This Project Consists of 3000 marks and has to be submitted in .ipynb/PDF format in the upcoming session for evaluation.

TV Script Generation

In this project, you'll generate your own Simpsons (https://en.wikipedia.org/wiki/The_Simpsons) TV scripts using RNNs. You'll be using part of the Simpsons dataset (https://www.kaggle.com/wcukierski/the-simpsons-by-the-data) of scripts from 27 seasons. The Neural Network you'll build will generate a new TV script for a scene at Moe's Tavern (https://simpsonswiki.com/wiki/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 [0]:

```
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 l
ess
    """
    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 [0]:

```
# 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 [0]:

```
view_sentence_range = (0, 10)

import numpy as np
```

```
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. M
ike Rotch. Hey, has anybody seen Mike Rotch, lately?
Moe_Szyslak: (INTO PHONE) Listen you little puke. One of these days I'm go
nna 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 [0]:

```
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 [0]:

```
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
    """
```

Preprocess all the data and save it

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

In [0]:

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

In [0]:

```
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 (https://www.tensorflow.org/api_docs/python/tf/placeholder) `name` parameter.
- Targets placeholder
- Learning Rate placeholder

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

In [0]:

```
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` (https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/BasicLSTMCell) in a `MultiRNNCell` (https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/MultiRNNCell).

- The Rnn size should be set using `rnn_size`
- Initalize Cell State using the MultiRNNCell's `zero_state()` (https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/MultiRNNCell#zero_state) function
  - Apply the name "initial_state" to the initial state using `tf.identity()` (https://www.tensorflow.org/api_docs/python/tf/identity)

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

In [0]:

```
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)
    """
```

Word Embedding

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

In [0]:

```
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.
    """
```

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()` (https://www.tensorflow.org/api_docs/python/tf/nn/dynamic_rnn)
  - Apply the name "final_state" to the final state using `tf.identity()` (https://www.tensorflow.org/api_docs/python/tf/identity)

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

In [0]:

```
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)
    """
```

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)

In [0]:

```
def build_nn(cell, rnn_size, input_data, vocab_size):
    """
    Build part of the neural network
    :param cell: RNN cell
    :param rnn_size: Size of rnns
    :param input_data: Input data
    :param vocab_size: Vocabulary size
    :return: Tuple (Logits, FinalState)
    """
```

In [0]:

```
def get_batches(int_text, batch_size, seq_length):
    """
    Return batches of input and target
    :param int_text: Text with the words replaced by their ids
    :param batch_size: The size of batch
    :param seq_length: The length of sequence
    :return: Batches as a Numpy array
    """
    total_batch = len(int_text)//(batch_size * seq_length)
    len_to_consider = int(total_batch*batch_size*seq_length)

    input_text = np.array(int_text[:len_to_consider])
    label_text = np.array(int_text[1:len_to_consider+1])

    input_text = np.split(input_text, total_batch*batch_size)
    label_text = np.split(label_text, total_batch*batch_size)

    output = np.empty((total_batch, 2, batch_size, seq_length))


    for i in range(batch_size):
        for j in range(total_batch):
            output[j][0][i] = input_text[total_batch*(i)+j]


    for i in range(batch_size):
        for j in range(total_batch):
            output[j][1][i] = label_text[total_batch*(i)+j]

    return output
```

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.

In [0]:

```
# Number of Epochs
num_epochs = 150
# Batch Size
batch_size = 128
# RNN Size
rnn_size = 128
# Sequence Length
seq_length = 32
# Learning Rate
learning_rate = 0.01
# Show stats for every n number of batches
show_every_n_batches = 20

save_dir = './save'
```

Build the Graph

Build the graph using the neural network you implemented.

