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

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

    # vocab_to_int
    vocab = sorted(set(text))
    vocab_to_int = {c: i for i, c in enumerate(vocab)}


    # int_to_vocab
    int_to_vocab = dict(enumerate(vocab))

    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 th
e value is the token
    """
    # TODO: Implement Function
    punc_tokenizer = { '.':'||Period||', ',':'||Comma||', '"':'||Quotati
on', ';':'SemiColon', '!':'||Exclamation||',
                  '?':'||Question||', '(':'||LeftParen||', ')':'||RightP
aren||', '--':'||Dash||', '\n':'||Ret||'   }
    return punc_tokenizer

"""
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
 TensorFlow 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 n
etwork.')
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, shape = [None,None], name = 'input' )
    Targets = tf.placeholder(   tf.int32, shape = [None,None], name = 'targets' )
    LearningRate = tf.placeholder(   tf.float32, name = 'learning_rate' )
    return Input, Targets, LearningRate


"""
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

    # build basic LSTM "cell"/layer
    def build_lstm_cell(lstm_size, keep_prob):
        # basic LSTM layer
        lstm = tf.contrib.rnn.BasicLSTMCell(lstm_size)

        # add dropout
        drop = tf.contrib.rnn.DropoutWrapper(lstm, output_keep_prob=keep
_prob)
        return drop

    # create RNN "multi-cell" by stacking LSTM layers
    num_layers = 3 # arbitrary
    keep_prob = 0.8 # arbitrary
    multi_cell = tf.contrib.rnn.MultiRNNCell([build_lstm_cell(rnn_size,
keep_prob) for _ in range(num_layers)])
    initial_state = multi_cell.zero_state(batch_size, tf.float32)
    initial_state = tf.identity( initial_state, name='initial_state')



    return multi_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.
    """

    #print(input_data)
    #print("vocab_size: " + str(vocab_size) )
    #print("embed_dim: " + str(embed_dim) )
    embedding = tf.Variable(tf.random_uniform((vocab_size, embed_dim), -1, 1) )
    embed = tf.nn.embedding_lookup(embedding, input_data )
    #print(embed)

    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

    # Run each sequence step through the RNN and collect the outputs
    outputs, final_state = tf.nn.dynamic_rnn(cell, inputs, dtype = tf.fl
oat32 )
    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 [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
    #print( cell.output_size)
    #print( rnn_size)
    inputs = get_embed( input_data, vocab_size, embed_dim )
    rnn_output, final_state = build_rnn(cell, inputs)
    logits = tf.contrib.layers.fully_connected( rnn_output, num_outputs
= vocab_size, activation_fn=None)

    if cell.output_size != rnn_size:
        print("Warning: RNN cell size mismatch ")
        print("actual output size: " + str(cell.output_size) )
        print("expected size (rnn_size): " + str(rnn_size) )

    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.

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 [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
    #print("batch_size: " + str(batch_size))
    #print("seq_length: " + str(seq_length))
    #print(int_text)
    #print( "\n")

    # test input
    #int_text = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
    #print(int_text)
    #batch_size = 3
    #seq_length = 2

    # use solution from AnnaKaRNNa, modified for specified output format
    n_seqs = batch_size
    characters_per_batch = n_seqs * seq_length
    n_batches = len(int_text)//characters_per_batch

    # Keep only enough characters to make full batches
    int_text = int_text[:n_batches * characters_per_batch]

    # Reshape into n_seqs rows
    int_text = np.asarray( int_text )
    int_text = int_text.reshape((n_seqs, -1))
    print(int_text)

    # output array
    Z = np.zeros(  (n_batches, 2, batch_size, seq_length) )

    # loop through
    for n in range(0, int_text.shape[1], seq_length):
        # The features
        x = int_text[:, n:n+seq_length]
        if n == 0:
          x_save = x[:,0]  # save 1st column of x for final wrap-around
        #print(x)

        # The targets, shifted by one
        y = np.zeros_like(x)
        #y[:, :-1 ] = x[:, 1:]   # y up to but excluding last column = x starting at oolumn 1 (shift)
        y = np.roll(x,-1,axis = 1)
        if n+seq_length < int_text.shape[1]:
            x_shift =  int_text[:, n+seq_length]
            y[:,-1] = x_shift
        else:
            y[:,-1] =  np.roll(x_save,-1, axis= 0)
            #print('else')
```

