machine_translation

May 6, 2018

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.

```
In [2]: %load_ext autoreload
        %aimport helper, tests
        %autoreload 1
```

```
In [3]: import collections

        import helper
        import numpy as np
        import project_tests as tests

        from keras.preprocessing.text import Tokenizer
        from keras.preprocessing.sequence import pad_sequences
        from keras.models import Model, Sequential
        from keras.layers import GRU, Input, Dense, TimeDistributed, Activation, RepeatVector, B
        from keras.layers.embeddings import Embedding
        from keras.optimizers import Adam
        from keras.losses import sparse_categorical_crossentropy
```

```
Using TensorFlow backend.
```

1.2.1 Verify access to the GPU

The following test applies only if you expect to be using a GPU, e.g., while running in a Udacity Workspace or using an AWS instance with GPU support. Run the next cell, and verify that the device_type is "GPU". - If the device is not GPU & you are running from a Udacity Workspace, then save your workspace with the icon at the top, then click "enable" at the bottom of the workspace. - If the device is not GPU & you are running from an AWS instance, then refer to the cloud computing instructions in the classroom to verify your setup steps.

```
In [4]: from tensorflow.python.client import device_lib
        print(device_lib.list_local_devices())
```

```
[name: "/cpu:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 12205182139204782502
, name: "/gpu:0"
device_type: "GPU"
memory_limit: 357171200
locality {
  bus_id: 1
}
incarnation: 10140026463398979671
physical_device_desc: "device: 0, name: Tesla K80, pci bus id: 0000:00:04.0"
]
```

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 [5]: # 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 [6]: 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 a
small_vocab_en Line 2:  the united states is usually chilly during july , and it is usually free
small_vocab_fr Line 2:  les états-unis est généralement froid en juillet , et il gèle habituelle
```

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 [7]: english_words_counter = collections.Counter([word for sentence in english_sentences for
        french_words_counter = collections.Counter([word for sentence in french_sentences for wo

        print('{} English words.'.format(len([word for sentence in english_sentences for word in
        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 s
        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 [8]: 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(lower=True)
            tokenizer.fit_on_texts(x)
            x_tokenized = tokenizer.texts_to_sequences(x)
            print(x_tokenized)
            return x_tokenized, 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))
```

```
[[1, 2, 4, 5, 6, 7, 1, 8, 9], [10, 11, 12, 2, 13, 14, 15, 16, 3, 17], [18, 19, 3, 20, 21]]
[[1, 2, 4, 5, 6, 7, 1, 8, 9], [10, 11, 12, 2, 13, 14, 15, 16, 3, 17], [18, 19, 3, 20, 21]]
{'the': 1, 'quick': 2, 'a': 3, 'brown': 4, 'fox': 5, 'jumps': 6, 'over': 7, 'lazy': 8, 'dog': 9,

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 [9]:
```
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 sequen
    :return: Padded numpy array of sequences
    """
    max_len = max([len(item) for item in x])
    if length is None:
        length = max_len
    x_pad = pad_sequences(x, maxlen=length, dtype='int32', padding='post', truncating='p
    return x_pad
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 [10]: 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 d
             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_tokenize
             preprocess(english_sentences, french_sentences)

         max_english_sequence_length = preproc_english_sentences.shape[1]
         max_french_sequence_length = preproc_french_sentences.shape[1]
         english_vocab_size = len(english_tokenizer.word_index)
         french_vocab_size = len(french_tokenizer.word_index)

         print('Data Preprocessed')
         print("Max English sentence length:", max_english_sequence_length)
         print("Max French sentence length:", max_french_sequence_length)
         print("English vocabulary size:", english_vocab_size)
         print("French vocabulary size:", french_vocab_size)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
```

```
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

```
Data Preprocessed
Max English sentence length: 15
Max French sentence length: 21
English vocabulary size: 199
French vocabulary size: 344
```

