UDACITY

< Back to Deep Learning

Generate TV Scripts

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Hey there!
Awesome implementation of the TV scripts generation. I have passed the review points here although please implement the suggestions given to get an even better result. Other than that, it is a pretty good project implementation. Kudos!!
And finally stay hungry for knowledge and never give up!!

Required Files and Tests

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

Contains the required files for check.

All the unit tests in project have passed.

The unit tests provide some very insightful information on analyzing the correctness of the code you have written in terms of analyzing a few basic things like shape, kind of variable being used etc. Do have a good look at it as it would be useful in the future too

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)

Concise and simple.
Great work on implementing the create_lookup_table. This step is extremely important as we need to convert every word character in the text input to numbers for processing to be done by our computer and to convert the numbers (outputs after computation) to text back again.

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

This too is crucial step in our pre-processing as it helps in improving the result from our model.

P.S: Always remember, your output efficiency depends on how good your input data is. So we usually spend 70% of our time cleaning the data in real world.

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)

Perfectly done. Do try to get a good grasp on tensorflow as you will now use more of it soon.

I would suggest you to look up sentdex's tutorials. Has some of the best tutorials for library usage for matplotlib, pandas and also for ML/DL related topics. He has explained the usage of a few basic tensorflow concepts too in the end of his ML tutorials

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)

Very well done here. But try implementing the dropout for this with a value greater than 0.6 as that is affecting your netwroks ability to learn which we can tell from the way the loss is stabilizing just under 1.0 value. Try using 0.9 or at maximum 0.8 would be a good value for the dropout here. They are a very helpful method to overcome the over-fitting problem in neural networks. Do try to play around with the hyper parameters, which will help you with understanding the effects of these parameters on the model which will be of significant knowledge from my point of view.

Also increase the lstm_layers to 2 as it would make your network more sophisticated and a complex one which will be able to increase the level of the model being able to mimic human level interactions.

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

Great implementation here. I'd like to suggest an alternative approach that is just as useful as the one implemented here.

Suggestion

I'd like you to take a look at tf.contrib.layers.embed_sequence it can also be used for embedding.

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)

Perfect implementation.

Please note: Tensorflow provides 2 RNN components:
• tf.nn.rnn
• tf.nn.dynamic_rnn

1. tf.nn.rnn is used for applications where there is a fixed length unrolled RNN.
2. The tf.nn.dynamic_rnn uses a tf.While loop to dynamically construct the graph at runtime.

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)

Feedback:  Excellent work, here. Very straight-forward and easy to understand!

Insight:  After all the emphasis on non-linear activation functions in this course, you might wonder why you have been asked to use a linear activation function in this project. Neural networks often use non-linear activation functions (such as sigmoid or RELU) in hidden layers but may use a linear activation function in the final output layer. A linear activation in the final layer scales the outputs continuously across the full range of values and is the best way to compare relative values of the various outputs. In this script-writing RNN, you are not trying to identify a single-best word to select (a RELU or other non-

linear activation function would be good for this purpose). Rather, you want the output to represent the relative probabilities of each potential word to be selected – so a linear activation is appropriate here.

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]**

Batching is the most trickiest part in any implementation. Very good implementation of it with the code.

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.

Well like i had mentioned before, please try tuning the hyper-parameters here and play around to observe how a model behaves with changing parameters. The hyper parameters look good for this network though i would like you to try a few changes that i suggest:

1. Try a lower learning rate of 0.001 as that would help your network move towards a global convergence.

The other hyper parameters look well tuned to the network you are building.

The project gets a loss less than 1.0

0.352 is a low training loss but can be further reduced significantly with the above suggested changes. Do try them out👍

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 a word function implemented here returns the word with the highest probability(np.argmax(probabilities)). This in turn makes that, when you try to predict a script, it'll always generate the same script depending on what the first word is. I mark it as meet specifications mainly because it didn't affect the generated script too much. Please, keep in mind that the aim is to mimic the human aspect of the script as much as possible.

Suggestions

What you could do is to pick a word randomly based on it's probability distribution. Thus, after enough training, the model will be giving out slightly different scripts each time, making it look very human. All this is to help reduce the risks of predictions falling into a loop of the same words. Please take a look at numpy's random.choice()

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

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

Well the quality can definitely be increased with the improvements suggested above. However it is sufficient to pass the rubric points of the project. Congratulations on completing the project👍

DOWNLOAD PROJECT

RETURN TO PATH