UDACITY

PROJECT

Generate TV Scripts

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Great work on this project. You have met all the requirements of the project, and your network produces very good results.
Good luck with the nanodegree!

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)

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)

Good. Technically, you didn't have to name targets and learning_rate here explicitly.

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)

Good job on stacking 1 BasicLSTMCell in a MultiRNNCell

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

Good. You could have also written it as: `return tf.contrib.layers.embed_sequence(input_data, vocab_size, embed_dim)`

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)

Great

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)

Good. You remembered to explicitly set activation function to None in contrib.layers.fully_connected.

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]

Great, this was a tricky one.

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 number of epochs is set adequately, as the training loss is low and not improving much with more training.
Batch size is large enough to train efficiently, and it is obviously small enough to fit in memory.
RNN size seems adequate.
The sequence length seems too high. It should be around the size of the length of sentences you want to generate, and be around the average length of sentences in training data.
Learning rate looks fine. Not too large, not too small.

The project gets a loss less than 1.0

Great performance.

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.

You use deterministic strategy for picking words by choosing the words with the highest probability. You should consider implementing random approach (using probabilities parameter), otherwise the predictions will 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.

Looks quite close. All the lines start with character names, and most of the sentences have proper punctuation.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

Student FAQ