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..

In [1]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import helper

data_dir = './data/simpsons/moes_tavern_lines.txt'
text = helper.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 [2]:
```
view_sentence_range = (0, 10)

"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import numpy as np

print('Dataset Stats')
print('Roughly the number of unique words: {}'.format(len({word: None fo
r 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_ra
nge[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 Rotc
h. 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 p
ick.
Moe_Szyslak: What's the matter Homer? You're not your normal effervesce
nt 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 skill
s.
```

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)`

In [3]:
```
import numpy as np
import problem_unittests as tests
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
    word_counts = Counter(text)
    sorted_vocab = sorted(word_counts, key=word_counts.get, reverse=True
)
    int_to_vocab = {ii: word for ii, word in enumerate(sorted_vocab)}
    vocab_to_int = {word: ii for ii, word in int_to_vocab.items()}
    return vocab_to_int, int_to_vocab


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_create_lookup_tables(create_lookup_tables)
```

```
Tests Passed
```

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 [5]:
```
def token_lookup():
    """
    Generate a dict to turn punctuation into a token.
    :return: Tokenize dictionary where the key is the punctuation and th
e value is the token
    """
    # TODO: Implement Function
    tokenize_dict = {
        '.': '||period||',
        ',': '||comma||',
        '"': '||quotation_mark||',
        ';': '||semicolon||',
        '!': '||exclamation_mark||',
        '?': '||question_mark||',
        '(': '||left_parentheses||',
        ')': '||right_parentheses||',
        '--': '||dash||',
        '\n': '||return||'
    }

    return tokenize_dict

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_tokenize(token_lookup)
```

```
Tests Passed
```

Preprocess all the data and save it

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

In [6]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
# Preprocess Training, Validation, and Testing Data
helper.preprocess_and_save_data(data_dir, token_lookup, create_lookup_ta
bles)
```

Check Point

This is your first checkpoint. If you ever decide to come back to this notebook or have to restart the notebook, you can start from here. The preprocessed data has been saved to disk.

In [1]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import helper
import numpy as np
import problem_unittests as tests

int_text, vocab_to_int, int_to_vocab, token_dict = helper.load_preproces
s()
```

Build the Neural Network

You'll build the components necessary to build a RNN by implementing the following functions below:

- get_inputs
- get_init_cell
- get_embed
- build_rnn
- build_nn
- get_batches

Check the Version of TensorFlow and Access to GPU

In [2]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
from distutils.version import LooseVersion
import warnings
import tensorflow as tf

# Check TensorFlow Version
assert LooseVersion(tf.__version__) >= LooseVersion('1.3'), 'Please use
 TensorFlow version 1.3 or newer'
print('TensorFlow Version: {}'.format(tf.__version__))

# Check for a GPU
if not tf.test.gpu_device_name():
    warnings.warn('No GPU found. Please use a GPU to train your neural n
etwork.')
else:
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
```

```
TensorFlow Version: 1.3.0
Default GPU Device: /gpu:0
```

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 tuple `(Input, Targets, LearningRate)`

In [3]:
```
def get_inputs():
    """
    Create TF Placeholders for input, targets, and learning rate.
    :return: Tuple (input, targets, learning rate)
    """
    # TODO: Implement Function
    inputs = tf.placeholder(tf.int32, shape=(None,None), name='input')
    targets = tf.placeholder(tf.int32, shape=(None,None), name='targets'
)
    learning_rate = tf.placeholder(tf.float32, name='learning_rate')
    return inputs, targets, learning_rate


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_get_inputs(get_inputs)
```

```
Tests Passed
```

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 [4]:
```
def add_dropout(lstm_cell, keep_prob):
    ''' Add dropout to the cell.

        Arguments
        ---------
        lstm_cell: tensorflow BasicLSTMCell() object
        keep_prob: Scalar tensor (tf.placeholder) for the
                   dropout keep probability

    '''
    return tf.contrib.rnn.DropoutWrapper(lstm_cell, output_keep_prob=kee
p_prob)

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)
    """
    # TODO: Implement Function

    # Probability an LSTM cell will not be dropped.
    keep_prob = 0.5
    # Create an LSTM cell
    cell = tf.contrib.rnn.BasicLSTMCell(rnn_size)

    # Stack multiple LSTM cells according to the number
    # of layers (rnn_layers). Wrap each LSTM cell in a
    # dropout layer.
    cell = tf.nn.rnn_cell.MultiRNNCell([add_dropout(cell, keep_prob)])

