CSC413 Assignment 3. Text Denoising Autoencoder for News Headlines

Ming Gong 1004709130
Hongyu Chen 1005398197

March 2021

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import matplotlib.pyplot as plt
import numpy as np
import random
%matplotlib inline
```

1 Question 1 Data exploration

```
from google.colab import drive
drive.mount('/content/gdrive')

train_path = '/content/gdrive/My Drive/CSC413/A/A3/Data/
                                  reuters_train.txt'
valid_path = '/content/gdrive/My Drive/CSC413/A/A3/Data/
                                  reuters_valid.txt'
```

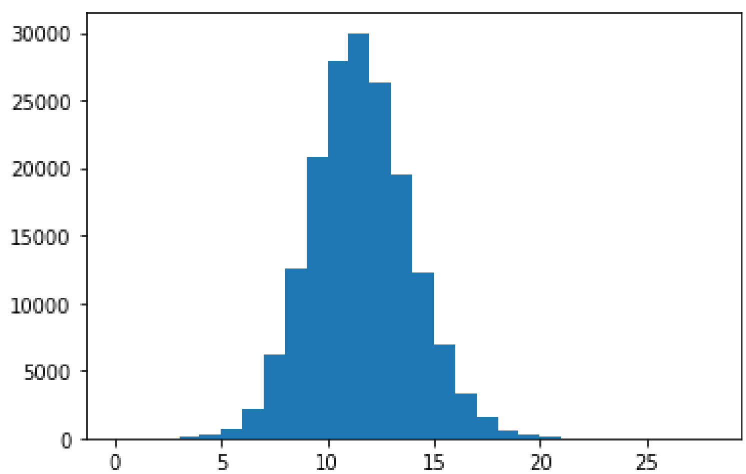
```
import torchtext
from torchtext.legacy import data
# Tokenization function to separate a headline into words
def tokenize_headline(headline):
    """Returns the sequence of words in the string headline. We
                                  also
    prepend the "<bos>" or beginning-of-string token, and append
                                  the
    "<eos>" or end-of-string token to the headline.
    """
    return ("<bos> " + headline + " <eos>").split()

# Data field (column) representing our *text*.
text_field = data.Field(
    sequential=True,            # this field consists of a sequence
    tokenize=tokenize_headline, # how to split sequences into words
```

```
    include_lengths=True,        # to track the length of sequences,
                                   for batching
    batch_first=True,            # similar to batch_first=True in nn
                                   .RNN demonstrated in lecture
    use_vocab=True)              # to turn each character into an
                                   integer index
train_data = data.TabularDataset(
    path=train_path,                 # data file path
    format="tsv",                    # fields are separated by a tab
    fields=[('title', text_field)]) # list of fields (we have only
                                       one)
```

1.1 Part (a)

```
num_words = []
for i in train_data:
  num_words.append(len(i.title)-2)
plt.hist(num_words, bins=np.arange(0,max(num_words)))
plt.show()
```



We would be interested in such histograms is because we want to know the number of words per headline in our training set so we will know if there exists some sentences are too long or too short compare to other sentences. We can then normalize these sentences using some of the techniques such as padding to let these sentences' length match closer to the average length.

1.2 Part (b)

```
# Report your values here. Make sure that you report the actual
                                   values,
# and not just the code used to get those values

# You might find the python class Counter from the collections
                                   package useful
from collections import Counter
words = []
for i in train_data:
  words.extend(i.title)
num_distinct = len(Counter(words).keys()) - 2
print("The number of distinct words: ", num_distinct)
```

```
The number of distinct words:  51298
```

1.3 Part (c)

```
# Report your values here. Make sure that you report the actual
                                   values,
# and not just the code used to get those values
counter = Counter(words)
freq = list(counter.values())
distinct = list(counter.keys())
once = []
twice = []
for i in range(len(distinct)):
  if freq[i] == 1:
    once.append(distinct[i])
  elif freq[i] == 2:
    twice.append(distinct[i])
print("The number of words appear exactly once: ", len(once))
print("The number of words appear exactly twice: ", len(twice))
```

```
The number of words appear exactly once:  19854
The number of words appear exactly twice:  7193
```

1.4 Part (d)

We may wish to replace these infrequent words with an <unk> tag is because these words are really rare in both our training set and also validation set. They might not even appears in the validation set so learning embedding for these rare words will not help the training process and also making predictions. These rare words can also affect our model in bad ways. So during the training we can treat these words as <unk> so in the validation set when we met these kind of rare words we can also use <unk> to represent these words.

1.5 Part (e)

```
# Report your values here. Make sure that you report the actual
                                    values,
# and not just the code used to get those values
top = counter.most_common(9997)
total = 0
num_useless = 0
for i in top:
  if i[0] != "<bos>" and i[0] != "<eos>":
    total += i[1]
  else:
    num_useless += i[1]
print("The percentage of word occurrences will be supported:", ((
                                    total/(len(words) - num_useless))
                                     * 100), "%")
print("The percentage of word occurrences in the training set will
                                    be set to the <unk> tag:", (100-(
                                    total/(len(words) - num_useless))
                                     * 100), "%")
```