```
        #print(y)
        Z[ n//seq_length, 0] = x
        Z[ n//seq_length, 1] = y


    #print(Z)
    return Z


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

```
[[   0    1    2 ...,   32   33   34]
 [  35   36   37 ...,   67   68   69]
 [  70   71   72 ...,  102  103  104]
 ...,
 [4375 4376 4377 ..., 4407 4408 4409]
 [4410 4411 4412 ..., 4442 4443 4444]
 [4445 4446 4447 ..., 4477 4478 4479]]
Tests Passed
```

In [26]:
```
# Number of Epochs
num_epochs = 1000
# Batch Size
batch_size = 128
# RNN Size
rnn_size = 512
# Embedding Dimension Size
embed_dim = 100
# Sequence Length
seq_length = 128
# Learning Rate
learning_rate = 0.005 #1e-2
# Show stats for every n number of batches
show_every_n_batches = 10

"""
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 [27]:
```
"""
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 [28]:
```
"""
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}'
.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')
```

```
[[3823 6772 3044 ..., 3169 6769 5963]
 [3185 6769 3923 ..., 1963 6315 5963]
 [1117 3573 6773 ...,   63 1208 4111]
 ...,
 [4134 1452 2800 ..., 6216 3759 2917]
 [5045 6774 6776 ...,  691  875 3923]
 [1338 1079   97 ..., 6776 6776 6746]]
Epoch   0 Batch    0/4   train_loss = 8.822
Epoch   2 Batch    2/4   train_loss = 6.099
Epoch   5 Batch    0/4   train_loss = 5.959
Epoch   7 Batch    2/4   train_loss = 5.971
Epoch  10 Batch    0/4   train_loss = 5.921
Epoch  12 Batch    2/4   train_loss = 5.955
Epoch  15 Batch    0/4   train_loss = 5.911
Epoch  17 Batch    2/4   train_loss = 5.949
Epoch  20 Batch    0/4   train_loss = 5.902
Epoch  22 Batch    2/4   train_loss = 5.946
Epoch  25 Batch    0/4   train_loss = 5.901
Epoch  27 Batch    2/4   train_loss = 5.944
Epoch  30 Batch    0/4   train_loss = 5.896
Epoch  32 Batch    2/4   train_loss = 5.936
Epoch  35 Batch    0/4   train_loss = 5.895
Epoch  37 Batch    2/4   train_loss = 5.934
Epoch  40 Batch    0/4   train_loss = 5.892
Epoch  42 Batch    2/4   train_loss = 5.931
Epoch  45 Batch    0/4   train_loss = 5.886
Epoch  47 Batch    2/4   train_loss = 5.929
Epoch  50 Batch    0/4   train_loss = 5.890
Epoch  52 Batch    2/4   train_loss = 5.922
Epoch  55 Batch    0/4   train_loss = 5.883
Epoch  57 Batch    2/4   train_loss = 5.924
Epoch  60 Batch    0/4   train_loss = 5.879
Epoch  62 Batch    2/4   train_loss = 5.921
Epoch  65 Batch    0/4   train_loss = 5.876
Epoch  67 Batch    2/4   train_loss = 5.913
Epoch  70 Batch    0/4   train_loss = 5.873
Epoch  72 Batch    2/4   train_loss = 5.909
Epoch  75 Batch    0/4   train_loss = 5.860
Epoch  77 Batch    2/4   train_loss = 5.894
Epoch  80 Batch    0/4   train_loss = 5.855
Epoch  82 Batch    2/4   train_loss = 5.888
Epoch  85 Batch    0/4   train_loss = 5.832
Epoch  87 Batch    2/4   train_loss = 5.867
Epoch  90 Batch    0/4   train_loss = 5.819
Epoch  92 Batch    2/4   train_loss = 5.857
Epoch  95 Batch    0/4   train_loss = 5.808
Epoch  97 Batch    2/4   train_loss = 5.831
Epoch 100 Batch    0/4   train_loss = 5.778
Epoch 102 Batch    2/4   train_loss = 5.802
Epoch 105 Batch    0/4   train_loss = 5.745
Epoch 107 Batch    2/4   train_loss = 5.762
