UDACITY

‹ Return to Classroom

Generate TV Scripts

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

🎉 Great work on the project! Nice job tuning the hyperparameters to get the losses low enough. As you can see, LSTM-generated text cannot create meaningful long-term text yet, but is an active area of research. I understand you completed the pytorch version even though this rubric is for the keras version, so I adapted it (the main big thing was the loss value is different - 1.0 for the keras version and 3.5 for pytorch).

Other libraries for text generation

As with most things in Python, someone has already created libraries to do a similar thing as in this project, but in a few lines of code. Here's one library (textgenrnn) which does just that, although with TensorFlow and Keras. texar is another one that is a bit more general-purpose.

Best of luck in the rest of the nanodegree!

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 with this one, but the counter you used actually has some extra overhead we don't need, so you could just use a set instead: `set(text)`. The sorting is also unnecessary. You also only need to enumerate once, if you create both dicts in a for loop. All of these things will save some compute power/time.

I would do it like this:

```
vocab = set(text)
vocab_to_int, int_to_vocab = {}, {}
for i, w in enumerate(vocab):
    vocab_to_int[w] = i
    int_to_vocab[i] = w

return (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)

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)

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)

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]

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.

You got it! Everything is in a good range for the most part.

LR

Your learning rate is good at 0.001; this usually has to be kind of small for LSTMs. You could also use learning rate schedules and decay to optimize the learning rate a bit better.

Embed dims

Your embed dims are slightly big at 100, but work just fine. Google's news word vectors, the GloVe vectors, and other word vectors are usually in the range 50 to 300, so I like to use embed_dims in that range for words typically. However, Google's blog post here suggests using the 4th root of the vocab size or number of categories, which in this case ends up being around 15. And you can get about the same accuracy as a higher embedding dim with only 15 as the embedding dimension.

More efficient way to search hyperparameter space

You could also use something like Bayesian search to optimize your hyperpamaters. However, having a good idea of starting points for the hyperparameters helps a lot in shortening search times and costs.

The project gets a loss less than 1.0

The torch loss spec is under 3.5, so 2.93 is actually quite good!

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

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

DOWNLOAD PROJECT

RETURN TO PATH