Assignment 4: Language Models and Machine Translation

Due **Friday, 22/10/21 at 11:59pm [no late penalty until 30/10/20]**

French philosopher Jean-Paul Sartre had many profound things to say about the universe. However, many of his greatest musings are in French!
Therefore, to get at the meaning of existence (or lack thereof), we will need to train a model that can translate from French to English! To get there we need to understand the meaning of words!

This sentence is words. These words have meaning. One way to understand the meaning of words is by generating the correct expected words in a language. This assignment begins by building a model that can generate language just as coherent as this as coherent as this as coherent as this is!

Getting the stencil

You can find the files on learn under assignment 4. The files are compressed into a ZIP file, and to unzip the ZIP file, you can just double click on the ZIP file. There, you should find the files: preprocess.py, assignment_part1.py, assignment_part2.py, rnn_model.py, README, and data folder.

Logistics

Work on this assignment off of the stencil code provided, but **you do not need to change the stencil except where specified.** **The only code you need to add/change is in rnn_model.py.**
You will run the main/test functions in assignment_part1.py and assignment_part2.py. You are welcome to change code elsewhere if you want to test or experiment with things.

This assignment has 2 main parts. You will implement a Recurrent Neural Network language generation model and train the RNN on the modified Wall Street Journal corpus we have provided. You will also implement a sequence to sequence language translation model based on the RNN.

Part 1: RNN Language Model...with Keras Layers!

The goal of part 1 is to build a model that can generate the next word when given a sequence of words. A simple example is the poem:

> Jack and Jill went up the hill
> To fetch a pail of water
> Jack fell down and broke his crown
> And Jill came tumbling after

If this is the only text we know about for this language, then a perfect language model would be able to generate the poem given only the following sentence stems:

> Jack and
> To
> Jack fell
> And

because all subsequent words are predictable given the stems. Notice that we cannot just provide "Jack" as a stem because there are two potential sequences that could follow. The general problem we are solving is *given N words in a language, what should the N+1 word be?* Breaking down one of the stems above a correct model would generate:

N words [input]	**N+1 word [output]**
To	fetch
To fetch	a
To fetch a	pail
To fetch a pail	of
To fetch a pail of	water

The Corpus [Part 1]

When it comes to language modeling, there is a LOT of data out there! In language modeling, we take some set of words, and train our model to predict the next, so almost any large text corpus will do.

Our data comes from the WSJ corpus, and comprises sentences taken from the newspaper--in fact, if you look at the text, you will see lots of references to finance, stocks, etc. Another thing you will see in the corpus is things that look like this: UNK-.

Uncommon words have been UNKed. This is a common preprocessing technique to remove rare words, and help the model focus more on learning patterns in general, common language. Because of this 'UNKing' there are no words in the testing corpus that are not in the training corpus! Lastly, you will notice that each text file has a sentence per line, followed by a 'STOP'.

Preprocessing

The preprocessing file contains a get_part1_data function. This function:

Step 1. Loads the file words and split the words on whitespace.
Step 2. Removes the STOP word at the end of every sentence and pads any sentence less than WINDOW_SIZE with extra PAD_TOKEN
Step 3. Creates a vocab dict [if not provided] that maps a unique index (id) to each word in the corpus
Step 4. Converts the list of words to their indices, making a 1-d list/array for each (this is called Tokenization)
Step 5. Returns an iterable of ids and the vocab dict

Roadmap [i.e. what you actually have to do]

Now that we are working with more complex architectures, like LSTMs and GRUs, it's time to add in a little more abstraction. You should use keras layers in your model instead of directly creating tensorflow Variables and calling low level math functions (just don't use the sequential API -- you need to subclass Model as before). For this assignment, the relevant layers are:

- tf.keras.layers.Embedding
- tf.keras.layers.GRU OR tf.keras.layers.LSTM for your RNN
- tf.keras.layers.Dense for feedforward layers.