1.4 Models

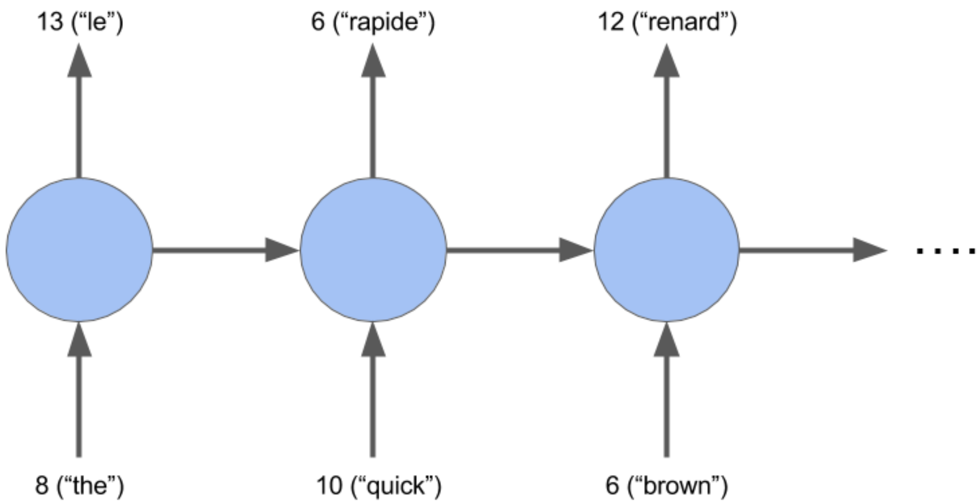
In this section, you will experiment with various neural network architectures. You will begin by training four relatively simple architectures. - Model 1 is a simple RNN - Model 2 is a RNN with Embedding - Model 3 is a Bidirectional RNN - Model 4 is an optional Encoder-Decoder RNN

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 [11]: 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)



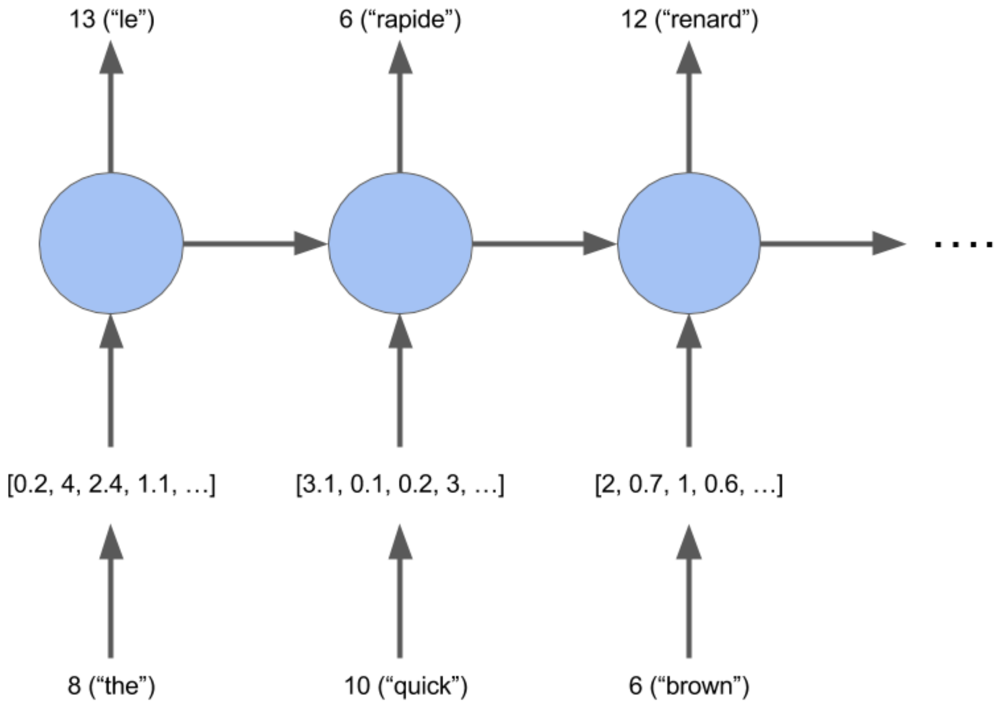
A basic RNN model is a good baseline for sequence data. In this model, you'll build a RNN that translates English to French.

```
In [11]: 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
             """
             # TODO: Build the layers
             learning_rate = 0.001
             model = Sequential()
             model.add(GRU(units=256, input_shape=input_shape[1:], return_sequences=True))
             model.add(TimeDistributed(Dense(french_vocab_size)))
             model.add(Activation('softmax'))
             model.compile(loss=sparse_categorical_crossentropy,
                           optimizer=Adam(learning_rate),
                           metrics=['accuracy'])
             return model
         tests.test_simple_model(simple_model)

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

         # Train the neural network
         simple_rnn_model = simple_model(
             tmp_x.shape,
             max_french_sequence_length,
```

```
    english_vocab_size,
    french_vocab_size+1)
simple_rnn_model.fit(tmp_x, preproc_french_sentences, batch_size=1024, epochs=10, valid

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