```
The percentage of word occurrences will be supported: 93.
                                    97857393100142 %
The percentage of word occurrences in the training set will be set
                                    to the <unk> tag: 6.
                                    021426068998579 %
```

```
# Build the vocabulary based on the training data. The vocabulary
# can have at most 9997 words (9995 words + the <bos> and <eos>
                                    token)
text_field.build_vocab(train_data, max_size=9997)

# This vocabulary object will be helpful for us
vocab = text_field.vocab
print(vocab.stoi["hello"]) # for instances, we can convert from
                                    string to (unique) index
print(vocab.itos[10])      # ... and from word index to string

# The size of our vocabulary is actually 10000
vocab_size = len(text_field.vocab.stoi)
print(vocab_size) # should be 10000

# The reason is that torchtext adds two more tokens for us:
print(vocab.itos[0]) # <unk> represents an unknown word not in our
                                    vocabulary
print(vocab.itos[1]) # <pad> will be used to pad short sequences
                                    for batching
```

```
0
on
10000
<unk>
<pad>
```

2 Question 2 Background Math

2.1 Part (a)

We know that for the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ and $f'(x) = \frac{e^x}{(e^x+1)^2}$

Then for any $i \in N$, Since here we know that $(x_{t+1} = f(Wx_t)$ we have:

$\frac{\partial x_{t+1}}{\partial x_t} = \frac{\partial f(Wx_t)}{\partial x_t} = \frac{e^{Wx_t}}{(e^{Wx_t}+1)^2}W$

By the hint provided in the question, Let
$C = \frac{\partial x_{t+1}}{\partial x_t}$, $A = \frac{e^{Wx_t}}{(e^{Wx_t}+1)^2}$ and $B = W$.

Since here f'(x) is in range (0,0.25), so here A must be a matrix in range (0,0.25), then to get the singular value we need to calculate the eigenvalue of AA^T and clearly all of the eigenvalues of AA^T is between (0,1). The singular values for matrix A is the square root of the eigenvalues of AA^T and they are clearly in range(0,1).

So then by the hint we will get:

$\sigma_{max}(\frac{\partial x_{t+1}}{\partial x_t}) \leq \sigma_{max}(\frac{e^{Wx_t}}{(e^{Wx_t}+1)^2})\sigma_{max}(W) = \sigma_{max}(\frac{e^{Wx_t}}{(e^{Wx_t}+1)^2})\frac{1}{2} \leq \frac{1}{2}$

Then according to the chain rule, we will have:

$\frac{\partial x_n}{\partial x_1} = \prod_{t=1}^{n-1} \frac{e^{Wx_t}}{(e^{Wx_t}+1)^2}$

Then we can apply the hint again so we have :

$\sigma_{max}(\prod_{t=1}^{n-1} \frac{x_{t+1}}{x_t}) \leq \prod_{t=1}^{n-1} \sigma_{max}(\frac{x_{t+1}}{x_t}) = (\frac{1}{2}t)^{n-1}$ (According to the result above)

Then we know that singular values are always greater or equal to 0, so finally we have:

$0 \leq \sigma_{\max}(\frac{\partial x_n}{\partial x_1}) \leq (\frac{1}{2})^n$

Which is what we want to prove in this question. This tell us that the input-output Jacobian as $n \to \infty$, $\frac{\partial x_n}{\partial x_1} = 0$

2.2 Part (b)

$$\frac{\partial \mathcal{L}}{\partial W_x} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial W_x} \tag{1}$$

Apply equation (1) to our RNN model then we will get

$$\frac{\partial \mathcal{L}}{\partial W_x} = \sum_{k=1}^{T} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_x} \tag{2}$$

$\frac{\partial h_t}{\partial h_k}$ refers to the partial derivative of h_t with respect to all previous k timesteps.

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \tag{3}$$

Put equation (1), (2), (3) together then we will get

$$\frac{\partial \mathcal{L}}{\partial W_x} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \left(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W_x} = \sum_{t=1}^{T} \sum_{k=1}^{t} (y_t - o_t) W_y \left(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) (1 - h_t^2) x_k \tag{4}$$

$$\frac{\partial h_j}{\partial h_{j-1}} = (1 - h_j^2) W_h \tag{5}$$

plug equation (5) into (4) we get

$$\frac{\partial \mathcal{L}}{\partial W_x} = \sum_{t=1}^{T} \sum_{k=1}^{t} (y_t - o_t) W_y \left(\prod_{j=k+1}^{t} (1 - h_j^2) W_h \right) (1 - h_t^2) x_k \tag{6}$$

The vanishing gradient problem happens when the term $\prod_{j=k+1}^{t} (1 - h_j^2) W_h$ becomes very small through timesteps. And the exploding gradient problem happens when the term is very large through timesteps.

2.3 Part (c)

$$\frac{\partial \mathcal{L}}{\partial W_x} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial W_x} \tag{7}$$

Apply equation (10) to our GRU

$$\frac{\partial \mathcal{L}}{\partial W_x} = \sum_{k=1}^{T} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial z_k} \frac{\partial z_k}{\partial W_x} \tag{8}$$

$\frac{\partial h_t}{\partial h_k}$ refers to the partial derivative of h_t with respect to all previous k timesteps.

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \tag{9}$$

Put equation (7), (8), (9) together then we will get

$$\frac{\partial \mathcal{L}}{\partial W_x} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \left(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial z_k} \frac{\partial z_k}{\partial W_x} \tag{10}$$

$$\frac{\partial h_k}{\partial z_k}\frac{\partial z_k}{\partial W_x} = (\hat{h}_k - h_{k-1})z_k(1-z_k)x_k \tag{11}$$

$$\frac{\partial h_j}{\partial h_{j-1}} = \frac{\partial h_j}{\partial \hat{h}_j}\frac{\partial \hat{h}_j}{\partial h_{t-1}} + \frac{\partial h_j}{\partial z_j}\frac{\partial z_j}{\partial h_{j-1}} + z_j = \frac{\partial h_j}{\partial \hat{h}_j}(\frac{\partial \hat{h}_j}{\partial r_j}\frac{\partial r_j}{\partial h_{j-1}} + (1-\hat{h}_j^2)U_z r_j) + \frac{\partial h_j}{\partial z_j}\frac{\partial z_j}{\partial h_{j-1}} + z_j \tag{12}$$

$$= z_j((1-\hat{h}_j^2)U_h h_{j-1} r_j(1-r_j)U_h h_{j-1} + (1-\hat{h}_j^2)U_z r_j) + (\hat{h}_j - h_{j-1})z_j(1-z_j)U_z + z_j \tag{13}$$

$$\frac{\partial \mathcal{L}_t}{\partial y_t} = y_t - o_t \tag{14}$$

$$\frac{\partial y_t}{\partial h_t} = W_y \tag{15}$$

Put equation (11), (13), (14), (15) into (10) and we will get the final result. In GRU $\frac{\partial h_j}{\partial h_{j-1}}$ is 1 for $z_j = 1$ and $\frac{\partial h_j}{\partial h_{j-1}}$ is z_j for $r_j = 0$. Shutting the update gate lets us essentially skip layers when calculating the gradient. This ameliorates the vanishing, exploding gradient problem.

3 Question 3 Building the autoencoder

3.1 Part (a)

