Assignment 6 - Exercise 2

Daniel Mehta

In [1]:
```
import numpy as np
import pandas as pd


import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [2]:
```
import os

import pandas as pd
df = pd.read_csv('raw_partner_headlines.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Unnamed: 0	headline	url	publisher	date	stock
0	2	Agilent Technologies Announces Pricing of $5........	http://www.gurufocus.com/news/1153187/agilent-...	GuruFocus	2020-06-01 00:00:00	A
1	3	Agilent (A) Gears Up for Q2 Earnings: What's i...	http://www.zacks.com/stock/news/931205/agilent...	Zacks	2020-05-18 00:00:00	A
2	4	J.P. Morgan Asset Management Announces Liquida...	http://www.gurufocus.com/news/1138923/jp-morga...	GuruFocus	2020-05-15 00:00:00	A
3	5	Pershing Square Capital Management, L.P. Buys ...	http://www.gurufocus.com/news/1138704/pershing...	GuruFocus	2020-05-15 00:00:00	A
4	6	Agilent Awards Trilogy Sciences with a Golden ...	http://www.gurufocus.com/news/1134012/agilent-...	GuruFocus	2020-05-12 00:00:00	A

In [4]:
```
news = []
for i, j in df.iterrows():
    news.append(j['headline'])

print(len(news))
```

```
1845559
```

In [5]:
```
news[:1]
```

Out[5]: ['Agilent Technologies Announces Pricing of $5..... Million of Senior Notes']

In [6]:
```
len(news)
```

Out[6]: 1845559

In [7]:
```
news = news[:109233]
```

In [8]:
```
len(news)
```

Out[8]: 109233

In [9]:
```
import os
file_path = os.path.join(os.getcwd(), 'finance_news.txt')
```

In [10]:
```
with open(file_path, 'w') as f:
    f.write('\n'.join(news))
```

In [11]:
```
import os
import pickle
import torch


SPECIAL_WORDS = {'PADDING': '<PAD>'}


def load_data(path):
    """
    Load Dataset from File
    """
    input_file = os.path.join(path)
    with open(input_file, "r") as f:
        data = f.read()

    return data


def preprocess_and_save_data(dataset_path, token_lookup, create_lookup_tables):
    """
    Preprocess Text Data
    """
    text = load_data(dataset_path)

    # Ignore notice, since we don't use it for analysing the data
    text = text[81:]

    token_dict = token_lookup()
    for key, token in token_dict.items():
        text = text.replace(key, ' {} '.format(token))

    text = text.lower()
```

```
    text = text.split()

    vocab_to_int, int_to_vocab = create_lookup_tables(text + list(SPECIAL_WORDS.values()))
    int_text = [vocab_to_int[word] for word in text]
    pickle.dump((int_text, vocab_to_int, int_to_vocab, token_dict), open('preprocess.p', 'wb'))


def load_preprocess():
    """
    Load the Preprocessed Training data and return them in batches of <batch_size> or less
    """
    return pickle.load(open('preprocess.p', mode='rb'))


def save_model(filename, decoder):
    save_filename = os.path.splitext(os.path.basename(filename))[0] + '.pt'
    torch.save(decoder, save_filename)


def load_model(filename):
    save_filename = os.path.splitext(os.path.basename(filename))[0] + '.pt'
    return torch.load(save_filename)
```

In [12]:
```
data_dir = 'finance_news.txt'
text = load_data(data_dir)
```

In [13]:
```
view_line_range = (0, 10)

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: 58299
Number of lines: 109233
Average number of words in each line: 9.44242124632666

The lines 0 to 10:
Agilent Technologies Announces Pricing of $5…… Million of Senior Notes
Agilent (A) Gears Up for Q2 Earnings: What's in the Cards?
J.P. Morgan Asset Management Announces Liquidation of Six Exchange-Traded Funds
Pershing Square Capital Management, L.P. Buys Agilent Technologies Inc, The Howard Hughes Corp, ...
Agilent Awards Trilogy Sciences with a Golden Ticket at LabCentral
Agilent Technologies Inc (A) CEO and President Michael R. Mcmullen Sold $-.4 million of Shares
' Stocks Growing Their Earnings Fast
Cypress Asset Management Inc Buys Verizon Communications Inc, United Parcel Service Inc, ...
Hendley & Co Inc Buys American Electric Power Co Inc, Agilent Technologies Inc, Paychex ...
Teacher Retirement System Of Texas Buys Hologic Inc, Vanguard Total Stock Market, Agilent ...
```

