UDACITY

PROJECT

Generate TV Scripts

A part of the Deep Learning Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

This time, you've implemented this project with near perfection. It's clear you have an excellent understanding of the basics. Keep up the good work!!

Required Files and Tests

✓

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

The project includes the notebook and the required helper files

✓

All the unit tests in project have passed.

Great Job!!

Unit Testing is a reliable method to test the code of all the modules independently before integration. That way we can be more or less certain that our project is free from error at any point. I really encourage you to take to heart this paradigm of developing code for unit testing along with your project code. This would enable clean, confident and fast development. You can read up more on this if you wish.

Though it's also important to remember that unit testing can't catch all the bugs in your code. So your code may have bugs even though all tests pass

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)

Clear and concise.
While working with text it's always a good idea to map each character to an id and vice versa. This plays an important role when we decide to generate novel text.

✓

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

Perfect.

You can find more information about other pre-processing techniques that one might use 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)

Correct.
In Tensorflow, placeholders are the foundations in the computational graph of any neural network.

Often I see students confused between tf.Variable and tf.placeholder. This provides correct usecases for the both of them

✓

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 Job!

✓

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

Yep. Nicely done.

Another way to do this concisely would be

```
return tf.contrib.layers.embed_sequence(input_data, vocab_size, embed_dim)
```

Embeddings are particularly important because they map the words to a numerical representation for the neural network to process.

Word2Vec was a landmark paper on embedding representations. The famous example King - Man + Woman = Queen ! This is a terrific explanation about the mechanics behind them

✓

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)**

Nice!
For your future reference , tensorflow provides a variety of other methods to construct an RNN.

✓

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)**

Impressive!

You've put everything together perfectly.

C Olah and Andrej are two researchers who have explained these concepts wonderfully.

✓

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]

get_batches is difficult to figure out. Great Job!

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.

Whoa that was some choice of parameters! It's evident that you've got some really great intuition.It's great that you took the previous reviewer's advice and decided to keep parameter values in the power of two. This helps tensorflow to computations more efficiently. The only minor critique I would offer is to keep your sequence_length to be somewhere around the dataset average. 5 is bit on the lower side

Great Job!! (I need new superlatives, this is sounding repetitive now. :"D)

The project gets a loss less than 1.0

True That.

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 on using np.random.choice to select the probablity of the words. Throwing in a bit of randomness in the selection process will improve the novelty of the produced script :)

✓

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

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

Having a somewhat unhealthy obsession with the Simpsons, it's almost scary to know that this script has been produced by an RNN. I'm sure this would be even more realistic if you trained this over the entire series.

Google is doing some very exciting work with RNNs to generate art and even music. Check it out.

For the final time Great Job!!

DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