machine_translation

July 13, 2017

1 Artificial Intelligence Nanodegree

1.1 Machine Translation Project

In this notebook, sections that end with **'(IMPLEMENTATION)'** in the header indicate that the following blocks of code will require additional functionality which you must provide. Please be sure to read the instructions carefully!

1.2 Introduction

In this notebook, you will build a deep neural network that functions as part of an end-to-end machine translation pipeline. Your completed pipeline will accept English text as input and return the French translation.

- **Preprocess** - You'll convert text to sequence of integers.
- **Models** Create models which accepts a sequence of integers as input and returns a probability distribution over possible translations. After learning about the basic types of neural networks that are often used for machine translation, you will engage in your own investigations, to design your own model!
- **Prediction** Run the model on English text.

1.3 Dataset

We begin by investigating the dataset that will be used to train and evaluate your pipeline. The most common datasets used for machine translation are from WMT. However, that will take a long time to train a neural network on. We'll be using a dataset we created for this project that contains a small vocabulary. You'll be able to train your model in a reasonable time with this dataset. ### Load Data The data is located in `data/small_vocab_en` and `data/small_vocab_fr`. The `small_vocab_en` file contains English sentences with their French translations in the `small_vocab_fr` file. Load the English and French data from these files from running the cell below.

```
In [3]: import helper


        # Load English data
        english_sentences = helper.load_data('data/small_vocab_en')
        # Load French data
```

```
french_sentences = helper.load_data('data/small_vocab_fr')

print('Dataset Loaded')
```

```
Dataset Loaded
```

1.3.1 Files

Each line in `small_vocab_en` contains an English sentence with the respective translation in each line of `small_vocab_fr`. View the first two lines from each file.

```
In [2]: for sample_i in range(2):
            print('small_vocab_en Line {}:  {}'.format(sample_i + 1, english_sentences[sample_i
            print('small_vocab_fr Line {}:  {}'.format(sample_i + 1, french_sentences[sample_i]
```

```
small_vocab_en Line 1:  new jersey is sometimes quiet during autumn , and it is snowy in april
small_vocab_fr Line 1:  new jersey est parfois calme pendant l' automne , et il est neigeux en
small_vocab_en Line 2:  the united states is usually chilly during july , and it is usually fre
small_vocab_fr Line 2:  les états-unis est généralement froid en juillet , et il gèle habituell
```

From looking at the sentences, you can see they have been preprocessed already. The puncuations have been delimited using spaces. All the text have been converted to lowercase. This should save you some time, but the text requires more preprocessing. ### Vocabulary The complexity of the problem is determined by the complexity of the vocabulary. A more complex vocabulary is a more complex problem. Let's look at the complexity of the dataset we'll be working with.

```
In [4]: import collections


        english_words_counter = collections.Counter([word for sentence in english_sentences fo
        french_words_counter = collections.Counter([word for sentence in french_sentences for

        print('{} English words.'.format(len([word for sentence in english_sentences for word i
        print('{} unique English words.'.format(len(english_words_counter)))
        print('10 Most common words in the English dataset:')
        print('"' + '" "'.join(list(zip(*english_words_counter.most_common(10)))[0]) + '"')
        print()
        print('{} French words.'.format(len([word for sentence in french_sentences for word in
        print('{} unique French words.'.format(len(french_words_counter)))
        print('10 Most common words in the French dataset:')
        print('"' + '" "'.join(list(zip(*french_words_counter.most_common(10)))[0]) + '"')
```

```
1823250 English words.
227 unique English words.
10 Most common words in the English dataset:
"is" "," "." "in" "it" "during" "the" "but" "and" "sometimes"
```

```
1961295 French words.
355 unique French words.
10 Most common words in the French dataset:
"est" "." "," "en" "il" "les" "mais" "et" "la" "parfois"
```

For comparison, *Alice's Adventures in Wonderland* contains 2,766 unique words of a total of 15,500 words. ## Preprocess For this project, you won't use text data as input to your model. Instead, you'll convert the text into sequences of integers using the following preprocess methods: 1. Tokenize the words into ids 2. Add padding to make all the sequences the same length.

Time to start preprocessing the data... ### Tokenize (IMPLEMENTATION) For a neural network to predict on text data, it first has to be turned into data it can understand. Text data like "dog" is a sequence of ASCII character encodings. Since a neural network is a series of multiplication and addition operations, the input data needs to be number(s).

We can turn each character into a number or each word into a number. These are called character and word ids, respectively. Character ids are used for character level models that generate text predictions for each character. A word level model uses word ids that generate text predictions for each word. Word level models tend to learn better, since they are lower in complexity, so we'll use those.