Step 1. Create your model

- Fill out the RNN_Part1 init function by defining your layers. A basic language model has the following structure: Embedding -> RNN -> Dense(vocab size, with softmax activation)

When you define your RNN keras layer, make sure that initialize it with return_sequences=True

- Fill out the call function using the layers you've created.
- Fill out the loss function by calculating the average softmax cross-entropy loss on the probabilities compared to the labels. (These should NOT be one hot vectors).

We recommend using tf.keras.losses.sparse_categorical_crossentropy.

Note that since some words are special (padding, stops, etc) the loss is not as accurate as it should be and our model learns to predict things like *UNK*. You can improve this by computing a mask over the labels and excluding non-words from the loss function using tf.boolean_mask. One way to create such a mask is:

```
mask = tf.less(labels, self.vocab[FIRST_SPECIAL]) | tf.greater(labels,
self.vocab[LAST_SPECIAL])
```

Step 2. Train and test

- In the main function of assignment_part1.py there are three tests that it runs:
    1. The Jack and Jill poem. You should see a good approximation of the poem output as text if your model works. It should only take a minute or so to train and test with 100 epochs.
    2. A short version of the Wall Street Journal training set. As your model trains over each epoch the generated debug and test sentences should start making more sense around training loss of 2. This test also outputs loss and perplexity on the test data set -- this should score relatively poor because the words it trained on are a very small set. Indeed you should see the test perplexity go down to start but then as the model overfits to the training data the perplexity will go back up. This may take around 10 minutes to train for 100 epochs.
    3. The long version of the Wall Street Journal training set. Training one epoch may take around 5 minutes but this will give the best general model. Your model should achieve around 500 perplexity on the test set after one epoch for partial credit, <250 perplexity for full credit.
- You can modify any of the training code but please make sure to turn in something that takes about <= 20 minutes to run total.

Sample output from my reference implementation:

```
#test 1
Jack and Jill went up the hill
To fetch a pail of water
Jack fell down and broke his crown
And Jill came after after
test loss 0.222 perplexity 1.248
…

# test 2
debug output:
#output:The I had been U.S. of be . '' says the to to to to to
#actual:`` Nobody had the *UNKS* to complain . '' *STOP* *PAD* *PAD* *PAD* *PAD*
*PAD* *PAD*
Epoch 14 training loss 3.732
```

```
test output:
#output:The I and the , to be the than to the the U.S. year York concern
#actual:`` Trade is *UNK* going to be more politically sensitive over the next six or
*STOP*
test loss 5.949 perplexity 383.214
…
debug output:
#output:The said the indictment on local political thin , who with his aggressive efforts to
be
#actual:He *UNKS* the indictment on local political *UNKING* , *UNKS* with his
aggressive efforts to *STOP*
Epoch 99 training loss 0.523
test output:
#output:The judge of gone considered to a , the manager of , of of their ,
#actual:The *UNKION* had been raised in Parliament by the *UNK* *CUNKAL* *UNK*
*UNK* following *UNK* *STOP*
test loss 8.879 perplexity 7182.084
```

# test 3

```
debug output:
#output:The , the , a to be the 2 million . '' the it are a
#actual:Citibank and Chase had agreed to *UNK* $ 3 billion , and said they were
*STOP*
Epoch 0 loss 5.966
test output:
#output:`` It 's a a to $ market 's , '' says Paul , a ''
#actual:`` It is almost *UNKAL* to the Sony product , '' Mr. *CUNK* *UNKED* , *STOP*
test loss 5.494 perplexity 243.293
```

Part 2: RNN Machine Translation

The Corpus [Part 2]

By law, the official records, (Hansards) of the Canadian Parliament must be transcribed in both English and French. Therefore, they provide a fantastic set of mappings for a machine translation model. We are providing you with a modified version of the corpus that only includes sentences shorter than 12 words.

