Sequence-to-Sequence, seq2seq

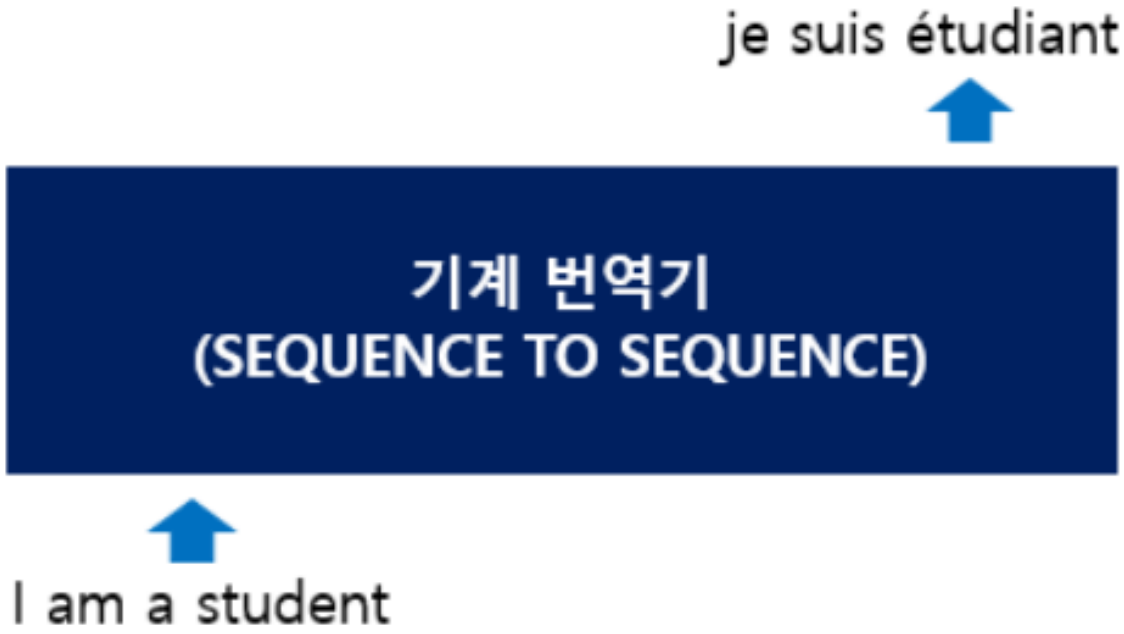
Simple RNN and LSTM

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

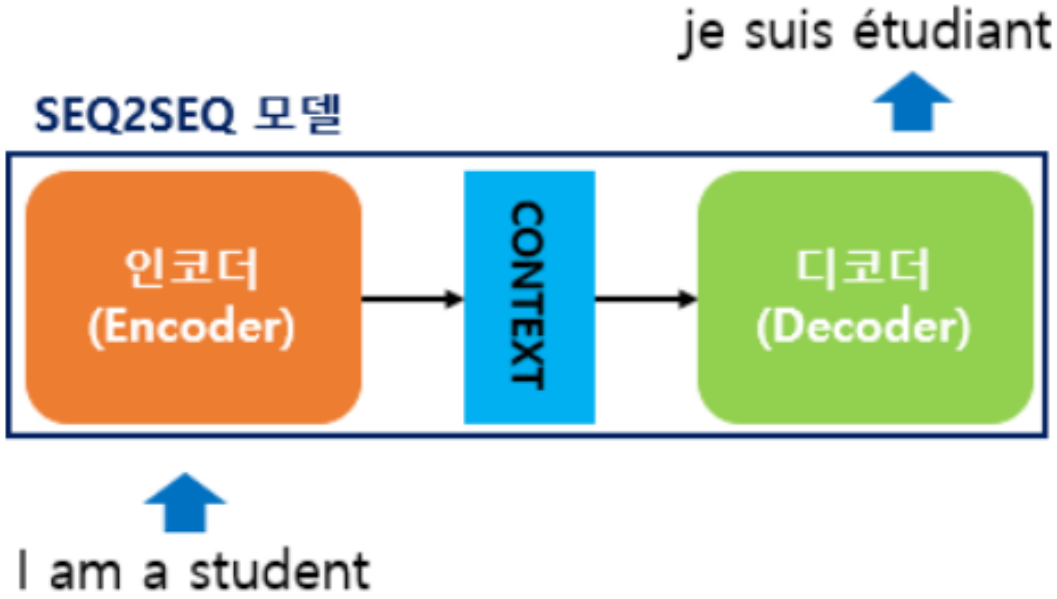


seq2seq는 인코더와 디코더 구조

➢ 인코더는 단어들을 순차적으로 입력받고, 이 모든 단어 정보들을 압축해서 벡터를 만든다.