In [0]:

```
import tensorflow as tf
from tensorflow.contrib import seq2seq

train_graph = tf.Graph()
with train_graph.as_default():
    vocab_size = len(int_to_vocab)
    input_text, targets, lr = get_inputs()
    input_data_shape = tf.shape(input_text)
    cell, initial_state = get_init_cell(input_data_shape[0], rnn_size)
    logits, final_state = build_nn(cell, rnn_size, input_text, vocab_size)

    # Probabilities for generating words
    probs = tf.nn.softmax(logits, name='probs')

    # Loss function
    cost = seq2seq.sequence_loss(
        logits,
        targets,
        tf.ones([input_data_shape[0], input_data_shape[1]]))

    # Optimizer
    optimizer = tf.train.AdamOptimizer(lr)

    # Gradient Clipping
    gradients = optimizer.compute_gradients(cost)
    capped_gradients = [(tf.clip_by_value(grad, -1., 1.), var) for grad, var in gradien
ts]
    train_op = optimizer.apply_gradients(capped_gradients)
```

In [0]:

```
batches = get_batches(int_text, batch_size, seq_length)

with tf.Session(graph=train_graph) as sess:
    sess.run(tf.global_variables_initializer())

    for epoch_i in range(num_epochs):
        state = sess.run(initial_state, {input_text: batches[0][0]})

        for batch_i, (x, y) in enumerate(batches):
            feed = {
                input_text: x,
                targets: y,
                initial_state: state,
                lr: learning_rate}
            train_loss, state, _ = sess.run([cost, final_state, train_op], feed)

            # Show every <show_every_n_batches> batches
            if (epoch_i * len(batches) + batch_i) % show_every_n_batches == 0:
                print('Epoch {:>3} Batch {:>4}/{}   train_loss = {:.3f}'.format(
                    epoch_i,
                    batch_i,
                    len(batches),
                    train_loss))

    # Save Model
    saver = tf.train.Saver()
    saver.save(sess, save_dir)
    print('Model Trained and Saved')
```

```
Epoch   0 Batch    0/16   train_loss = 8.822
Epoch   1 Batch    4/16   train_loss = 6.057
Epoch   2 Batch    8/16   train_loss = 6.062
Epoch   3 Batch   12/16   train_loss = 5.810
Epoch   5 Batch    0/16   train_loss = 5.679
Epoch   6 Batch    4/16   train_loss = 5.567
Epoch   7 Batch    8/16   train_loss = 5.430
Epoch   8 Batch   12/16   train_loss = 5.185
Epoch  10 Batch    0/16   train_loss = 5.062
Epoch  11 Batch    4/16   train_loss = 4.973
Epoch  12 Batch    8/16   train_loss = 4.887
Epoch  13 Batch   12/16   train_loss = 4.729
Epoch  15 Batch    0/16   train_loss = 4.623
Epoch  16 Batch    4/16   train_loss = 4.567
Epoch  17 Batch    8/16   train_loss = 4.500
Epoch  18 Batch   12/16   train_loss = 4.354
Epoch  20 Batch    0/16   train_loss = 4.209
Epoch  21 Batch    4/16   train_loss = 4.166
Epoch  22 Batch    8/16   train_loss = 4.096
Epoch  23 Batch   12/16   train_loss = 3.957
Epoch  25 Batch    0/16   train_loss = 3.809
Epoch  26 Batch    4/16   train_loss = 3.752
Epoch  27 Batch    8/16   train_loss = 3.755
Epoch  28 Batch   12/16   train_loss = 3.632
Epoch  30 Batch    0/16   train_loss = 3.429
Epoch  31 Batch    4/16   train_loss = 3.368
Epoch  32 Batch    8/16   train_loss = 3.298
Epoch  33 Batch   12/16   train_loss = 3.232
Epoch  35 Batch    0/16   train_loss = 3.066
Epoch  36 Batch    4/16   train_loss = 3.047
Epoch  37 Batch    8/16   train_loss = 3.054
Epoch  38 Batch   12/16   train_loss = 3.049
Epoch  40 Batch    0/16   train_loss = 2.804
Epoch  41 Batch    4/16   train_loss = 2.715
Epoch  42 Batch    8/16   train_loss = 2.643
Epoch  43 Batch   12/16   train_loss = 2.611
Epoch  45 Batch    0/16   train_loss = 2.448
Epoch  46 Batch    4/16   train_loss = 2.399
Epoch  47 Batch    8/16   train_loss = 2.383
