TV Script Generation

In this project, you'll generate your own [Seinfeld (https://en.wikipedia.org/wiki/Seinfeld)](https://en.wikipedia.org/wiki/Seinfeld) TV scripts using RNNs. You'll be using part of the [Seinfeld dataset (https://www.kaggle.com/thec03u5/seinfeld-chronicles#scripts.csv)](https://www.kaggle.com/thec03u5/seinfeld-chronicles#scripts.csv) of scripts from 9 seasons. The Neural Network you'll build will generate a new ,"fake" TV script, based on patterns it recognizes in this training data.

Get the Data

The data is already provided for you in `./data/Seinfeld_Scripts.txt` and you're encouraged to open that file and look at the text.

> - As a first step, we'll load in this data and look at some samples.
> - Then, you'll be tasked with defining and training an RNN to generate a new script!

In [1]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
# load in data
import helper
data_dir = './data/Seinfeld_Scripts.txt'
text = helper.load_data(data_dir)
```

Explore the Data

Play around with `view_line_range` to view different parts of the data. This will give you a sense of the data you'll be working with. You can see, for example, that it is all lowercase text, and each new line of dialogue is separated by a newline character `\n`.

In [2]:
```
view_line_range = (0, 10)

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
import numpy as np

print('Dataset Stats')
print('Roughly the number of unique words: {}'.format(len({word: None for word
in text.split()})))

lines = text.split('\n')
print('Number of lines: {}'.format(len(lines)))
word_count_line = [len(line.split()) for line in lines]
print('Average number of words in each line: {}'.format(np.average(word_count_
line)))

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

```
Dataset Stats
Roughly the number of unique words: 46367
Number of lines: 109233
Average number of words in each line: 5.544240293684143

The lines 0 to 10:
jerry: do you know what this is all about? do you know, why were here? to be
out, this is out...and out is one of the single most enjoyable experiences of
life. people...did you ever hear people talking about we should go out? this
is what theyre talking about...this whole thing, were all out now, no one is
home. not one person here is home, were all out! there are people trying to f
ind us, they dont know where we are. (on an imaginary phone) did you ring?, i
cant find him. where did he go? he didnt tell me where he was going. he must
have gone out. you wanna go out you get ready, you pick out the clothes, righ
t? you take the shower, you get all ready, get the cash, get your friends, th
e car, the spot, the reservation...then youre standing around, what do you d
o? you go we gotta be getting back. once youre out, you wanna get back! you w
anna go to sleep, you wanna get up, you wanna go out again tomorrow, right? w
here ever you are in life, its my feeling, youve gotta go.

jerry: (pointing at georges shirt) see, to me, that button is in the worst po
ssible spot. the second button literally makes or breaks the shirt, look at i
t. its too high! its in no-mans-land. you look like you live with your mothe
r.

george: are you through?

jerry: you do of course try on, when you buy?

george: yes, it was purple, i liked it, i dont actually recall considering th
e buttons.
```

Implement Pre-processing Functions

The first thing to do to any dataset is pre-processing. Implement the following pre-processing 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`

In [3]:
```
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 = tuple(set(text))
    vocab_to_int = {word: ii for ii, word in enumerate(vocab)}
    int_to_vocab = dict(enumerate(vocab))

    # return tuple
    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 can create multiple ids for the same word. For example, "bye" and "bye!" would generate two different word ids.

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 tokenize the symbols and add the delimiter (space) around it. This separates each symbols as its own word, making it easier for the neural network to predict the next word. Make sure you don't use a value that could be confused as a word; for example, instead of using the value "dash", try using something like "||dash||".

In [4]:
```
def token_lookup():
    """
    Generate a dict to turn punctuation into a token.
    :return: Tokenized dictionary where the key is the punctuation and the val
ue is the token
    """
    # TODO: Implement Function
    symbols_dict = {}
    symbols_dict['.'] = '||period||'
    symbols_dict[','] = '||comma||'
    symbols_dict['"'] = '||quotation_mark||'
    symbols_dict[';'] = '||semicolon||'
    symbols_dict['!'] = '||exclamation_mark||'
    symbols_dict['?'] = '||question_mark||'
    symbols_dict['('] = '||left_parentheses||'
    symbols_dict[')'] = '||right_parentheses||'
    symbols_dict['-'] = '||dash||'
    symbols_dict['\n'] = '||return||'

    return symbols_dict

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

```
Tests Passed
```

Pre-process all the data and save it

Running the code cell below will pre-process all the data and save it to file. You're encouraged to look at the code for `preprocess_and_save_data` in the `helpers.py` file to see what it's doing in detail, but you do not need to change this code.