    #Create initial state.
    initial_state = tf.identity(cell.zero_state(batch_size, tf.float32),
 name='initial_state')
    return cell, initial_state


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_get_init_cell(get_init_cell)
```

```
Tests Passed
```

Word Embedding

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

In [5]:
```
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.
    """
    # TODO: Implement Function
    embedding = tf.Variable(tf.random_uniform(shape=(vocab_size, embed_d
im),
                                              minval=-1, maxval=1))
    embed = tf.nn.embedding_lookup(params=embedding, ids=input_data)
    return embed


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_get_embed(get_embed)
```

```
Tests Passed
```

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 [6]:
```
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)
    """
    # TODO: Implement Function
    outputs, final_state = tf.nn.dynamic_rnn(cell, inputs, dtype=tf.floa
t32)
    final_state = tf.identity(final_state, name='final_state')

    return outputs, final_state


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_build_rnn(build_rnn)
```

```
Tests Passed
```

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 [7]:
```
def build_nn(cell, rnn_size, input_data, vocab_size, embed_dim):
    """
    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
    :param embed_dim: Number of embedding dimensions
    :return: Tuple (Logits, FinalState)
    """
    # TODO: Implement Function
    embedding_layer = get_embed(input_data, vocab_size, embed_dim)
    rnn_outputs, final_state = build_rnn(cell, embedding_layer)
    logits = tf.contrib.layers.fully_connected(inputs=rnn_outputs, num_o
utputs=vocab_size,
                                               activation_fn=None)

    return logits, final_state


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_build_nn(build_nn)
```

```
Tests Passed
```

Batches

Implement `get_batches` to create batches of input and targets using `int_text`. The batches should be a Numpy array with the shape `(number of batches, 2, batch size, sequence length)`. Each batch contains two elements:

- The first element is a single batch of **input** with the shape `[batch size, sequence length]`
- The second element is a single batch of **targets** with the shape `[batch size, sequence length]`

If you can't fill the last batch with enough data, drop the last batch.

For example, `get_batches([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20], 3, 2)` would return a Numpy array of the following:

```
[
  # First Batch
  [
    # Batch of Input
    [[ 1  2], [ 7  8], [13 14]]
    # Batch of targets
    [[ 2  3], [ 8  9], [14 15]]
  ]

  # Second Batch
  [
    # Batch of Input
    [[ 3  4], [ 9 10], [15 16]]
    # Batch of targets
    [[ 4  5], [10 11], [16 17]]
  ]

  # Third Batch
  [
    # Batch of Input
    [[ 5  6], [11 12], [17 18]]
    # Batch of targets
    [[ 6  7], [12 13], [18  1]]
  ]
]
```

Notice that the last target value in the last batch is the first input value of the first batch. In this case, `1`. This is a common technique used when creating sequence batches, although it is rather unintuitive.

In [8]:
```
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
    """
    # TODO: Implement Function

    # Keep only enough words in int_text to make full batches.
    words_per_batch = batch_size * seq_length
    number_of_batches = len(int_text) // words_per_batch
    words_to_keep = words_per_batch * number_of_batches
    int_text = np.array(int_text[:words_to_keep])

    # Prepare for creation of the target values:
    # Target value of an element at a given position in int_text
    # is that of the element residing at the subsequent position.
    target_int_text = int_text.copy()
    # Move the first element of the input text to the final position.
    # This handles the edge case of how to choose a target value for
    # the final element of int_text.
    target_int_text[0:-1], target_int_text[-1] = int_text[1:], int_text[
0]

    # Reshape int_text to have the same number of rows
    # as the size of the batches.
    int_text = int_text.reshape((batch_size, -1))

    # Also reshape target_int_text to have the same number of rows
    # as the size of the batches.
    target_int_text = target_int_text.reshape((batch_size, -1))

    # An array to contain all the batches
    all_batches = []

    # Slice features (x) and targets (y) columns from
    # their respective arrays (int_text, target_int_text)
    # in order to populate the batches. Each batch contains
    # one sequence of values from each row of the arrays,
    # for both features and targets.
    # The start and stop of each sequence (or the indices of
    # of the columns that are sliced) correspond to the
    # seq_length parameter's value.
    for n in range (0, int_text.shape[1], seq_length):
        x_batch = int_text[:, n:n+seq_length]
        y_batch = target_int_text[:, n:n+seq_length]
        all_batches.append([x_batch, y_batch])