In [14]:
```
from collections import Counter

def create_lookup_tables(text):
    """
    Create lookup tables for vocabulary
    :param text: The text of tv scripts split into words
    :return: A tuple of dicts (vocab_to_int, int_to_vocab)
    """
    # TODO: Implement Function
    word_count = Counter(text)
    sorted_vocab = sorted(word_count, key = word_count.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 tuple
    return (vocab_to_int, int_to_vocab)
```

In [15]:
```
def token_lookup():
    """
    Generate a dict to turn punctuation into a token.
    :return: Tokenized dictionary where the key is the punctuation and the value is the token
```

```
    """
    # TODO: Implement Function
    token = dict()
    token['.'] = '<PERIOD>'
    token[','] = '<COMMA>'
    token['"'] = 'QUOTATION_MARK'
    token[';'] = 'SEMICOLON'
    token['!'] = 'EXCLAIMATION_MARK'
    token['?'] = 'QUESTION_MARK'
    token['('] = 'LEFT_PAREN'
    token[')'] = 'RIGHT_PAREN'
    token['-'] = 'QUESTION_MARK'
    token['\n'] = 'NEW_LINE'
    return token
```

In [16]:
```
preprocess_and_save_data(data_dir, token_lookup, create_lookup_tables)
```

In [17]:
```
int_text, vocab_to_int, int_to_vocab, token_dict = load_preprocess()
```

In [18]:
```
train_on_gpu = torch.cuda.is_available()
```

In [19]:
```
from torch.utils.data import TensorDataset, DataLoader
import torch
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 batch
    :return: DataLoader with batched data
    """
    # TODO: Implement function
    n_batches = len(words)//batch_size
    x, y = [], []
    words = words[:n_batches*batch_size]
```

```
    for ii in range(0, len(words)-sequence_length):
        i_end = ii+sequence_length
        batch_x = words[ii:ii+sequence_length]
        x.append(batch_x)
        batch_y = words[i_end]
        y.append(batch_y)

    data = TensorDataset(torch.from_numpy(np.asarray(x)), torch.from_numpy(np.asarray(y)))
    data_loader = DataLoader(data, shuffle=True, batch_size=batch_size)


    # return a dataloader
    return data_loader
```

In [20]:

```
# 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 = next(data_iter)  # using built in next() function instead of .next() method for iterators

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

```
torch.Size([10, 5])
tensor([[35, 36, 37, 38, 39],
        [19, 20, 21, 22, 23],
        [17, 18, 19, 20, 21],
        [ 7,  8,  9, 10, 11],
        [ 3,  4,  5,  6,  7],
        [23, 24, 25, 26, 27],
        [18, 19, 20, 21, 22],
        [31, 32, 33, 34, 35],
        [39, 40, 41, 42, 43],
        [38, 39, 40, 41, 42]])

torch.Size([10])
tensor([40, 24, 22, 12,  8, 28, 23, 36, 44, 43])
```

In [21]:

```python
import torch.nn as nn

class RNN(nn.Module):

    def __init__(self, vocab_size, output_size, embedding_dim, hidden_dim, n_layers, dropout=0.5):
        """
        Initialize the PyTorch RNN Module
        :param vocab_size: The number of input dimensions of the neural network (the size of the vocabulary)
        :param output_size: The number of output dimensions of the neural network
        :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

        # define embedding layer
        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        # define lstm layer
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, n_layers, dropout=dropout, batch_first=True)


        # set class variables
        self.vocab_size = vocab_size
        self.output_size = output_size
```

```
        self.embedding_dim = embedding_dim
        self.hidden_dim = hidden_dim
        self.n_layers = n_layers