```
class AutoEncoder(nn.Module):
    def __init__(self, vocab_size, emb_size, hidden_size):
        """
        A text autoencoder. The parameters
            - vocab_size: number of unique words/tokens in the
                                 vocabulary
            - emb_size: size of the word embeddings $x^{(t)}$
            - hidden_size: size of the hidden states in both the
                           encoder RNN ($h^{(t)}$) and the
                           decoder RNN ($m^{(t)}$)
        """
        super().__init__()
        self.embed = nn.Embedding(num_embeddings=vocab_size,
                                  embedding_dim=emb_size)
        self.encoder_rnn = nn.GRU(input_size=emb_size,
                                  hidden_size=hidden_size,
                                  batch_first=True)
        self.decoder_rnn = nn.GRU(input_size=hidden_size,
                                  hidden_size=emb_size,
                                  batch_first=True)
        self.proj = nn.Linear(in_features=emb_size,
                              out_features=vocab_size)

    def encode(self, inp):
        """
        Computes the encoder output given a sequence of words.
        """
        emb = self.embed(inp)
        out, last_hidden = self.encoder_rnn(emb)
```

```python
        return last_hidden

    def decode(self, inp, hidden=None):
        """
        Computes the decoder output given a sequence of words, and
        (optionally) an initial hidden state.
        """
        emb = self.embed(inp)
        out, last_hidden = self.decoder_rnn(emb, hidden)
        out_seq = self.proj(out)
        return out_seq, last_hidden

    def forward(self, inp):
        """
        Compute both the encoder and decoder forward pass
        given an integer input sequence inp with shape [batch_size,
                                  seq_length],
        with inp[a,b] representing the (index in our vocabulary of)
                                  the b-th word
        of the a-th training example.

        This function should return the logits $z^{(t)}$ in a
                                  tensor of shape
        [batch_size, seq_length - 1, vocab_size], computed using *
                                  teaching forcing*.