✓ 입력 정보가 하나의 컨텍스트 벡터로 압축되면 인코더는 컨텍스트 벡터를 디코더로 전송합니다.

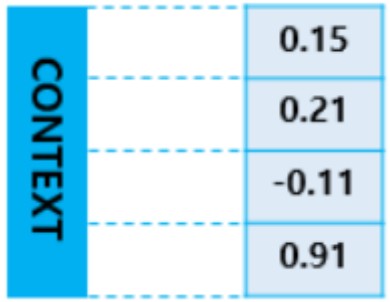
✓ 디코더는 컨텍스트 벡터를 받아서 번역된 단어를 한 개씩 순차적으로 출력합니다.



컨텍스트 벡터(context vector)

➢ 컨텍스트 벡터의 간단한 예제

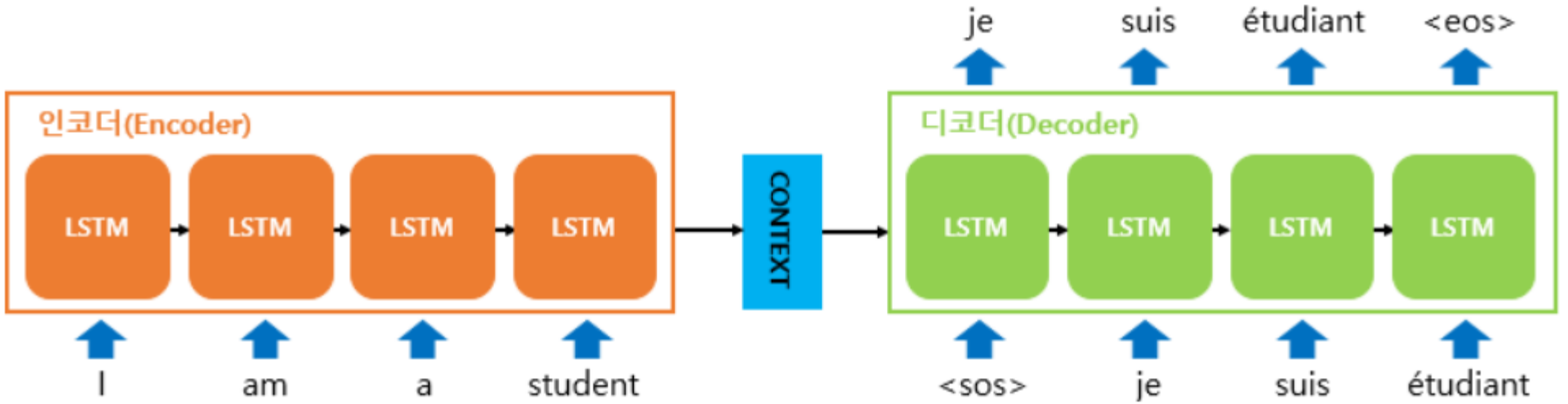
✓ 크기는 4로 표현하였다. 보통 수백 이상의 차원을 갖고있습니다.



인코더와 디코더는 RNN 아키텍처이다.

➢ 컨텍스트 벡터(context vector)

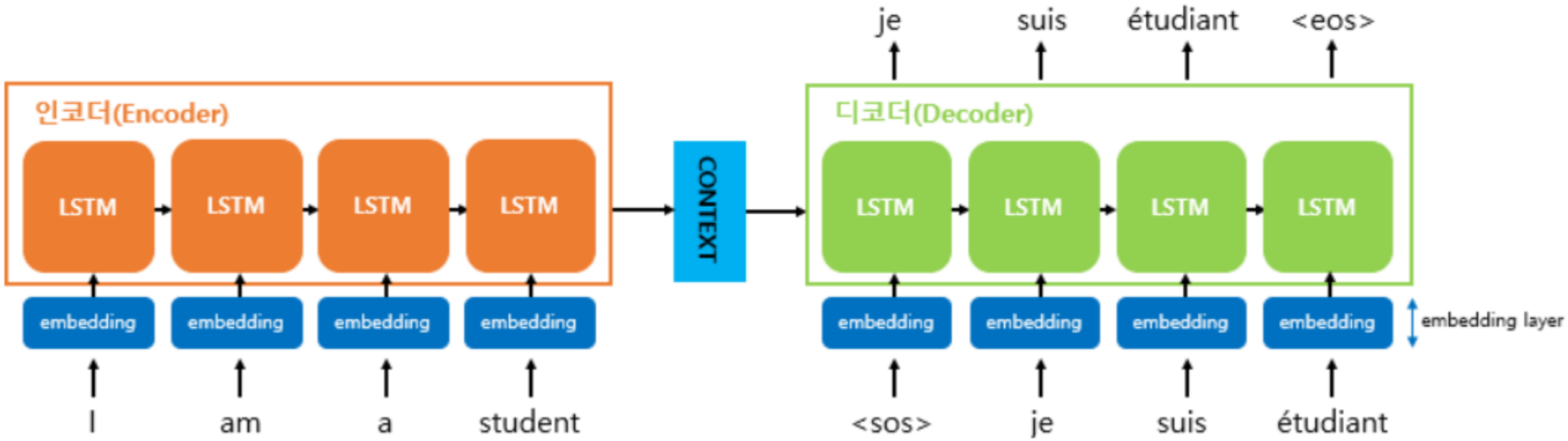
- ✓ 인코더는 문장의 모든 단어들을 순차적으로 입력받은 뒤에 모든 단어 정보들을 압축한다.
- ✓ 디코더는 컨텍스트 벡터를 받아서 번역된 단어를 한 개씩 순차적으로 출력합니다.
- ✓ 컨텍스트 벡터는 디코더 RNN 셀의 첫번째 은닉 상태로 사용됩니다.



