Session 42

Project Question

Session 42: Project

Table of Contents

1. Introduction

This assignment will help you to consolidate the concepts learnt in the session.

2. Problem Statement

TV Script Generation

In this project, you'll generate your own Simpsons TV scripts using RNNs. You'll be using part of the Simpsons dataset of scripts from 27 seasons. The Neural Network you'll build will generate a new TV script for a scene at Moe's Tavern.

Get the Data

The data is already provided for you. You'll be using a subset of the original dataset. It consists of only the scenes in Moe's Tavern. This doesn't include other versions of the tavern, like "Moe's Cavern", "Flaming Moe's", "Uncle Moe's Family Feed-Bag", etc..

The following are some helper functions students can use in their code

In [1]:

```
import os
import pickle


def load_data(path):
    """
    Load Dataset from File
    """
    input_file = os.path.join(path)
    with open(input_file, "r") as f:
        data = f.read()

    return data


def preprocess_and_save_data(dataset_path, token_lookup, create_lookup_tables):
```

```
    """
    Preprocess Text Data
    """
    text = load_data(dataset_path)

    # Ignore notice, since we don't use it for analysing the data
    text = text[81:]

    token_dict = token_lookup()
    for key, token in token_dict.items():
        text = text.replace(key, ' {} '.format(token))

    text = text.lower()
    text = text.split()

    vocab_to_int, int_to_vocab = create_lookup_tables(text)
    int_text = [vocab_to_int[word] for word in text]
    pickle.dump((int_text, vocab_to_int, int_to_vocab, token_dict), open('preprocess.p', 'wb'))


def load_preprocess():
    """
    Load the Preprocessed Training data and return them in batches of <batch_size> or less
    """
    return pickle.load(open('preprocess.p', mode='rb'))


def save_params(params):
    """
    Save parameters to file
    """
    pickle.dump(params, open('params.p', 'wb'))


def load_params():
    """
    Load parameters from file
    """
    return pickle.load(open('params.p', mode='rb'))
```

In [3]:

```
# Load data

data_dir = './data/simpsons/moes_tavern_lines.txt'
text = load_data(data_dir)
# Ignore notice, since we don't use it for analysing the data
text = text[81:]
```

Explore the Data

Play around with view_sentence_range to view different parts of the data.

In [4]:

```
view_sentence_range = (0, 10)

import numpy as np

print('Dataset Stats')
print('Roughly the number of unique words: {}'.format(len({word: None for word in text.split()})))
scenes = text.split('\n\n')
print('Number of scenes: {}'.format(len(scenes)))
sentence_count_scene = [scene.count('\n') for scene in scenes]
print('Average number of sentences in each scene: {}'.format(np.average(sentence_count_scene)))

sentences = [sentence for scene in scenes for sentence in scene.split('\n')]
print('Number of lines: {}'.format(len(sentences)))
word_count_sentence = [len(sentence.split()) for sentence in sentences]
print('Average number of words in each line: {}'.format(np.average(word_count_sentence)))

print()
print('The sentences {} to {}:'.format(*view_sentence_range))
print('\n'.join(text.split('\n')[view_sentence_range[0]:view_sentence_range[1]]))
```

```
Dataset Stats
Roughly the number of unique words: 11492
Number of scenes: 262
Average number of sentences in each scene: 15.248091603053435
Number of lines: 4257
Average number of words in each line: 11.50434578341555
```

The sentences 0 to 10:
Moe_Szyslak: (INTO PHONE) Moe's Tavern. Where the elite meet to drink.
Bart_Simpson: Eh, yeah, hello, is Mike there? Last name, Rotch.
Moe_Szyslak: (INTO PHONE) Hold on, I'll check. (TO BARFLIES) Mike Rotch. Mike Rotch. Hey, has anybody seen Mike Rotch, lately?
Moe_Szyslak: (INTO PHONE) Listen you little puke. One of these days I'm gonna catch you, and I'm gonna carve my name on your back with an ice pick.
Moe_Szyslak: What's the matter Homer? You're not your normal effervescent self.
Homer_Simpson: I got my problems, Moe. Give me another one.
Moe_Szyslak: Homer, hey, you should not drink to forget your problems.
Barney_Gumble: Yeah, you should only drink to enhance your social skills.