        # define model layers
        self.fc = nn.Linear(hidden_dim, output_size)


    def forward(self, x, 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 = x.size(0)
        x=x.long()

        # embedding and lstm_out
        embeds = self.embedding(x)
        lstm_out, hidden = self.lstm(embeds, hidden)

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

        # dropout, fc layer and final sigmoid layer
        out = self.fc(lstm_out)

        # reshaping out layer to batch_size * seq_length * output_size
        out = out.view(batch_size, -1, self.output_size)

        # return last batch
        out = out[:, -1]

        # return one batch of output word scores and the hidden state
        return out, 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)
        '''
        # create 2 new zero tensors of size n_layers * batch_size * hidden_dim
        weights = next(self.parameters()).data
        if(train_on_gpu):
            hidden = (weights.new(self.n_layers, batch_size, self.hidden_dim).zero_().cuda(),
                      weights.new(self.n_layers, batch_size, self.hidden_dim).zero_().cuda())
        else:
            hidden = (weights.new(self.n_layers, batch_size, self.hidden_dim).zero_(),
                      weights.new(self.n_layers, batch_size, self.hidden_dim).zero_())

        # initialize hidden state with zero weights, and move to GPU if available

        return hidden
```

In [22]:
```
def forward_back_prop(rnn, optimizer, criterion, inp, target, hidden):
    """
    Forward and backward propagation on the neural network
    :param decoder: The PyTorch Module that holds the neural network
    :param decoder_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):
        rnn.cuda()

    # creating variables for hidden state to prevent back-propagation
    # of historical states
    h = tuple([each.data for each in hidden])

    rnn.zero_grad()
    # move inputs, targets to GPU
```

```
    inputs, targets = inp.cuda(), target.cuda()

    output, h = rnn(inputs, h)

    loss = criterion(output, targets)

    # perform backpropagation and optimization
    loss.backward()
    nn.utils.clip_grad_norm_(rnn.parameters(), 5)
    optimizer.step()

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

In [23]:
```
def train_rnn(rnn, batch_size, optimizer, criterion, n_epochs, show_every_n_batches=100):
    batch_losses = []

    rnn.train()

    print("Training for %d epoch(s)..." % n_epochs)
    for epoch_i in 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)

            # 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
```

In [24]:
```
# Data params
# Sequence Length
sequence_length = 10  # of words in a sequence
# Batch Size
batch_size = 128

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

In [25]:
```
# Training parameters
# Number of Epochs
num_epochs = 10
# Learning Rate
learning_rate = 0.001

# Model parameters
# Vocab size
vocab_size = len(vocab_to_int)
# Output size
output_size = vocab_size
# Embedding Dimension
embedding_dim = 200
# Hidden Dimension
hidden_dim = 250
# Number of RNN Layers
n_layers = 2

# Show stats for every n number of batches
show_every_n_batches = 500
```

In [26]:
```
# create model and move to gpu if available
rnn = RNN(vocab_size, output_size, embedding_dim, hidden_dim, n_layers, dropout=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
save_model('trained_rnn', trained_rnn)
print('Model Trained and Saved')
```

