UDACITY

‹ Return to 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,especially the batching, model architecture and hyperparameters.
- Very good hyperparameters and decreasing cross entropy loss. It is great that you have got everything right in first review 👍
- Please go through the additional suggestions in the rubric below.
- I wish you all the best for next adventures 🚀

Few references to explore:

- Colah's Blog A clear explanation of LSTMs you want to look at to understand it more.
- Andrej Karpathy Andrej is director of AI and AutoPilot Vision at Tesla, this link explains the unreasonable effectiveness of RNN
- Rohan Kapur , This link is an overall explanation of RNNs that you may find useful and insightful.

Few project/research that uses RNNs:

- Generate your own Bach music using DeepBach. https://arxiv.org/pdf/1612.01010.pdf
- Generate and customize realistic images https://heartbeat.fritz.ai/stylegans-use-machine-learning-to-generate-and-customize-realistic-images-c943388dc672
- Art Generation : https://aiartists.org/ai-generated-art-tools

(feel free to reach out to mentors in the knowledge forum regarding any question or confusion)

Keep up the good work 👍 Stay Udacious 🎶

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 👍

Bonus tips:

- 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.

Well Done ! 👍 Donald Knuth (a famous computer science pioneer) once famously said

> "Beware of bugs in the above code; I have only proved it correct, not tried it."

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).

Good job! 👏

- The Counter is a convenient way to get the information needed for that approach, but it 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 **inside a single for loop**. All of these things will save some compute power/time.
  Alternatively, it could be implemented 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.

👍

- Converting each punctuation into explicit token is very handy when working with RNNs.
- All 10 entries are present
- Do read up this link to understand what other pre-processing steps are carried out before feeding text data to RNNs.
- Good list of Tokenization Options https://graphics.cs.wisc.edu/WP/vep/tokenization/

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.

It is recommended to add explanatory comments in between, look at following implementation :

```
    # get number of targets we can make
    n_targets = len(words) - sequence_length
    # initialize feature and target
    feature, target = [], []
    # loop through all targets we can make
    for i in range(n_targets):
        x = words[i : i+sequence_length]    # get some words from the given list
        y = words[i+sequence_length]        # get the next word to be the target
        feature.append(x)
```

```
        target.append(y)

    feature_tensor, target_tensor = torch.from_numpy(np.array(feature)), torch.fr
om_numpy(np.array(target))
    # create data
    data = TensorDataset(feature_tensor, target_tensor)
    # create dataloader
    dataloader = DataLoader(data, batch_size=batch_size, shuffle=True)
     # return a dataloader
    return dataloader
```

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

The unit test output tensor verifies the implementation 👍 `shuffle=True` will add randomness in the training sequences.

Build the RNN

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

👍

- `__init__`, `forward` and `init_hidden` functions are complete, a good model architecture
- it is recommended to remove unnecessary/unused methods from the code.

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.
- It is also **recommended to not use 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 and use dropout between two 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)
```

- check this Illustrated guide to lstms and gru- a step by step explanation

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. Taking into account that there are approximately an average of 11.504 words per line and 15.248 sentences in each scene
- Size of embedding is in the range of [200-300]
- Learning rate seems good based on other hyper parameter
- Hidden Dimension can be selected like this https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046

```
hidden_dim = 5*int(len(train_loader.dataset) / (embedding_dim + output_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 decreasing loss ..

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

detailed answer..

> the act of elaborating your approach often leads to a deeper understanding of the material 😊

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! 👏🏻 almost all lines are making sense and good grammatical structure ..

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START