Turn each sentence into a sequence of words ids using Keras's `Tokenizer` function. Use this function to tokenize `english_sentences` and `french_sentences` in the cell below.

Running the cell will run `tokenize` on sample data and show output for debugging.

In [10]:
```
import project_tests as tests
from keras.preprocessing.text import Tokenizer


def tokenize(x):
    """
    Tokenize x
    :param x: List of sentences/strings to be tokenized
    :return: Tuple of (tokenized x data, tokenizer used to tokenize x)
    """
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(x)
    text_sequences = tokenizer.texts_to_sequences(x)
    return text_sequences, tokenizer
tests.test_tokenize(tokenize)

# Tokenize Example output
text_sentences = [
    'The quick brown fox jumps over the lazy dog .',
    'By Jove , my quick study of lexicography won a prize .',
    'This is a short sentence .']
text_tokenized, text_tokenizer = tokenize(text_sentences)
print(text_tokenizer.word_index)
print()
for sample_i, (sent, token_sent) in enumerate(zip(text_sentences, text_tokenized)):
```

```
        print('Sequence {} in x'.format(sample_i + 1))
        print('  Input:  {}'.format(sent))
        print('  Output: {}'.format(token_sent))
```

```
Using TensorFlow backend.


{'fox': 5, 'brown': 4, 'jumps': 6, 'lexicography': 15, 'lazy': 8, 'by': 10, 'a': 3, 'is': 19,

Sequence 1 in x
  Input:  The quick brown fox jumps over the lazy dog .
  Output: [1, 2, 4, 5, 6, 7, 1, 8, 9]
Sequence 2 in x
  Input:  By Jove , my quick study of lexicography won a prize .
  Output: [10, 11, 12, 2, 13, 14, 15, 16, 3, 17]
Sequence 3 in x
  Input:  This is a short sentence .
  Output: [18, 19, 3, 20, 21]
```

1.3.2 Padding (IMPLEMENTATION)

When batching the sequence of word ids together, each sequence needs to be the same length. Since sentences are dynamic in length, we can add padding to the end of the sequences to make them the same length.

Make sure all the English sequences have the same length and all the French sequences have the same length by adding padding to the **end** of each sequence using Keras's `pad_sequences` function.

```
In [11]: import numpy as np
         from keras.preprocessing.sequence import pad_sequences


         def pad(x, length=None):
             """
             Pad x
             :param x: List of sequences.
             :param length: Length to pad the sequence to.  If None, use length of longest seq
             :return: Padded numpy array of sequences
             """
             # TODO: Implement
             padded_sequences = pad_sequences(x,maxlen=length,padding='post')
             return padded_sequences
         tests.test_pad(pad)

         # Pad Tokenized output
         test_pad = pad(text_tokenized)
         for sample_i, (token_sent, pad_sent) in enumerate(zip(text_tokenized, test_pad)):
             print('Sequence {} in x'.format(sample_i + 1))
```

```
            print('  Input:  {}'.format(np.array(token_sent)))
            print('  Output: {}'.format(pad_sent))
```

```
Sequence 1 in x
  Input:  [1 2 4 5 6 7 1 8 9]
  Output: [1 2 4 5 6 7 1 8 9 0]
Sequence 2 in x
  Input:  [10 11 12  2 13 14 15 16  3 17]
  Output: [10 11 12  2 13 14 15 16  3 17]
Sequence 3 in x
  Input:  [18 19  3 20 21]
  Output: [18 19  3 20 21  0  0  0  0  0]
```

1.3.3 Preprocess Pipeline

Your focus for this project is to build neural network architecture, so we won't ask you to create a preprocess pipeline. Instead, we've provided you with the implementation of the `preprocess` function.

```
In [12]: def preprocess(x, y):
             """
             Preprocess x and y
             :param x: Feature List of sentences
             :param y: Label List of sentences
             :return: Tuple of (Preprocessed x, Preprocessed y, x tokenizer, y tokenizer)
             """
             preprocess_x, x_tk = tokenize(x)
             preprocess_y, y_tk = tokenize(y)

             preprocess_x = pad(preprocess_x)
             preprocess_y = pad(preprocess_y)

             # Keras's sparse_categorical_crossentropy function requires the labels to be in 3
             preprocess_y = preprocess_y.reshape(*preprocess_y.shape, 1)

             return preprocess_x, preprocess_y, x_tk, y_tk

         preproc_english_sentences, preproc_french_sentences, english_tokenizer, french_tokeni
             preprocess(english_sentences, french_sentences)

         print('Data Preprocessed')
```

```
Data Preprocessed
```