Epoch 110 Batch    0/4   train_loss = 5.712
Epoch 112 Batch    2/4   train_loss = 5.730
Epoch 115 Batch    0/4   train_loss = 5.686
Epoch 117 Batch    2/4   train_loss = 5.698
Epoch 120 Batch    0/4   train_loss = 5.654
Epoch 122 Batch    2/4   train_loss = 5.675
```

```
Epoch 125 Batch    0/4   train_loss = 5.619
Epoch 127 Batch    2/4   train_loss = 5.665
Epoch 130 Batch    0/4   train_loss = 5.591
Epoch 132 Batch    2/4   train_loss = 5.606
Epoch 135 Batch    0/4   train_loss = 5.573
Epoch 137 Batch    2/4   train_loss = 5.582
Epoch 140 Batch    0/4   train_loss = 5.531
Epoch 142 Batch    2/4   train_loss = 5.543
Epoch 145 Batch    0/4   train_loss = 5.410
Epoch 147 Batch    2/4   train_loss = 5.347
Epoch 150 Batch    0/4   train_loss = 5.211
Epoch 152 Batch    2/4   train_loss = 5.165
Epoch 155 Batch    0/4   train_loss = 4.976
Epoch 157 Batch    2/4   train_loss = 4.921
Epoch 160 Batch    0/4   train_loss = 4.770
Epoch 162 Batch    2/4   train_loss = 4.711
Epoch 165 Batch    0/4   train_loss = 4.589
Epoch 167 Batch    2/4   train_loss = 4.516
Epoch 170 Batch    0/4   train_loss = 4.395
Epoch 172 Batch    2/4   train_loss = 4.323
Epoch 175 Batch    0/4   train_loss = 4.216
Epoch 177 Batch    2/4   train_loss = 4.179
Epoch 180 Batch    0/4   train_loss = 4.098
Epoch 182 Batch    2/4   train_loss = 4.049
Epoch 185 Batch    0/4   train_loss = 3.988
Epoch 187 Batch    2/4   train_loss = 3.918
Epoch 190 Batch    0/4   train_loss = 3.841
Epoch 192 Batch    2/4   train_loss = 3.770
Epoch 195 Batch    0/4   train_loss = 3.696
Epoch 197 Batch    2/4   train_loss = 3.660
Epoch 200 Batch    0/4   train_loss = 3.622
Epoch 202 Batch    2/4   train_loss = 3.585
Epoch 205 Batch    0/4   train_loss = 3.491
Epoch 207 Batch    2/4   train_loss = 3.459
Epoch 210 Batch    0/4   train_loss = 3.383
Epoch 212 Batch    2/4   train_loss = 3.334
Epoch 215 Batch    0/4   train_loss = 3.286
Epoch 217 Batch    2/4   train_loss = 3.264
Epoch 220 Batch    0/4   train_loss = 3.217
Epoch 222 Batch    2/4   train_loss = 3.148
Epoch 225 Batch    0/4   train_loss = 3.074
Epoch 227 Batch    2/4   train_loss = 3.027
Epoch 230 Batch    0/4   train_loss = 2.973
Epoch 232 Batch    2/4   train_loss = 2.940
Epoch 235 Batch    0/4   train_loss = 2.972
Epoch 237 Batch    2/4   train_loss = 2.872
Epoch 240 Batch    0/4   train_loss = 2.827
Epoch 242 Batch    2/4   train_loss = 2.732
Epoch 245 Batch    0/4   train_loss = 2.686
Epoch 247 Batch    2/4   train_loss = 2.609
Epoch 250 Batch    0/4   train_loss = 2.587
Epoch 252 Batch    2/4   train_loss = 2.524
Epoch 255 Batch    0/4   train_loss = 2.488
Epoch 257 Batch    2/4   train_loss = 2.485
Epoch 260 Batch    0/4   train_loss = 2.466
Epoch 262 Batch    2/4   train_loss = 2.346
Epoch 265 Batch    0/4   train_loss = 2.292
```

```
Epoch 267 Batch    2/4   train_loss = 2.217
Epoch 270 Batch    0/4   train_loss = 2.188
Epoch 272 Batch    2/4   train_loss = 2.125
Epoch 275 Batch    0/4   train_loss = 2.146
Epoch 277 Batch    2/4   train_loss = 2.121
Epoch 280 Batch    0/4   train_loss = 2.092
Epoch 282 Batch    2/4   train_loss = 1.989
Epoch 285 Batch    0/4   train_loss = 1.950