```
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 13s 117us/step - loss: 2.5624 - acc: 0.4821 - v
Epoch 2/10
110288/110288 [==============================] - 11s 99us/step - loss: 1.6549 - acc: 0.5848 - va
Epoch 3/10
110288/110288 [==============================] - 11s 99us/step - loss: 1.4314 - acc: 0.6133 - va
Epoch 4/10
110288/110288 [==============================] - 11s 99us/step - loss: 1.3107 - acc: 0.6315 - va
Epoch 5/10
110288/110288 [==============================] - 11s 99us/step - loss: 1.2232 - acc: 0.6458 - va
Epoch 6/10
110288/110288 [==============================] - 11s 99us/step - loss: 1.1582 - acc: 0.6578 - va
Epoch 7/10
110288/110288 [==============================] - 11s 99us/step - loss: 1.1088 - acc: 0.6663 - va
Epoch 8/10
110288/110288 [==============================] - 11s 99us/step - loss: 1.0693 - acc: 0.6717 - va
Epoch 9/10
110288/110288 [==============================] - 11s 99us/step - loss: 1.0369 - acc: 0.6752 - va
Epoch 10/10
110288/110288 [==============================] - 11s 99us/step - loss: 1.0077 - acc: 0.6801 - va
new jersey est parfois calme en mois de il est il en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD
```

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.

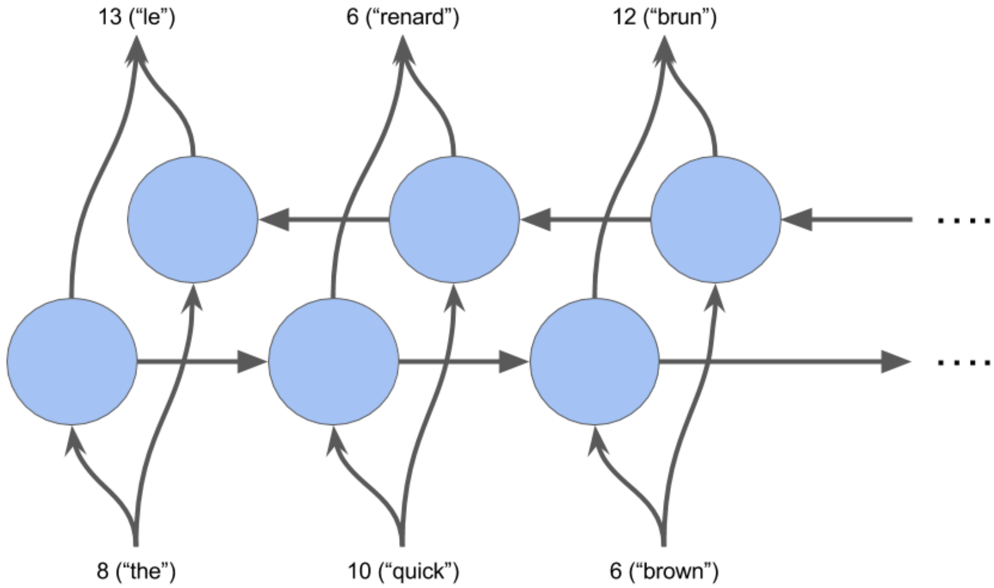
```
In [12]: def embed_model(input_shape, output_sequence_length, english_vocab_size, french_vocab_s
             """
             Build and train a RNN model using word embedding on x and y
             :param input_shape: Tuple of input shape
             :param output_sequence_length: Length of output 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 = 0.01
             model = Sequential()
             model.add(Embedding(english_vocab_size+1, 100, input_shape=input_shape[1:] ))
             model.add(GRU(units=256, return_sequences=True))
             model.add(TimeDistributed(Dense(french_vocab_size)))
             model.add(Activation('softmax'))
             model.compile(loss=sparse_categorical_crossentropy,
                           optimizer=Adam(learning_rate),
                           metrics=['accuracy'])
             model.summary()
             return model
         tests.test_embed_model(embed_model)
```

