PROJECT

Generate TV Scripts

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW | CODE REVIEW | NOTES

Meets Specifications

SHARE YOUR ACCOMPLISHMENT

Hello!

Congratulations on passing the third Deep Learning project

Truly amazing work! I have to admit the best P3 submission I have ever seen.

Really enjoyed checking it along with the reading of the thought process/reasoning behind from the notes you have provided.

There is not much I can add, you went way up above and beyond on what was expected.
Thank you very much for the great submission!

Amazing progress. Keep it up!
Kudos and happy learning

Required Files and Tests

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

iPython Notebook is present.

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

Your code passed the unit tests. Great job!

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)

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

Interesting idea of shortening the tokens. In out case that would not affect the results as we train the model on the word level and the word are converted to a number and the number does not reflect on what the token is. But that is definitely great for the character level model.
Here is some related reading about character level RNN model CLICK, the original code in lua and replication of in tensorflow here

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)

Very nice job here

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

✓ 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)

Flawless implementation of model components

✓ 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]

Nice job here!

As you may know the for loop or any kind of iterations are slow in python, so I am just sharing with you a way it could be done using the **numpy** library. It can drastically decrease the time for computation. Well even though for this project it is not that important it is still nice to align with programming good practices, more over numpy is a lready imported and used.
Here is the way:

```
num_words = len(int_text)
    n_batches = len(int_text)//(batch_size*seq_length)
    x_data = np.array(int_text[:n_batches*batch_size*seq_length])
    y_data = np.array(int_text[1:n_batches*batch_size*seq_length+1])
    x_batches = np.split(x_data.reshape(batch_size,-1),n_batches,1)
    y_batches = np.split(y_data.reshape(batch_size,-1),n_batches,1)
    return np.array(list(zip(x_batches, y_batches)))
```

It produces results different from yours (results it produces do match the example give in the notebook), but still worth checking

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**

Amazing results

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.

Very nice setup

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

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

Please share the results you got from Harry Potter in the review evaluation section if possible. I really wonder what you have got

DOWNLOAD PROJECT

RETURN TO PATH