Epoch 287 Batch    2/4   train_loss = 1.882
Epoch 290 Batch    0/4   train_loss = 1.893
Epoch 292 Batch    2/4   train_loss = 1.819
Epoch 295 Batch    0/4   train_loss = 1.792
Epoch 297 Batch    2/4   train_loss = 1.712
Epoch 300 Batch    0/4   train_loss = 1.713
Epoch 302 Batch    2/4   train_loss = 1.647
Epoch 305 Batch    0/4   train_loss = 1.632
Epoch 307 Batch    2/4   train_loss = 1.543
Epoch 310 Batch    0/4   train_loss = 1.562
Epoch 312 Batch    2/4   train_loss = 1.504
Epoch 315 Batch    0/4   train_loss = 1.546
Epoch 317 Batch    2/4   train_loss = 1.474
Epoch 320 Batch    0/4   train_loss = 1.454
Epoch 322 Batch    2/4   train_loss = 1.341
Epoch 325 Batch    0/4   train_loss = 1.368
Epoch 327 Batch    2/4   train_loss = 1.323
Epoch 330 Batch    0/4   train_loss = 1.303
Epoch 332 Batch    2/4   train_loss = 1.207
Epoch 335 Batch    0/4   train_loss = 1.218
Epoch 337 Batch    2/4   train_loss = 1.141
Epoch 340 Batch    0/4   train_loss = 1.160
Epoch 342 Batch    2/4   train_loss = 1.126
Epoch 345 Batch    0/4   train_loss = 1.113
Epoch 347 Batch    2/4   train_loss = 1.031
Epoch 350 Batch    0/4   train_loss = 1.063
Epoch 352 Batch    2/4   train_loss = 1.001
Epoch 355 Batch    0/4   train_loss = 0.988
Epoch 357 Batch    2/4   train_loss = 0.935
Epoch 360 Batch    0/4   train_loss = 0.941
Epoch 362 Batch    2/4   train_loss = 0.887
Epoch 365 Batch    0/4   train_loss = 0.921
Epoch 367 Batch    2/4   train_loss = 0.836
Epoch 370 Batch    0/4   train_loss = 0.846
Epoch 372 Batch    2/4   train_loss = 0.788
Epoch 375 Batch    0/4   train_loss = 0.790
Epoch 377 Batch    2/4   train_loss = 0.747
Epoch 380 Batch    0/4   train_loss = 0.776
Epoch 382 Batch    2/4   train_loss = 0.727
Epoch 385 Batch    0/4   train_loss = 0.717
Epoch 387 Batch    2/4   train_loss = 0.663
Epoch 390 Batch    0/4   train_loss = 0.664
Epoch 392 Batch    2/4   train_loss = 0.607
Epoch 395 Batch    0/4   train_loss = 0.609
Epoch 397 Batch    2/4   train_loss = 0.572
Epoch 400 Batch    0/4   train_loss = 0.582
Epoch 402 Batch    2/4   train_loss = 0.535
Epoch 405 Batch    0/4   train_loss = 0.548
Epoch 407 Batch    2/4   train_loss = 0.525
```

```
Epoch 410 Batch    0/4   train_loss = 0.531
Epoch 412 Batch    2/4   train_loss = 0.489
Epoch 415 Batch    0/4   train_loss = 0.514
Epoch 417 Batch    2/4   train_loss = 0.474
Epoch 420 Batch    0/4   train_loss = 0.468
Epoch 422 Batch    2/4   train_loss = 0.432
Epoch 425 Batch    0/4   train_loss = 0.439
Epoch 427 Batch    2/4   train_loss = 0.412
Epoch 430 Batch    0/4   train_loss = 0.409
Epoch 432 Batch    2/4   train_loss = 0.383
Epoch 435 Batch    0/4   train_loss = 0.399
Epoch 437 Batch    2/4   train_loss = 0.363
Epoch 440 Batch    0/4   train_loss = 0.369
Epoch 442 Batch    2/4   train_loss = 0.342
Epoch 445 Batch    0/4   train_loss = 0.335
Epoch 447 Batch    2/4   train_loss = 0.314
Epoch 450 Batch    0/4   train_loss = 0.320
Epoch 452 Batch    2/4   train_loss = 0.306
Epoch 455 Batch    0/4   train_loss = 0.306
Epoch 457 Batch    2/4   train_loss = 0.290
Epoch 460 Batch    0/4   train_loss = 0.285
Epoch 462 Batch    2/4   train_loss = 0.271
