UDACITY

‹ Return to Classroom

Generate TV Scripts

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Good job on the project. I hope you had fun implementing RNNs. Let's see how you do on the GANs. All the best!

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.

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)

Good work, here. :+1:

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)

Good work, here. :+1:
Naming each tensor is good practice, as it helps in debugging.

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)

Good work!

You have used a drop_out layer in the architecture, the drop out layer is not really needed as there is no performance metric to measure overfitting (i.e no validation set). We are mostly concerned with generating text for our project.
Try making the keep_prob=1 and the network will converge faster and loss will be lower.

To learn more about LSTM
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

A simpler way to stack up multiple LSTM, for deep learning
lstm_cells = [
tf.contrib.rnn.BasicLSTMCell(rnn_size) for i in range(0, 1)]

The function `get_embed` applies embedding to `input_data` and returns embedded 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)

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)

Good work putting the previously defined functions together 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]

Great implementation!
You have not a used a loop to achieve the batching, most of the student generally go for loops.

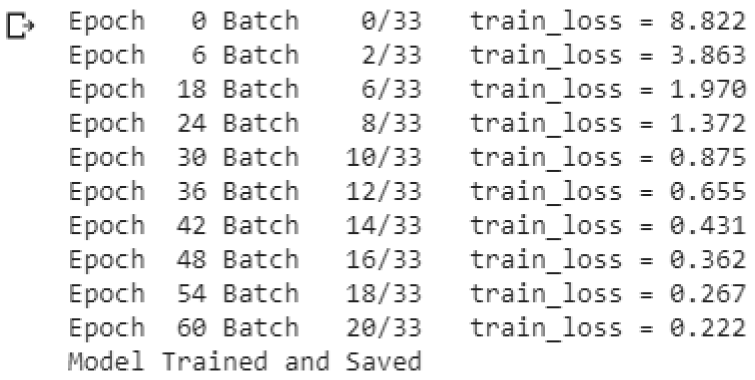
Neural Network Training

- 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.

Perfect Job.

- The number of epochs is perfect
- Batch size(128) is again on the spot.

- The sequence length (16) is perfect for the length of the sentences we want to generate - at the start of the exercise you calculated the average line length to be 11.5.
- The embedding size(256) and RNN(512) size are big enough to handle the complexity of the sentences.
- The learning rate 0.003 is perfect for the project and hyperparameters you have used
  Great work on the hyperparameters. You have nailed it on the first attempt.
  TIP: Try making keep_prob=1 and the error will reduce more(see below)



```
Epoch   0 Batch    0/33   train_loss = 8.822
Epoch   6 Batch    2/33   train_loss = 3.863
Epoch  18 Batch    6/33   train_loss = 1.970
Epoch  24 Batch    8/33   train_loss = 1.372
Epoch  30 Batch   10/33   train_loss = 0.875
Epoch  36 Batch   12/33   train_loss = 0.655
Epoch  42 Batch   14/33   train_loss = 0.431
Epoch  48 Batch   16/33   train_loss = 0.362
Epoch  54 Batch   18/33   train_loss = 0.267
Epoch  60 Batch   20/33   train_loss = 0.222
Model Trained and Saved
```

The project gets a loss less than 1.0

Your loss is below 1.0, Awesome work

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 adding randomness, This will ensure that an unique scripts is generated everytime

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

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

Seems Done! 😊

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

DOWNLOAD PROJECT

RETURN TO PATH

Rate this project

START