```
Training for 10 epoch(s)...
Epoch:    1/10    Loss: 6.993944655418396

Epoch:    1/10    Loss: 6.1529925899505615

Epoch:    1/10    Loss: 5.768744532585144

Epoch:    1/10    Loss: 5.521613512992859

Epoch:    1/10    Loss: 5.35321697807312

Epoch:    1/10    Loss: 5.216994276046753

Epoch:    1/10    Loss: 5.087665128707886

Epoch:    1/10    Loss: 4.985864153862

Epoch:    1/10    Loss: 4.9125787000656125

Epoch:    1/10    Loss: 4.819999813556671

Epoch:    1/10    Loss: 4.785653531551361

Epoch:    1/10    Loss: 4.675312551498413

Epoch:    1/10    Loss: 4.630319984436035

Epoch:    1/10    Loss: 4.589827892303467

Epoch:    1/10    Loss: 4.520943288326263

Epoch:    1/10    Loss: 4.504897451877594

Epoch:    1/10    Loss: 4.460526263713836

Epoch:    1/10    Loss: 4.418950723171234

Epoch:    1/10    Loss: 4.3891025171279905

Epoch:    1/10    Loss: 4.357330962657929
```

```
Epoch:    2/10    Loss: 4.189208025261476

Epoch:    2/10    Loss: 4.10013459777832

Epoch:    2/10    Loss: 4.077091027259827

Epoch:    2/10    Loss: 4.055426326274872

Epoch:    2/10    Loss: 4.051941301345825

Epoch:    2/10    Loss: 4.027236931800842

Epoch:    2/10    Loss: 4.038839217185974

Epoch:    2/10    Loss: 4.02114538192749

Epoch:    2/10    Loss: 4.002659096717834

Epoch:    2/10    Loss: 3.992010589599609

Epoch:    2/10    Loss: 3.9921094298362734

Epoch:    2/10    Loss: 3.983956072330475

Epoch:    2/10    Loss: 4.011128291130066

Epoch:    2/10    Loss: 3.957461054801941

Epoch:    2/10    Loss: 3.9485399923324587

Epoch:    2/10    Loss: 3.942023030757904

Epoch:    2/10    Loss: 3.957240201473236

Epoch:    2/10    Loss: 3.9657868704795836

Epoch:    2/10    Loss: 3.9279524698257444

Epoch:    2/10    Loss: 3.918103006362915

Epoch:    3/10    Loss: 3.757539758513031
```

```
Epoch:    3/10    Loss: 3.6790250129699706

Epoch:    3/10    Loss: 3.6784533581733703

Epoch:    3/10    Loss: 3.6833298907279968

Epoch:    3/10    Loss: 3.7099530897140505

Epoch:    3/10    Loss: 3.6990194683074953

Epoch:    3/10    Loss: 3.691033864021301

Epoch:    3/10    Loss: 3.6963361291885377

Epoch:    3/10    Loss: 3.696784559249878

Epoch:    3/10    Loss: 3.6761943950653078

Epoch:    3/10    Loss: 3.690470257282257

Epoch:    3/10    Loss: 3.6814261226654055

Epoch:    3/10    Loss: 3.690713716983795

Epoch:    3/10    Loss: 3.6857989077568054

Epoch:    3/10    Loss: 3.716593376159668

Epoch:    3/10    Loss: 3.7060396580696104

Epoch:    3/10    Loss: 3.7227933435440064

Epoch:    3/10    Loss: 3.6932522168159485

Epoch:    3/10    Loss: 3.6860303072929383

Epoch:    3/10    Loss: 3.7337316503524782

Epoch:    4/10    Loss: 3.5524714210640793
```

```
Epoch:    4/10    Loss: 3.4709064354896544

Epoch:    4/10    Loss: 3.4818826422691345

Epoch:    4/10    Loss: 3.4830974969863893

Epoch:    4/10    Loss: 3.4983982071876527

Epoch:    4/10    Loss: 3.491863907337189

Epoch:    4/10    Loss: 3.529422034263611

Epoch:    4/10    Loss: 3.504852892398834

Epoch:    4/10    Loss: 3.516963713169098

Epoch:    4/10    Loss: 3.505481177806854

Epoch:    4/10    Loss: 3.50170051240921

Epoch:    4/10    Loss: 3.5350561804771425

Epoch:    4/10    Loss: 3.519514736175537

Epoch:    4/10    Loss: 3.5131672954559328

Epoch:    4/10    Loss: 3.5451217436790468

Epoch:    4/10    Loss: 3.5367036352157593

Epoch:    4/10    Loss: 3.559509958267212

Epoch:    4/10    Loss: 3.5625045561790465