Epoch 465 Batch    0/4   train_loss = 0.265
Epoch 467 Batch    2/4   train_loss = 0.265
Epoch 470 Batch    0/4   train_loss = 0.261
Epoch 472 Batch    2/4   train_loss = 0.245
Epoch 475 Batch    0/4   train_loss = 0.247
Epoch 477 Batch    2/4   train_loss = 0.233
Epoch 480 Batch    0/4   train_loss = 0.240
Epoch 482 Batch    2/4   train_loss = 0.226
Epoch 485 Batch    0/4   train_loss = 0.227
Epoch 487 Batch    2/4   train_loss = 0.212
Epoch 490 Batch    0/4   train_loss = 0.220
Epoch 492 Batch    2/4   train_loss = 0.205
Epoch 495 Batch    0/4   train_loss = 0.203
Epoch 497 Batch    2/4   train_loss = 0.194
Epoch 500 Batch    0/4   train_loss = 0.198
Epoch 502 Batch    2/4   train_loss = 0.191
Epoch 505 Batch    0/4   train_loss = 0.197
Epoch 507 Batch    2/4   train_loss = 0.187
Epoch 510 Batch    0/4   train_loss = 0.179
Epoch 512 Batch    2/4   train_loss = 0.173
Epoch 515 Batch    0/4   train_loss = 0.179
Epoch 517 Batch    2/4   train_loss = 0.168
Epoch 520 Batch    0/4   train_loss = 0.162
Epoch 522 Batch    2/4   train_loss = 0.160
Epoch 525 Batch    0/4   train_loss = 0.165
Epoch 527 Batch    2/4   train_loss = 0.153
Epoch 530 Batch    0/4   train_loss = 0.155
Epoch 532 Batch    2/4   train_loss = 0.146
Epoch 535 Batch    0/4   train_loss = 0.149
Epoch 537 Batch    2/4   train_loss = 0.138
Epoch 540 Batch    0/4   train_loss = 0.143
Epoch 542 Batch    2/4   train_loss = 0.138
Epoch 545 Batch    0/4   train_loss = 0.137
Epoch 547 Batch    2/4   train_loss = 0.131
Epoch 550 Batch    0/4   train_loss = 0.136
```

```
Epoch 552 Batch    2/4   train_loss = 0.133
Epoch 555 Batch    0/4   train_loss = 0.129
Epoch 557 Batch    2/4   train_loss = 0.126
Epoch 560 Batch    0/4   train_loss = 0.127
Epoch 562 Batch    2/4   train_loss = 0.128
Epoch 565 Batch    0/4   train_loss = 0.123
Epoch 567 Batch    2/4   train_loss = 0.120
Epoch 570 Batch    0/4   train_loss = 0.120
Epoch 572 Batch    2/4   train_loss = 0.121
Epoch 575 Batch    0/4   train_loss = 0.119
Epoch 577 Batch    2/4   train_loss = 0.114
Epoch 580 Batch    0/4   train_loss = 0.114
Epoch 582 Batch    2/4   train_loss = 0.117
Epoch 585 Batch    0/4   train_loss = 0.114
Epoch 587 Batch    2/4   train_loss = 0.112
Epoch 590 Batch    0/4   train_loss = 0.110
Epoch 592 Batch    2/4   train_loss = 0.107
Epoch 595 Batch    0/4   train_loss = 0.105
Epoch 597 Batch    2/4   train_loss = 0.106
Epoch 600 Batch    0/4   train_loss = 0.108
Epoch 602 Batch    2/4   train_loss = 0.104
Epoch 605 Batch    0/4   train_loss = 0.103
Epoch 607 Batch    2/4   train_loss = 0.098
Epoch 610 Batch    0/4   train_loss = 0.098
Epoch 612 Batch    2/4   train_loss = 0.100
Epoch 615 Batch    0/4   train_loss = 0.096
Epoch 617 Batch    2/4   train_loss = 0.098
Epoch 620 Batch    0/4   train_loss = 0.096
Epoch 622 Batch    2/4   train_loss = 0.100
Epoch 625 Batch    0/4   train_loss = 0.091
Epoch 627 Batch    2/4   train_loss = 0.091
Epoch 630 Batch    0/4   train_loss = 0.093
Epoch 632 Batch    2/4   train_loss = 0.092
Epoch 635 Batch    0/4   train_loss = 0.090
Epoch 637 Batch    2/4   train_loss = 0.086
Epoch 640 Batch    0/4   train_loss = 0.085
Epoch 642 Batch    2/4   train_loss = 0.083