Implement Preprocessing Functions

The first thing to do to any dataset is preprocessing. Implement the following preprocessing functions below:

- Lookup Table
- Tokenize Punctuation

Lookup Table

To create a word embedding, you first need to transform the words to ids. In this function, 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

Return these dictionaries in the following tuple (vocab_to_int, int_to_vocab)

Solution

In [5]:

```
import numpy as np

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)
    """
    vocab = set(text)
    vocab_to_int = {c: i for i, c in enumerate(vocab)}
    int_to_vocab = dict(enumerate(vocab))
    return vocab_to_int, int_to_vocab
```

Tokenize Punctuation

We'll be splitting the script into a word array using spaces as delimiters. However, punctuations like periods and exclamation marks make it hard for the neural network to distinguish between the word "bye" and "bye!".

Implement the function token_lookup to return a dict that will be used to tokenize symbols like "!" into "||Exclamation_Mark||". Create a dictionary for the following symbols where the symbol is the key and value is the token:

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

This dictionary will be used to token the symbols and add the delimiter (space) around it. This separates the symbols as it's own word, making it easier for the neural network to predict on the next word. Make sure you don't use a token that could be confused as a word. Instead of using the token "dash", try using something like "||dash||".

In [6]:

```
def token_lookup():
    """
    Generate a dict to turn punctuation into a token.
    :return: Tokenize dictionary where the key is the punctuation and the value is the token
    """
    dict = {'.':'||Period||',
            ',':'||Comma||',
            '"':'||Quotation_Mark||',
            ';':'||Semicolon||',
            '!':'||Exclamation_mark||',
            '?':'||Question_mark||',
            '(':'||Left_Parentheses||',
            ')':'||Right_Parentheses||',
            '--':'||Dash||',
            '\n':'||Return||',
           }
    return dict
```

Preprocess all the data and save it

Running the code cell below will preprocess all the data and save it to file.

In [8]:

```
preprocess_and_save_data(data_dir, token_lookup, create_lookup_tables)
```

In [9]:

```
import numpy as np


int_text, vocab_to_int, int_to_vocab, token_dict = load_preprocess()
```

Input

Implement the get_inputs() function to create TF Placeholders for the Neural Network. It should create the following placeholders:

- Input text placeholder named "input" using the TF Placeholder name parameter.
- Targets placeholder
- Learning Rate placeholder

Return the placeholders in the following the tuple (Input, Targets, LearingRate)

In [10]:

```
def get_inputs():
    """
    Create TF Placeholders for input, targets, and learning rate.
    :return: Tuple (input, targets, learning rate)
    """
    Input = tf.placeholder(tf.int32, [None, None] , name='input')
    Targets = tf.placeholder(tf.int32, [None, None])
    LearningRate = tf.placeholder(tf.float32)
    return Input, Targets, LearningRate
```

Build RNN Cell and Initialize

Stack one or more BasicLSTMCells in a MultiRNNCell.

- The Rnn size should be set using rnn_size
- Initalize Cell State using the MultiRNNCell's zero_state() function
    - Apply the name "initial_state" to the initial state using tf.identity()

Return the cell and initial state in the following tuple (Cell, InitialState)

In [11]:

```
def get_init_cell(batch_size, rnn_size):
    """
    Create an RNN Cell and initialize it.
    :param batch_size: Size of batches
    :param rnn_size: Size of RNNs
    :return: Tuple (cell, initialize state)
    """
```

```
    lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
    lstm_layers = 2
    cell = tf.contrib.rnn.MultiRNNCell([lstm] * lstm_layers)
    initial_state = cell.zero_state(batch_size, tf.float32)
    initial_state = tf.identity(initial_state, name="initial_state")
    return cell, initial_state
```

Word Embedding

Apply embedding to input_data using TensorFlow. Return the embedded sequence.