Epoch  48 Batch   12/16   train_loss = 2.361
Epoch  50 Batch    0/16   train_loss = 2.244
Epoch  51 Batch    4/16   train_loss = 2.169
Epoch  52 Batch    8/16   train_loss = 2.170
Epoch  53 Batch   12/16   train_loss = 2.121
Epoch  55 Batch    0/16   train_loss = 1.967
Epoch  56 Batch    4/16   train_loss = 1.885
Epoch  57 Batch    8/16   train_loss = 1.883
Epoch  58 Batch   12/16   train_loss = 1.830
Epoch  60 Batch    0/16   train_loss = 1.753
Epoch  61 Batch    4/16   train_loss = 1.668
Epoch  62 Batch    8/16   train_loss = 1.710
Epoch  63 Batch   12/16   train_loss = 1.682
Epoch  65 Batch    0/16   train_loss = 1.587
Epoch  66 Batch    4/16   train_loss = 1.550
Epoch  67 Batch    8/16   train_loss = 1.606
Epoch  68 Batch   12/16   train_loss = 1.582
Epoch  70 Batch    0/16   train_loss = 1.543
Epoch  71 Batch    4/16   train_loss = 1.524
Epoch  72 Batch    8/16   train_loss = 1.550
Epoch  73 Batch   12/16   train_loss = 1.457
Epoch  75 Batch    0/16   train_loss = 1.392
```

```
Epoch  76 Batch    4/16   train_loss = 1.312
Epoch  77 Batch    8/16   train_loss = 1.336
Epoch  78 Batch   12/16   train_loss = 1.293
Epoch  80 Batch    0/16   train_loss = 1.265
Epoch  81 Batch    4/16   train_loss = 1.243
Epoch  82 Batch    8/16   train_loss = 1.296
Epoch  83 Batch   12/16   train_loss = 1.300
Epoch  85 Batch    0/16   train_loss = 1.252
Epoch  86 Batch    4/16   train_loss = 1.208
Epoch  87 Batch    8/16   train_loss = 1.213
Epoch  88 Batch   12/16   train_loss = 1.190
Epoch  90 Batch    0/16   train_loss = 1.211
Epoch  91 Batch    4/16   train_loss = 1.110
Epoch  92 Batch    8/16   train_loss = 1.046
Epoch  93 Batch   12/16   train_loss = 1.012
Epoch  95 Batch    0/16   train_loss = 0.974
Epoch  96 Batch    4/16   train_loss = 0.919
Epoch  97 Batch    8/16   train_loss = 0.934
Epoch  98 Batch   12/16   train_loss = 0.932
Epoch 100 Batch    0/16   train_loss = 0.874
Epoch 101 Batch    4/16   train_loss = 0.878
Epoch 102 Batch    8/16   train_loss = 0.910
Epoch 103 Batch   12/16   train_loss = 0.876
Epoch 105 Batch    0/16   train_loss = 0.893
Epoch 106 Batch    4/16   train_loss = 0.849
Epoch 107 Batch    8/16   train_loss = 0.820
Epoch 108 Batch   12/16   train_loss = 0.854
Epoch 110 Batch    0/16   train_loss = 0.824
Epoch 111 Batch    4/16   train_loss = 0.765
Epoch 112 Batch    8/16   train_loss = 0.808
Epoch 113 Batch   12/16   train_loss = 0.749
Epoch 115 Batch    0/16   train_loss = 0.749
Epoch 116 Batch    4/16   train_loss = 0.741
Epoch 117 Batch    8/16   train_loss = 0.718
Epoch 118 Batch   12/16   train_loss = 0.670
Epoch 120 Batch    0/16   train_loss = 0.681
Epoch 121 Batch    4/16   train_loss = 0.687
Epoch 122 Batch    8/16   train_loss = 0.693
Epoch 123 Batch   12/16   train_loss = 0.630
Epoch 125 Batch    0/16   train_loss = 0.600
Epoch 126 Batch    4/16   train_loss = 0.575
Epoch 127 Batch    8/16   train_loss = 0.548
Epoch 128 Batch   12/16   train_loss = 0.504
Epoch 130 Batch    0/16   train_loss = 0.508
Epoch 131 Batch    4/16   train_loss = 0.488
Epoch 132 Batch    8/16   train_loss = 0.472
Epoch 133 Batch   12/16   train_loss = 0.448
Epoch 135 Batch    0/16   train_loss = 0.456
Epoch 136 Batch    4/16   train_loss = 0.451
Epoch 137 Batch    8/16   train_loss = 0.442
Epoch 138 Batch   12/16   train_loss = 0.430
Epoch 140 Batch    0/16   train_loss = 0.449
Epoch 141 Batch    4/16   train_loss = 0.484
Epoch 142 Batch    8/16   train_loss = 0.460
Epoch 143 Batch   12/16   train_loss = 0.492
Epoch 145 Batch    0/16   train_loss = 0.518
Epoch 146 Batch    4/16   train_loss = 0.521
Epoch 147 Batch    8/16   train_loss = 0.552
Epoch 148 Batch   12/16   train_loss = 0.589
Model Trained and Saved
```

