UDACITY

‹ Return to "Deep Learning" in the classroom

DISCUSS ON STUDENT HUB

Generate TV Scripts

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations 🎉

- Your submission reveals that you have made an **excellent effort** in finishing this project. It is an important milestone in learning about RNNs
- Very good hyperparameters and loss . It is great that you have got everything right in first review 👍
- I wish you all the best for next adventures 🚀

- Nice Read: (Colah's Blog) http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Nice Read: (Andrej Karpathy) : http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- Nice Read: (Rohan Kapur) https://ayearofai.com/rohan-lenny-3-recurrent-neural-networks-10300100899b

Keep up the good work 👍 Stay Udacious

All Required Files and Tests

The project submission contains the project notebook, called "dlnd_tv_script_generation.ipynb".
All required files are present 👍

- It is recommended to export your conda environment into environment.yaml file. command **conda env export -f environment.yaml**, so that you can recreate your conda environment later.
- While submitting this to any version control system like Github, make sure to include helper, data and environment files and exclude and temp files. It will help you in future if you want to re-execute it. Some guideline for best practice.

All the unit tests in project have passed.

Pre-processing Data

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 as a tuple (vocab_to_int, int_to_vocab).

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

Simply you can implement it like this :

```
return {
       '.'  : '||period||',
       ','  : '||comma||',
       '"'  : '||quotationmark||',
       ';'  : '||semicolon||',
       '!'  : '||exclamationmark||',
       '?'  : '||questionmark||',
       '('  : '||leftparentheses',
       ')'  : '||rightparentheses',
       '--' : '||doubledash||',
       '\n' : '||return||'
  }
```

Batching Data

The function `batch_data` breaks up word id's into the appropriate sequence lengths, such that only complete sequence lengths are constructed.

Good Job 👏

- breaks up word id's into the appropriate sequence lengths

In the function `batch_data`, data is converted into Tensors and formatted with TensorDataset.

Finally, `batch_data` returns a DataLoader for the batched training data.

Build the RNN

The RNN class has complete `__init__`, `forward`, and `init_hidden` functions.

👍

- `__init__`, `forward` and `init_hidden` functions are complete

The RNN must include an LSTM or GRU and at least one fully-connected layer. The LSTM/GRU should be correctly initialized, where relevant.

The ideal structure is as follows:

- Embedding layer (nn.Embedding) before the LSTM or GRU layer.
- The fully-connected layer comes at the end to get our desired number of outputs.
- Extra marks for not using a dropout after LSTM and before FC layer, as the drop out is already incorporated in the LSTMs, A lot of students will add it and then end up finding convergence difficult
- You can try to add more than one fc:

```
# init
self.fcc=nn.Linear(self.hidden_dim, self.hidden_dim)
self.fcc2=nn.Linear(self.hidden_dim,self.output_size)
....
# forward
output,hidden=self.lstm(embedded,hidden)
lstm_output = output.contiguous().view(-1, self.hidden_dim)
```

```
output= self.fcc(output)
output=self.dropout(output)
output=self.fcc2(output)
```

RNN 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.**
- **Embedding dimension, significantly smaller than the size of the vocabulary, if you choose to use word embeddings**
- **Hidden dimension (number of units in the hidden layers of the RNN) is large enough to fit the data well. Again, no real "best" value.**
- **n_layers (number of layers in a GRU/LSTM) is between 1-3.**
- **The sequence length (seq_length) here should be about the size of the length of sentences you want to look at before you generate the next word.**
- **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.**

🚀

- Enough epochs to get near a minimum in the training loss. Do not hesitate to use a value as big as needed till the loss the improving
- Batch size is large enough to train efficiently
- Sequence length is about the size of the length of sentences we want to generate
- Size of embedding is in the range of [200-300]
- Learning rate seems good based on other hyper parameter
- simply write `output_size=vocab_size`

> Your efforts shows that you have really have executed it again and again to get an optimized value 🔥

The printed loss should decrease during training. The loss should reach a value lower than 3.5.

🏁 excellent !

There is a provided answer that justifies choices about model size, sequence length, and other parameters.

Generate TV Script

The generated script can vary in length, and should look structurally similar to the TV script in the dataset.

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

well generated fun script ! 😊

DOWNLOAD PROJECT

RETURN TO PATH