PROJECT

Generate TV Scripts

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW | CODE REVIEW | NOTES

Meets Specifications

SHARE YOUR ACCOMPLISHMENT

> After playing for a long time with these values, I have found that lower seq_length = 10 performs better, but I'm not sure why. Also I'd like to have some input about the number of LSTM cells, in this case LSTM above 2 performs better than 3 or 4, it's not really worthy the time spent vs the improvement achived on the train_loss value.

A smaller sequence length generates more training batches per epoch, and also determines the output length of the generation sequence. It's best to use a sequence length that matches the average word length of the input text. With the MultiRNNCell, you are stacking layers of LSTMs, which has a similar effect of stacking layers in other networks - a larger model means higher variance or complexity, higher difficulty of training, and possibility for overfitting.

> Please could you provide any further reading related to LSTM applied for text generation?? Thanks!!

Here are some of my favorite blog posts:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

Great job! You've correctly implemented all of the functions in this project. See below for some additional materials.
Can't wait to see what you build next. Keep it up! 🏅 🚀 😀

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.**

Preprocessing

✓ **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 in the a tuple (vocab_to_int, int_to_vocab)

Great job! 😀

In other projects, you can also perform a similar operation using tool from scikit-learn called LabelBinarizer: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelBinarizer.html

Here's another guide on working with text data with help from scikit-learn: http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

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

Build the Neural Network

✓ **Implemented the `get_inputs` function to create TF Placeholders for the Neural Network with the following placeholders:**

- **Input text placeholder named "input" using the TF Placeholder name parameter.**
- **Targets placeholder**
- **Learning Rate placeholder**

The `get_inputs` function return the placeholders in the following the tuple (Input, Targets, LearingRate)

✓ **The `get_init_cell` function does the following:**

- **Stacks one or more BasicLSTMCells in a MultiRNNCell using the RNN size `rnn_size`.**
- **Initializes Cell State using the MultiRNNCell's `zero_state` function**
- **The name "initial_state" is applied to the initial state.**
- **The `get_init_cell` function return the cell and initial state in the following tuple (Cell, InitialState)**

Nice! ✅

If you'd like to learn more about how LSTM/RNN networks work, and how to apply them to language data, check out these blog posts:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

✓ **The function `get_embed` applies embedding to `input_data` and returns embedded sequence.**

Awesome!

In future projects, instead of using our simple lookup table to generate embeddings for our text data, we might choose to use word embeddings to achieve better results. While using IDs in our lookup table means we have a long list of numbers that have little relation to each other, word embeddings can map words into a continuous space, where distances have semantic meaning. This can also help with large data sets and computation. You can check out more here:
https://www.tensorflow.org/tutorials/word2vec#motivation_why_learn_word_embeddings http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

✓ **The function `build_rnn` does the following:**

- **Builds the RNN using the `tf.nn.dynamic_rnn`.**
- **Applies the name "final_state" to the final state.**
- **Returns the outputs and final_state state in the following tuple (Outputs, FinalState)**

✓ **The `build_nn` function does the following in order:**

- **Apply embedding to `input_data` using `get_embed` function.**
- **Build RNN using cell using `build_rnn` function.**
- **Apply a fully connected layer with a linear activation and `vocab_size` as the number of outputs.**
- **Return the logits and final state in the following tuple (Logits, FinalState)**

✓ **The `get_batches` function create batches of input and targets using `int_text`. The batches should be a Numpy array of tuples. Each tuple is (batch of input, batch of target).**

- **The first element in the tuple is a single batch of input with the shape [batch size, sequence length]**
- **The second element in the tuple is a single batch of targets with the shape [batch size, sequence length]**

Neural Network 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.**
- **Size of the RNN cells (number of units in the hidden layers) is large enough to fit the data well. Again, no real "best" value.**
- **The sequence length (seq_length) here should be about the size of the length of sentences you want to generate. Should match the structure of the data.**
  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.
  Set show_every_n_batches to the number of batches the neural network should print progress.

✓ **The project gets a loss less than 1.0**

Generate TV Script

✓ **"input:0", "initial_state:0", "final_state:0", and "probs:0" are all returned by `get_tensor_by_name`, in that order, and in a tuple**

✓ **The `pick_word` function predicts the next word correctly.**

Great job!

You added a bit of randomness when choosing the next word. the predictions will be the same each time they are generated and may fall into a loop of the same words.

✓ **The generated script looks similar to the TV script in the dataset.**

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

Awesome work!
As mentioned in the project, we expected to see interesting but nonsensical results. In order to get better results, you'll have to use a smaller vocabulary or get more data. Feel free to experiment with the full dataset here:
https://www.kaggle.com/wcukierski/the-simpsons-by-the-data

DOWNLOAD PROJECT

RETURN TO PATH