1.4 Models

In this section, you will experiment with various neural network architectures. You will begin by training four relatively simple architectures. - Model 1 is a simple RNN - Model 2 is a RNN with

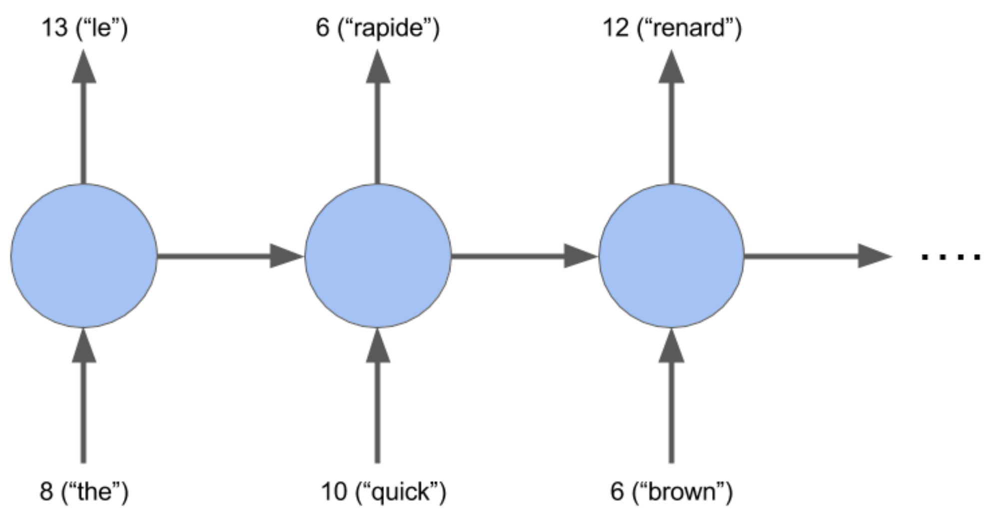
Embedding - Model 3 is a Bidirectional RNN - Model 4 is an optional Encoder-Decoder RNN

After experimenting with the four simple architectures, you will construct a deeper architecture that is designed to outperform all four models. ### Ids Back to Text The neural network will be translating the input to words ids, which isn't the final form we want. We want the French translation. The function `logits_to_text` will bridge the gab between the logits from the neural network to the French translation. You'll be using this function to better understand the output of the neural network.

```
In [13]: def logits_to_text(logits, tokenizer):
             """
             Turn logits from a neural network into text using the tokenizer
             :param logits: Logits from a neural network
             :param tokenizer: Keras Tokenizer fit on the labels
             :return: String that represents the text of the logits
             """
             index_to_words = {id: word for word, id in tokenizer.word_index.items()}
             index_to_words[0] = '<PAD>'

             return ' '.join([index_to_words[prediction] for prediction in np.argmax(logits, 1

         print('`logits_to_text` function loaded.')
```

```
`logits_to_text` function loaded.
```

1.4.1 Model 1: RNN (IMPLEMENTATION)



A basic RNN model is a good baseline for sequence data. In this model, you'll build a RNN that translates English to French.

```
In [21]: from keras.layers import GRU, Input, Dense, TimeDistributed
         from keras.models import Model, Sequential
         from keras.layers import Activation, Dropout
         from keras.optimizers import Adam
```