In [5]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
# pre-process training 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 [1]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import helper
import problem_unittests as tests

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

Build the Neural Network

In this section, you'll build the components necessary to build an RNN by implementing the RNN Module and forward and backpropagation functions.

Check Access to GPU

In [2]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import torch

# Check for a GPU
train_on_gpu = torch.cuda.is_available()
if not train_on_gpu:
    print('No GPU found. Please use a GPU to train your neural network.')
```

```
No GPU found. Please use a GPU to train your neural network.
```

Input

Let's start with the preprocessed input data. We'll use [TensorDataset (http://pytorch.org/docs/master/data.html#torch.utils.data.TensorDataset)](http://pytorch.org/docs/master/data.html#torch.utils.data.TensorDataset) to provide a known format to our dataset; in combination with [DataLoader (http://pytorch.org/docs/master/data.html#torch.utils.data.DataLoader)](http://pytorch.org/docs/master/data.html#torch.utils.data.DataLoader), it will handle batching, shuffling, and other dataset iteration functions.

You can create data with TensorDataset by passing in feature and target tensors. Then create a DataLoader as usual.

```
data = TensorDataset(feature_tensors, target_tensors)
data_loader = torch.utils.data.DataLoader(data,
                                          batch_size=batch_size)
```

Batching

Implement the `batch_data` function to batch `words` data into chunks of size `batch_size` using the `TensorDataset` and `DataLoader` classes.

> You can batch words using the DataLoader, but it will be up to you to create `feature_tensors` and `target_tensors` of the correct size and content for a given `sequence_length`.

For example, say we have these as input:

```
words = [1, 2, 3, 4, 5, 6, 7]
sequence_length = 4
```

Your first `feature_tensor` should contain the values:

```
[1, 2, 3, 4]
```

And the corresponding `target_tensor` should just be the next "word"/tokenized word value:

```
5
```

This should continue with the second `feature_tensor`, `target_tensor` being:

```
[2, 3, 4, 5]  # features
6             # target
```

In [7]:
```
from torch.utils.data import TensorDataset, DataLoader
import numpy as np

def batch_data(words, sequence_length, batch_size):
    """
    Batch the neural network data using DataLoader
    :param words: The word ids of the TV scripts
    :param sequence_length: The sequence length of each batch
    :param batch_size: The size of each batch; the number of sequences in a ba
tch
    :return: DataLoader with batched data
    """
    # TODO: Implement function
    batch_size_total = batch_size * sequence_length
    n_batches = len(words)//batch_size_total
    max_batch_size = len(words)//sequence_length

    features = np.zeros((max_batch_size, sequence_length), dtype=int)
    targets = np.zeros((max_batch_size,), dtype=int)

    npArrayWords = np.array(words)

    index = 0
    for n in range(max_batch_size):
        features[n,:] = npArrayWords[index:index+sequence_length]

        if ( index+sequence_length < len(npArrayWords) ):
            targets[n] = npArrayWords[index+sequence_length]
        else:
            targets[n] = npArrayWords[0] #last element target equals to first
 word

        index += sequence_length

    data = TensorDataset(torch.from_numpy(features), torch.from_numpy(targets
))
    data_loader = DataLoader(data, shuffle=True, batch_size=batch_size)
    return data_loader

# there is no test for this function, but you are encouraged to create
# print statements and tests of your own
```

Test your dataloader

You'll have to modify this code to test a batching function, but it should look fairly similar.

Below, we're generating some test text data and defining a dataloader using the function you defined, above. Then, we are getting some sample batch of inputs `sample_x` and targets `sample_y` from our dataloader.

Your code should return something like the following (likely in a different order, if you shuffled your data):