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

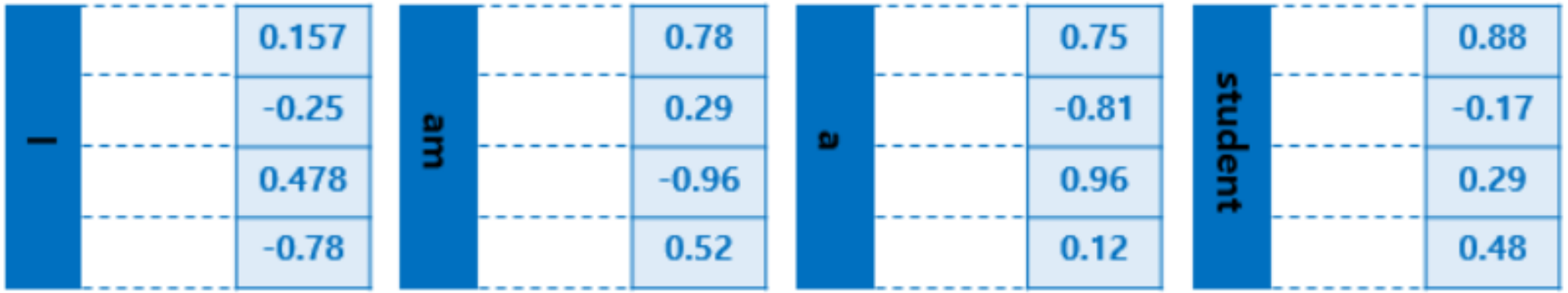
인코더와 디코더

- 단어들은 워드 임베딩 벡터이다.

