UDACITY

PROJECT

Generate TV Scripts

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

4 SPECIFICATIONS REQUIRE CHANGES

Amazing submission! There are some minor changes that need to be taken care off, other than that you are good to go. Best of luck for your next submission, I am confident that you will do great in your next submission.

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.

Great work. Unit testing is very good practice to ensure that your code is free from all bugs without getting confused and prevent you from wasting a lot of time while debugging minor things. Unit test also help in improving our code standards. For more details read this(https://cgoldberg.github.io/python-unittest-tutorial/) and I really hope that you will continue to use unit testing in every module that you write to keep it clean and speed up your development and for quality development. Unit testing is highly motivated in industries.

It is not always that if you passed unit test your code is okay, there can be some errors.

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)

(Following abstract is from Tensorflow documentation)
TensorFlow programs use a tensor data structure to represent all data -- only tensors are passed between operations in the computation graph. You can think of a TensorFlow tensor as an n-dimensional array or list. A tensor has a static type, a rank, and a shape.

In the TensorFlow system, tensors are described by a unit of dimensionality known as rank. Tensor rank is not the same as matrix rank. Tensor rank (sometimes referred to as order or degree or n-dimension) is the number of dimensions of the tensor. For example, the following tensor (defined as a Python list) has a rank of 2:

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

A rank two tensor is what we typically think of as a matrix, a rank one tensor is a vector. For a rank two tensor you can access any element with the syntax t[i, j]. For a rank three tensor you would need to address an element with t[i, j, k].

Link! that might help you with better understanding of rank in Tensor.

Fantastic! You've perfectly used "None"'s to adequately create a dynamic sized placeholder variables.

get_inputs function is correctly implemented. Seems that you have good grasp of TF Placeholder, I appreciate your efforts.
There is a very good course offered by Stanford University, CS 20SI: Tensorflow for Deep Learning Research. This would help you in further understanding of Tensorflow and its application.

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)

What the difference between an LSTM memory cell and an LSTM layer?
Answer: Link!

Correct code would have been:

```
    lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
    drop = tf.contrib.rnn.DropoutWrapper(lstm, output_keep_prob = 0.5)
    cell = tf.contrib.rnn.MultiRNNCell([drop]*number_of_layers)
    initial_state_ = cell.zero_state(batch_size, tf.float32)
    initial_state= tf.identity(initial_state_, name="initial_state")
    return (cell, initial_state)
```

Number of layers for this project can be kept in range [2-3].

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

Fantastic work! And great use of the dimension to set the variance.

You can also use single line TF wrapper tf.contrib.layers.embed_sequence()
Here's a documentation for TF wrapper tf.contrib.layers.embed_sequence();
https://www.tensorflow.org/api_docs/python/tf/contrib/layers/embed_sequence

We will map each word into a real vector domain, a popular technique when working with text called word embedding. This is a technique where words are encoded as real-valued vectors in a high dimensional space, where the similarity between words in terms of meaning translates to closeness in the vector space.

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 `rnn_size` parameter has been passed by mistake, please ignore it. Previously students were encouraged to put `rnn_size` instead of `embed_size` as `embed_size` was not there.

Good Work!! Taken care off activation function in fully connected layer function.

More on embedding layer:

```
It might be necessary to add an Embedding layer just before the RNN to embed the words into a smaller dimensional space. Since the LSTM
 operates linearly over its input X, then linearly embedding the 1-of-K words to some embedding dimension D first corresponds to the ori
ginal case you tried to get working here, except the matrix is factored through a rank D-dimensional bottleneck. If that makes sense.

Also, it is common to keep track of frequency of all words and discard words that appear, say, less than 5 times in the dataset.
(Words from Andrej karpathy)
```

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]

Good work!!

Code for good use of Numpy Library and more pythonic.

```
n_batches = int(len(int_text) / (batch_size * seq_length))

# Drop the last few characters to make only full batches
xdata = np.array(int_text[: n_batches * batch_size * seq_length])
ydata = np.array(int_text[1: n_batches * batch_size * seq_length + 1])
ydata[-1] = xdata[0]

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

#print(np.array(list(zip(x_batches, y_batches))))

return np.array(list(zip(x_batches, y_batches)))
```

If there is still some doubt, this video will definitely help you better understand how batching operates.

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.

- Chosen `rnn_size` is small, it would be hard for network to learn something good. First try increasing and see if your model converges better and has a lower training loss or not. Consider changing according to your system configuration. (Please read tips also)

Some Tips

The values of hyper parameters should be power of 2. Tensorflow optimizes our computation if we do so.

The `seq_length` should be kept similar to average size of sentence or say according to the structure of our data, this would help in getting better results. Link: http://stats.stackexchange.com/questions/158834/what-is-a-feasible-sequence-length-for-an-rnn-to-model (for this project 12-15 is a desirable range)

Select a learning rate such the model converges well with minimum oscillations/spikes (where the training loss mostly decreases and doesn't "jitter around" in value).

Some links that would help in fine tune model.
http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html

- **rnn_size** : For each LSTM cell that we initialise, we need to supply a value for the hidden dimension(`rnn_size`), or as some people like to call it, the number of units in the LSTM cell. The value of it is it up to you, too high a value may lead to overfitting or a very low value may yield extremely poor results.

The project gets a loss less than 1.0

We need 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.

Good Work!! correctly implemented this function without any bug.
Really appreciate the use of randomness when choosing the next word.
If you don't have used randomness, the predictions would have 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.

Please work upon above suggestions and then we will evaluate this point.

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

Student FAQ Reviewer Agreement