```
torch.Size([10, 5])
tensor([[ 28,  29,  30,  31,  32],
        [ 21,  22,  23,  24,  25],
        [ 17,  18,  19,  20,  21],
        [ 34,  35,  36,  37,  38],
        [ 11,  12,  13,  14,  15],
        [ 23,  24,  25,  26,  27],
        [  6,   7,   8,   9,  10],
        [ 38,  39,  40,  41,  42],
        [ 25,  26,  27,  28,  29],
        [  7,   8,   9,  10,  11]])

torch.Size([10])
tensor([ 33,  26,  22,  39,  16,  28,  11,  43,  30,  12])
```

Sizes

Your sample_x should be of size `(batch_size, sequence_length)` or (10, 5) in this case and sample_y should just have one dimension: batch_size (10).

Values

You should also notice that the targets, sample_y, are the *next* value in the ordered test_text data. So, for an input sequence `[ 28,  29,  30,  31,  32]` that ends with the value `32` , the corresponding output should be `33` .

In [9]:
```
# test dataloader

test_text = range(50)
t_loader = batch_data(test_text, sequence_length=5, batch_size=10)

data_iter = iter(t_loader)
sample_x, sample_y = data_iter.next()

print(sample_x.shape)
print(sample_x)
print()
print(sample_y.shape)
print(sample_y)
```

```
torch.Size([10, 5])
tensor([[  5,   6,   7,   8,   9],
        [ 25,  26,  27,  28,  29],
        [ 20,  21,  22,  23,  24],
        [ 15,  16,  17,  18,  19],
        [ 35,  36,  37,  38,  39],
        [  0,   1,   2,   3,   4],
        [ 30,  31,  32,  33,  34],
        [ 45,  46,  47,  48,  49],
        [ 10,  11,  12,  13,  14],
        [ 40,  41,  42,  43,  44]])

torch.Size([10])
tensor([ 10,  30,  25,  20,  40,   5,  35,   0,  15,  45])
```

Build the Neural Network

Implement an RNN using PyTorch's [Module class (http://pytorch.org/docs/master/nn.html#torch.nn.Module)](http://pytorch.org/docs/master/nn.html#torch.nn.Module). You may choose to use a GRU or an LSTM. To complete the RNN, you'll have to implement the following functions for the class:

- `__init__` - The initialize function.
- `init_hidden` - The initialization function for an LSTM/GRU hidden state
- `forward` - Forward propagation function.

The initialize function should create the layers of the neural network and save them to the class. The forward propagation function will use these layers to run forward propagation and generate an output and a hidden state.

The output of this model should be the *last* batch of word scores after a complete sequence has been processed. That is, for each input sequence of words, we only want to output the word scores for a single, most likely, next word.

Hints

1. Make sure to stack the outputs of the lstm to pass to your fully-connected layer, you can do this with `lstm_output = lstm_output.contiguous().view(-1, self.hidden_dim)`
2. You can get the last batch of word scores by shaping the output of the final, fully-connected layer like so:

```
# reshape into (batch_size, seq_length, output_size)
output = output.view(batch_size, -1, self.output_size)
# get last batch
out = output[:, -1]
```

In [3]:
```
import torch.nn as nn

class RNN(nn.Module):

    def __init__(self, vocab_size, output_size, embedding_dim, hidden_dim, n_l
ayers, dropout=0.5):
        """
        Initialize the PyTorch RNN Module
        :param vocab_size: The number of input dimensions of the neural networ
k (the size of the vocabulary)
        :param output_size: The number of output dimensions of the neural netw
ork
        :param embedding_dim: The size of embeddings, should you choose to use
them
        :param hidden_dim: The size of the hidden layer outputs
        :param dropout: dropout to add in between LSTM/GRU layers
        """
        super(RNN, self).__init__()
        # TODO: Implement function

        # set class variables
        self.output_size = output_size
        self.n_layers = n_layers
        self.hidden_dim = hidden_dim

        # define model layers
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, n_layers,
                            dropout=dropout, batch_first=True)

        # dropout layer
        self.dropout = nn.Dropout(dropout)

        # linear and sigmoid layers
        self.fc = nn.Linear(hidden_dim, output_size)

    def forward(self, nn_input, hidden):
        """
        Forward propagation of the neural network
        :param nn_input: The input to the neural network
        :param hidden: The hidden state
        :return: Two Tensors, the output of the neural network and the latest
 hidden state
        """
        # TODO: Implement function
        batch_size = nn_input.size(0)

        # embeddings and lstm_out
        nn_input = nn_input.long()
        embeds = self.embedding(nn_input)
        lstm_out, hidden = self.lstm(embeds, hidden)

        # stack up lstm outputs
        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

        # dropout and fully-connected layer
```

```
        out = self.dropout(lstm_out)
        out = self.fc(out)

