PROJECT

Generate TV Scripts

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW | CODE REVIEW | NOTES

Meets Specifications

SHARE YOUR ACCOMPLISHMENT

Excellent job! You seem to have understood theoretically and in practice (coding) how RNNs work for this kind of task.
If you are curious keep playing around with more data or improving parameters.
As an extra bonus:

- This tutorial shows how to use RNNs for speech recognition https://svds.com/tensorflow-rnn-tutorial/
- Check this article: https://www.wired.com/story/a-sons-race-to-give-his-dying-father-artificial-immortality/

Hope you enjoy it
Keep up with the good work!

Required Files and Tests

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

Alright, file submitted correctly!

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

Achievement unlocked! All tests passed successfully.

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! Good use of `Counter`, list comprehension, `enumerate`, and `dict`, to generate both lookup tables.

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

Good job completing the assigned task

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)

TF Placeholders created correctly, bidimensional placeholders for Input and Targets, and scalar for Learning Rate.

✓ **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)**

Excellent, although you could try stacking two or more `BasicLSTMCells` in a `MultiRNNCell` and compare results/performance. Initial state correctly initialized to zeros and "initial_state" name applied to it.

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

Good job! Also, notice that `tf.contrib.layers.embed_sequence` is the most straightforward way to complete this part.
You can read more about its options 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)**

Parameters correctly assigned to `tf.nn.dynamic_rnn` and name correctly applied.

✓ **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)**

You built the RNN correctly, storing its outputs and final state. For `tf.contrib.layers.fully_connected`, the parameter activation_function should be set as `None` to maintain a linear activation as required.

✓ **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 job creating the batches! Also, notice that you could use `numpy` functions such as `split` and `reshape` to avoid using `for` loops (also take a look at python's native `zip`)

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.

There are no hard values for most of the parameters, the most important thing is that your network trains in a reasonable amount of time (given that we are working on a small dataset - hint: if your training takes more than 3-5 minutes on GPU to achieve a training loss lower than 1.0, you are doing it wrong)
The number of epochs shouldn't be too high that it keeps training with no additional gain in accuracy (your training results seem ok). Batch size depends on your GPU memory, as large as you can, but small enough to fit on it. Same with the size of the RNN cells.
On the other hand, the sequence length depends on the structure of your data. Should be about the size of the length of the sentences (hint, hint)
The learning rate as you already should know, should find a balance: large enough to avoid an infinite training, but not so large that it won't converge.

A little suggestion to consider: As a rule of thumb it's a good idea to use an rnn size twice the size of the embedding dimension. So you could try using 512 and 256 respectively.

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

Excellent! You loss stabilizes below 1.0 which is the expected metric for this project.
Congratulations!

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**

Ok! Tensors obtained using the names given in order.

✓ **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.

Excellent job!

DOWNLOAD PROJECT

RETURN TO PATH