UDACITY

PROJECT

Generate TV Scripts

A part of the Deep Learning Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Finally, you have done some great work on this project. This will help you in fields like NLP and continuous image/video processing by understanding their techniques.

Required Files and Tests

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

Great job there. You have included all the necessary files in your submission.

All the unit tests in project have passed.

All the unit test have passed successfully. Also make sure that you will have a look at the problem_unittests.py as it gives you more insights.

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 implementation of the function create_lookup_tables. You have successfully created a dictionary to go from words to id's and another to go from id's to word. They are successfully returned from your method.

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

Code can be a bit more cleaner by creating the token dictionary for all the tokens simulatneously. Nevertheless, it achieves the required result.

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)

Tensor-flow is now the most used framework for deep learning along with keras (for abstract usage). This is also one of the most basic lines of code you will write if you become an deep learning engineer. So congratulations on hitting the first step:)

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 work on the usage of BasicLSTMCells in MultiRNNCell's and creation of cell stages.

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

Great implementation here. I'd like to suggest an another approach that is just as useful as the one implemented here.

I'd like to invite you to take a look at tf.contrib.layers.embed_sequence it can also be used for embedding.

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)**

Great work here. You have successfully applied embedding on the input data and also built a RNN.

One thing I want to note here is that, when using the fully_connected layer from the contrib.layers package of TensorFlow, do not forget that, when you don't specify the activation, ReLu activation is used by default (which is non-linear). In order to use a linear activation, you must explicitly set activation_fn = None which you have done perfectly.

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]**

Batching is one of the most important techniques that should be understood for successful implementation of RNN's.

Great job setting up the number of epochs. With 100, your network trains accurately.
If you have checked for all the conditions mentioned in the table, that is really amazing and appreciable. You have the patience and the hunger to learn.
Coming to the error which you have experienced, it is very nice test case actually.
Hint: Maybe because of a similar size of the two parameters used ?

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.

I see an abnormality here. Did you train the network with rnn_size and embed_dim as 512? If it threw an error according to your table given, how come it trained well with loss going below 1?

Also the batch_size could be a bit higher. Set it to maybe 500 to see how the training loss varies. In your network the training loss sort of remains constant coming to the end.

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.

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

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

DOWNLOAD PROJECT

RETURN TO PATH