Epoch:    4/10    Loss: 3.5684971590042114

Epoch:    4/10    Loss: 3.576594874382019

Epoch:    5/10    Loss: 3.4195049056837616

Epoch:    5/10    Loss: 3.3419065127372742
```

```
Epoch:    5/10    Loss: 3.374010527610779

Epoch:    5/10    Loss: 3.3253595323562624

Epoch:    5/10    Loss: 3.362663249492645

Epoch:    5/10    Loss: 3.3648437700271607

Epoch:    5/10    Loss: 3.3824815311431884

Epoch:    5/10    Loss: 3.397530748844147

Epoch:    5/10    Loss: 3.360007179737091

Epoch:    5/10    Loss: 3.385705491542816

Epoch:    5/10    Loss: 3.4034339265823363

Epoch:    5/10    Loss: 3.394639880180359

Epoch:    5/10    Loss: 3.434655624389648

Epoch:    5/10    Loss: 3.4195758385658266

Epoch:    5/10    Loss: 3.419886960029602

Epoch:    5/10    Loss: 3.432862590789795

Epoch:    5/10    Loss: 3.437755522251129

Epoch:    5/10    Loss: 3.442976296424866

Epoch:    5/10    Loss: 3.4706301856040955

Epoch:    5/10    Loss: 3.45311723279953

Epoch:    6/10    Loss: 3.3147154386203703

Epoch:    6/10    Loss: 3.2438850336074827
```

```
Epoch:    6/10    Loss: 3.2753872771263124

Epoch:    6/10    Loss: 3.2648511233329773

Epoch:    6/10    Loss: 3.2717924079895018

Epoch:    6/10    Loss: 3.286987283706665

Epoch:    6/10    Loss: 3.282788242340088

Epoch:    6/10    Loss: 3.320651725769043

Epoch:    6/10    Loss: 3.283897407054901

Epoch:    6/10    Loss: 3.3035778160095215

Epoch:    6/10    Loss: 3.2987783522605896

Epoch:    6/10    Loss: 3.308003444671631

Epoch:    6/10    Loss: 3.325775638580322

Epoch:    6/10    Loss: 3.3243118500709534

Epoch:    6/10    Loss: 3.3516270656585694

Epoch:    6/10    Loss: 3.3465147891044618

Epoch:    6/10    Loss: 3.3556078910827636

Epoch:    6/10    Loss: 3.3375739755630494

Epoch:    6/10    Loss: 3.3597601532936094

Epoch:    6/10    Loss: 3.369690434932709

Epoch:    7/10    Loss: 3.2324840762618225

Epoch:    7/10    Loss: 3.1617148151397707

Epoch:    7/10    Loss: 3.1622398438453674
```

```
Epoch:    7/10    Loss: 3.183980472564697

Epoch:    7/10    Loss: 3.186767496585846

Epoch:    7/10    Loss: 3.210755100727081

Epoch:    7/10    Loss: 3.207624593257904

Epoch:    7/10    Loss: 3.2116354274749757

Epoch:    7/10    Loss: 3.2185562801361085

Epoch:    7/10    Loss: 3.242059384346008

Epoch:    7/10    Loss: 3.2538273367881776

Epoch:    7/10    Loss: 3.238495846271515

Epoch:    7/10    Loss: 3.2427316989898682

Epoch:    7/10    Loss: 3.2580183119773864

Epoch:    7/10    Loss: 3.2772063665390014

Epoch:    7/10    Loss: 3.296006287574768

Epoch:    7/10    Loss: 3.306935854911804

Epoch:    7/10    Loss: 3.2765626463890074

Epoch:    7/10    Loss: 3.2786044850349425

Epoch:    7/10    Loss: 3.3104314770698546

Epoch:    8/10    Loss: 3.1804552084290636

Epoch:    8/10    Loss: 3.103109248638153

Epoch:    8/10    Loss: 3.1078985562324526
```

```
Epoch:    8/10    Loss: 3.12465931224823

Epoch:    8/10    Loss: 3.1350995082855224

Epoch:    8/10    Loss: 3.1685340185165405

Epoch:    8/10    Loss: 3.1492422065734864

Epoch:    8/10    Loss: 3.151209388256073

Epoch:    8/10    Loss: 3.161052990436554

Epoch:    8/10    Loss: 3.1811849036216735

Epoch:    8/10    Loss: 3.174163818359375

Epoch:    8/10    Loss: 3.187234085083008

Epoch:    8/10    Loss: 3.1788876676559448

Epoch:    8/10    Loss: 3.204271884918213

Epoch:    8/10    Loss: 3.2245695185661316

Epoch:    8/10    Loss: 3.219024398326874

