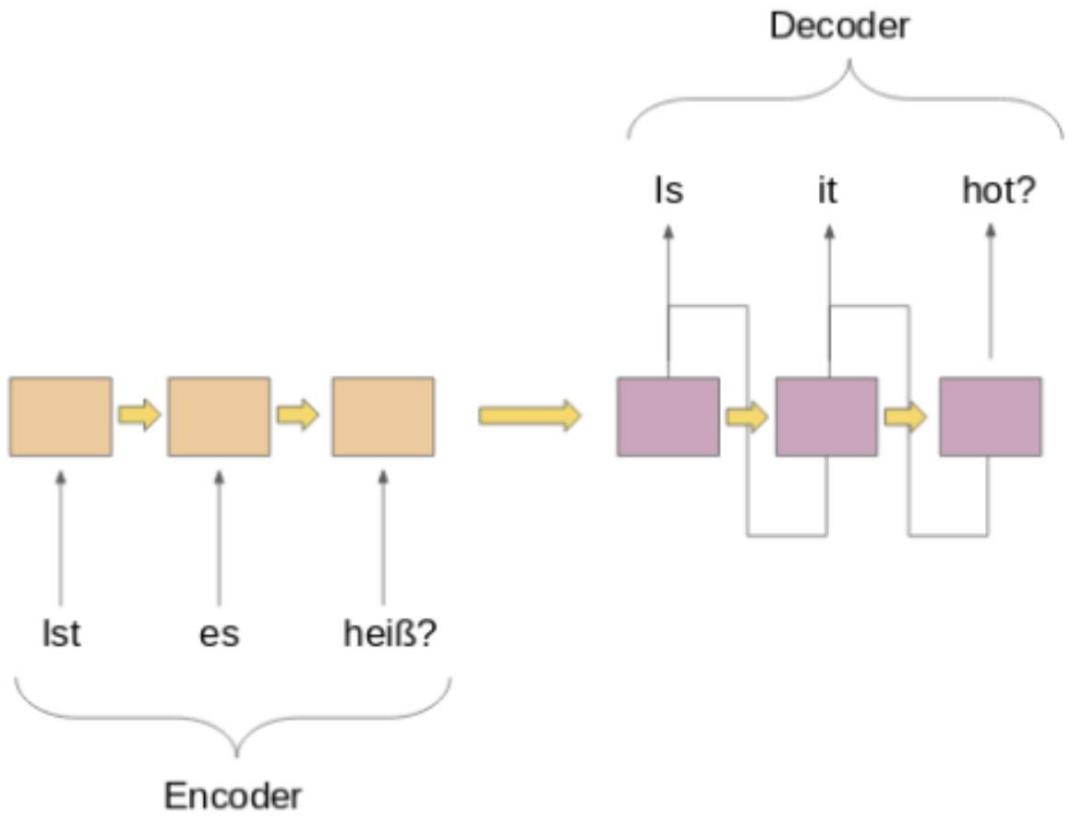
# 디코더


je
suis
étudiant
<eos>
softmax
softmax
softmax
softmax
Dense
Dense
Dense
Dense
디코더(Decoder)
LSTM
LSTM
LSTM
LSTM
embedding
embedding
embedding
embedding
<sos>
je
suis
étudiant

# 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
```

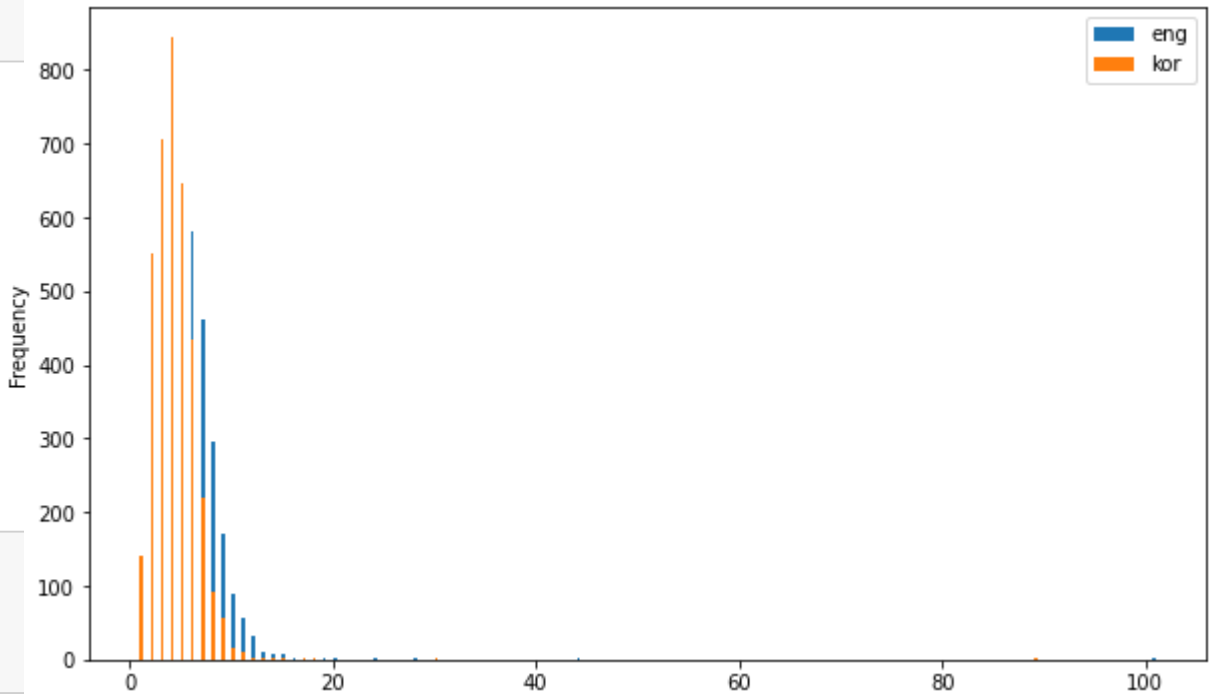
```
인 것 같아.',
        'CC-BY 2.0 (France) Attribution: tatoeba.org #953635 (CK) & #8384140 (Eunhee)'],
       ["If someone who doesn't know your background says that you sound like a native speaker, it means t
hey probably noticed something about your speaking that made them realize you weren't a native 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 깨닫
게 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 v
ersa; 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 존재
하지. 그런데 인간이 수백 명의 사람만 알고 지내는 사이가 될 기회를 갖는다고 생각해 보면, 또 그 수백 명 중
열여 명 쯤 이하만 잘 알 수 있고, 그리고 나서 그 열여 명 중에 한두 명만 친구가 될 수 있다면, 그리고 또 만일
우리가 이 세상에 살고 있는 수백만 명의 사람들만 기억하고 있다면, 딱 맞는 남자는 지구가 생겨난 이래로 딱
```

