Meets Specifications

Congratulations! Great model architecture and choice of hyper-parameters. Keep up the good work!

All

Required Files and Tests

The project submission contains the project notebook, called "dlnd_tv_script_generation.ipynb".

All the unit tests in project have passed.

Congratulations!

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).

The function `token_lookup` returns a dict that can correctly tokenizes the provided symbols.

Batching Data

The function `batch_data` breaks up word id's into the appropriate sequence lengths, such that only complete sequence lengths are constructed.

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.

Great model architecture!

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

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.**

Great choice of hyper-parameters. Some suggestions:

- Batch size -> in order to use the GPU more efficiently, always try to set a value that is a power of two (e.g. 64 or 128).
- RNN size and embed dim -> nice choices! Bigger values might improve the training results in exchange of a longer training, due to the added complexity.
- Number of layers -> Despite the comments in the project recommending a value between 1-3, some students managed to get great results with 4 or 5 layers.

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

Awesome!

You do not need to modify the starting code or use pre-trained models to achieve these results, but it is great that you were resourceful enough to reach good loss values by your own means.

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

Great, very detailed explanation. Informed trial an error is a good strategy to come up with reasonable values.

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.

Reads great!