        # reshape into (batch_size, seq_length, output_size)
        output = out.view(batch_size, -1, self.output_size)
        # get last batch
        output = output[:, -1]
        return output, hidden


    def init_hidden(self, batch_size):
        '''
        Initialize the hidden state of an LSTM/GRU
        :param batch_size: The batch_size of the hidden state
        :return: hidden state of dims (n_layers, batch_size, hidden_dim)
        '''
        # Implement function

        # initialize hidden state with zero weights, and move to GPU if availa
ble
        weight = next(self.parameters()).data

        if (train_on_gpu):
            hidden = (weight.new(self.n_layers, batch_size, self.hidden_dim).z
ero_().cuda(),
                  weight.new(self.n_layers, batch_size, self.hidden_dim).zero_
().cuda())
        else:
            hidden = (weight.new(self.n_layers, batch_size, self.hidden_dim).z
ero_(),
                      weight.new(self.n_layers, batch_size, self.hidden_dim).z
ero_())

        return hidden

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_rnn(RNN, train_on_gpu)
```

```
Tests Passed
```

Define forward and backpropagation

Use the RNN class you implemented to apply forward and back propagation. This function will be called, iteratively, in the training loop as follows:

```
loss = forward_back_prop(decoder, decoder_optimizer, criterion, inp, target)
```

And it should return the average loss over a batch and the hidden state returned by a call to `RNN(inp, hidden)`. Recall that you can get this loss by computing it, as usual, and calling `loss.item()`.

If a GPU is available, you should move your data to that GPU device, here.

In [11]:
```
def forward_back_prop(rnn, optimizer, criterion, inp, target, hidden):
    """
    Forward and backward propagation on the neural network
    :param rnn: The PyTorch Module that holds the neural network
    :param optimizer: The PyTorch optimizer for the neural network
    :param criterion: The PyTorch loss function
    :param inp: A batch of input to the neural network
    :param target: The target output for the batch of input
    :return: The loss and the latest hidden state Tensor
    """
    # TODO: Implement Function

    # move data to GPU, if available
    if(train_on_gpu):
        inp, target = inp.cuda(), target.cuda()

    # Creating new variables for the hidden state, otherwise
    # we'd backprop through the entire training history
    hidden = tuple([each.data for each in hidden])

    # zero accumulated gradients
    rnn.zero_grad()

    # get the output from the model
    output, hidden = rnn(inp, hidden)

    # perform backpropagation and optimization

    # calculate the loss and perform backprop
    loss = criterion(output.squeeze(), target)
    loss.backward()

    # `clip_grad_norm` helps prevent the exploding gradient problem in RNNs /
 LSTMs.
    #  gradient clipping
    clip=5
    nn.utils.clip_grad_norm_(rnn.parameters(), clip)
    optimizer.step()

    # return the loss over a batch and the hidden state produced by our model
    return loss.item(), hidden

# Note that these tests aren't completely extensive.
# they are here to act as general checks on the expected outputs of your funct
ions
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_forward_back_prop(RNN, forward_back_prop, train_on_gpu)
```

```
Tests Passed
```

Neural Network Training

With the structure of the network complete and data ready to be fed in the neural network, it's time to train it.

Train Loop

The training loop is implemented for you in the `train_decoder` function. This function will train the network over all the batches for the number of epochs given. The model progress will be shown every number of batches. This number is set with the `show_every_n_batches` parameter. You'll set this parameter along with other parameters in the next section.

In [12]:
```
from workspace_utils import keep_awake

"""
DON'T MODIFY ANYTHING IN THIS CELL
"""

def train_rnn(rnn, batch_size, optimizer, criterion, n_epochs, show_every_n_ba
tches=100):

    batch_losses = []

    rnn.train()

    print("Training for %d epoch(s)..." % n_epochs)
    for epoch_i in keep_awake(range(1, n_epochs + 1)):

        # initialize hidden state
        hidden = rnn.init_hidden(batch_size)

        for batch_i, (inputs, labels) in enumerate(train_loader, 1):

            # make sure you iterate over completely full batches, only
            n_batches = len(train_loader.dataset)//batch_size
            if(batch_i > n_batches):
                break

            # forward, back prop
            loss, hidden = forward_back_prop(rnn, optimizer, criterion, inputs
, labels, hidden)
            # record loss
            batch_losses.append(loss)

            # record valid loss per iteration
            valid_loss_per_iteration = np.average(batch_losses)

            # printing loss stats
            if batch_i % show_every_n_batches == 0:
                print('Epoch: {:>4}/{:<4}  Loss: {}\n'.format(
                    epoch_i, n_epochs, np.average(batch_losses)))
                batch_losses = []