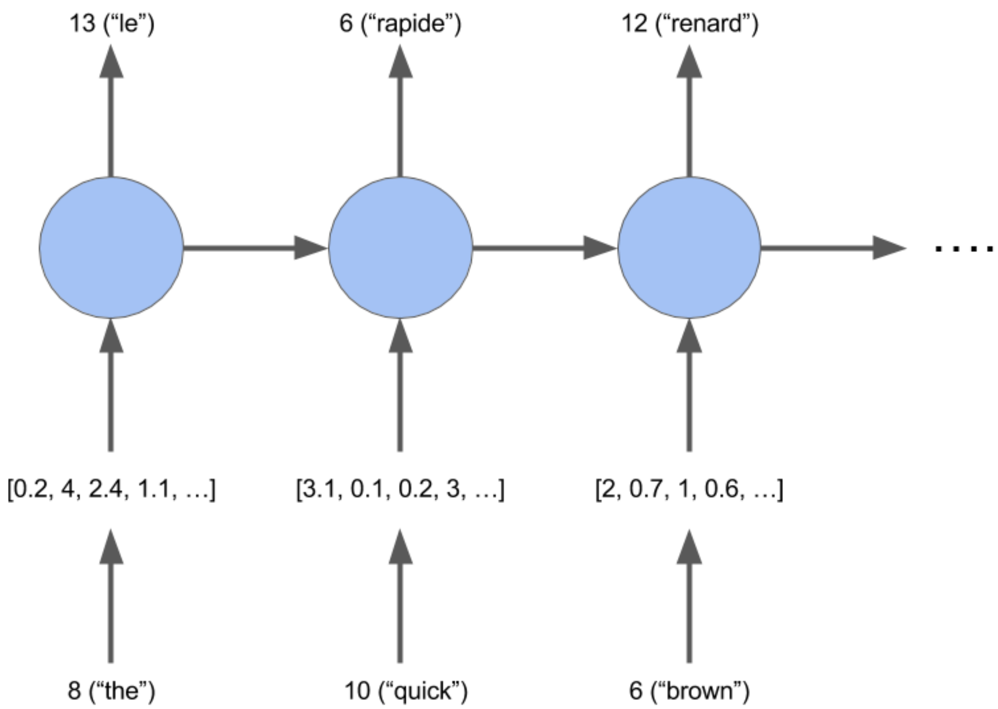
```python
from keras.losses import sparse_categorical_crossentropy


def simple_model(input_shape, output_sequence_length, english_vocab_size, french_vocab
    """
    Build and train a basic RNN on x and y
    :param input_shape: Tuple of input shape
    :param output_sequence_length: Length of output sequence
    :param english_vocab_size: Number of unique English words in the dataset
    :param french_vocab_size: Number of unique French words in the dataset
    :return: Keras model built, but not trained
    """
    learning_rate = 0.001
    units = output_sequence_length
    model = Sequential()
    model.add(
        GRU(
            # Model-specific parameters
            units, # Output space dimensionality
            activation='tanh',
            # Recurrent Abstract class parameters
            return_sequences=True, # Set to True since feeds other recurrent layers
            input_shape=input_shape[1:])) # Add input shape only to first layer

    # Dense Layers
    model.add(Dropout(0.2))
    model.add(TimeDistributed(Dense(2 * french_vocab_size, activation='relu')))


    model.add(TimeDistributed(Dense(french_vocab_size, activation='softmax')))
    model.compile(loss=sparse_categorical_crossentropy,
      # https://keras.io/optimizers/
      optimizer=Adam(learning_rate), # Learning Rate
      metrics=['sparse_categorical_accuracy']) # or 'accuracy'
    model.summary()
    return model

tests.test_simple_model(simple_model)


# Reshaping the input to work with a basic RNN
tmp_x = pad(preproc_english_sentences, preproc_french_sentences.shape[1])
tmp_x = tmp_x.reshape((-1, preproc_french_sentences.shape[-2], 1))

# Train the neural network
simple_rnn_model = simple_model(
    tmp_x.shape,
    preproc_french_sentences.shape[1],
```

```
            len(english_tokenizer.word_index),
            len(french_tokenizer.word_index))

        print ('training')
        simple_rnn_model.fit(tmp_x, preproc_french_sentences, batch_size=1024, epochs=10, vali

        # Print prediction(s)
        print(logits_to_text(simple_rnn_model.predict(tmp_x[:1])[0], french_tokenizer))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_12 (SimpleRNN)    (None, 21, 21)            483
_________________________________________________________________
dropout_14 (Dropout)         (None, 21, 21)            0
_________________________________________________________________
time_distributed_27 (TimeDis (None, 21, 688)           15136
_________________________________________________________________
time_distributed_28 (TimeDis (None, 21, 344)           237016
=================================================================
Total params: 252,635
Trainable params: 252,635
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_13 (SimpleRNN)    (None, 21, 21)            483
_________________________________________________________________
dropout_15 (Dropout)         (None, 21, 21)            0
_________________________________________________________________
time_distributed_29 (TimeDis (None, 21, 688)           15136
_________________________________________________________________
time_distributed_30 (TimeDis (None, 21, 344)           237016
=================================================================
Total params: 252,635
Trainable params: 252,635
Non-trainable params: 0
_________________________________________________________________
training
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 11s - loss: 3.1329 - sparse_categorical_accura
Epoch 2/10
110288/110288 [==============================] - 11s - loss: 2.2347 - sparse_categorical_accura
Epoch 3/10
110288/110288 [==============================] - 11s - loss: 2.0866 - sparse_categorical_accura
Epoch 4/10
```

```
110288/110288 [==============================] - 11s - loss: 1.9339 - sparse_categorical_accura
Epoch 5/10
110288/110288 [==============================] - 11s - loss: 1.7901 - sparse_categorical_accura
Epoch 6/10
110288/110288 [==============================] - 11s - loss: 1.6938 - sparse_categorical_accura
Epoch 7/10
110288/110288 [==============================] - 11s - loss: 1.6372 - sparse_categorical_accura
Epoch 8/10
110288/110288 [==============================] - 11s - loss: 1.5959 - sparse_categorical_accura
Epoch 9/10
110288/110288 [==============================] - 11s - loss: 1.5628 - sparse_categorical_accura
Epoch 10/10
110288/110288 [==============================] - 11s - loss: 1.5345 - sparse_categorical_accura
new jersey est généralement parfois en en et il est est en en <PAD> <PAD> <PAD> <PAD> <PAD> <PA
```