    # Make sure an np.array is returned
    return np.array(all_batches)


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
```

```
"""
tests.test_get_batches(get_batches)
```

```
Tests Passed
```

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 `embed_dim` to the size of the embedding.
- 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 [52]:
```
# Number of Epochs
num_epochs = 1000
# Batch Size
batch_size = 1000
# RNN Size
rnn_size = 512
# Embedding Dimension Size
embed_dim = 300
# Sequence Length
seq_length = 10
# Learning Rate
learning_rate = 0.001
# Show stats for every n number of batches
show_every_n_batches = 60

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
save_dir = './save'
```

Build the Graph

Build the graph using the neural network you implemented.

In [53]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
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_siz
e, embed_dim)

    # 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 gradients if grad is not None]
    train_op = optimizer.apply_gradients(capped_gradients)
```

Train

Train the neural network on the preprocessed data. If you have a hard time getting a good loss, check the forums (https://discussions.udacity.com/) to see if anyone is having the same problem.

In [54]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
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(1, num_epochs+1):
        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-1) * len(batches) + batch_i+1) % show_every_n_b
atches == 0:
                print('Epoch {:>3} Batch {:>4}/{}   train_loss = {:.3f}'
.format(
                    epoch_i,
                    batch_i+1,
                    len(batches),
                    train_loss))