    # returns a trained rnn
    return rnn
```

Hyperparameters

Set and train the neural network with the following parameters:

- Set `sequence_length` to the length of a sequence.
- Set `batch_size` to the batch size.
- Set `num_epochs` to the number of epochs to train for.
- Set `learning_rate` to the learning rate for an Adam optimizer.
- Set `vocab_size` to the number of unique tokens in our vocabulary.
- Set `output_size` to the desired size of the output.
- Set `embedding_dim` to the embedding dimension; smaller than the vocab_size.
- Set `hidden_dim` to the hidden dimension of your RNN.
- Set `n_layers` to the number of layers/cells in your RNN.
- Set `show_every_n_batches` to the number of batches at which the neural network should print progress.

If the network isn't getting the desired results, tweak these parameters and/or the layers in the `RNN` class.

In [9]:
```
# Data params
# Sequence Length
sequence_length = 6  #Average number of words in each line: 5.544240293684143
# Batch Size
batch_size = 1000

# data loader - do not change
train_loader = batch_data(int_text, sequence_length, batch_size)
```

In [10]:
```
# Training parameters
# Number of Epochs
num_epochs = 20
# Learning Rate
learning_rate = 0.001

# Model parameters
# Vocab size
vocab_size = len(vocab_to_int)
# Output size
# plus one for the SPECIAL_WORDS "<PAD>" in the vocab
output_size = len(set(int_text))+1
# Embedding Dimension
embedding_dim = 200
# Hidden Dimension
hidden_dim = 512
# Number of RNN Layers
n_layers = 2

# Show stats for every n number of batches
show_every_n_batches = 100
```

Train

In the next cell, you'll train the neural network on the pre-processed data. If you have a hard time getting a good loss, you may consider changing your hyperparameters. In general, you may get better results with larger hidden and n_layer dimensions, but larger models take a longer time to train.

> **You should aim for a loss less than 3.5.**

You should also experiment with different sequence lengths, which determine the size of the long range dependencies that a model can learn.

In [15]:
```
from workspace_utils import active_session

with active_session():
    """
    DON'T MODIFY ANYTHING IN THIS CELL
    """

    # create model and move to gpu if available
    rnn = RNN(vocab_size, output_size, embedding_dim, hidden_dim, n_layers, dr
opout=0.5)
    if train_on_gpu:
        rnn.cuda()

    # defining loss and optimization functions for training
    optimizer = torch.optim.Adam(rnn.parameters(), lr=learning_rate)
    criterion = nn.CrossEntropyLoss()

    # training the model
    trained_rnn = train_rnn(rnn, batch_size, optimizer, criterion, num_epochs,
show_every_n_batches)

    # saving the trained model
    helper.save_model('./save/trained_rnn', trained_rnn)
    print('Model Trained and Saved')
```