Save Parameters

Save `seq_length` and `save_dir` for generating a new TV script.

In [0]:

```
save_params((seq_length, save_dir))
```

Checkpoint

In [0]:

```
_, vocab_to_int, int_to_vocab, token_dict = load_preprocess()
seq_length, load_dir = load_params()
```

Implement Generate Functions

Get Tensors

Get tensors from `loaded_graph` using the function `get_tensor_by_name()` (https://www.tensorflow.org/api_docs/python/tf/Graph#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)`

In [0]:

```
def get_tensors(loaded_graph):
    """
    Get input, initial state, final state, and probabilities tensor from <loaded_graph>
    :param loaded_graph: TensorFlow graph loaded from file
    :return: Tuple (InputTensor, InitialStateTensor, FinalStateTensor, ProbsTensor)
    """
    InputTensor = loaded_graph.get_tensor_by_name("input:0")
    InitialStateTensor = loaded_graph.get_tensor_by_name("initial_state:0")
    FinalStateTensor = loaded_graph.get_tensor_by_name("final_state:0")
    ProbsTensor = loaded_graph.get_tensor_by_name("probs:0")
    return InputTensor, InitialStateTensor, FinalStateTensor, ProbsTensor
```

Choose Word

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

In [0]:

```
def pick_word(probabilities, int_to_vocab):
    """
    Pick the next word in the generated text
    :param probabilities: Probabilites of the next word
    :param int_to_vocab: Dictionary of word ids as the keys and words as the values
    :return: String of the predicted word
    """
```

Generate TV Script

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

In [0]:

```
gen_length = 200
# homer_simpson, moe_szyslak, or Barney_Gumble
prime_word = 'homer_simpson'


with tf.Session(graph=loaded_graph) as sess:
    # Load saved model
    loader = tf.train.import_meta_graph(load_dir + '.meta')


    # Get Tensors from loaded model
    input_text, initial_state, final_state, probs = get_tensors(loaded_graph)


    # Sentences generation setup
    gen_sentences = [prime_word + ':']


    # Generate sentences

        # Get Prediction


        gen_sentences.append(pred_word)

    # Remove tokens
    tv_script = ' '.join(gen_sentences)


    print(tv_script)
```

```
homer_simpson:(awkwardly) you dropped somethin' that he was drinking, bar
t.(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 t
oo. 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
```