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

```
Epoch  10 Batch    6/6   train_loss = 5.506
Epoch  20 Batch    6/6   train_loss = 4.901
Epoch  30 Batch    6/6   train_loss = 4.466
Epoch  40 Batch    6/6   train_loss = 4.127
Epoch  50 Batch    6/6   train_loss = 3.835
Epoch  60 Batch    6/6   train_loss = 3.574
Epoch  70 Batch    6/6   train_loss = 3.344
Epoch  80 Batch    6/6   train_loss = 3.125
Epoch  90 Batch    6/6   train_loss = 2.937
Epoch 100 Batch    6/6   train_loss = 2.755
Epoch 110 Batch    6/6   train_loss = 2.597
Epoch 120 Batch    6/6   train_loss = 2.465
Epoch 130 Batch    6/6   train_loss = 2.310
Epoch 140 Batch    6/6   train_loss = 2.198
Epoch 150 Batch    6/6   train_loss = 2.082
Epoch 160 Batch    6/6   train_loss = 1.978
Epoch 170 Batch    6/6   train_loss = 1.855
Epoch 180 Batch    6/6   train_loss = 1.780
Epoch 190 Batch    6/6   train_loss = 1.657
Epoch 200 Batch    6/6   train_loss = 1.582
Epoch 210 Batch    6/6   train_loss = 1.505
Epoch 220 Batch    6/6   train_loss = 1.442
Epoch 230 Batch    6/6   train_loss = 1.382
Epoch 240 Batch    6/6   train_loss = 1.283
Epoch 250 Batch    6/6   train_loss = 1.231
Epoch 260 Batch    6/6   train_loss = 1.167
Epoch 270 Batch    6/6   train_loss = 1.121
Epoch 280 Batch    6/6   train_loss = 1.069
Epoch 290 Batch    6/6   train_loss = 1.013
Epoch 300 Batch    6/6   train_loss = 0.965
Epoch 310 Batch    6/6   train_loss = 0.925
Epoch 320 Batch    6/6   train_loss = 0.893
Epoch 330 Batch    6/6   train_loss = 0.869
Epoch 340 Batch    6/6   train_loss = 0.808
Epoch 350 Batch    6/6   train_loss = 0.787
Epoch 360 Batch    6/6   train_loss = 0.765
Epoch 370 Batch    6/6   train_loss = 0.741
Epoch 380 Batch    6/6   train_loss = 0.710
Epoch 390 Batch    6/6   train_loss = 0.684
Epoch 400 Batch    6/6   train_loss = 0.668
Epoch 410 Batch    6/6   train_loss = 0.654
Epoch 420 Batch    6/6   train_loss = 0.649
Epoch 430 Batch    6/6   train_loss = 0.623
Epoch 440 Batch    6/6   train_loss = 0.595
Epoch 450 Batch    6/6   train_loss = 0.591
Epoch 460 Batch    6/6   train_loss = 0.579
Epoch 470 Batch    6/6   train_loss = 0.563
Epoch 480 Batch    6/6   train_loss = 0.561
Epoch 490 Batch    6/6   train_loss = 0.554
Epoch 500 Batch    6/6   train_loss = 0.545
Epoch 510 Batch    6/6   train_loss = 0.529
Epoch 520 Batch    6/6   train_loss = 0.525
Epoch 530 Batch    6/6   train_loss = 0.512
Epoch 540 Batch    6/6   train_loss = 0.505
Epoch 550 Batch    6/6   train_loss = 0.508
Epoch 560 Batch    6/6   train_loss = 0.503
Epoch 570 Batch    6/6   train_loss = 0.493
```

```
Epoch 580 Batch    6/6   train_loss = 0.491
Epoch 590 Batch    6/6   train_loss = 0.483
Epoch 600 Batch    6/6   train_loss = 0.481
Epoch 610 Batch    6/6   train_loss = 0.467
Epoch 620 Batch    6/6   train_loss = 0.467
Epoch 630 Batch    6/6   train_loss = 0.465
Epoch 640 Batch    6/6   train_loss = 0.466
Epoch 650 Batch    6/6   train_loss = 0.456
Epoch 660 Batch    6/6   train_loss = 0.460
Epoch 670 Batch    6/6   train_loss = 0.451
Epoch 680 Batch    6/6   train_loss = 0.444
Epoch 690 Batch    6/6   train_loss = 0.444
Epoch 700 Batch    6/6   train_loss = 0.444
Epoch 710 Batch    6/6   train_loss = 0.432
Epoch 720 Batch    6/6   train_loss = 0.442
Epoch 730 Batch    6/6   train_loss = 0.432
Epoch 740 Batch    6/6   train_loss = 0.444
Epoch 750 Batch    6/6   train_loss = 0.436
Epoch 760 Batch    6/6   train_loss = 0.433
Epoch 770 Batch    6/6   train_loss = 0.426
Epoch 780 Batch    6/6   train_loss = 0.426
Epoch 790 Batch    6/6   train_loss = 0.418
Epoch 800 Batch    6/6   train_loss = 0.423
Epoch 810 Batch    6/6   train_loss = 0.424
Epoch 820 Batch    6/6   train_loss = 0.422
Epoch 830 Batch    6/6   train_loss = 0.417
Epoch 840 Batch    6/6   train_loss = 0.417
Epoch 850 Batch    6/6   train_loss = 0.413
Epoch 860 Batch    6/6   train_loss = 0.414
Epoch 870 Batch    6/6   train_loss = 0.416
Epoch 880 Batch    6/6   train_loss = 0.405
Epoch 890 Batch    6/6   train_loss = 0.409
Epoch 900 Batch    6/6   train_loss = 0.410
Epoch 910 Batch    6/6   train_loss = 0.407
Epoch 920 Batch    6/6   train_loss = 0.402
Epoch 930 Batch    6/6   train_loss = 0.403
Epoch 940 Batch    6/6   train_loss = 0.402
Epoch 950 Batch    6/6   train_loss = 0.399
Epoch 960 Batch    6/6   train_loss = 0.404
Epoch 970 Batch    6/6   train_loss = 0.396
Epoch 980 Batch    6/6   train_loss = 0.399
Epoch 990 Batch    6/6   train_loss = 0.396
Epoch 1000 Batch    6/6   train_loss = 0.391
Model Trained and Saved
```

Save Parameters

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

In [55]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
# Save parameters for checkpoint
helper.save_params((seq_length, save_dir))
```

Checkpoint

In [56]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import tensorflow as tf
import numpy as np
import helper
import problem_unittests as tests

_, vocab_to_int, int_to_vocab, token_dict = helper.load_preprocess()
seq_length, load_dir = helper.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 [57]:
```
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, P
robsTensor)
    """
    # TODO: Implement Function
    input_tensor = loaded_graph.get_tensor_by_name('input:0')
    initial_state_tensor = loaded_graph.get_tensor_by_name('initial_stat
e:0')
    final_state_tensor = loaded_graph.get_tensor_by_name('final_state:0'
)
    probs_tensor = loaded_graph.get_tensor_by_name('probs:0')
    return input_tensor, initial_state_tensor, final_state_tensor, probs
_tensor


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_get_tensors(get_tensors)
```

```
Tests Passed
```

Choose Word

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

In [58]:
```
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
    """
    # TODO: Implement Function

    # Randomly choose a word from the four words that have
    # the highest probabilities.
    top_n = 4

    # Get the indices corresponding to the top probilities
    idx_top_probs = np.argpartition(probabilities, -top_n)[-top_n:]