        The (seq_length - 1) part is not a typo. If you don't
                                  understand why
        we need to subtract 1, refer to the teacher-forcing diagram
                                  above.
        """

        last_encode_hidden = self.encode(inp)
        out_seq, last_decode_hidden = self.decode(inp,
                                  last_encode_hidden)
        return out_seq, last_decode_hidden
```

3.2 Part (b)

```python
headline = train_data[42].title
input_seq = torch.Tensor([vocab.stoi[w] for w in headline]).long().
                                  unsqueeze(0)
```

```python
model = AutoEncoder(vocab_size, 128, 128)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

for it in range(300):

    optimizer.zero_grad()
    output, _ = model(input_seq[:,:-1])
    target = input_seq[:,1:]
    loss = criterion(output.reshape(-1, vocab_size), target.reshape
                                  (-1))
    loss.backward()
```

```
    optimizer.step()
    if (it+1) % 50 == 0:
        print("[Iter %d] Loss %f" % (it+1, float(loss)))
```

```
[Iter 50] Loss 0.095599
[Iter 100] Loss 0.027365
[Iter 150] Loss 0.017290
[Iter 200] Loss 0.012128
[Iter 250] Loss 0.009070
[Iter 300] Loss 0.007097
```

3.3 Part (c)

```
def sample_sequence(model, hidden, max_len=20, temperature=1):
    """
    Return a sequence generated from the model's decoder
        - model: an instance of the AutoEncoder model
        - hidden: a hidden state (e.g. computed by the encoder)
        - max_len: the maximum length of the generated sequence
        - temperature: described in Part (d)
    """
    # We'll store our generated sequence here
    generated_sequence = []
    # Set input to the <BOS> token
    inp = torch.Tensor([text_field.vocab.stoi["<bos>"]]).long()
    for p in range(max_len):
        # compute the output and next hidden unit
        output, hidden = model.decode(inp.unsqueeze(0), hidden)
        # Sample from the network as a multinomial distribution
        output_dist = output.data.view(-1).div(temperature).exp()
        top_i = int(torch.multinomial(output_dist, 1)[0])
        # Add predicted word to string and use as next input
        word = text_field.vocab.itos[top_i]
        # Break early if we reach <eos>
        if word == "<eos>":
            break
        generated_sequence.append(word)
        inp = torch.Tensor([top_i]).long()
    return generated_sequence

hidden = model.encode(input_seq)
print(sample_sequence(model, hidden))
print(sample_sequence(model, hidden))
print(sample_sequence(model, hidden))
print(sample_sequence(model, hidden))
print(sample_sequence(model, hidden))
```

```
['zambian', 'president', 'swears', 'in', 'new', 'army', 'chief']
['zambian', 'president', 'swears', 'in', 'new', 'army', 'chief']
['zambian', 'president', 'swears', 'in', 'new', 'army', 'chief']
['zambian', 'president', 'swears', 'in', 'new', 'army', 'chief']
['zambian', 'president', 'swears', 'in', 'new', 'army', 'chief']
```

3.4 Part (d)

```
# Include the generated sequences and explanation in your PDF
                                report.
temperature = [1.5, 2,5]
for i in temperature:
  print("Current Temperature: ", i)
  print(sample_sequence(model, hidden, temperature=i))
  print(sample_sequence(model, hidden, temperature=i))
  print(sample_sequence(model, hidden, temperature=i))
  print(sample_sequence(model, hidden, temperature=i))
  print(sample_sequence(model, hidden, temperature=i))
```

```
Current Temperature:  1.5
['zambian', 'president', 'swears', 'in', 'expel', 'army', 'chief',
                                'significant', 'stresses', 'norsk
                                ', 'measles', 'twins', 'house', '
                                repatriated', 'goldman', '
                                recognize', 'army', 'chief']
['embrace', 'barring', 'apology', 'president', 'in', 'new', 'non-
                                essential', 'allegedly', 'assad',
                                 'chief', 'poaches', 'better', '
                                president-elect', 'keytruda', '
                                epic', 'hathaway', 'sacked', 'in'
                                , 'new', 'army']
['zambian', 'prime', 'weakened', 'prayer', 'coronavirus', 'flavored
                                ', 'airliner', 'gloomy', 'prince'
                                , 'pais', 'isner', 'president', '
                                swears', 'stocks-energy', '
                                present', 'blackface', 'fails', '
                                themes', 'valero', 'brexit']
['zambian', 'lavrov', 'in', 'new', 'army', 'chief', 'surcharge', '
                                landmark', 'fourteen', 'lease', '
                                _num_-amazon', 'in', 'graft', '
                                open', 'your', 'plotting', '
                                arrivals', 'grocery', 'reel']
['zambian', 'hindu', 'carrefour', 'lula', 'in', 'new', 'army', '
                                chief']
Current Temperature:  2
['president', 'brexiteer', 'visas', 'violates', 'klm', '
                                unacceptable', 'offer', '
                                renewable', 'renewal', '_num_', '
                                nation', 'arm', 'shown', '
                                manufacturing', 'swaps', 'dc', '
                                bnp', 'lauren', 'kurds', '
                                bushfire']
['lining', 'corrected-update', 'emir', 'restaurants', 'taste', '
                                shells', 'tentative', 'treasuries
                                -yields', 'marlins', 'minimum', '
                                shipper', 'interested', 'vessels'
                                , 'cautious', 'text', 'zambian',
                                'juul', 'lifts', 'meo', 'youngest
                                ']
['resuming', 'dents', 'army', 'slowly', 'avoids', 'slots', 'cover',
                                 'wealth', '_num_-vw', 'surgery',
                                 'wreckage', 'past', 'army', '
                                army', 'wheels', 'actors', '
```