Epoch 645 Batch    0/4   train_loss = 0.082
Epoch 647 Batch    2/4   train_loss = 0.081
Epoch 650 Batch    0/4   train_loss = 0.084
Epoch 652 Batch    2/4   train_loss = 0.085
Epoch 655 Batch    0/4   train_loss = 0.083
Epoch 657 Batch    2/4   train_loss = 0.079
Epoch 660 Batch    0/4   train_loss = 0.081
Epoch 662 Batch    2/4   train_loss = 0.082
Epoch 665 Batch    0/4   train_loss = 0.078
Epoch 667 Batch    2/4   train_loss = 0.073
Epoch 670 Batch    0/4   train_loss = 0.073
Epoch 672 Batch    2/4   train_loss = 0.074
Epoch 675 Batch    0/4   train_loss = 0.077
Epoch 677 Batch    2/4   train_loss = 0.076
Epoch 680 Batch    0/4   train_loss = 0.078
Epoch 682 Batch    2/4   train_loss = 0.077
Epoch 685 Batch    0/4   train_loss = 0.077
Epoch 687 Batch    2/4   train_loss = 0.076
Epoch 690 Batch    0/4   train_loss = 0.073
Epoch 692 Batch    2/4   train_loss = 0.073
```

```
Epoch 695 Batch    0/4   train_loss = 0.077
Epoch 697 Batch    2/4   train_loss = 0.074
Epoch 700 Batch    0/4   train_loss = 0.069
Epoch 702 Batch    2/4   train_loss = 0.070
Epoch 705 Batch    0/4   train_loss = 0.072
Epoch 707 Batch    2/4   train_loss = 0.069
Epoch 710 Batch    0/4   train_loss = 0.070
Epoch 712 Batch    2/4   train_loss = 0.070
Epoch 715 Batch    0/4   train_loss = 0.075
Epoch 717 Batch    2/4   train_loss = 0.067
Epoch 720 Batch    0/4   train_loss = 0.066
Epoch 722 Batch    2/4   train_loss = 0.066
Epoch 725 Batch    0/4   train_loss = 0.068
Epoch 727 Batch    2/4   train_loss = 0.069
Epoch 730 Batch    0/4   train_loss = 0.066
Epoch 732 Batch    2/4   train_loss = 0.068
Epoch 735 Batch    0/4   train_loss = 0.066
Epoch 737 Batch    2/4   train_loss = 0.066
Epoch 740 Batch    0/4   train_loss = 0.065
Epoch 742 Batch    2/4   train_loss = 0.065
Epoch 745 Batch    0/4   train_loss = 0.068
Epoch 747 Batch    2/4   train_loss = 0.061
Epoch 750 Batch    0/4   train_loss = 0.060
Epoch 752 Batch    2/4   train_loss = 0.062
Epoch 755 Batch    0/4   train_loss = 0.062
Epoch 757 Batch    2/4   train_loss = 0.062
Epoch 760 Batch    0/4   train_loss = 0.062
Epoch 762 Batch    2/4   train_loss = 0.057
Epoch 765 Batch    0/4   train_loss = 0.061
Epoch 767 Batch    2/4   train_loss = 0.061
Epoch 770 Batch    0/4   train_loss = 0.060
Epoch 772 Batch    2/4   train_loss = 0.060
Epoch 775 Batch    0/4   train_loss = 0.062
Epoch 777 Batch    2/4   train_loss = 0.058
Epoch 780 Batch    0/4   train_loss = 0.059
Epoch 782 Batch    2/4   train_loss = 0.058
Epoch 785 Batch    0/4   train_loss = 0.056
Epoch 787 Batch    2/4   train_loss = 0.061
Epoch 790 Batch    0/4   train_loss = 0.062
Epoch 792 Batch    2/4   train_loss = 0.057
Epoch 795 Batch    0/4   train_loss = 0.057
Epoch 797 Batch    2/4   train_loss = 0.061
Epoch 800 Batch    0/4   train_loss = 0.059
Epoch 802 Batch    2/4   train_loss = 0.062
Epoch 805 Batch    0/4   train_loss = 0.061
Epoch 807 Batch    2/4   train_loss = 0.059
Epoch 810 Batch    0/4   train_loss = 0.058
Epoch 812 Batch    2/4   train_loss = 0.061
Epoch 815 Batch    0/4   train_loss = 0.062
Epoch 817 Batch    2/4   train_loss = 0.058
Epoch 820 Batch    0/4   train_loss = 0.057
