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 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(senten
ce_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 sel
f.
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)`

In [3]:
```
import numpy as np
from collections import Counter
import problem_unittests as tests

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 [4]:
```
def token_lookup():
    """
    Generate a dict to turn punctuation into a token.
    :return: Tokenize dictionary where the key is the punctuation and the valu
e is the token
    """
    # TODO: Implement Function
    token_dict = {
        '.' : "||Period||",
        ',' : "||Comma||",
        '"' : "||Quotation_Mark||",
        ';' : "||Semicolon||",
        '!' : "||Exclamation_Mark||",
        '?' : "||Question_Mark||",
        '(' : "||Left_Parentheses||",
        ')' : "||Right_Parentheses||",
        '--': "||Dash||",
        '\n': "||Return||"
        }
    return token_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 [5]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
# Preprocess Training, Validation, and Testing Data
helper.preprocess_and_save_data(data_dir, token_lookup, create_lookup_tables)
```

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 [6]:
```
"""
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_preprocess()
```

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 [7]:
```
"""
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('1.0'), 'Please use Tensor
Flow version 1.0 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 networ
k.')
else:
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
```

```
TensorFlow Version: 1.0.0
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 [8]:
```
def get_inputs():
    """
    Create TF Placeholders for input, targets, and learning rate.
    :return: Tuple (input, targets, learning rate)
    """
    # TODO: Implement Function
    input = tf.placeholder(tf.int32, [None, None], name='input')
    targets = tf.placeholder(tf.int32, [None, None], name='targets')
    learning_rate = tf.placeholder(tf.float32, name='learning_rate')
    return input, 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 [9]:
```
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
    lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)

    # Add dropout to the cell
    drop = tf.contrib.rnn.DropoutWrapper(lstm, 0.5)

    # multiple LSTM layers
    #cell = tf.contrib.rnn.MultiRNNCell([lstm] * 1)

    cell = tf.contrib.rnn.MultiRNNCell([drop] * 1)

    # initialize LSTM cell state and identify as 'initial_state'
    initial_state = tf.identity(cell.zero_state(batch_size, tf.float32), na
me='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 [10]:
```
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((vocab_size, embed_dim), -1,
1))
    embed = tf.nn.embedding_lookup(embedding, 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 [11]:
```
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 = tf.nn.dynamic_rnn(cell, inputs, dtype=tf.float32)
    final_state = tf.identity(input=final, 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 [12]:
```
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
    emb = get_embed(input_data, vocab_size, embed_dim)
    outputs, final_state = build_rnn(cell, emb)
    logits = tf.contrib.layers.fully_connected(outputs, 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 exmple, `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 [13]:
```
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
    n_batches = int(len(int_text) / (batch_size * seq_length))

    # Drop the last few characters to make only full batches
    xdata = np.array(int_text[: n_batches * batch_size * seq_length])
    ydata = np.array(int_text[1: n_batches * batch_size * seq_length + 1])
    ydata[-1] = xdata[0]

    x_batches = np.split(xdata.reshape(batch_size, -1), n_batches, 1)
    y_batches = np.split(ydata.reshape(batch_size, -1), n_batches, 1)