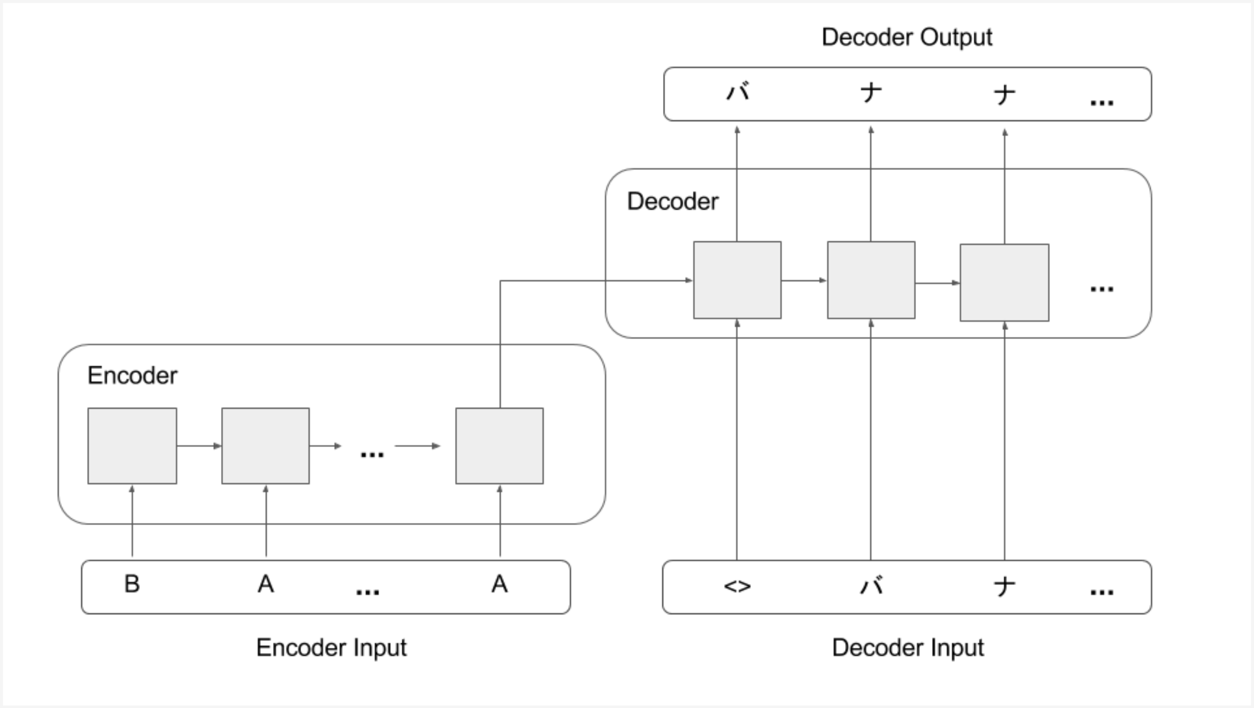
Here's what you should find in the data folder:

fls.txt - french_training_file
els.txt - english_training_file
flt.txt - french_test_file
elt.txt - english_test_file

Preprocessing
The preprocessing file contains a get_part2_data function.

Step 1. Loads the french and english file words and splits the words on whitespace.
Step 2. Pads any sentence less than WINDOW_SIZE with extra PAD_TOKEN
Step 3. Creates a french and english vocab dict [if not provided] that maps a unique index (id) to each word in the corpus
Step 4. Converts the list of words to their indices, making a 1-d list/array for each (this is called Tokenization)
Step 5. Returns an iterable of ids and the vocab dict

Roadmap
For this part you will build an RNN based encoder-decoder architecture that encodes the French sentence and then decodes the English translation. Here is a general picture of the architecture we are creating:



In this diagram the gray boxes are a single RNN encoder and a single RNN decoder which recursively processes a sequence of input for both encoding and decoding. In language translation the input sequence is known (for us it is the French sentence). During training we also know the decoder input and decoder output (the English sentence). Note how the Decoder Output is shifted by 1 from the Decoder Input. We implement the training process shown in the diagram which is called Teacher Forcing which stablizes training (see https://machinelearningmastery.com/teacher-forcing-for-recurrent-neural-networks/ for more details). When we actually translate a sentence for testing, we will only pass in the *START* English word (the <> in the diagram) and the encoded hidden state of the

French sentence to generate one English word at a time. You can see the code for this in assignment_part2.py's translate_sentence function.