```python
# TODO: Reshape the input

tmp_x = pad(preproc_english_sentences, max_french_sequence_length)
print('tmp_x.shape ',tmp_x.shape)
#tmp_x = tmp_x.reshape((-1, preproc_french_sentences.shape[-2], 1))

print('english_vocab_size ',english_vocab_size)
print('french_vocab_size ',french_vocab_size)
print('tmp_x.shape ',tmp_x.shape)
# TODO: Train the neural network
embed_rnn_model = embed_model(
    tmp_x.shape,
    max_french_sequence_length,
    english_vocab_size,
    french_vocab_size+1)
embed_rnn_model.fit(tmp_x, preproc_french_sentences, batch_size=1024, epochs=10, valida

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 21, 100)           20000
_________________________________________________________________
gru_3 (GRU)                  (None, 21, 256)           274176
_________________________________________________________________
time_distributed_3 (TimeDist (None, 21, 344)           88408
_________________________________________________________________
activation_3 (Activation)    (None, 21, 344)           0
=================================================================
Total params: 382,584
Trainable params: 382,584
Non-trainable params: 0
_________________________________________________________________
tmp_x.shape  (137861, 21)
english_vocab_size  199
french_vocab_size  344
tmp_x.shape  (137861, 21)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 21, 100)           20000
_________________________________________________________________
gru_4 (GRU)                  (None, 21, 256)           274176
_________________________________________________________________
time_distributed_4 (TimeDist (None, 21, 345)           88665
```

```
_________________________________________________________________
activation_4 (Activation)    (None, 21, 345)           0
=================================================================
Total params: 382,841
Trainable params: 382,841
Non-trainable params: 0
_________________________________________________________________
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 13s 119us/step - loss: 1.7628 - acc: 0.6240 - v
Epoch 2/10
110288/110288 [==============================] - 13s 115us/step - loss: 0.3882 - acc: 0.8780 - v
Epoch 3/10
110288/110288 [==============================] - 13s 115us/step - loss: 0.2617 - acc: 0.9136 - v
Epoch 4/10
110288/110288 [==============================] - 13s 115us/step - loss: 0.2239 - acc: 0.9240 - v
Epoch 5/10
110288/110288 [==============================] - 13s 115us/step - loss: 0.2066 - acc: 0.9287 - v
Epoch 6/10
110288/110288 [==============================] - 13s 115us/step - loss: 0.1958 - acc: 0.9315 - v
Epoch 7/10
110288/110288 [==============================] - 13s 115us/step - loss: 0.1903 - acc: 0.9328 - v
Epoch 8/10
110288/110288 [==============================] - 13s 115us/step - loss: 0.1844 - acc: 0.9345 - v
Epoch 9/10
110288/110288 [==============================] - 13s 115us/step - loss: 0.1833 - acc: 0.9348 - v
Epoch 10/10
110288/110288 [==============================] - 13s 115us/step - loss: 0.1853 - acc: 0.9342 - v
new jersey est parfois calme en l' automne et il est neigeux en avril <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 [14]: def bd_model(input_shape, output_sequence_length, english_vocab_size, french_vocab_size
             """
             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
             """
             learning_rate = 0.01
             model = Sequential()
             model.add(Bidirectional(GRU(units=256, return_sequences=True), input_shape=input_sh
             model.add(TimeDistributed(Dense(french_vocab_size)))
             model.add(Activation('softmax'))
             model.compile(loss=sparse_categorical_crossentropy,
                           optimizer=Adam(learning_rate),
                           metrics=['accuracy'])
             model.summary()
             return model
         tests.test_bd_model(bd_model)