    return np.array(list(zip(x_batches, y_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 [14]:
```
# Number of Epochs
num_epochs = 500
# Batch Size
batch_size = 100
# RNN Size
rnn_size = 500
# Embedding Dimension Size
embed_dim = 256
# Sequence Length
seq_length = 30
# Learning Rate
learning_rate = 0.001
# Show stats for every n number of batches
show_every_n_batches = 100

"""
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 [15]:
```
"""
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_size,
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, va
r 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 [16]:
```
"""
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(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}'.fo
rmat(
                    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    0/23   train_loss = 8.820
Epoch   4 Batch    8/23   train_loss = 5.406
Epoch   8 Batch   16/23   train_loss = 5.007
Epoch  13 Batch    1/23   train_loss = 4.508
Epoch  17 Batch    9/23   train_loss = 4.246
Epoch  21 Batch   17/23   train_loss = 4.037
Epoch  26 Batch    2/23   train_loss = 3.952
Epoch  30 Batch   10/23   train_loss = 3.786
Epoch  34 Batch   18/23   train_loss = 3.650
Epoch  39 Batch    3/23   train_loss = 3.482
Epoch  43 Batch   11/23   train_loss = 3.397
Epoch  47 Batch   19/23   train_loss = 3.240
Epoch  52 Batch    4/23   train_loss = 3.072
Epoch  56 Batch   12/23   train_loss = 2.926
Epoch  60 Batch   20/23   train_loss = 2.839
Epoch  65 Batch    5/23   train_loss = 2.807
Epoch  69 Batch   13/23   train_loss = 2.738
Epoch  73 Batch   21/23   train_loss = 2.630
Epoch  78 Batch    6/23   train_loss = 2.567
Epoch  82 Batch   14/23   train_loss = 2.506
Epoch  86 Batch   22/23   train_loss = 2.414
Epoch  91 Batch    7/23   train_loss = 2.364
Epoch  95 Batch   15/23   train_loss = 2.220
Epoch 100 Batch    0/23   train_loss = 2.199
Epoch 104 Batch    8/23   train_loss = 2.116
Epoch 108 Batch   16/23   train_loss = 2.028
Epoch 113 Batch    1/23   train_loss = 1.949
Epoch 117 Batch    9/23   train_loss = 1.906
Epoch 121 Batch   17/23   train_loss = 1.813
Epoch 126 Batch    2/23   train_loss = 1.774
Epoch 130 Batch   10/23   train_loss = 1.813
Epoch 134 Batch   18/23   train_loss = 1.692
Epoch 139 Batch    3/23   train_loss = 1.624
Epoch 143 Batch   11/23   train_loss = 1.602
Epoch 147 Batch   19/23   train_loss = 1.552
Epoch 152 Batch    4/23   train_loss = 1.513
Epoch 156 Batch   12/23   train_loss = 1.409
Epoch 160 Batch   20/23   train_loss = 1.399
Epoch 165 Batch    5/23   train_loss = 1.379
Epoch 169 Batch   13/23   train_loss = 1.321
Epoch 173 Batch   21/23   train_loss = 1.355
Epoch 178 Batch    6/23   train_loss = 1.297
Epoch 182 Batch   14/23   train_loss = 1.278
Epoch 186 Batch   22/23   train_loss = 1.206
Epoch 191 Batch    7/23   train_loss = 1.169
Epoch 195 Batch   15/23   train_loss = 1.068
Epoch 200 Batch    0/23   train_loss = 1.114
Epoch 204 Batch    8/23   train_loss = 1.097
Epoch 208 Batch   16/23   train_loss = 0.996
Epoch 213 Batch    1/23   train_loss = 0.990
Epoch 217 Batch    9/23   train_loss = 0.994
Epoch 221 Batch   17/23   train_loss = 0.912
Epoch 226 Batch    2/23   train_loss = 0.903
Epoch 230 Batch   10/23   train_loss = 0.926
Epoch 234 Batch   18/23   train_loss = 0.844
Epoch 239 Batch    3/23   train_loss = 0.797
Epoch 243 Batch   11/23   train_loss = 0.813
```

```
Epoch 247 Batch   19/23   train_loss = 0.775
Epoch 252 Batch    4/23   train_loss = 0.752
Epoch 256 Batch   12/23   train_loss = 0.709
Epoch 260 Batch   20/23   train_loss = 0.701
Epoch 265 Batch    5/23   train_loss = 0.664
Epoch 269 Batch   13/23   train_loss = 0.657
Epoch 273 Batch   21/23   train_loss = 0.686
Epoch 278 Batch    6/23   train_loss = 0.627
Epoch 282 Batch   14/23   train_loss = 0.642
Epoch 286 Batch   22/23   train_loss = 0.607
Epoch 291 Batch    7/23   train_loss = 0.596
Epoch 295 Batch   15/23   train_loss = 0.518
Epoch 300 Batch    0/23   train_loss = 0.551
Epoch 304 Batch    8/23   train_loss = 0.544
Epoch 308 Batch   16/23   train_loss = 0.482
Epoch 313 Batch    1/23   train_loss = 0.512
Epoch 317 Batch    9/23   train_loss = 0.499
Epoch 321 Batch   17/23   train_loss = 0.449
Epoch 326 Batch    2/23   train_loss = 0.464
Epoch 330 Batch   10/23   train_loss = 0.485
Epoch 334 Batch   18/23   train_loss = 0.435
Epoch 339 Batch    3/23   train_loss = 0.402
Epoch 343 Batch   11/23   train_loss = 0.408
Epoch 347 Batch   19/23   train_loss = 0.401
Epoch 352 Batch    4/23   train_loss = 0.371
Epoch 356 Batch   12/23   train_loss = 0.372
Epoch 360 Batch   20/23   train_loss = 0.356
Epoch 365 Batch    5/23   train_loss = 0.353
Epoch 369 Batch   13/23   train_loss = 0.332
Epoch 373 Batch   21/23   train_loss = 0.351
Epoch 378 Batch    6/23   train_loss = 0.325
Epoch 382 Batch   14/23   train_loss = 0.336
Epoch 386 Batch   22/23   train_loss = 0.331
Epoch 391 Batch    7/23   train_loss = 0.322
Epoch 395 Batch   15/23   train_loss = 0.294
Epoch 400 Batch    0/23   train_loss = 0.295
Epoch 404 Batch    8/23   train_loss = 0.295
Epoch 408 Batch   16/23   train_loss = 0.267
Epoch 413 Batch    1/23   train_loss = 0.265
Epoch 417 Batch    9/23   train_loss = 0.274
Epoch 421 Batch   17/23   train_loss = 0.241
Epoch 426 Batch    2/23   train_loss = 0.250
Epoch 430 Batch   10/23   train_loss = 0.278
Epoch 434 Batch   18/23   train_loss = 0.245
Epoch 439 Batch    3/23   train_loss = 0.245
Epoch 443 Batch   11/23   train_loss = 0.248
Epoch 447 Batch   19/23   train_loss = 0.235
Epoch 452 Batch    4/23   train_loss = 0.244
Epoch 456 Batch   12/23   train_loss = 0.218
Epoch 460 Batch   20/23   train_loss = 0.208
Epoch 465 Batch    5/23   train_loss = 0.219
Epoch 469 Batch   13/23   train_loss = 0.217
Epoch 473 Batch   21/23   train_loss = 0.217
Epoch 478 Batch    6/23   train_loss = 0.203
Epoch 482 Batch   14/23   train_loss = 0.222
Epoch 486 Batch   22/23   train_loss = 0.212
Epoch 491 Batch    7/23   train_loss = 0.210
```

```
Epoch 495 Batch   15/23   train_loss = 0.185
Model Trained and Saved
```

