Generate TV Scripts

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

🎉 Great work on the project! Nice job tuning the hyperparameters to get the losses low enough. As you can see, LSTM-generated text cannot create meaningful long-term text yet, but is an active area of research.

Other libraries for text generation

As with most things in Python, someone has already created libraries to do a similar thing as in this project, but in a few lines of code. Here's one library (textgenrnn) which does just that, although with TensorFlow and Keras. texar is another one that is a bit more general-purpose.

Best of luck in the rest of the nanodegree!

All Required Files and Tests

The project submission contains the project notebook, called "dlnd_tv_script_generation.ipynb".
Yes, the jupyter notebook is included.

All the unit tests in project have passed.

Yes, all tests passed - good work!

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 with this one, but the counter you used actually has some extra overhead we don't need, so you could just use a set instead: `set(text)`. The sorting from the example is also unnecessary. Your use of enumerating only once is good, creating both dicts in the same loop. All of these things will save some compute power/time.

I would do it like this:

```
vocab = set(text)
vocab_to_int, int_to_vocab = {}, {}
for i, w in enumerate(vocab):
    vocab_to_int[w] = i
    int_to_vocab[i] = w

return (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

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.

Yes, a `DataLoader` has been returned.

Build the RNN

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

Good work! The 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.

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.

You got it! Everything is in a good range for the most part.

LR

Your learning rate is good at 0.001; this usually has to be kind of small for LSTMs. You could also use learning rate schedules and decay to optimize the learning rate a bit better.

Embed dims

Your embed dims are a bit on the big side at 200, but work. Google's news word vectors, the GloVe vectors, and other word vectors are usually in the range 50 to 300, so I like to use embed_dims in that range for words

typically. However, Google's blog post here suggests using the 4th root of the vocab size or number of categories, which in this case ends up being around 15. And you can get about the same accuracy as a higher embedding dim with only 15 as the embedding dimension.

More efficient way to search hyperparameter space

You could also use something like Bayesian search to optimize your hyperpamaters. However, having a good idea of starting points for the hyperparameters helps a lot in shortening search times and costs.

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

3.41 💯

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

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.

DOWNLOAD PROJECT

Rate this review

START