         # TODO: Train and Print prediction(s)
         # Reshaping the input to work with a basic RNN
         tmp_x = pad(preproc_english_sentences, max_french_sequence_length)
         tmp_x = tmp_x.reshape((-1, preproc_french_sentences.shape[-2], 1))
```

```python
print('english_vocab_size ',english_vocab_size)
print('french_vocab_size ',french_vocab_size)
print('tmp_x.shape ',tmp_x.shape)

# Train the neural network
bd_rnn_model = bd_model(
    tmp_x.shape,
    max_french_sequence_length,
    english_vocab_size,
    french_vocab_size+1)
bd_rnn_model.fit(tmp_x, preproc_french_sentences, batch_size=1024, epochs=10, validatio

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_3 (Bidirection (None, 21, 512)           396288
_________________________________________________________________
time_distributed_7 (TimeDist (None, 21, 344)           176472
_________________________________________________________________
activation_7 (Activation)    (None, 21, 344)           0
=================================================================
Total params: 572,760
Trainable params: 572,760
Non-trainable params: 0
_________________________________________________________________
english_vocab_size  199
french_vocab_size  344
tmp_x.shape  (137861, 21, 1)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_4 (Bidirection (None, 21, 512)           396288
_________________________________________________________________
time_distributed_8 (TimeDist (None, 21, 345)           176985
_________________________________________________________________
activation_8 (Activation)    (None, 21, 345)           0
=================================================================
Total params: 573,273
Trainable params: 573,273
Non-trainable params: 0
_________________________________________________________________
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 19s 173us/step - loss: 1.4536 - acc: 0.6208 - v
Epoch 2/10
```

```
110288/110288 [==============================] - 18s 166us/step - loss: 0.9837 - acc: 0.6869 - v
Epoch 3/10
110288/110288 [==============================] - 18s 166us/step - loss: 0.8616 - acc: 0.7068 - v
Epoch 4/10
110288/110288 [==============================] - 18s 166us/step - loss: 0.7940 - acc: 0.7184 - v
Epoch 5/10
110288/110288 [==============================] - 18s 166us/step - loss: 0.7352 - acc: 0.7364 - v
Epoch 6/10
110288/110288 [==============================] - 18s 166us/step - loss: 0.6981 - acc: 0.7487 - v
Epoch 7/10
110288/110288 [==============================] - 18s 166us/step - loss: 0.6701 - acc: 0.7629 - v
Epoch 8/10
110288/110288 [==============================] - 18s 166us/step - loss: 0.6482 - acc: 0.7596 - v
Epoch 9/10
110288/110288 [==============================] - 18s 165us/step - loss: 0.6377 - acc: 0.7646 - v
Epoch 10/10
110288/110288 [==============================] - 18s 166us/step - loss: 0.6251 - acc: 0.7751 - v
new jersey est parfois calme au mois et il est neigeux en avril <PAD> <PAD> <PAD> <PAD> <PAD> <P
```

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 [15]:
```
def encdec_model(input_shape, output_sequence_length, english_vocab_size, french_vocab_
    """
    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
    """
    learning_rate = 0.01