Epoch:    8/10    Loss: 3.1868942914009093

Epoch:    8/10    Loss: 3.2357496848106386

Epoch:    8/10    Loss: 3.2361267704963685

Epoch:    8/10    Loss: 3.2235345001220703

Epoch:    9/10    Loss: 3.093372000185407

Epoch:    9/10    Loss: 3.0271588859558105

Epoch:    9/10    Loss: 3.045473005771637

Epoch:    9/10    Loss: 3.046944664001465
```

```
Epoch:   9/10   Loss: 3.055320095062256
Epoch:   9/10   Loss: 3.094630542755127
Epoch:   9/10   Loss: 3.1152786622047426
Epoch:   9/10   Loss: 3.0821052112579346
Epoch:   9/10   Loss: 3.1092875213623046
Epoch:   9/10   Loss: 3.102768180847168
Epoch:   9/10   Loss: 3.1256769275665284
Epoch:   9/10   Loss: 3.124481111526489
Epoch:   9/10   Loss: 3.1167268490791322
Epoch:   9/10   Loss: 3.175630630016327
Epoch:   9/10   Loss: 3.1422842245101927
Epoch:   9/10   Loss: 3.161958396434784
Epoch:   9/10   Loss: 3.194431569099426
Epoch:   9/10   Loss: 3.1797687463760376
Epoch:   9/10   Loss: 3.174331483364105
Epoch:   9/10   Loss: 3.2016622443199156
Epoch:  10/10   Loss: 3.0666787881210578
Epoch:  10/10   Loss: 2.975358413219452
Epoch:  10/10   Loss: 3.0065314846038818
Epoch:  10/10   Loss: 3.0024210953712465
```

```
Epoch:   10/10    Loss: 3.0132005343437194

Epoch:   10/10    Loss: 3.0244974813461303

Epoch:   10/10    Loss: 3.041077748298645

Epoch:   10/10    Loss: 3.0541318411827088

Epoch:   10/10    Loss: 3.0660813546180723

Epoch:   10/10    Loss: 3.072396270751953

Epoch:   10/10    Loss: 3.0773998289108278

Epoch:   10/10    Loss: 3.1026220908164976

Epoch:   10/10    Loss: 3.1100303130149842

Epoch:   10/10    Loss: 3.098790253162384

Epoch:   10/10    Loss: 3.098059878349304

Epoch:   10/10    Loss: 3.133239384651184

Epoch:   10/10    Loss: 3.114422559738159

Epoch:   10/10    Loss: 3.1404697575569154

Epoch:   10/10    Loss: 3.1519603643417358

Epoch:   10/10    Loss: 3.1421021361351014

Model Trained and Saved
```

In [29]:

```python
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import torch
```

```
_, vocab_to_int, int_to_vocab, token_dict = load_preprocess()
trained_rnn = torch.load('trained_rnn.pt', weights_only=False)
```

In [30]:
```
import torch.nn.functional as F

def generate(rnn, prime_id, int_to_vocab, token_dict, pad_value, predict_len=100):
    """
    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)

        # the generated word becomes the next "current sequence" and the cycle can continue
        current_seq = np.roll(current_seq.cpu(), -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
```

In [31]:

```
gen_length = 50 # modify the length to your preference
prime_words = ['tesla'] # name for starting the script

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
for prime_word in prime_words:
    pad_word = SPECIAL_WORDS['PADDING']
    generated_script = generate(trained_rnn, vocab_to_int[prime_word], int_to_vocab, token_dict, vocab_to_int[pad_word], gen_l
    print(generated_script)
```

```
tesla
american tower(amt) gains from market anticipated cagr of —4%...
global organic photovoltaics market —…—… global industry analysis, industry share...
the vetr community has downgraded $amx to 2? stars
the vetr community has downgraded $anf to
```

Conclusion

- Used 100k+ financial news headlines as training data
- Preprocessed text and converted it to integer sequences
- Built and trained an LSTM model for word level text generation
- Generated sample headlines using topk sampling