UDACITY

PROJECT

Generate TV Scripts

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Hi, This is a great submission. You created a RNN that was able to come up with new sentences based on a relatively small training set. This time you trained on words. In the Udacity class room you probably saw the lesson on doing the same thing based on character sequences. This video in which Andrej Karpathy elaborates on the LSTM provides much more insight in what such a character sequence based RNN can do. Especially the examples between 22:30 - 31:15 show in a humoristic way what LSTMs are capable of doing.

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.

Well done, your code runs flawlessly.

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)

A good 'under the hood' implementation. A solution using more abstract Python functionality would go like:

```
    vocab = set(text)
    vocab_to_int = {c: i for i, c in enumerate(vocab)}
    int_to_vocab = dict(enumerate(vocab)
    return 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)

Well done, allowing for variable batch size and sequence length.

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)

Indeed, just one cell does the job here. Great that you also use dropouts.

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

Excellent implementation. An alternative would have been the even shorter:
`return tf.contrib.layers.embed_sequence(input_data, vocab_size, embed_dim)`.

You also might look at this video for some more info on embeddings.

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]

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.

Good choices, leading to a low loss and a nice script with sentences of roughly the same size as those in the training set.

The project gets a loss less than 1.0

Nice value achieved.

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 implementation, causing some variability in the selection of words.

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

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

Great script. Could you already imagine this one being broadcasted on TV?

DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