1.4.2 Model 2: Embedding (IMPLEMENTATION)



You’ve turned the words into ids, but there’s a better representation of a word. This is called word embeddings. An embedding is a vector representation of the word that is close to similar words in n-dimensional space, where the n represents the size of the embedding vectors.

In this model, you’ll create a RNN model using embedding.

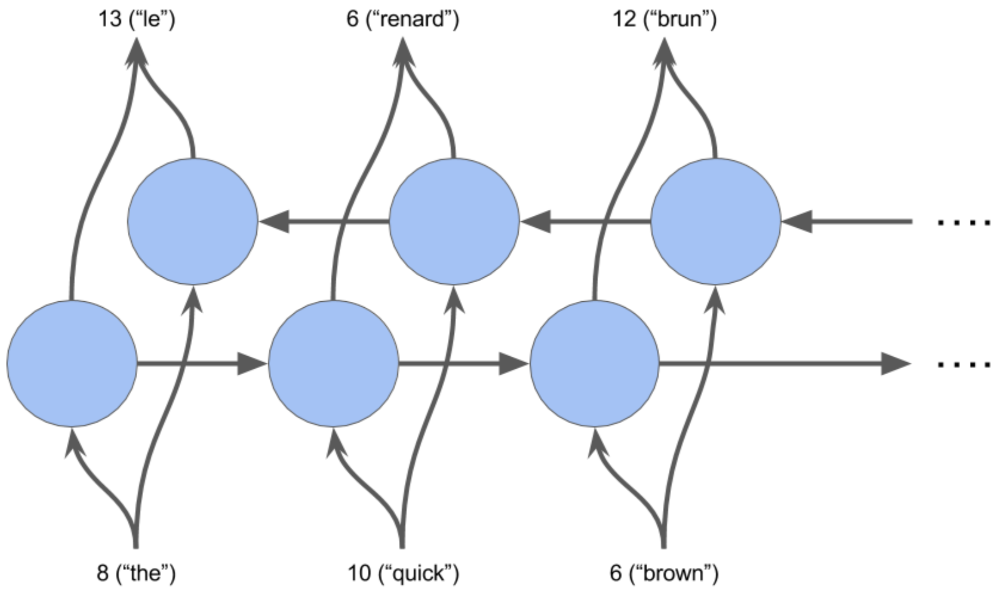
```
In [42]: from keras.layers.embeddings import Embedding


         def embed_model(input_shape, output_sequence_length, english_vocab_size, french_vocab
             """
```

```
    Build and train a RNN model using word embedding on x and y
    :param input_shape: Tuple of input shape
    :param output_sequence_length: Length of output sequence
    :param english_vocab_size: Number of unique English words in the dataset
    :param french_vocab_size: Number of unique French words in the dataset
    :return: Keras model built, but not trained
    """
    learning_rate=0.001
    model = Sequential()
    model.add(Embedding(english_vocab_size,french_vocab_size,input_length = input_sha

    model.add(GRU(output_sequence_length, return_sequences=True, input_shape= input_s
    model.add(TimeDistributed(Dense(french_vocab_size, activation='softmax')))


    model.compile(
        loss=sparse_categorical_crossentropy,
        optimizer=Adam(learning_rate),
        metrics=['sparse_categorical_accuracy'] # or 'accuracy'
    )
    return model

tests.test_embed_model(embed_model)


# TODO: Reshape the input
tmp_x = pad(preproc_english_sentences, preproc_french_sentences.shape[1])
# TODO: Train the neural network
embed_model = embed_model(
    tmp_x.shape,
    preproc_french_sentences.shape[1],
    len(english_tokenizer.word_index),
    len(french_tokenizer.word_index))

print ('training')
embed_model.fit(tmp_x, preproc_french_sentences, batch_size=1024, epochs=10, validati
# TODO: Print prediction(s)
print(logits_to_text(embed_model.predict(tmp_x[:1])[0], french_tokenizer))
```

```
training
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 11s - loss: 4.2948 - sparse_categorical_accur
Epoch 2/10
110288/110288 [==============================] - 10s - loss: 2.6301 - sparse_categorical_accur
Epoch 3/10
110288/110288 [==============================] - 10s - loss: 1.9154 - sparse_categorical_accur
Epoch 4/10
```

```
110288/110288 [==============================] - 10s - loss: 1.5270 - sparse_categorical_accura
Epoch 5/10
110288/110288 [==============================] - 10s - loss: 1.3325 - sparse_categorical_accura
Epoch 6/10
110288/110288 [==============================] - 10s - loss: 1.2085 - sparse_categorical_accura
Epoch 7/10
110288/110288 [==============================] - 10s - loss: 1.1120 - sparse_categorical_accura
Epoch 8/10
110288/110288 [==============================] - 10s - loss: 1.0337 - sparse_categorical_accura
Epoch 9/10
110288/110288 [==============================] - 10s - loss: 0.9690 - sparse_categorical_accura
Epoch 10/10
110288/110288 [==============================] - 10s - loss: 0.9148 - sparse_categorical_accura
new jersey est parfois calme en l' et et il est en en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <P
```