```
                                    damaging', 'breaking', 'triggers'
                                    , '_num_-investor']
['robotic', 'camps', 'bryant', 'ryder', 'trips', 'default', '
                                    adelaide', 'arts', 'thousands', '
                                    pullback', 'tin', 'whales', '
                                    president', 'reporter', 'rate', '
                                    sears', 'format', 'swears', 'pe',
                                     'investigations']
['tumble', 'pop', 'regain', 'in', 'ex', 'bbc', 'forecasts', '_num_-
                                    turkey', 'autumn', 'murdered', '
                                    expense', 'maersk', 'varadkar', '
                                    bourses', 'removing', 'broadly',
                                    'suspected', 'riding', '/', '
                                    apollo']
Current Temperature:  5
['cypress', 'lagerfeld', 'jays', 'counted', 'pat', 'gauff', '
                                    crowded', 'footage', 'activism',
                                    'post', 'nyc', 'burial', '
                                    passports', 'sony', 'bankers', '
                                    bibi', 'modestly', 'products', '
                                    sensor', 'ammunition']
['ankara', 'chesapeake', 'public', 'mid-2020', 'suntrust', 'grows',
                                     'snapshot-wall', '76ers', '
                                    dreams', 'stops', '_num_-tesla',
                                    'bidens', 'dassault', 'menaces',
                                    'pedo', 'fifa', 'ramping', '
                                    withdrawal', 'damaged', 'button']
['advises', 'mobil', 'defuse', 'feb.', 'jetblue', 'resumption', '
                                    _num_-bunge', 'basin', 'pumps', '
                                    dangerous', 'air', '_num_-after',
                                     'lula', 'cocoa', 'm', 'debris',
                                    'speaks', 'angels', 'deposits', '
                                    uphold']
['hottest', 'commercial', 'breath', 'believed', 'occidental', 'peer
                                    ', '_num_-global', 'd', '_num_-
                                    world', 'stocks', 'wave', '
                                    cripple', 'evacuation', 'bearish'
                                    , 'child', 'increase', 'agree', '
                                    anti-trust', 'oks', 'apec']
['m', 'panasonic', 'papua', 'celebrate', 'apache', 'refile-us', '
                                    scuffle', "p'nyang", 'liquidation
                                    ', 'montpellier', 'point', '
                                    addressed', 'recovery', 'palace',
                                     'rematch', 'high-stakes', '
                                    import', 'heir', 'stick', 'wheat'
                                    ]
```

If we set the temperature to be high, then that means that we will get a more diverse sample, with potentially more mistakes

4 Question 4 Training the autoencoder using data augmentation

4.1 Part (a)

1) we can scale the image and padding with noise
2) we can rotate the image and padding with noise
3) we can edit the RGB channel in order to change the brightness of the image

4.2 Part (b)

```
def tokenize_and_randomize(headline,
                           drop_prob=0.1,  # probability of
                                              dropping
                                               a
                                              word
                           blank_prob=0.1, # probability of "
                                              blanking
                                              " out
                                              a word
                           sub_prob=0.1,   # probability of
                                              substituting
                                               a
                                              word
                                              with a

                                              random
                                               one
                           shuffle_dist=3): # maximum distance to
                                              shuffle
                                               a
                                              word
    """
    Add 'noise' to a headline by slightly shuffling the word order,
    dropping some words, blanking out some words (replacing with
                                 the <pad> token)
    and substituting some words with random ones.
    """
    headline = [vocab.stoi[w] for w in headline.split()]
    n = len(headline)
    # shuffle
    headline = [headline[i] for i in get_shuffle_index(n,
                                 shuffle_dist)]

    new_headline = [vocab.stoi['<bos>']]
    for w in headline:
        if random.random() < drop_prob:
            # drop the word
            pass
        elif random.random() < blank_prob:
            # replace with blank word
            new_headline.append(vocab.stoi["<pad>"])
        elif random.random() < sub_prob:
```

```
            # substitute word with another word
            new_headline.append(random.randint(0, vocab_size - 1))
        else:
            # keep the original word
            new_headline.append(w)
    new_headline.append(vocab.stoi['<eos>'])
    return new_headline

def get_shuffle_index(n, max_shuffle_distance):
    """ This is a helper function used to shuffle a headline with n
                                    words,
    where each word is moved at most max_shuffle_distance. The
                                    function does
    the following:
       1. start with the *unshuffled* index of each word, which
          is just the values [0, 1, 2, ..., n]
       2. perturb these "index" values by a random floating-point
                                    value between
          [0, max_shuffle_distance]
       3. use the sorted position of these values as our new index
    """
    index = np.arange(n)
    perturbed_index = index + np.random.rand(n) * 3
    new_index = sorted(enumerate(perturbed_index), key=lambda x: x[
                                    1])
    return [index for (index, pert) in new_index]
```