Save Parameters

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

In [17]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
# Save parameters for checkpoint
helper.save_params((seq_length, save_dir))
```

Checkpoint

In [18]:
```
"""
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 [19]:
```
def get_tensors(loaded_graph):
    """
    Get input, initial state, final state, and probabilities tensor from <l
oaded_graph>
    :param loaded_graph: TensorFlow graph loaded from file
    :return: Tuple (InputTensor, InitialStateTensor, FinalStateTensor, Prob
sTensor)
    """
    # TODO: Implement Function
    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


"""
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 [20]:
```
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 v
alues
    :return: String of the predicted word
    """
    # TODO: Implement Function
    word = np.random.choice(list(int_to_vocab.values()), p=probabilities)
    return 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 [21]:
```
gen_length = 200
# 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_grap
h)

    # 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_len
gth:]]]
        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[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)
```

```
moe_szyslak: gee, i can't--
barney_gumble: hey!, i've been even come from" face and" and" bonfire"?
moe_szyslak: hey, barney. my name put on the way.
seymour_skinner: we got caught in this special?
flea:(aside) and you all hate that last night / bartender he doesn't stand th
ere and be able to bring her out!
homer_simpson:(gasp)" african princess."
gil_gunderson:(low voice) really?
marge_simpson: gimme that on the nose...
moe_szyslak:(sings) good king wenceslas looked out on their mouth."
moe_szyslak:(embarrassed) whoa, tha...(laughs) that, no. you're a fabulous ca
tch. see the day of speech never need your name, and on the bar, then you're
just tryin' to.
homer_simpson: well, you really think i could do these with marge?
lenny_leonard: quiet, we're great.
moe_szyslak: look, i really a little trick. i pictured everyone in their unde
rwear. the judge,
```

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. Luckly there's more data! As we mentioned in the begging 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.