1.4.3 Model 3: Bidirectional RNNs (IMPLEMENTATION)



One restriction of a RNN is that it can’t see the future input, only the past. This is where bidirectional recurrent neural networks come in. They are able to see the future data.

```
In [44]: from keras.layers import Bidirectional


         def bd_model(input_shape, output_sequence_length, english_vocab_size, french_vocab_si
             """
             Build and train a bidirectional RNN model on x and y
             :param input_shape: Tuple of input shape
             :param output_sequence_length: Length of output sequence
             :param english_vocab_size: Number of unique English words in the dataset
             :param french_vocab_size: Number of unique French words in the dataset
             :return: Keras model built, but not trained
```

```python
    """
    # TODO: Implement
    learning_rate = 0.001

    model = Sequential()
    model.add(Bidirectional(GRU(output_sequence_length, return_sequences=True), input_
    model.add(Dense(french_vocab_size, activation='relu'))
    model.add(Dense(french_vocab_size, activation='softmax'))

    model.compile(loss=sparse_categorical_crossentropy,
                  optimizer=Adam(learning_rate),
                  metrics=['accuracy'])

    return model
tests.test_bd_model(bd_model)


# TODO: Train and Print prediction(s)
tmp_x = pad(preproc_english_sentences, preproc_french_sentences.shape[1])
tmp_x = tmp_x.reshape((-1, preproc_french_sentences.shape[-2], 1))

bd_rnn_model = bd_model(
    tmp_x.shape,
    preproc_french_sentences.shape[1],
    len(english_tokenizer.word_index),
    len(french_tokenizer.word_index))

bd_rnn_model.fit(tmp_x, preproc_french_sentences, batch_size=1024, epochs=10, validati

print(logits_to_text(bd_rnn_model.predict(tmp_x[:1])[0], french_tokenizer))
```

```
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 17s - loss: 3.3048 - acc: 0.4655 - val_loss: 
Epoch 2/10
110288/110288 [==============================] - 15s - loss: 1.9657 - acc: 0.5461 - val_loss: 
Epoch 3/10
110288/110288 [==============================] - 15s - loss: 1.6411 - acc: 0.5885 - val_loss: 
Epoch 4/10
110288/110288 [==============================] - 15s - loss: 1.5077 - acc: 0.6048 - val_loss: 
Epoch 5/10
110288/110288 [==============================] - 15s - loss: 1.4320 - acc: 0.6205 - val_loss: 
Epoch 6/10
110288/110288 [==============================] - 15s - loss: 1.3805 - acc: 0.6291 - val_loss: 
Epoch 7/10
110288/110288 [==============================] - 15s - loss: 1.3429 - acc: 0.6357 - val_loss: 
Epoch 8/10
110288/110288 [==============================] - 15s - loss: 1.3125 - acc: 0.6394 - val_loss: 
```

```
Epoch 9/10
110288/110288 [==============================] - 15s - loss: 1.2870 - acc: 0.6428 - val_loss: 
Epoch 10/10
110288/110288 [==============================] - 15s - loss: 1.2640 - acc: 0.6467 - val_loss: 
new jersey est parfois parfois en en et il est est en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <
```

1.4.4 Model 4: Encoder-Decoder (OPTIONAL)

Time to look at encoder-decoder models. This model is made up of an encoder and decoder. The encoder creates a matrix representation of the sentence. The decoder takes this matrix as input and predicts the translation as output.

