UDACITY

Back to Deep Learning Nanodegree

# Generate TV Scripts

REVIEW

CODE REVIEW

HISTORY

## Requires Changes

5 SPECIFICATIONS REQUIRE CHANGES

I really enjoyed reviewing your project. It is truly a remarkable sight that your network is able to generate the script as shown. It is amazing indeed. Good job here :). Just minor edit required.

Great submission!. Keep up the good work.

### Required Files and Tests

**The project submission contains the project notebook, called “dlnd_tv_script_generation.ipynb”.**

You are on the right track, the project contains the required notebook. Great submission!

**All the unit tests in project have passed.**

Great work. Unit testing is one of the most reliable methods to ensure that your code is free from all bugs without getting confused with the interactions with all the other code. But always keep in mind, that unit tests cannot catch every issue in the code. So your code could have bugs even though unit 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)

Fantastic job here! The create_lookup_tables function was implemented and the function succeeded in creating both vocab_to_int and int_to_vocab dictionaries.

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

All the 10 symbols are taken as key and the tokens of those symbols are taken as values into the dictionary.

## 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)

Outstanding job done! You did well setting up TF Placeholders for the Neural Network

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)

The issue is that you have mixed up with rnn_size and num_layers values.

```
lstm = tf.contrib.rnn.BasicLSTMCell(num_layers)
cell = tf.contrib.rnn.MultiRNNCell([lstm] * rnn_size)
```

Instead, you should have the below code

Instead, you should have the below code:

```
def build_cell(lstm_size):
lstm = tf.contrib.rnn.BasicLSTMCell(lstm_size)
return lstm
cell = tf.contrib.rnn.MultiRNNCell([build_cell(rnn_size) for _ in range(nu
m_layers)])
```

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

Good effort. Alternatively you could also return tf.contrib.layers.embed_sequence(input_data, vocab_size, embed_dim). You can read more about this function here:
https://www.tensorflow.org/api_docs/python/tf/contrib/layers/embed_sequence

**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 Implementation on using tf.nn.dynamic_rnn function to build the 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)**

So basically, the fully connecter layer use the relu activation layer by default which is a non linear activation function (read here) as presented below:

```
logits = tf.contrib.layers.fully_connected(outputs, vocab_size, activation
_fn=tf.nn.relu)
```

You must set the activation_fn paramter to None to have a linear activation. Replace your previous line with:

```
logits = tf.contrib.layers.fully_connected(outputs, vocab_size, activation
_fn = None)
```

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]

Awesome! Great implementation here!

## 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.

Generally, you've made good choices for the various hyper-parameters, but a few could be adjusted to better values. This is something that just takes experience...there are generally no hard and fast/definite values, but rather a range based on both past experience and the particular problem at hand.

The number of epochs(10) is on the lower side, you should increase them to 80 to 100 and train the network better and training loss should go down further.
rnn_size: This number of cells in your RNN would not work, but something in the range of 200-300 is more

reasonable for this small project and would likely produce a much better script.
learning_rate: Looks very reasonable. This is something that you need to trade off with num_epochs.

(Optional)
Batch size(should normally be chosen in the sizes of 128, 256, 512 etc, however, nobody is stopping you to use 100) and learning rate are chosen such that the network trains quickly.

**The project gets a loss less than 1.0**

Work on the changes and experiment around hyperparameters to achieve the loss less that 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 pick_word function predicts the next word correctly.

You could also simplify the code

```
idx = np.random.choice(len(int_to_vocab),p=probabilities)
return int_to_vocab[idx]
```

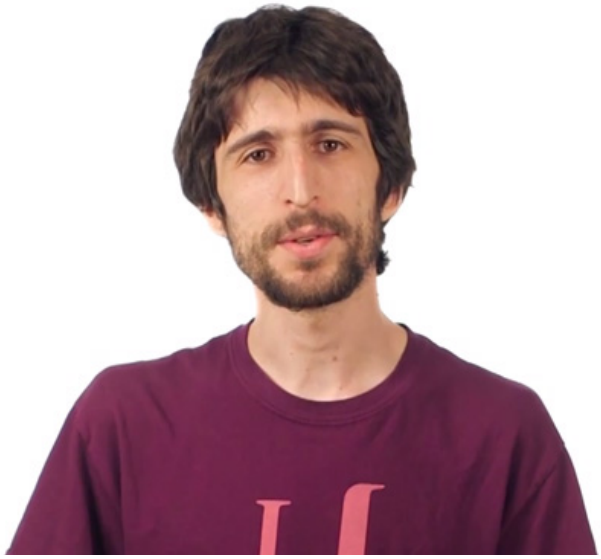
The generated script looks similar to the TV script in the dataset.

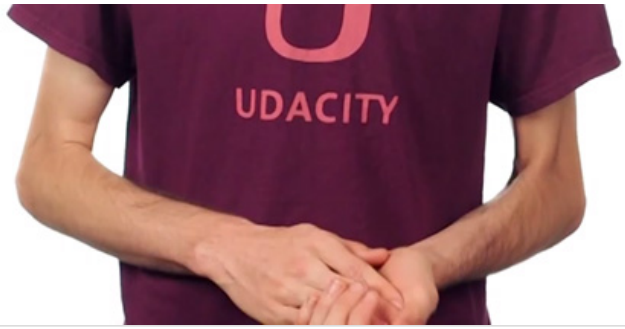
It doesn't have to be grammatically correct or make sense.

Once your training loss is less than 1.0, your generated script will look much more similar to the TV script in the dataset.

RESUBMIT

DOWNLOAD PROJECT






## Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Rate this review