```
Training for 20 epoch(s)...
Epoch:    1/20    Loss: 5.860833530426025

Epoch:    2/20    Loss: 4.961379205858385

Epoch:    3/20    Loss: 4.646563133677921

Epoch:    4/20    Loss: 4.430735987585944

Epoch:    5/20    Loss: 4.287778867257608

Epoch:    6/20    Loss: 4.167826734684609

Epoch:    7/20    Loss: 4.050776998738985

Epoch:    8/20    Loss: 3.9656474509754696

Epoch:    9/20    Loss: 3.858438659358669

Epoch:   10/20    Loss: 3.7810412081512244

Epoch:   11/20    Loss: 3.68198374316499

Epoch:   12/20    Loss: 3.5995311865935453

Epoch:   13/20    Loss: 3.5094605136562036

Epoch:   14/20    Loss: 3.4322989969640165

Epoch:   15/20    Loss: 3.3570909902856156

Epoch:   16/20    Loss: 3.274692699715898

Epoch:   17/20    Loss: 3.20470113689835

Epoch:   18/20    Loss: 3.1312133283228487

Epoch:   19/20    Loss: 3.073596733647424

Epoch:   20/20    Loss: 3.0001834727622367

Model Trained and Saved

/opt/conda/lib/python3.6/site-packages/torch/serialization.py:193: UserWarnin
g: Couldn't retrieve source code for container of type RNN. It won't be check
ed for correctness upon loading.
  "type " + obj.__name__ + ". It won't be checked "
```

Question: How did you decide on your model hyperparameters?

For example, did you try different sequence_lengths and find that one size made the model converge faster? What about your hidden_dim and n_layers; how did you decide on those?

Answer: I have decided my model hyperparameters based on the following observations:

1. For the sequence_length, it is a value of 6 because the average number of words in each line in the Seinfield script is 5.54 words.
2. I have tried increase the number of words (e.g. to 7-10 words) which did not improve on how fast the training loss converge. In fact, increasing the number of words from 6 seems to slow down on how fast the training loss converge.
3. The batch_size is 1000 which is limited by the available memory for running the workspace in CPU mode (I have decided to run this project using CPU in order to save GPU hours for later projects).
4. The num_epochs is 20 which is sufficient for the loss to fall below 3.5 while still decreasing at the last epoch.
5. The learning_rate is 0.001 which is not too little for the training loss to decrease below 3.5 after running 20 epochs.
6. The embedding_dim is 200 because study found performance increase as the embedding size increase until it reaches 200.
7. The hidden_dim is 512. Initially, the hidden_dim is 256 which is not enough for the training loss to get below 3.5 after 20 epochs. Therefore, it is increased by a factor of 2, to hidden_dim = 512 which is a good value for achieving training loss below 3.5 after 20 epochs.
8. The n_layers is 2 which is sufficient for our purpose to achieve loss less than 3.5. If n_layers is increased to 3, this will improve the accuracy but it will increase the training time (we only have limited number of GPU hours).

Checkpoint

After running the above training cell, your model will be saved by name, `trained_rnn` , and if you save your notebook progress, **you can pause here and come back to this code at another time**. You can resume your progress by running the next cell, which will load in our word:id dictionaries *and* load in your saved model by name!

In [4]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import torch
import helper
import problem_unittests as tests

_, vocab_to_int, int_to_vocab, token_dict = helper.load_preprocess()
trained_rnn = helper.load_model('./save/trained_rnn')
```

Generate TV Script

With the network trained and saved, you'll use it to generate a new, "fake" Seinfeld TV script in this section.

Generate Text

To generate the text, the network needs to start with a single word and repeat its predictions until it reaches a set length. You'll be using the `generate` function to do this. It takes a word id to start with, `prime_id`, and generates a set length of text, `predict_len`. Also note that it uses topk sampling to introduce some randomness in choosing the most likely next word, given an output set of word scores!

In [5]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
import torch.nn.functional as F

def generate(rnn, prime_id, int_to_vocab, token_dict, pad_value, predict_len=1
00):
    """
    Generate text using the neural network
    :param decoder: The PyTorch Module that holds the trained neural network
    :param prime_id: The word id to start the first prediction
    :param int_to_vocab: Dict of word id keys to word values
    :param token_dict: Dict of puncuation tokens keys to puncuation values
    :param pad_value: The value used to pad a sequence
    :param predict_len: The length of text to generate
    :return: The generated text
    """
    rnn.eval()

    # create a sequence (batch_size=1) with the prime_id
    current_seq = np.full((1, sequence_length), pad_value)
    current_seq[-1][-1] = prime_id
    predicted = [int_to_vocab[prime_id]]

    for _ in range(predict_len):
        if train_on_gpu:
            current_seq = torch.LongTensor(current_seq).cuda()
        else:
            current_seq = torch.LongTensor(current_seq)

        # initialize the hidden state
        hidden = rnn.init_hidden(current_seq.size(0))

        # get the output of the rnn
        output, _ = rnn(current_seq, hidden)

        # get the next word probabilities
        p = F.softmax(output, dim=1).data
        if(train_on_gpu):
            p = p.cpu() # move to cpu

        # use top_k sampling to get the index of the next word
        top_k = 5
        p, top_i = p.topk(top_k)
        top_i = top_i.numpy().squeeze()

        # select the likely next word index with some element of randomness
        p = p.numpy().squeeze()
        word_i = np.random.choice(top_i, p=p/p.sum())

        # retrieve that word from the dictionary
        word = int_to_vocab[word_i]
        predicted.append(word)

        if(train_on_gpu):
            current_seq = current_seq.cpu() # move to cpu
```

```
        # the generated word becomes the next "current sequence" and the cycle
can continue
        if train_on_gpu:
            current_seq = current_seq.cpu()
        current_seq = np.roll(current_seq, -1, 1)
        current_seq[-1][-1] = word_i

    gen_sentences = ' '.join(predicted)

