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 job and congrats on finishing Project 3! I hope you're enjoying the course and learning some valuable skills. Good luck on your next project

Required Files and Tests

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

Very nice

All the unit tests in project have passed.

Great work

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)

Very nice

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 job

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)

2 layers is great. Good to see students experimenting with more than one LSTM cell.

Here are some great resources for learning more about RNN and LSTM

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

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

Cool. This can be simplified down to just

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

See the docs at https://www.tensorflow.org/api_docs/python/tf/contrib/layers/embed_sequence

Also, here's a great blog post on word embeddings that might further your understanding. http://veredshwartz.blogspot.in/2016/01/representing-words.html

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]

Nice job. `get_batches` is a great function to learn some of the powerful features of numpy. Reducing the number of loops done in Python and vectorization really improves performance. `reshape` and `zip` are very useful here. Check out this implementation:

```
    num_batches = len(int_text) // (batch_size * seq_length)

    xdata = np.array(int_text[:n_batches * batch_size * seq_length])
    ydata = np.array(int_text[1:n_batches * batch_size * seq_length + 1])

    x_batches = np.split(xdata.reshape(batch_size, -1), n_batches, 1)
    y_batches = np.split(ydata.reshape(batch_size, -1), n_batches, 1)
```

```
    batches = np.array(list(zip(y_batches, x_batches)))

    return batches
```

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.

These hyperparameters look really good.

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

This is great!

The `pick_word` function predicts the next word correctly.

Very cool. Picking the top 5 is a great idea.

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

Student FAQ