UDACITY

PROJECT

Generate TV Scripts

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Amazing submission! 👏 You've gotten a full fledged RNN to function correctly. And more importantly, you just generated a small Simpson's episode all using a mini AI! That's truly impressive.

What's more, I loved your concise coding style and very clever use of pre-built libraries to keep you code clean. This is a great start and with RNN's the limits are only your imagination. I hope that this project has inspired you to look at more data-sets (Kaggle is your friend here) and try your own awesome projects.

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 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. If you are interested, you can read up more and I hope that you will continue to use unit testing in every module that you write to keep it clean and speed up your development.

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)

Awesome work! 👏 And great use of counters to quickly implement this. You've a good grasp of data structures and that's going to come in very handy.

✓

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)

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

Truly awesome! You've

1. Stacked 3 😮 RNNs cell for a more complex behaviour. Which is fantastic.
2. You've added dropout layer to curb over-fitting. This is great work!!
3. Initialised the cell state correctly.
4. Used the identity function to name the state (which is very clever) 👏🏽

✓

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

Great! Did you know that tensorflow has a handy one line API that will do this for you `tf.contrib.layers.embed_sequence(input_data, vocab_size, embed_dim)`

✓

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)

Fantastic! And great use of the contrib layer's fully connected module! You could also hand craft this part if you desire.

✓

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]

This is tricky, so it's great that you've gotten it perfect! You use the i+1th word as the expected output for the ith word. 👏🏽

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.

Awesome selection of all hyper parameters. And it shows in the loss that you've achieved.

Suggestion

1. Try to add comments in your selection that discusses your thoughts on why you selected these parameters and how you came about these values. It really helps a reader.
2. 3 RNN's each of 1024 size is a really big model. You can try to see if a smaller model achieves the same performances as yours done (which should speed up training)

✓

The project gets a loss less than 1.0

Fantastic! Absolutely no comments, the loss you've achieved is phenomenal. You should be really proud of your work here!

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.

Fantastic work here picking out a random word based on it's probability distribution. I can see that you are trying to pick the top 10 words after re-normalising the probability, but probability itself would automatically take care of this. The less probable words would anyway come only very infrequently and when they do come, it makes the model sound more human. So in my opinion, I would simply pass the output probabilities into random.choice and get the predicted word. But this is debatable 😄

✓

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

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

Crazy isn't it! And that's the power of RNNs. And like the problem states, although it doesn't give us semantically correct scripts, it's still terrific that it gets most of the tone right! But you can see that sometimes, it starts a brace and then forgets to end it, or that it adds a question mark to a sentence that's not really a question, etc... With a lot more data to train, you'll get much better results.

Hope that this has inspired you to generate awesome stuff using RNNs. You can look at Magenta (https://magenta.tensorflow.org/welcome-to-magenta) for some really awesome stuff that Google is doing in trying to generate art (music, sketches) using deep neural nets.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

Student FAQ