    model = Sequential()
    model.add(GRU(units=256,input_shape=input_shape[1:], return_sequences=False))
    model.add(RepeatVector(output_sequence_length))
    model.add(GRU(french_vocab_size, return_sequences=True))
    model.add(TimeDistributed(Dense(french_vocab_size)))
    model.add(Activation('softmax'))
    model.compile(loss=sparse_categorical_crossentropy,
                  optimizer=Adam(learning_rate),
                  metrics=['accuracy'])
    model.summary()
```

```
    return model
tests.test_encdec_model(encdec_model)


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

print('english_vocab_size ',english_vocab_size)
print('french_vocab_size ',french_vocab_size)
print('tmp_x.shape ',tmp_x.shape)

# Train the neural network
encdec_rnn_model = encdec_model(
    tmp_x.shape,
    max_french_sequence_length,
    english_vocab_size,
    french_vocab_size+1)
encdec_rnn_model.fit(tmp_x, preproc_french_sentences, batch_size=1024, epochs=10, valid

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_9 (GRU)                  (None, 256)               198144
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 21, 256)           0
_________________________________________________________________
gru_10 (GRU)                 (None, 21, 344)           620232
_________________________________________________________________
time_distributed_9 (TimeDist (None, 21, 344)           118680
_________________________________________________________________
activation_9 (Activation)    (None, 21, 344)           0
=================================================================
Total params: 937,056
Trainable params: 937,056
Non-trainable params: 0
_________________________________________________________________
english_vocab_size  199
french_vocab_size  344
tmp_x.shape  (137861, 21, 1)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_11 (GRU)                 (None, 256)               198144
```