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

    # return all the sentences
    return gen_sentences
```

Generate a New Script

It's time to generate the text. Set `gen_length` to the length of TV script you want to generate and set `prime_word` to one of the following to start the prediction:

- "jerry"
- "elaine"
- "george"
- "kramer"

You can set the prime word to *any word* in our dictionary, but it's best to start with a name for generating a TV script. (You can also start with any other names you find in the original text file!)

In [22]:

```
# run the cell multiple times to get different results!
gen_length = 2000 # modify the length to your preference
prime_word = 'george' # name for starting the script

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
pad_word = helper.SPECIAL_WORDS['PADDING']
generated_script = generate(trained_rnn, vocab_to_int[prime_word + ':'], int_t
o_vocab, token_dict, vocab_to_int[pad_word], gen_length)
print(generated_script)
```

george: you think you're going in there. i can't believe you...(to kramer) wh at are you doing here?

kramer: oh, yeah, i think i can get it.

kramer: well, i was in a while.

jerry:(quietly) no, that's a little man.

jerry:(cont'd) oh, yeah.(to kramer, george is trying to get a little problem.

jerry: i don't know.

kramer: oh, yeah. well, i'm sorry, what is that?

george: well, you know what i am, and i was going to get a couple of weeks in the middle of my house, but it's just the best...

elaine:(to george) oh, hi! you go out with a woman?

elaine: well, i don't know.

jerry: what?

jerry: i thought you were a good guy.

kramer: i don't know if i don't know. i think it's not a good driver.

jerry: i can't believe that, i got a man, and a few place in the middle of my life.

george: well, if i have to go through the car, and i was a good idea.

kramer:(to jerry) i know, it's not a long, and you go into her of the stree t... and i can't believe you were a good man.

jerry: what?

jerry:(quietly) no, you know, i don't want to see the rest of the car!

george: what, did you think you think i'm gonna go out to a tractor?

jerry: no, i don't know.

jerry: oh, well...

kramer: yeah, but i got the card of the first ones you know it was a big dea l?

kramer: yeah, yeah, i think i'm not going to do this.

kramer: well, i'm sorry, i'm not going to be careful.

jerry:(confused) i don't know.

george: i think you should be able to get away with her.

george: i can't get the whole thing!

george:(looking at the couch)

jerry: i think i'm very good cantaloupe, i'm going to be nice to do it. i know. i'm sure i think it's a little problem.

kramer: yeah. i think i can.

george: well you don't know, it's not going to have a great time...

jerry:(confused) well, i can't go out. i don't know how grateful i could get the same time.(she leaves, she walks down and the door) what are you going on? what are you talking about?

jerry: yeah, but it's not going.

kramer: i got the same one.

george: oh no, i got a job, but i'm not going to go see you...........

jerry: well, i'm sorry, it's a tough one.

elaine: oh, hey, you think that is my life...(opens the door) i don't know if i don't know how grateful you would be a little eccentric.

george:(to jerry) well, i can't get it in my house.

elaine: well, it's a big deal, and i have to do something to have to talk to you, but i don't think you know.

elaine: oh, i know what i am. i don't want you to do something else?

elaine:(quietly) well, what is he going?

jerry: i got the ticket. i think i'm not going to see this pathetic.

george: oh. well, i'm sorry, i can't get the door, and begins into the door) oh my god.

jerry:(to jerry) hey, george, i can't believe it, i got a job of my life. it's like a cup of times and i don't think i can do it.

george: i don't know.

george:(to george) what are you doing here?

jerry: i can't believe that i was a little steam. i mean i could have a man.

jerry: what do you want me to do that.

jerry: i don't know how much did he say?

kramer: well, i think i'm not sure.

jerry: what are you doing?

jerry: no.

kramer:(to george) oh, yeah.

jerry: what?

elaine: oh, yeah.

kramer:(to george) hey, jerry!(to jerry) hey, what is that?

jerry: yeah, well, i can't believe that i got the same time for a little spic
e.

elaine: oh, well, i'm sure it would be nice in this, i can't get the same of the road, but it's a little standoffish smaller emerge out of the hospital.

jerry: what do you think, did you get it?

jerry:(confused) i think you might see me.(he pushes her off) oh, hi, georg
e...

george: i can't believe i want a job.

jerry: what about you?

elaine: yeah, yeah. i got a job.

kramer:(quietly and exits)..........

jerry: i think you can tell me a little bit, and a stereo, you get a job.(to elaine) : what is that?

jerry: yeah?

jerry: well...

jerry: oh, i don't think so.

kramer: well, you know, i can't go with you.

jerry: well, i don't have any idea.

george:(cont'd) oh, that's nice.

george:(to jerry) what?

jerry: yeah.(he leaves)

kramer: i think i'm not going to do something for it.

elaine:(to george) oh, yeah, well.

kramer:(to elaine) hey, i think you should get a good cheek for you.

jerry: well, you know, i know what i think.

jerry: oh, yeah. i don't want it to get.

elaine: i don't want a job. i can't go to the end of the bathroom)

george: i got it.(she walks off)

george: what, you think?

george: no, no, i know you don't like to be in the car, and i want it to be i
n love.

elaine:(cont'd) what?

kramer: yeah, i got a little steam.

elaine:(to himself) hey, jerry, you think i have to do it!

jerry: i think you can tell me anything to get this off for the hospital.

george: i can't believe i got it. i don't even know how about the first thing
i got to see the whole time. i'm going to go.

george:(cont'd) what do you mean that was a very nice gift, but i'm gonna get
a lot of people, if i have a job, you know, i think i'm going to do something
else!

kramer: well, i'm gonna get an idea.

jerry:(cont'd) well, i know.

jerry: well, you know, i don't know, you know, i don't know.

elaine: i think i can do it.(she walks over to the bathroom. i got to get som
e real done.

elaine:(cont'd) i know i got a little standoffish player pie and he lies for
their apartment and the crowd.

jerry: what?

jerry: what are you doing?

kramer: oh, that's good!(laughs)

kramer: oh! i got the job.

george: what?

george:(looking at her face) i can't take a little clearly man for him on the
door)

jerry: i can't believe what happened.

elaine: well, i'm gonna get an old man!

kramer:(to elaine) i think you could get a good woman.

george: well, if i don't have one of those people.