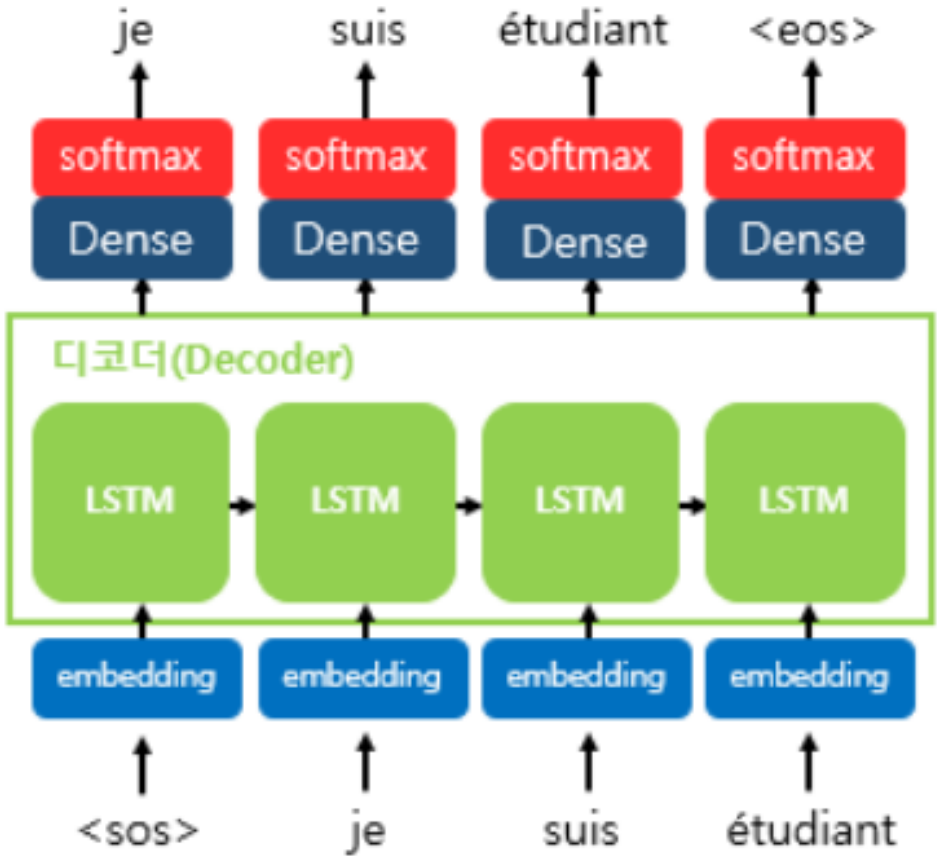


인코더와 디코더

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



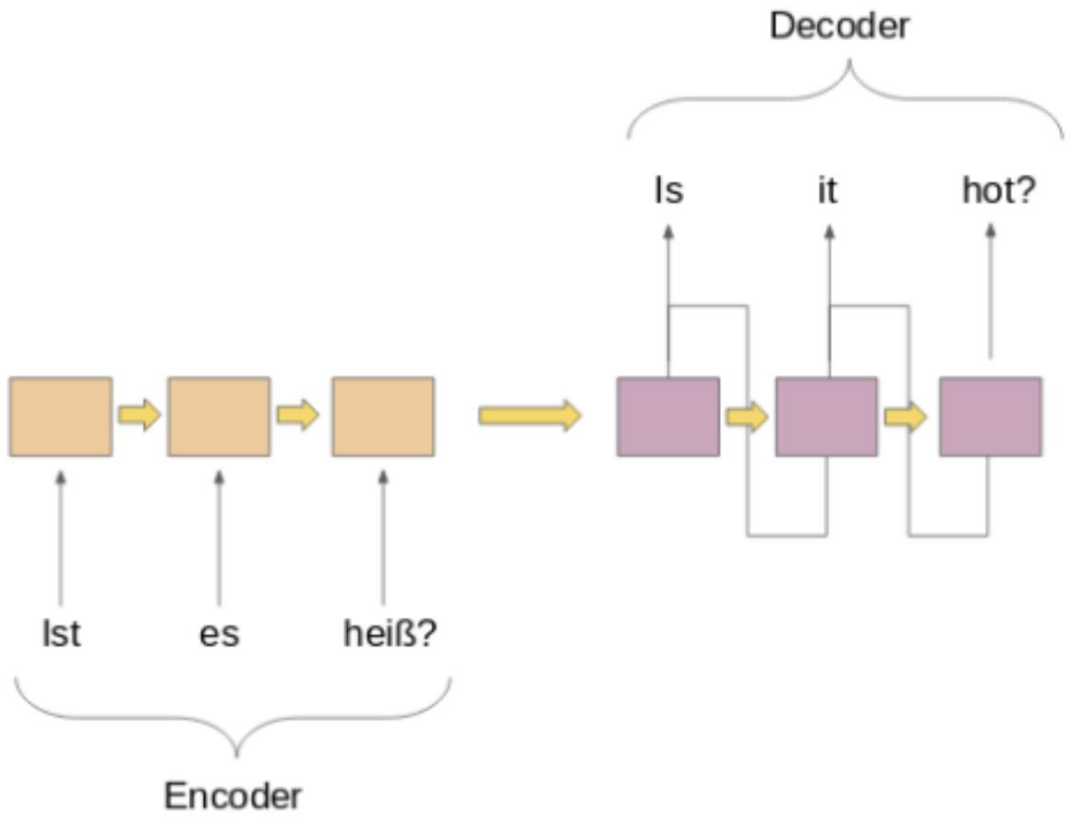
디코더


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

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
```

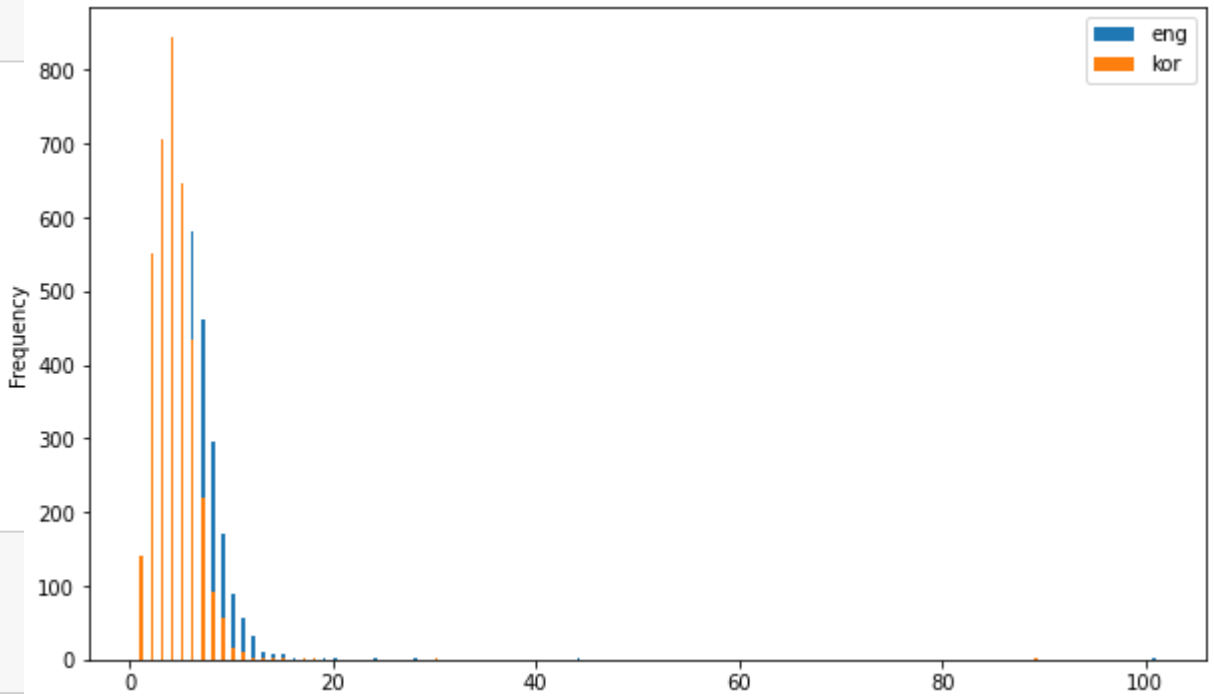
```
인 것 같아.',
        'CC-BY 2.0 (France) Attribution: tatoeba.org #953635 (CK) & #8384140 (Eunhee)'],