```
_________________________________________________________________
repeat_vector_2 (RepeatVecto (None, 21, 256)          0
_________________________________________________________________
gru_12 (GRU)                 (None, 21, 345)          623070
_________________________________________________________________
time_distributed_10 (TimeDis (None, 21, 345)          119370
_________________________________________________________________
activation_10 (Activation)   (None, 21, 345)          0
=================================================================
Total params: 940,584
Trainable params: 940,584
Non-trainable params: 0
_________________________________________________________________
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 24s 217us/step - loss: 3.0571 - acc: 0.4052 - v
Epoch 2/10
110288/110288 [==============================] - 23s 209us/step - loss: 2.5882 - acc: 0.4334 - v
Epoch 3/10
110288/110288 [==============================] - 23s 209us/step - loss: 2.5290 - acc: 0.4389 - v
Epoch 4/10
110288/110288 [==============================] - 23s 209us/step - loss: 2.4235 - acc: 0.4564 - v
Epoch 5/10
110288/110288 [==============================] - 23s 209us/step - loss: 2.3427 - acc: 0.4805 - v
Epoch 6/10
110288/110288 [==============================] - 23s 209us/step - loss: 2.3062 - acc: 0.4844 - v
Epoch 7/10
110288/110288 [==============================] - 23s 209us/step - loss: 2.2758 - acc: 0.4873 - v
Epoch 8/10
110288/110288 [==============================] - 23s 209us/step - loss: 2.5689 - acc: 0.4388 - v
Epoch 9/10
110288/110288 [==============================] - 23s 209us/step - loss: 2.5196 - acc: 0.4331 - v
Epoch 10/10
110288/110288 [==============================] - 23s 209us/step - loss: 2.5041 - acc: 0.4332 - v
new <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <P
```

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 [12]: def model_final(input_shape, output_sequence_length, english_vocab_size, french_vocab_s
             """
             Build and train a model that incorporates embedding, encoder-decoder, and bidirecti
             :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 = 0.01

        model = Sequential()
        model.add(Embedding(english_vocab_size+1, 100, input_shape=input_shape[1:] ))
        model.add(Bidirectional(GRU(units=256, return_sequences=False), name="encoder_gru")
        model.add(RepeatVector(output_sequence_length))
        model.add(Bidirectional(GRU(french_vocab_size, return_sequences=True), name="decode
        model.add(TimeDistributed(Dense(french_vocab_size)))
        model.add(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')
    # TODO: Train the final model
    tmp_x = pad(preproc_english_sentences, max_french_sequence_length)
    #tmp_x = tmp_x.reshape((-1, preproc_french_sentences.shape[-2], 1))

    print('english_vocab_size ',english_vocab_size)
    print('french_vocab_size ',french_vocab_size)
    print('tmp_x.shape ',tmp_x.shape)

    # Train the neural network
    model = model_final(tmp_x.shape,
        max_french_sequence_length,
        english_vocab_size,
        french_vocab_size+1)
    model.fit(tmp_x, preproc_french_sentences, batch_size=1024, epochs=10, validation_split
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 15, 100)           20000
_________________________________________________________________
encoder_gru (Bidirectional)  (None, 512)               548352
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 21, 512)           0
_________________________________________________________________
decoder_gru (Bidirectional)  (None, 21, 688)           1768848
_________________________________________________________________
time_distributed_1 (TimeDist (None, 21, 344)           237016
```

```
_________________________________________________________________
activation_1 (Activation)    (None, 21, 344)           0
=================================================================
Total params: 2,574,216
Trainable params: 2,574,216
Non-trainable params: 0
_________________________________________________________________
Final Model Loaded
english_vocab_size  199
french_vocab_size  344
tmp_x.shape  (137861, 21)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 21, 100)           20000
_________________________________________________________________
encoder_gru (Bidirectional)  (None, 512)               548352
_________________________________________________________________
repeat_vector_2 (RepeatVecto (None, 21, 512)           0
_________________________________________________________________
decoder_gru (Bidirectional)  (None, 21, 690)           1776060
_________________________________________________________________
time_distributed_2 (TimeDist (None, 21, 345)           238395
_________________________________________________________________
activation_2 (Activation)    (None, 21, 345)           0
=================================================================
Total params: 2,582,807
Trainable params: 2,582,807
Non-trainable params: 0
_________________________________________________________________
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 53s 479us/step - loss: 3.4889 - acc: 0.4324 - v
Epoch 2/10
110288/110288 [==============================] - 50s 455us/step - loss: 2.3375 - acc: 0.5546 - v
Epoch 3/10
110288/110288 [==============================] - 50s 456us/step - loss: 1.9269 - acc: 0.6606 - v
Epoch 4/10
110288/110288 [==============================] - 50s 456us/step - loss: 1.7899 - acc: 0.6877 - v
Epoch 5/10
110288/110288 [==============================] - 50s 455us/step - loss: 1.5513 - acc: 0.7483 - v
Epoch 6/10
110288/110288 [==============================] - 50s 456us/step - loss: 1.4228 - acc: 0.7821 - v
Epoch 7/10
110288/110288 [==============================] - 50s 456us/step - loss: 1.3051 - acc: 0.8180 - v
Epoch 8/10
110288/110288 [==============================] - 50s 456us/step - loss: 1.2219 - acc: 0.8440 - v
Epoch 9/10
```

```
110288/110288 [==============================] - 50s 456us/step - loss: 1.1400 - acc: 0.8718 - v
Epoch 10/10
110288/110288 [==============================] - 50s 457us/step - loss: 1.1206 - acc: 0.8770 - v
```

```
Out[12]: <keras.callbacks.History at 0x7f59d29c8da0>
```