```
elaine: well, you know i don't know if it's going to be nice.

george: i don't know.

elaine: well, i think i'm gonna get involved. i don't want to have to get out
of the way to see it.(kramer enters.)

jerry:(cont'd) yeah, yeah!

george: i think you could have a big deal for my friend and i was going to ge
t a couple of pie!

elaine: oh! i got a good cheek.

jerry: i don't know.

george:(looking) the guy from the door) : alright, alright, i don't want you
to do that!

elaine: oh, yeah.

jerry: what?

elaine: oh, well, i know what the hell was that the time, it was an accident.

jerry:(to elaine, he picks out of the middle of alabama, but i got a job of m
y life, and the jury's ones the same thing. i don't know if you think you're
not going to do you think i'm talking about this.

kramer: i think he would be a good idea.

elaine: oh, yeah. well, i know, i'm going to see this whole thing, if we wan
t.

george: i got a good idea, and i'm going to be able to get the car, the whole
thing is a good guy.

jerry: i thought we were in a great date?

jerry: i can't get it. i can't believe that.

kramer: well...

elaine:
```

Save your favorite scripts

Once you have a script that you like (or find interesting), save it to a text file!

In [20]:
```python
# save script to a text file
f =  open("generated_script_1.txt","w")
f.write(generated_script)
f.close()
```

The TV Script is Not Perfect

It's ok if the TV script doesn't make perfect sense. It should look like alternating lines of dialogue, here is one such example of a few generated lines.

Example generated script

> jerry: what about me?
>
> jerry: i don't have to wait.
>
> kramer:(to the sales table)
>
> elaine:(to jerry) hey, look at this, i'm a good doctor.
>
> newman:(to elaine) you think i have no idea of this...
>
> elaine: oh, you better take the phone, and he was a little nervous.
>
> kramer:(to the phone) hey, hey, jerry, i don't want to be a little bit.(to kramer and jerry) you can't.
>
> jerry: oh, yeah. i don't even know, i know.
>
> jerry:(to the phone) oh, i know.
>
> kramer:(laughing) you know...(to jerry) you don't know.

You can see that there are multiple characters that say (somewhat) complete sentences, but it doesn't have to be perfect! It takes quite a while to get good results, and often, you'll have to use a smaller vocabulary (and discard uncommon words), or get more data. The Seinfeld dataset is about 3.4 MB, which is big enough for our purposes; for script generation you'll want more than 1 MB of text, generally.

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 another copy as an HTML file by clicking "File" -> "Download as.."->"html". Include the "helper.py" and "problem_unittests.py" files in your submission. Once you download these files, compress them into one zip file for submission.

In [ ]:

```

```