Create an encoder-decoder model in the cell below.

```
In [48]: from keras.layers import RepeatVector


         def encdec_model(input_shape, output_sequence_length, english_vocab_size, french_vocal
             """
             Build and train an encoder-decoder model on x and y
             :param input_shape: Tuple of input shape
             :param output_sequence_length: Length of output sequence
             :param english_vocab_size: Number of unique English words in the dataset
             :param french_vocab_size: Number of unique French words in the dataset
             :return: Keras model built, but not trained
             """
             # OPTIONAL: Implement
             learning_rate = 0.01
             model = Sequential()

             model.add(GRU(output_sequence_length, return_sequences=False, input_shape= input_
             model.add(RepeatVector(output_sequence_length))
             model.add(GRU(output_sequence_length, return_sequences=True))

             model.add(Dense(french_vocab_size, activation='relu'))
             model.add(Dense(french_vocab_size, activation='softmax'))

             model.compile(loss=sparse_categorical_crossentropy,
                           optimizer=Adam(learning_rate),
                           metrics=['accuracy'])
             return model
         tests.test_encdec_model(encdec_model)


         # OPTIONAL: Train and Print prediction(s)
         tmp_x = pad(preproc_english_sentences, preproc_french_sentences.shape[1])
         tmp_x = tmp_x.reshape((-1, preproc_french_sentences.shape[-2], 1))
```

```
encdec_rnn_model = encdec_model(
    tmp_x.shape,
    preproc_french_sentences.shape[1],
    len(english_tokenizer.word_index),
    len(french_tokenizer.word_index))

encdec_rnn_model.fit(tmp_x, preproc_french_sentences, batch_size=1024, epochs=10, vali

print(logits_to_text(encdec_rnn_model.predict(tmp_x[:1])[0], french_tokenizer))
```

```
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 18s - loss: 2.8848 - acc: 0.4420 - val_loss: 
Epoch 2/10
110288/110288 [==============================] - 16s - loss: 2.0328 - acc: 0.5272 - val_loss: 
Epoch 3/10
110288/110288 [==============================] - 16s - loss: 1.8557 - acc: 0.5476 - val_loss: 
Epoch 4/10
110288/110288 [==============================] - 16s - loss: 1.7782 - acc: 0.5558 - val_loss: 
Epoch 5/10
110288/110288 [==============================] - 16s - loss: 1.5533 - acc: 0.5855 - val_loss: 
Epoch 6/10
110288/110288 [==============================] - 16s - loss: 1.4769 - acc: 0.5965 - val_loss: 
Epoch 7/10
110288/110288 [==============================] - 16s - loss: 1.4093 - acc: 0.6093 - val_loss: 
Epoch 8/10
110288/110288 [==============================] - 16s - loss: 1.3944 - acc: 0.6115 - val_loss: 
Epoch 9/10
110288/110288 [==============================] - 16s - loss: 1.3532 - acc: 0.6204 - val_loss: 
Epoch 10/10
110288/110288 [==============================] - 16s - loss: 1.3370 - acc: 0.6235 - val_loss: 
new jersey est parfois froid en mois et il est est en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <
```

1.4.5 Model 5: Custom (IMPLEMENTATION)

Use everything you learned from the previous models to create a model that incorporates embedding and a bidirectional rnn into one model.

In [65]:
```
def model_final(input_shape, output_sequence_length, english_vocab_size, french_vocab
    """
    Build and train a model that incorporates embedding, encoder-decoder, and bidirec
    :param input_shape: Tuple of input shape
    :param output_sequence_length: Length of output sequence
    :param english_vocab_size: Number of unique English words in the dataset
    :param french_vocab_size: Number of unique French words in the dataset
    :return: Keras model built, but not trained
    """
```

```python
    # TODO: Implement
    learning_rate = 0.01

    model = Sequential()

    model.add(Embedding(english_vocab_size,french_vocab_size,input_length = input_sha

    model.add(Bidirectional(GRU(output_sequence_length, return_sequences=False), inpu
    model.add(RepeatVector(output_sequence_length))
    model.add(Bidirectional(GRU(output_sequence_length, return_sequences=True)))
    model.add(Dropout(0.2))

    model.add(TimeDistributed(Dense(french_vocab_size, activation='relu')))
    model.add(TimeDistributed(Dense(french_vocab_size, activation='softmax')))

    model.compile(loss=sparse_categorical_crossentropy,
                  optimizer=Adam(learning_rate),
                  metrics=['accuracy'])

    model.summary()

    return model
tests.test_model_final(model_final)


print('Final Model Loaded')
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_47 (Embedding)     (None, 15, 344)           68456
_________________________________________________________________
bidirectional_33 (Bidirectio (None, 42)                46116
_________________________________________________________________
repeat_vector_21 (RepeatVect (None, 21, 42)            0
_________________________________________________________________
bidirectional_34 (Bidirectio (None, 21, 42)            8064
_________________________________________________________________
dropout_24 (Dropout)         (None, 21, 42)            0
_________________________________________________________________
time_distributed_70 (TimeDis (None, 21, 344)           14792
_________________________________________________________________
time_distributed_71 (TimeDis (None, 21, 344)           118680
=================================================================
Total params: 256,108
Trainable params: 256,108
Non-trainable params: 0
_________________________________________________________________
```

```
Final Model Loaded
```

