UDACITY

PROJECT

Generate TV Scripts

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Woohoo! Nice work. Here's a discussion on LSTMs for natural language generation if you're interested in reading more.

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)

Great job! A tiny improvement would be to loop through the dict creation in a for loop, and create both dicts in the loop.

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.

Great job! You could also do it like `tf.contrib.layers.embed_sequence(input_data, vocab_size, embed_dim)`
If the embedding_lookup function seems confusing, check out this.

Also, here's a video on word embeddings, and here's a nice blog post, if you want to learn more.

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]

That works! You could also do it similarly to Mat's example (here's the newest version which is different, and here's the version I used to solve this part):

```
n_batches = int(len(int_text) / (batch_size * seq_length))

# Drop the last few characters to make only full batches
xdata = np.array(int_text[: n_batches * batch_size * seq_length])
ydata = np.array(int_text[1: n_batches * batch_size * seq_length + 1])
ydata[-1] = xdata[0]

x_batches = np.split(xdata.reshape(batch_size, -1), n_batches, 1)
y_batches = np.split(ydata.reshape(batch_size, -1), n_batches, 1)

#print(np.array(list(zip(x_batches, y_batches))))

return np.array(list(zip(x_batches, y_batches)))
```

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 example, 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.

Your seq length is good; we base this off the avg sentence length (printed at the top) of 11 words/line. So you've got about 1 line as your seq_length, which is in a good range (0.5-5 lines range).

The project gets a loss less than 1.0

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

Student FAQ