Epoch 822 Batch    2/4   train_loss = 0.058
Epoch 825 Batch    0/4   train_loss = 0.060
Epoch 827 Batch    2/4   train_loss = 0.060
Epoch 830 Batch    0/4   train_loss = 0.058
Epoch 832 Batch    2/4   train_loss = 0.057
Epoch 835 Batch    0/4   train_loss = 0.057
```

```
Epoch 837 Batch    2/4   train_loss = 0.055
Epoch 840 Batch    0/4   train_loss = 0.056
Epoch 842 Batch    2/4   train_loss = 0.061
Epoch 845 Batch    0/4   train_loss = 0.057
Epoch 847 Batch    2/4   train_loss = 0.058
Epoch 850 Batch    0/4   train_loss = 0.055
Epoch 852 Batch    2/4   train_loss = 0.055
Epoch 855 Batch    0/4   train_loss = 0.056
Epoch 857 Batch    2/4   train_loss = 0.053
Epoch 860 Batch    0/4   train_loss = 0.055
Epoch 862 Batch    2/4   train_loss = 0.052
Epoch 865 Batch    0/4   train_loss = 0.056
Epoch 867 Batch    2/4   train_loss = 0.055
Epoch 870 Batch    0/4   train_loss = 0.053
Epoch 872 Batch    2/4   train_loss = 0.055
Epoch 875 Batch    0/4   train_loss = 0.055
Epoch 877 Batch    2/4   train_loss = 0.057
Epoch 880 Batch    0/4   train_loss = 0.053
Epoch 882 Batch    2/4   train_loss = 0.053
Epoch 885 Batch    0/4   train_loss = 0.057
Epoch 887 Batch    2/4   train_loss = 0.052
Epoch 890 Batch    0/4   train_loss = 0.051
Epoch 892 Batch    2/4   train_loss = 0.053
Epoch 895 Batch    0/4   train_loss = 0.053
Epoch 897 Batch    2/4   train_loss = 0.053
Epoch 900 Batch    0/4   train_loss = 0.051
Epoch 902 Batch    2/4   train_loss = 0.053
Epoch 905 Batch    0/4   train_loss = 0.054
Epoch 907 Batch    2/4   train_loss = 0.051
Epoch 910 Batch    0/4   train_loss = 0.054
Epoch 912 Batch    2/4   train_loss = 0.049
Epoch 915 Batch    0/4   train_loss = 0.051
Epoch 917 Batch    2/4   train_loss = 0.051
Epoch 920 Batch    0/4   train_loss = 0.049
Epoch 922 Batch    2/4   train_loss = 0.052
Epoch 925 Batch    0/4   train_loss = 0.053
Epoch 927 Batch    2/4   train_loss = 0.051
Epoch 930 Batch    0/4   train_loss = 0.050
Epoch 932 Batch    2/4   train_loss = 0.052
Epoch 935 Batch    0/4   train_loss = 0.050
Epoch 937 Batch    2/4   train_loss = 0.049
Epoch 940 Batch    0/4   train_loss = 0.052
Epoch 942 Batch    2/4   train_loss = 0.052
Epoch 945 Batch    0/4   train_loss = 0.052
Epoch 947 Batch    2/4   train_loss = 0.053
Epoch 950 Batch    0/4   train_loss = 0.055
Epoch 952 Batch    2/4   train_loss = 0.051
Epoch 955 Batch    0/4   train_loss = 0.051
Epoch 957 Batch    2/4   train_loss = 0.052
Epoch 960 Batch    0/4   train_loss = 0.050
Epoch 962 Batch    2/4   train_loss = 0.052
Epoch 965 Batch    0/4   train_loss = 0.051
Epoch 967 Batch    2/4   train_loss = 0.048
Epoch 970 Batch    0/4   train_loss = 0.052
Epoch 972 Batch    2/4   train_loss = 0.052
Epoch 975 Batch    0/4   train_loss = 0.053
Epoch 977 Batch    2/4   train_loss = 0.058
```

```
Epoch 980 Batch    0/4   train_loss = 0.051
Epoch 982 Batch    2/4   train_loss = 0.052
Epoch 985 Batch    0/4   train_loss = 0.059
Epoch 987 Batch    2/4   train_loss = 0.056
Epoch 990 Batch    0/4   train_loss = 0.055
Epoch 992 Batch    2/4   train_loss = 0.051
Epoch 995 Batch    0/4   train_loss = 0.052
Epoch 997 Batch    2/4   train_loss = 0.053
Model Trained and Saved
```