1.5 Prediction (IMPLEMENTATION)

```
In [66]: import numpy as np
         from keras.preprocessing.sequence import pad_sequences


         def final_predictions(x, y, x_tk, y_tk):
             """
             Gets predictions using the final model
             :param x: Preprocessed English data
             :param y: Preprocessed French data
             :param x_tk: English tokenizer
             :param y_tk: French tokenizer
             """
             # TODO: Train neural network using model_final
             model = model_final(x.shape, y.shape[1], len(x_tk.word_index), len(y_tk.word_inde
             model.fit(x, y, batch_size=1024, epochs=10, validation_split=0.2)

             ## DON'T EDIT ANYTHING BELOW THIS LINE
             y_id_to_word = {value: key for key, value in y_tk.word_index.items()}
             y_id_to_word[0] = '<PAD>'

             sentence = 'he saw a old yellow truck'
             sentence = [x_tk.word_index[word] for word in sentence.split()]
             sentence = pad_sequences([sentence], maxlen=x.shape[-1], padding='post')
             sentences = np.array([sentence[0], x[0]])
             predictions = model.predict(sentences, len(sentences))

             print('Sample 1:')
             print(' '.join([y_id_to_word[np.argmax(x)] for x in predictions[0]]))
             print('Il a vu un vieux camion jaune')
             print('Sample 2:')
             print(' '.join([y_id_to_word[np.argmax(x)] for x in predictions[1]]))
             print(' '.join([y_id_to_word[np.argmax(x)] for x in y[0]]))


         final_predictions(preproc_english_sentences, preproc_french_sentences, english_tokeni
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_48 (Embedding)     (None, 15, 344)           68456
_________________________________________________________________
bidirectional_35 (Bidirectio (None, 42)                46116
_________________________________________________________________
```

```
repeat_vector_22 (RepeatVect (None, 21, 42)           0
_________________________________________________________________
bidirectional_36 (Bidirectio (None, 21, 42)           8064
_________________________________________________________________
dropout_25 (Dropout)         (None, 21, 42)           0
_________________________________________________________________
time_distributed_72 (TimeDis (None, 21, 344)          14792
_________________________________________________________________
time_distributed_73 (TimeDis (None, 21, 344)          118680
=================================================================
Total params: 256,108
Trainable params: 256,108
Non-trainable params: 0
_________________________________________________________________
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 31s - loss: 2.2947 - acc: 0.4885 - val_loss:
Epoch 2/10
110288/110288 [==============================] - 21s - loss: 1.3597 - acc: 0.6239 - val_loss:
Epoch 3/10
110288/110288 [==============================] - 21s - loss: 1.0390 - acc: 0.6936 - val_loss:
Epoch 4/10
110288/110288 [==============================] - 21s - loss: 0.8824 - acc: 0.7311 - val_loss:
Epoch 5/10
110288/110288 [==============================] - 21s - loss: 0.7685 - acc: 0.7606 - val_loss:
Epoch 6/10
110288/110288 [==============================] - 21s - loss: 0.6788 - acc: 0.7847 - val_loss:
Epoch 7/10
110288/110288 [==============================] - 21s - loss: 0.6213 - acc: 0.8007 - val_loss:
Epoch 8/10
110288/110288 [==============================] - 21s - loss: 0.5605 - acc: 0.8201 - val_loss:
Epoch 9/10
110288/110288 [==============================] - 21s - loss: 0.5240 - acc: 0.8315 - val_loss:
Epoch 10/10
110288/110288 [==============================] - 21s - loss: 0.4833 - acc: 0.8447 - val_loss:
Sample 1:
il a vu un vieux camion jaune <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD
Il a vu un vieux camion jaune
Sample 2:
new jersey est parfois calme pendant l' automne automne il est neigeux en avril avril <PAD> <P
<PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD
```

1.6 Submission

When you are ready to submit your project, do the following steps: 1. Ensure you pass all points on the rubric. 2. Submit the following in a zip file. - `helper.py` - `machine_translation.ipynb` - `machine_translation.html` - You can export the notebook by navigating to **File -> Download as**

-> HTML (.html).