```
# Report your values here. Make sure that you report the actual
                                    values,
# and not just the code used to get those values
def listToString(s):
    ans = ""
    for i in s:
        ans += i
        ans += " "
    ans = ans [:-1]
    return ans

original = train_data[0].title
ori_str = listToString(original)
print("Original headlines: ",ori_str)
for i in range(5):
  idx = tokenize_and_randomize(ori_str)
  curr = []
  for j in idx:
    curr.append(vocab.itos[j])
  print("Headline", i, curr)
```

```
Original headlines:  <bos> dems move to end shutdown , without wall
                                    money <eos>
Headline 0 ['<bos>', 'move', '<bos>', 'dems', 'to', 'end', ',', '<
                                    pad>', 'wall', 'slow', 'money', '
                                    <eos>', '<eos>']
Headline 1 ['<bos>', 'net', 'move', 'solution', 'markets-dollar', '
                                    ,', 'wall', '<pad>', '<pad>', '<
                                    eos>']
```

```
Headline 2 ['<bos>', '<pad>', 'admit', 'to', 'end', 'shutdown', '
                                  wall', 'without', '<eos>', 'money
                                  ', '<eos>']
Headline 3 ['<bos>', '<bos>', 'dems', 'move', 'to', 'end', '
                                  shutdown', ',', 'without', 'money
                                  ', 'wall', '<eos>', '<eos>']
Headline 4 ['<bos>', 'dems', 'move', 'end', 'shutdown', ',', 'ask',
                                   'money', '<eos>', 'wall', '<eos>
                                  ']
```

4.3 Part (c)

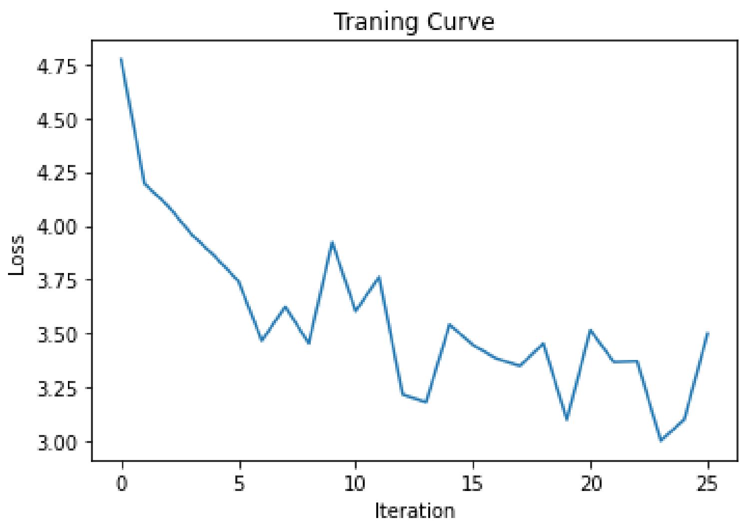
```
def train_autoencoder(model, batch_size=64, learning_rate=0.001,
                                  num_epochs=10):
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    criterion = nn.CrossEntropyLoss()
    losses = []
    for ep in range(num_epochs):
        # We will perform data augmentation by re-reading the input
                                           each time
        field = data.Field(sequential=True,
                              tokenize=tokenize_and_randomize,
                              include_lengths=True,
                              batch_first=True,
                              use_vocab=False,
                              pad_token=vocab.stoi['<pad>'])
        dataset = data.TabularDataset(train_path, "tsv", [('title',
                                           field)])

        # This BucketIterator will handle padding of sequences that
                                           are not of the same
                                          length
        train_iter = data.BucketIterator(dataset,
                                batch_size=batch_size,
                                sort_key=lambda x: len(x.title),
                                repeat=False)
        for it, ((xs, lengths), _) in enumerate(train_iter):
            # Fill in the training code here
            optimizer.zero_grad()
            output, _ = model(xs[:,:-1])
            target = xs[:,1:]
            loss = criterion(output.reshape(-1, vocab_size), target
                                             .reshape(-1))
            loss.backward()
            optimizer.step()
            if (it+1) % 100 == 0:
                losses.append(loss)
                print("[Iter %d] Loss %f" % (it+1, float(loss)))
    plt.plot(losses)
    plt.title("Traning Curve")
    plt.xlabel("Iteration")
    plt.ylabel("Loss")
    plt.show()
        # Optional: Compute and track validation loss
        #val_loss = 0
```