       ["If someone who doesn't know your background says that you sound like a native speaker, it means t
hey probably noticed something about your speaking that made them realize you weren't a native 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.

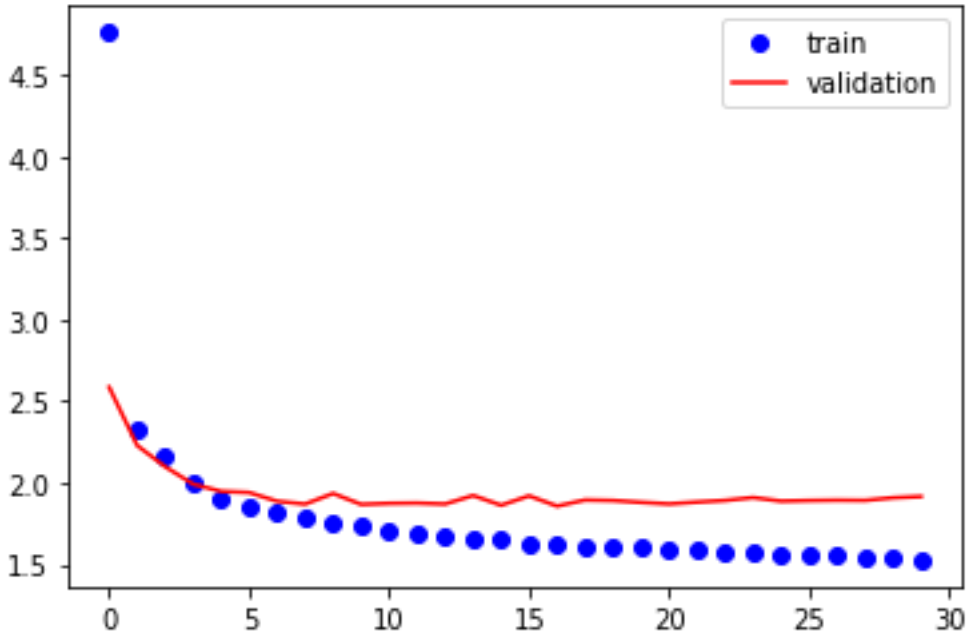
In [28]:
```
history_dict = history.history
history_dict.keys()
```

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