# (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.

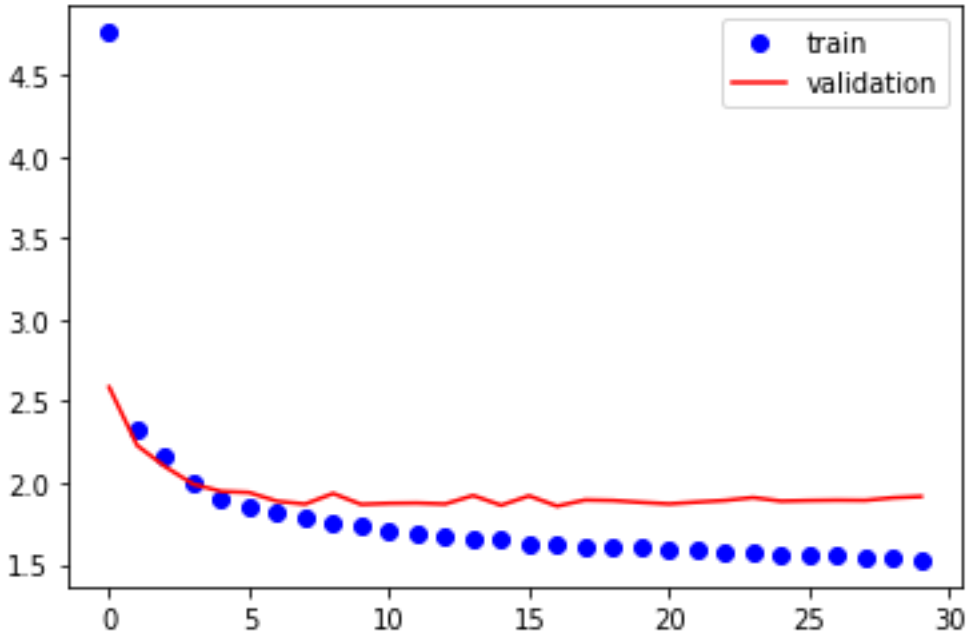
In [28]:
```
history_dict = history.history
history_dict.keys()
```

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

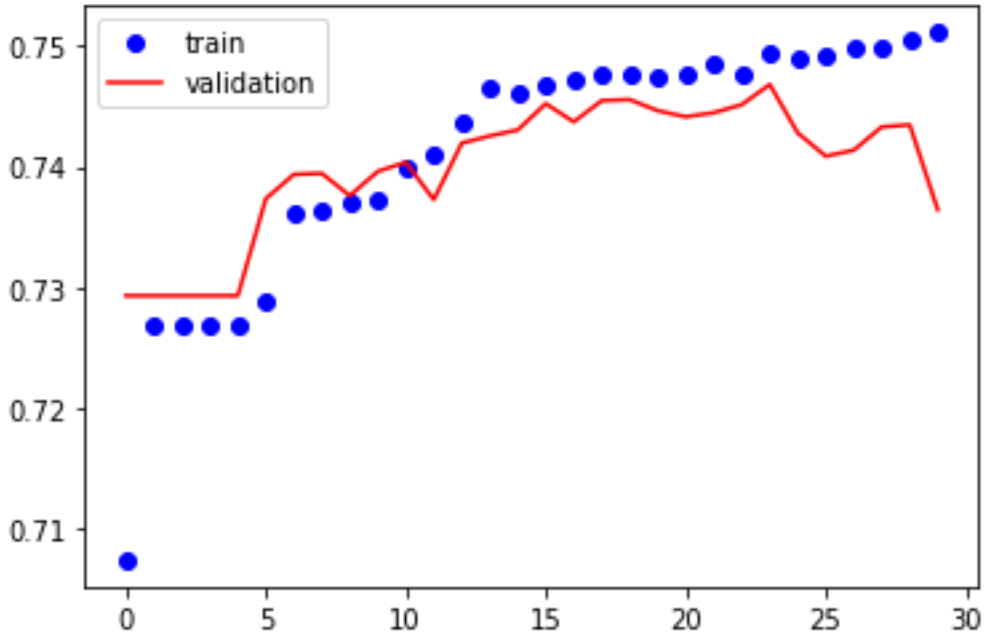
In [30]:
```
plt.plot(history.history['loss'],'bo')
plt.plot(history.history['val_loss'],'r')
plt.legend(['train','validation'])
plt.show()
```

# Loss and Accuracy

Loss




Accuracy



# Thank You!