```
        #val_n = 0
        #for it, ((xs, lengths), _) in enumerate(valid_iter):
        #    zs = model(xs)
        #    loss = None # TODO
        #    val_loss += float(loss)

# Include your training curve or output to show that your training
                                  loss is trending down
model = AutoEncoder(vocab_size, 128, 128)
train_autoencoder(model,64,0.001,1)
```

```
[Iter 100] Loss 4.773969
[Iter 200] Loss 4.194685
[Iter 300] Loss 4.092672
[Iter 400] Loss 3.960555
[Iter 500] Loss 3.856952
[Iter 600] Loss 3.742683
[Iter 700] Loss 3.465442
[Iter 800] Loss 3.624000
[Iter 900] Loss 3.452881
[Iter 1000] Loss 3.922388
[Iter 1100] Loss 3.602681
[Iter 1200] Loss 3.762640
[Iter 1300] Loss 3.215091
[Iter 1400] Loss 3.179993
[Iter 1500] Loss 3.539971
[Iter 1600] Loss 3.445369
[Iter 1700] Loss 3.382091
[Iter 1800] Loss 3.347859
[Iter 1900] Loss 3.452721
[Iter 2000] Loss 3.097118
[Iter 2100] Loss 3.514776
[Iter 2200] Loss 3.366509
[Iter 2300] Loss 3.369957
[Iter 2400] Loss 3.000375
[Iter 2500] Loss 3.098118
[Iter 2600] Loss 3.498921
```



4.4 Part (d)

```
model = AutoEncoder(10000, 128, 128)
checkpoint_path = '/content/gdrive/My Drive/CSC413/A/A3/Data/
                                    p4model.pk'
model.load_state_dict(torch.load(checkpoint_path))
```

```
# Include the generated sequences and explanation in your PDF
                                    report.

headline = train_data[10].title
input_seq = torch.Tensor([vocab.stoi[w] for w in headline]).
                                    unsqueeze(0).long()

# ...
hidden = model.encode(input_seq)
temperature = [0.7, 0.9,1.5]
for i in temperature:
  print("Current Temperature: ", i)
  print(sample_sequence(model, hidden, temperature=i))
  print(sample_sequence(model, hidden, temperature=i))
  print(sample_sequence(model, hidden, temperature=i))
  print(sample_sequence(model, hidden, temperature=i))
  print(sample_sequence(model, hidden, temperature=i))
```

```
Current Temperature:  0.7
['wall', 'street', 'rises', ',', 'limps', 'across', 'melbourne', '
                                    will', 'march', '<pad>', '
                                    australian', 'january', '
                                    political']
['wall', 'street', 'rises', ',', 'limps', 'across', 'the', 'finish'
                                    , 'line', 'of', 'a', '
                                    thyssenkrupp', 'hosts']
```

```
['wall', 'street', 'rises', ',', 'limps', 'die', 'win', 'at', 'of',
                                      'sciences', 'election', 'four',
                                      'airspace']
['wall', 'street', 'rises', ',', 'limps', 'die', 'win', 'at', 'of',
                                      'sciences', '<pad>', '
                                      presidential', 'point']
['wall', 'street', 'rises', ',', 'limps', 'die', 'win', 'at', '$',
                                      'election', 'to', 'bans', 'cheers
                                      ']
Current Temperature:  0.9
['wall', 'street', 'rises', ',', 'limps', 'initial', ',', 'every',
                                      '<unk>', 'oil', 'after', 'bans',
                                      'questions']
['wall', 'street', 'rises', ',', 'limps', 'across', 'the', 'finish'
                                      , 'line', 'of', 'a', 'turbulent',
                                      'year']
['wall', 'street', 'rises', ',', 'limps', 'open', 'sentence', ',',
                                      'africa', 'after', 'storm', '
                                      targeted', 'brokerage']
['wall', 'street', 'rises', ',', 'limps', 'die', 'win', 'at', '$',
                                      '<pad>', 'after', 'camps', '
                                      suitors']
['wall', 'street', 'rises', ',', 'limps', 'die', 'win', 'at', '$',
                                      'election', ';', 'bank', '
                                      nightmare']
Current Temperature:  1.5
['wall', 'street', 'rises', 'first-quarter', 'hope', 'government',
                                      'escalating', 'alaska', 'profit',
                                      'man', 'oil', 'big', 'this']
['wall', 'street', 'rises', ',', 'austrian', 'join', 'golf', '
                                      bitcoin', 'new', 'trash', ',', '
                                      frontier', 'curbs']
['wall', 'street', 'rises', ',', 'limps', 'form', 'cbs', 'man', '
                                      movies', 'for', 'after', 'crisis'
                                      , 'post']
['wall', 'street', 'rises', ',', 'limps', 'head', 'kick', 'for', '
                                      off', '<pad>', 'block', 'benefits
                                      ', 'thyssenkrupp']
['wall', 'street', 'rises', ',', 'scales', 'executives', 'greener',
                                      'profit', 'now', 'u.s.', '<pad>'
                                      , 'misconduct', 'earthquake']
```

We don't want the termperature setting to be too small is because the lower temperature means that we may see repetitions of the same high probability sequence

5 Question 5 Analyzing the embeddings (interpolating between headlines)