    # Choose one of these indices at random
    choice = np.random.choice(idx_top_probs)

    # Select the word that belongs to the randomly
    # chosen index.
    next_word = int_to_vocab[choice]

    return next_word


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_pick_word(pick_word)
```

```
Tests Passed
```

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 [59]:
```
gen_length = 500
# homer_simpson, moe_szyslak, or Barney_Gumble
prime_word = 'moe_szyslak'

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
loaded_graph = tf.Graph()
with tf.Session(graph=loaded_graph) as sess:
    # Load saved model
    loader = tf.train.import_meta_graph(load_dir + '.meta')
    loader.restore(sess, load_dir)

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

    # Sentences generation setup
    gen_sentences = [prime_word + ':']
    prev_state = sess.run(initial_state, {input_text: np.array([[1]])})

    # Generate sentences
    for n in range(gen_length):
        # Dynamic Input
        dyn_input = [[vocab_to_int[word] for word in gen_sentences[-seq_
length:]]]
        dyn_seq_length = len(dyn_input[0])

        # Get Prediction
        probabilities, prev_state = sess.run(
            [probs, final_state],
            {input_text: dyn_input, initial_state: prev_state})

        pred_word = pick_word(probabilities[0][dyn_seq_length-1], int_to
_vocab)

        gen_sentences.append(pred_word)

    # Remove tokens
    tv_script = ' '.join(gen_sentences)
    for key, token in token_dict.items():
        ending = ' ' if key in ['\n', '(', '"'] else ''
        tv_script = tv_script.replace(' ' + token.lower(), key)
    tv_script = tv_script.replace('\n ', '\n')
    tv_script = tv_script.replace('( ', '(')

    print(tv_script)
```

INFO:tensorflow:Restoring parameters from ./save
moe_szyslak: yeah on the ball thing.
lenny_leonard: i see this for now since there who makes sense home she. while.
carl_carlson: pick up! what?
moe_szyslak: just what have money for the president of course that old friend, boxing i was! another moe
barney_gumble: hey! what, you're the fat one else stupid(to, lenny home r? hey) they the should put on how a daddy? 'cause you sayin' you look, but that you've had all a feeling moe_szyslak: him for / yeah with the" ned_flanders: hey guys stink, usin'. it's husband at
homer_simpson:(realizing at a man sorry homer then) there, didn't be" e ightball" problem?!
homer_simpson: thanks for homer?. uh, would.
apu_nahasapeemapetilon: i could this guy for your old man passed, two w anted ya got in good?
barney_gumble:(a beer to the real"-- then tonight, then singing as mo e's.. some with dirt.(counting sign at the bar)
, ah, who homer_simpson: on the phone and drink where? i! i own losers all better again
barney_gumble: that's alright what does she got about the money
homer_simpson: and i are you! 'cause i will say no trouble. could pull it off my face! ow? my poor man!
david_byrne: wait right the party at this bar is(a beer) what are you?, didn't? the one, hey(determined mean a day that's drive a body? where y ou could do up because they enveloped me with marge 'cause" joe could" save no".
moe_szyslak: the super are my best friend is my one? you've have to be second, booze like(with each woman).(art). i could really do it, you! a m(raises prank watch) better were outta the the man. the only" no girl, of any? hey that's what its fondest me last with alcohol again!. whee h ave in there you open up? but. not to their choice. and it's

lisa_simpson: a duff!
barney_gumble: you like for be sorry, homer the clone). you know, no wa s from here?.. it, uh., i wanna have a job for well that you still in" here no" i wanna need? another.
seymour_skinner:(sings, disgusted lloyd to us sits there around, stick tonight. i'll have. to uh.(takes out cell phone
bart_simpson: hi while did yourself is here puff in the car!
moe_szyslak: i mean. now i let,

The TV Script is Nonsensical

It's ok if the TV script doesn't make any sense. We trained on less than a megabyte of text. In order to get good results, you'll have to use a smaller vocabulary or get more data. Luckily there's more data! As we mentioned in the beggining of this project, this is a subset of another dataset (https://www.kaggle.com/wcukierski/the-simpsons-by-the-data). We didn't have you train on all the data, because that would take too long. However, you are free to train your neural network on all the data. After you complete the project, of course.

Submitting This Project

When submitting this project, make sure to run all the cells before saving the notebook. Save the notebook file as "dlnd_tv_script_generation.ipynb" and save it as a HTML file under "File" -> "Download as". Include the "helper.py" and "problem_unittests.py" files in your submission.