Generate TV Scripts

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

Congratulation!!, I think you've done a perfect job of implementing a recurrent neural net fully. It's very clear that you have a good understanding of the basics. Keep improving and keep learning. 👍

All Required Files and Tests

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

The `dlnd_tv_script_generation.ipynb` , `helper.py` , `problem_unittests.py` and `HTML` files are included.

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

Great work. Unit testing is one of the most reliable methods to ensure that your code is free from all bugs without getting confused with the interactions with all the other code. But always keep in mind, that unit tests cannot catch every issue in the code. So your code could have bugs even though unit tests pass.

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).

Clean and concise.
Mapping each char to unique identifier (int) and vice-versa is always a good approach when working with text data. Further, when generating new text, this will be of utmost importance.

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

Nicely done, as required!
Converting each punctuation into explicit token is very handy when working with RNNs.

Batching Data

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

A good implementation here.

✓ **In the function `batch_data` , data is converted into Tensors and formatted with TensorDataset.**

Converting data into tensors is crucial for torch.

✓ **Finally, `batch_data` returns a DataLoader for the batched training data.**

Correctly implemented again.

Build the RNN

✓ **The RNN class has complete `__init__`, `forward`, and `init_hidden` functions.**

RNN Cell and the hidden state looks all good to me.
Dropout layers have been added in order to reduce network overfitting.

Dropout is a regularization method where input and recurrent connections to LSTM units are probabilistically excluded from activation and weight updates while training a network. This has the effect of reducing overfitting and improving model performance.

Moreover, embedding are necessary because Neural Nets need numbers to crunch instead of words.

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

Congrats on placing all the puzzles together ! You correctly built your RNN :)

One point of note is, torch abstracts a lot of detailed theory behind RNN/LSTM. In the long run, you would definitely want to understand these concepts by heart. C Olah and Andrej are two researchers who explains these concepts wonderfully.

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.**

15 `epochs` with around 0.001 `learning_rate` took you places. The hyperparams chosen are very good as evident from your training loss.
Further, setting `batch_size` as a power of 2 (200 in your case) is handled efficiently by torch (better so on GPU).

Everything worked perfectly with these setting of hyperparams. Seems to me, you had your Deep learning hat on while choosing these. 😉

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

Good job !

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

Good explanation evidenced from experiments.

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.

Finally!, the generated script is similar to the script in the dataset.
These conversations are amazing knowing they are produced by an RNN. I am sure training on the whole series will produce better results, who knows, an episode itself.