In [12]:

```
def get_embed(input_data, vocab_size, embed_dim):
    """
    Create embedding for <input_data>.
    :param input_data: TF placeholder for text input.
    :param vocab_size: Number of words in vocabulary.
    :param embed_dim: Number of embedding dimensions
    :return: Embedded input.
    """
    embedding = tf.Variable(tf.random_uniform((vocab_size, embed_dim), -1, 1))
    embed = tf.nn.embedding_lookup(embedding, input_data)
    return embed
```

Build RNN

You created a RNN Cell in the get_init_cell() function. Time to use the cell to create a RNN.

- Build the RNN using the tf.nn.dynamic_rnn()
    - Apply the name "final_state" to the final state using tf.identity()

Return the outputs and final_state state in the following tuple (Outputs, FinalState)

In [13]:

```
def build_rnn(cell, inputs):

    """
    Create a RNN using a RNN Cell
    :param cell: RNN Cell
    :param inputs: Input text data
    :return: Tuple (Outputs, Final State)
    """
```

```
    Outputs, FinalState = tf.nn.dynamic_rnn(cell, inputs, initial_state=None, dtype=tf.float32)
    FinalState = tf.identity(FinalState, name = 'final_state')
    return Outputs, FinalState
```

Build the Neural Network

Apply the functions you implemented above to:

- Apply embedding to input_data using your get_embed(input_data, vocab_size, embed_dim) function.
- Build RNN using cell and your build_rnn(cell, inputs) 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)

Build the Neural Network - **CODE HERE**

Neural Network Training

Hyperparameters

Tune the following parameters:

- Set num_epochs to the number of epochs.
- Set batch_size to the batch size.
- Set rnn_size to the size of the RNNs.
- Set seq_length to the length of sequence.
- Set learning_rate to the learning rate.
- Set show_every_n_batches to the number of batches the neural network should print progress.

Neural Network Training - **CODE HERE**

Build the Graph

Build the graph using the neural network you implemented.

Build the graph - **CODE HERE**

Save Parameters

Save seq_length and save_dir for generating a new TV script.

Save Parameters - **CODE HERE**

Checkpoint

Checkpoint - **CODE HERE**

Implement Generate Functions

Get Tensors

Get tensors from loaded_graph using the function get_tensor_by_name(). Get the tensors using the following names:

- "input:0"
- "initial_state:0"
- "final_state:0"
- "probs:0"

Return the tensors in the following tuple (InputTensor, InitialStateTensor, FinalStateTensor, ProbsTensor)

Get Tensors - **CODE HERE**

Choose Word

Implement the pick_word() function to select the next word using probabilities.

Choose Word - **CODE HERE**

Generate TV Script

This will generate the TV script for you. Set gen_length to the length of TV script you want to generate.

Generate TV Script - **CODE HERE**

Output:

homer_simpson:(awkwardly) you dropped somethin' that he was drinking, bart.(looks at no) no, uh, no. malabar gregor at her.(sobs) voice marge, i've always wanted to go up.
marge_simpson: homer, i'm just a guy who shouldn't have some more / look too. i'm gonna stop the time.
kemi: i comes her back.
moe_szyslak: and i need your kids' store i saw this idea.
homer_simpson:(to moe) i guess there's not so bad. there's an drunk, but i am so moe?! i don't care if i put this bottle is, i just had this outside of your limits?
moe_szyslak: thank you, this was on the bow!
moe_szyslak: homer, you're rather like money, maybe they are using go all crazy.
homer_simpson:(loud) hey, i don't want to go to the music store ruled.
ned_flanders: i need from the last company and you man! i gotta take that again.
homer_simpson: well, i ain't changin' it

NOTE: The solution shared through Github should contain the source code used and the screenshot of the output.

3. Output

N/A