Save Parameters

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

In [29]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
# Save parameters for checkpoint
helper.save_params((seq_length, save_dir))
```

Checkpoint

In [30]:
```
"""
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 [31]:
```
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)
    """
    # 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 [35]:
```
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
    #print(probabilities)
    #print(np.argmax(probabilities) )
    pred_word = int_to_vocab[ np.argmax(probabilities)  ]
    return pred_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 [41]:
```
gen_length = 400 #200
# homer_simpson, moe_szyslak, or Barney_Gumble
prime_word = 'homer_simpson' #'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[dyn_seq_length-1], int_to_vo
cab)

        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)
```

```
homer_simpson: helllp me!
moe_szyslak: that uh, that's me. i've been taking ventriloquism lesson
s.(nervous laugh)
homer_simpson: help me or kill me!
moe_szyslak:(looking for approval) heh? heh?
moe_szyslak: now, let dr. moe cure what ails you.
marge_simpson: mm, there's something odd about this beer.
marge_simpson:(talk-sings) it tastes like... cuddling! / it tastes like
clean clothes!
marge_simpson: it tastes like hot steaming cocoa mixed with rainbows...
moe_szyslak:(surprised/thrilled) it does?
lenny_leonard:(sings) full-bodied...
carl_carlson:(sings) full-blooded...
barney_gumble:(sings) greetings, moe. it's,... and i am i live for than
king you done up. i'm why hey, uh, i've gotta pour him into an last bow
l!
homer_simpson:(cheery) joey gone swigmore so rude.(looks like door) act
ually, and looks like you see? it's funny and the moe-lennium.
moe_szyslak: she took a drink.
moe_szyslak: yeah, so we're under the of the" 'cause it might be time?
(chuckles)
moe_szyslak: it's too much of my friends!
marge_simpson: oh.
moe_szyslak: thank you-- 'cause i'm on arrest for a mr. um.
homer_simpson: woo! hey, whatever... at great. it's not song.
lenny_leonard: thank.? every thing sound like these left to make our wa
y in the phrase oh and kill... i'm going to make the school desperate t
o" drink it like that.
lenny_leonard: what means that was over.(sniffs) oh, god, it's true! 'c
ause no-- it's official.(ice card) we're not got with 'em of you, let's
going to pursue the drinks for my life.(sobs)


moe_szyslak:(cutting right on his stool.
barney_gumble: hey, better me, who's your senator coming out for someth
ing. i'm one's an soul at that.
homer_simpson: aw, if you just gettin' homesick in my best friend.
```

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.