Congratulations 

* Your submission reveals that you have made an **excellent effort** in finishing this project. It is an important milestone in learning about RNNs
* Very good hyperparameters and loss . It is great that you have got everything right in first review 
* I wish you all the best for next adventures 
* Nice Read: (Colah's Blog) <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
* Nice Read: (Andrej Karpathy) : <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
* Nice Read: (Rohan Kapur) <https://ayearofai.com/rohan-lenny-3-recurrent-neural-networks-10300100899b>

Keep up the good work  Stay Udacious :udacious:

**All Required Files and Tests**

**The project submission contains the project notebook, called “dlnd\_tv\_script\_generation.ipynb”.**

All required files are present 

* It is recommended to export your conda environment into environment.yaml file. command **conda env export -f environment.yaml**, so that you can recreate your conda environment later.
* While submitting this to any version control system like Github, make sure to include helper, data and environment files and exclude temp files. It will help you in future if you want to re-execute it. Some [guideline](https://udacity.github.io/git-styleguide/) for best practice.

**All the unit tests in project have passed.**

Well Done !  Donald Knuth (a famous computer science pioneer) once famously said “Beware of bugs in the above code; I have only proved it correct, not tried it.”

**Pre-processing Data**

**The function create\_lookup\_tables create two dictionaries:**

* **Dictionary to go from the words to an id, we'll call vocab\_to\_int**
* **Dictionary to go from the id to word, we'll call int\_to\_vocab**

**The function create\_lookup\_tables return these dictionaries as a tuple (vocab\_to\_int, int\_to\_vocab).**

Good work 

* In particular, using a python set function to ensure each entry in the vocab list is unique is an excellent technique to remove duplicates from a list, and something that you will find yourself doing as part of almost any dataset-preparation work.
* when creating lookup tables it can also be helpful to index words by the frequency each word occurs in the text. The python Counter function (part of the collections library) is a convenient way to get the information needed for that approach.  
  <https://pymotw.com/2/collections/counter.html>
* from collections import Counter
* word\_counts = Counter(text)
* sorted\_words = sorted(word\_counts, key=word\_counts.get, reverse=True)
* vocab\_to\_int = dict()
* int\_to\_vocab = dict()
* for i, word in enumerate(sorted\_words):
* vocab\_to\_int[word] = i
* int\_to\_vocab[i] = word
* return vocab\_to\_int, int\_to\_vocab

**The function token\_lookup returns a dict that can correctly tokenizes the provided symbols.**

* You can keep entries in separate lines to make it more readable ..
* return {
* '.' : '||period||',
* ',' : '||comma||',
* '"' : '||quotationmark||',
* ';' : '||semicolon||',
* '!' : '||exclamationmark||',
* '?' : '||questionmark||',
* '(' : '||leftparentheses',
* ')' : '||rightparentheses',
* '--' : '||doubledash||',
* '\n' : '||return||'
* }

**Batching Data**

**The function batch\_data breaks up word id's into the appropriate sequence lengths, such that only complete sequence lengths are constructed.**

Good Job 

* breaks up word id's into the appropriate sequence lengths

**In the function batch\_data, data is converted into Tensors and formatted with TensorDataset.**

**Finally, batch\_data returns a DataLoader for the batched training data.**

**Build the RNN**

**The RNN class has complete \_\_init\_\_, forward , and init\_hidden functions.**



* \_\_init\_\_, forward and init\_hidden functions are complete

**The RNN must include an LSTM or GRU and at least one fully-connected layer. The LSTM/GRU should be correctly initialized, where relevant.**

The ideal structure is as follows:

* Embedding layer (nn.Embedding) before the LSTM or GRU layer.
* The fully-connected layer comes at the end to get our desired number of outputs.
* Extra marks for not using a dropout after LSTM and before FC layer, as the drop out is already incorporated in the LSTMs, A lot of students will add it and then end up finding convergence difficult
* You can try to add more than one fc:
* # init
* self.fcc=nn.Linear(self.hidden\_dim, self.hidden\_dim)
* self.fcc2=nn.Linear(self.hidden\_dim,self.output\_size)
* ....
* # forward
* output,hidden=self.lstm(embedded,hidden)
* lstm\_output = output.contiguous().view(-1, self.hidden\_dim)
* output=self.dropout(output)
* output= self.fcc(output)
* output=self.dropout(output)
* output=self.fcc2(output)

**RNN Training**

* **Enough epochs to get near a minimum in the training loss, no real upper limit on this. Just need to make sure the training loss is low and not improving much with more training.**
* **Batch size is large enough to train efficiently, but small enough to fit the data in memory. No real “best” value here, depends on GPU memory usually.**
* **Embedding dimension, significantly smaller than the size of the vocabulary, if you choose to use word embeddings**
* **Hidden dimension (number of units in the hidden layers of the RNN) is large enough to fit the data well. Again, no real “best” value.**
* **n\_layers (number of layers in a GRU/LSTM) is between 1-3.**
* **The sequence length (seq\_length) here should be about the size of the length of sentences you want to look at before you generate the next word.**
* **The learning rate shouldn’t be too large because the training algorithm won’t converge. But needs to be large enough that training doesn’t take forever.**



* Enough epochs to get near a minimum in the training loss. Do not hesitate to use a value as big as needed till the loss the improving
* Batch size is large enough to train efficiently
* Sequence length is about the size of the length of sentences we want to generate
* Size of embedding is in the range of [200-300]
* Learning rate seems good based on other hyper parameter

Your efforts shows that you have really have executed it again and again to get an optimized value 

**The printed loss should decrease during training. The loss should reach a value lower than 3.5.**

 excellent !

**There is a provided answer that justifies choices about model size, sequence length, and other parameters.**

well written answer. The act of explaining implementation by writing often leads to a deeper understanding of the concepts.

**Generate TV Script**

**The generated script can vary in length, and should look structurally similar to the TV script in the dataset.**

**It doesn’t have to be grammatically correct or make sense.**

well generated fun script ! 