Step 1. Create your RNN model

- Fill out the init function, and define your encoder and decoder layers. You should use at least one RNN layer to encode, one RNN layer to decode. Note that you need two embedding layers, one for each language, and at least one dense layer for the final output.
- Fill out the call function using your layers to recreate the logical flow shown in the diagram.
- Calculate the average loss using tf.keras.losses.sparse_categorical_crossentropy. You should use a mask that does not include labels that are padding. You also need to shift the labels and output probabilities by 1 so that the loss function compares index 1 of the output probabilities with index 0 of the labels (since the labels have an extra *START* tag -- see code comment for details).

Step 2. Train and test

- In the main function of assignment_part2.py there are two tests that it runs:
    1. A short version that trains on a shorter english and french corpus. It trains for 20 epochs and outputs sample translations just from the testing on the shorter training set itself. Because this is meant to overfit the data you should see its loss and perplexity go down towards zero after the 20 epochs as it begins to memorize the phrases. Your version should be below 10 perplexity after 20 epochs.
    2. A longer version that trains on the longer english and french corpus. It trains for 1 epoch total and will take a long time so it also outputs validation tests and debug sentences along the way. You should see it output perplexity on the full test set every 10 training batches.
- You can modify any of the training code but please make sure to turn in something that takes about <= 60 minutes to run total.

Sample output from my reference implementation:

```
#test 1
Epoch 0
test output:
#input:mme elsie wayne *STOP* *PAD* *PAD* *PAD* *PAD* *PAD* *PAD* *PAD* *PAD*
*PAD* *PAD* *PAD*
#output:mr. *STOP* *STOP* *STOP* *STOP* *STOP* *STOP* *STOP* *STOP* *STOP*
*STOP* *STOP* *STOP* *STOP* *STOP* *STOP*
#actual:mrs. elsie wayne *STOP* *PAD* *PAD* *PAD* *PAD* *PAD* *PAD* *PAD*
*PAD* *PAD* *PAD* *PAD* *PAD*
test loss 5.659 perplexity 286.924
…
Epoch 20
```

```
test output:
#input:l ' hon. paul martin *STOP* *PAD* *PAD* *PAD* *PAD* *PAD* *PAD* *PAD*
*PAD* *PAD*
#output:hon. jean martin *STOP* ) ) ) ) ) ) ) . . . . . .
#actual:hon. paul martin *STOP* *PAD* *PAD* *PAD* *PAD* *PAD* *PAD* *PAD* *PAD*
*PAD* *PAD* *PAD* *PAD*
test loss 1.240 perplexity 3.457
```

# test 2

```
debug output:
#input:quand donc les réformistes se sont-ils intéressés à la question ? *STOP* *PAD*
*PAD* *PAD*
#output:i we that speaker party . the on the is . *STOP* *STOP* *STOP* *STOP* .
#actual:when does the reform party become interested in this issue ? *STOP* *PAD*
*PAD* *PAD* *PAD*
test loss 3.420 perplexity 30.565
…
```

Part 3: Conceptual Questions

1. What are the dimensions of an embedding matrix? What do they represent?
2. Given the following 3 sentences, plot in 2d (or create an adjacency matrix) reasonable embeddings for Sam, Came, Theater, Store, and Went. (Hint: A simple graph with some clusters is fine and you don't need to specifically follow a CBOW or other algorithm to generate one). See lecture notes or https://medium.com/@Petuum/embeddings-a-matrix-of-meaning-4de877c9aa27 for an overview.

   Sam went to the store.
   Then Sam came to the theater.
   I went to the theater.

3. What are the limitations of feed forward networks for language modelling that RNNS solve? (3-6 sentences)
4. Explain in your own words how your RNN in this assignment predicts the next word in the sequence.
5. What are the limitations of RNNS that Transformers solve? (3-6 sentences)
6. Explain in your own words how your RNN encoder and RNN decoder work together to translate from French to English in this assignment.

Grading

Code: You will be primarily graded on functionality. **Your part1 RNN model should have a perplexity < 250. Your part2 RNN model should have a perplexity < 10 on the short set and < 10 after 1 epoch on the long set.**

Conceptual: You will be primarily graded on correctness (when applicable), thoughtfulness, and clarity.