1.5 Prediction (IMPLEMENTATION)

```
In [20]: 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
             """
             tmp_x = pad_sequences(x, maxlen=x.shape[-1], padding='post')
             model = model_final(tmp_x.shape,
                 max_french_sequence_length,
                 english_vocab_size,
                 french_vocab_size+1)
             model.fit(tmp_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.max(x)] for x in y[0]]))


         final_predictions(preproc_english_sentences, preproc_french_sentences, english_tokenize
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 15, 100)           20000
```

```
_________________________________________________________________
encoder_gru (Bidirectional)  (None, 512)              548352
_________________________________________________________________
repeat_vector_3 (RepeatVecto (None, 21, 512)          0
_________________________________________________________________
decoder_gru (Bidirectional)  (None, 21, 690)          1776060
_________________________________________________________________
time_distributed_3 (TimeDist (None, 21, 345)          238395
_________________________________________________________________
activation_3 (Activation)    (None, 21, 345)          0
=================================================================
Total params: 2,582,807
Trainable params: 2,582,807
Non-trainable params: 0
_________________________________________________________________
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 47s 427us/step - loss: 2.3492 - acc: 0.5097 - v
Epoch 2/10
110288/110288 [==============================] - 46s 418us/step - loss: 0.9825 - acc: 0.7205 - v
Epoch 3/10
110288/110288 [==============================] - 46s 418us/step - loss: 0.5484 - acc: 0.8357 - v
Epoch 4/10
110288/110288 [==============================] - 46s 418us/step - loss: 0.2883 - acc: 0.9172 - v
Epoch 5/10
110288/110288 [==============================] - 46s 418us/step - loss: 0.1925 - acc: 0.9445 - v
Epoch 6/10
110288/110288 [==============================] - 46s 418us/step - loss: 0.1439 - acc: 0.9582 - v
Epoch 7/10
110288/110288 [==============================] - 46s 418us/step - loss: 0.1265 - acc: 0.9629 - v
Epoch 8/10
110288/110288 [==============================] - 46s 418us/step - loss: 0.1073 - acc: 0.9686 - v
Epoch 9/10
110288/110288 [==============================] - 46s 418us/step - loss: 0.1010 - acc: 0.9705 - v
Epoch 10/10
110288/110288 [==============================] - 46s 418us/step - loss: 0.0913 - acc: 0.9733 - v
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 et il est neigeux en avril <PAD> <PAD> <PAD> <PA
new jersey est parfois calme pendant l' automne et il est neigeux en avril <PAD> <PAD> <PAD> <PA
```

1.6 Submission

When you're ready to submit, complete the following steps: 1. Review the rubric to ensure your submission meets all requirements to pass 2. Generate an HTML version of this notebook

- Run the next cell to attempt automatic generation (this is the recommended method in Workspaces)
- Navigate to **FILE -> Download as -> HTML (.html)**
- Manually generate a copy using `nbconvert` from your shell terminal

```
$ pip install nbconvert
$ python -m nbconvert machine_translation.ipynb
```

3. Submit the project

- If you are in a Workspace, simply click the "Submit Project" button (bottom towards the right)

- Otherwise, add the following files into a zip archive and submit them

- `helper.py`

- `machine_translation.ipynb`

- `machine_translation.html`

  - You can export the notebook by navigating to **File -> Download as -> HTML (.html)**.

1.6.1 Generate the html

Save your notebook before running the next cell to generate the HTML output. Then submit your project.

```
In [2]: # Save before you run this cell!
        !!jupyter nbconvert *.ipynb
```

```
Out[2]: ['[NbConvertApp] Converting notebook machine_translation.ipynb to html',
         '[NbConvertApp] Writing 305996 bytes to machine_translation.html']
```

1.7 Optional Enhancements

This project focuses on learning various network architectures for machine translation, but we don't evaluate the models according to best practices by splitting the data into separate test & training sets – so the model accuracy is overstated. Use the `sklearn.model_selection.train_test_split()` function to create separate training & test datasets, then retrain each of the models using only the training set and evaluate the prediction accuracy using the hold out test set. Does the "best" model change?