Generate TV Scripts

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

Dear Student,

Great work 👏 for completing the assignment.
I really like some implemenetation you have made such as optimized code for `create_lookup_table` function and optimized hyper-parameters mainly sequance_length, batch size and epochs.

I really appricieate your efforts to complete the assignement. Happy learing 📚

All Required Files and Tests

✓ The project submission contains the project notebook, called "dlnd_tv_script_generation.ipynb".

Good 👏 the notebook is available in the submission.
✅ noteook file

✓ All the unit tests in project have passed.

superb all unit tests are passing 🔥

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).

Perfect 👌
The implementation is one of the optimized ones where you have used counter and dictionary comprehension.

```
from collections import Counter
def create_lookup_tables(text):
    """
    Create lookup tables for vocabulary
    :param text: The text of tv scripts split into words
    :return: A tuple of dicts (vocab_to_int, int_to_vocab)
    """
    # TODO: Implement Function
    vocabs = Counter(text)

    vocabs_sorted = sorted(vocabs, key=vocabs.get, reverse = False)
    vocab_to_int = {word: i for i, word in enumerate(vocabs_sorted)}
    int_to_vocab = {i: word for i, word in enumerate(vocabs_sorted)}

    # return tuple
    return (vocab_to_int, int_to_vocab)
```

in the future, you can also use predefined word embedding for better results.
please refer the link for more about word embedding

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

great 👍 all tokens are considered in the function.

- Period (.)
- Comma (,)
- Quotation Mark (")
- Semicolon (;)
- Exclamation mark (!)
- Question mark (?)
- Left Parentheses (( )
- Right Parentheses ( ))
- Dash (-)
- Return (\n)

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 👏
Here, instead of using lists, you can directly use NumPy array to not converting datatype from list to array and array to tensor.

```
def batch_data(words, sequence_length, batch_size):

    features = np.array([words[idx:idx+sequence_length] for idx, word in enumerate(words[0:-sequence_length])])
    targets = np.array([words[idx+sequence_length] for idx, word in enumerate(words[0:-sequence_length])])
    data = TensorDataset(torch.from_numpy(features), torch.from_numpy(targets))
    data_loader = DataLoader(data,batch_size=batch_size)

    return data_loader
```

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

Nice 👌

```
data = TensorDataset(torch.from_numpy(np.asarray(feature_tensors)),
                     torch.from_numpy(np.asarray(target_tensors)))
```

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

Nice 👌

```
data_loader = DataLoader(data, shuffle=False, batch_size=batch_size)
```

Build the RNN

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

Good👏
for more explanation about suitable architecture for text generation here

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

Good that you have chosen LSTM model.
click here to undetstand difference between LSTM and GRU.

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.

all hyperparameters seems good.
you can also refer the link to improve the hyperparameters

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

perfect 🔥 the loss value is lower than 3.5.
`Epoch: 10/10 Loss: 3.1911292787790297`

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

good efforts to tune hyper-parameters. 👏
I really like that you have done hyper-parameter tunings such as changing sequence length, hidden_dim and n_layers.

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.

the generated script looks good

DOWNLOAD PROJECT

RETURN TO PATH