```
valid_data = data.TabularDataset(
    path=valid_path,                 # data file path
    format="tsv",                    # fields are separated by a tab
    fields=[('title', text_field)]) # list of fields (we have only
                                       one)
```

5.1 Part (a)

```
# Write your code here
# Show that your resulting PyTorch tensor has shape `[19046, 128]`
result = torch.Tensor([])
valid_iter = data.BucketIterator(valid_data,
                                 batch_size=128,
                                 sort_key=lambda x: len(x.title),
                                 repeat=False)
for it, ((xs, lengths), _) in enumerate(valid_iter):
  result = torch.cat((result, model.encode(xs)),1)
_,m,n = result.shape
result = result.reshape(m,n)
print(result.shape)
```

```
torch.Size([19046, 128])
```

5.2 Part (b)

```
# Write your code here. Make sure to include the actual 5 closest
                                  headlines.
headline_emb = result.detach().numpy()
norms = np.linalg.norm(headline_emb, axis=1)
headline_emb_norm = (headline_emb.T / norms).T
similarities = np.matmul(headline_emb_norm, headline_emb_norm.T)
def five_closest(num):
  similar = similarities[num]
  index = (np.argsort(similar))[::-1]
  print("5 closest headline for", valid_data[num].title,":")
  for i in range(5):
    print(valid_data[index[i]].title)

five_closest(13)
```

```
5 closest headline for ['<bos>', 'asia', 'takes', 'heart', 'from',
                                  'new', 'year', 'gains', 'in', 'u.
                                  s.', 'stock', 'futures', '<eos>']
                                   :
['<bos>', 'asia', 'takes', 'heart', 'from', 'new', 'year', 'gains',
                                   'in', 'u.s.', 'stock', 'futures'
                                  , '<eos>']
['<bos>', 'uk', "'s", 'johnson', 'on', 'track', 'for', '_num_-seat'
                                  , 'majority', ':', 'focaldata', '
                                  <eos>']
['<bos>', 'trump', 'administration', 'may', 'use', 'iran', 'threat'
                                  , 'to', 'sell', 'bombs', 'to', '
                                  saudis', 'without', 'congress', "
                                  '", 'approval', '-', 'senator', '
                                  <eos>']
['<bos>', 'eu-backed', 'group', 'urges', 'no', 'escalation', 'of',
                                  'tensions', 'in', 'venezuela', '<
                                  eos>']
['<bos>', 'bombardier', 'consortium', 'wins', '$', '_num_', 'bln',
                                  'monorail', 'contract', '<eos>']
```

5.3 Part (c)

```
# Write your code here.
# Make sure to include the original headline and the 5 closest
                                  headlines.
five_closest(0)
```

```
5 closest headline for ['<bos>', 'n.korea', "'s", 'kim', 'says', '
                                  new', 'path', 'inevitable', 'if',
                                   'u.s.', 'demands', 'unilateral',
                                   'action', '<eos>'] :
['<bos>', 'n.korea', "'s", 'kim', 'says', 'new', 'path', '
                                  inevitable', 'if', 'u.s.', '
                                  demands', 'unilateral', 'action',
                                   '<eos>']
['<bos>', 'wall', 'street', 'rally', 'pauses', 'amid', 'china', '
                                  virus', 'outbreak', ',', 'growth'
                                  , 'fears', '<eos>']
['<bos>', 'slain', 'north', 'carolina', 'college', 'student', '
                                  confronted', 'gunman', ',', '
                                  saved', 'lives', '<eos>']
['<bos>', 'google', 'target', 'of', 'new', 'u.s.', 'antitrust', '
                                  probe', 'by', 'state', 'attorneys
                                  ', 'general', '<eos>']
['<bos>', 'coronavirus', 'death', 'toll', 'leaps', 'in', 'china', "
                                  's", 'hubei', 'province', ',', '
                                  party', 'bosses', 'sacked', '<eos
                                  >']
```

5.4 Part (d)

```
# Write your code here. Include your generated sequences.
headline1 = train_data[0].title
input_seq1 = torch.Tensor([vocab.stoi[w] for w in headline1]).long
                                  ().unsqueeze(0)
headline2 = train_data[1].title
input_seq2 = torch.Tensor([vocab.stoi[w] for w in headline2]).long
                                  ().unsqueeze(0)
e0 = model.encode(input_seq1)
e4 = model.encode(input_seq2)
e1 = 0.75 * e0 + 0.25 * e4
e2 = 0.50 * e0 + 0.50 * e4
e3 = 0.25 * e0 + 0.75 * e4
print("e1:")
for i in range(5):
  print(sample_sequence(model, e1, temperature=1))
print("e2:")
for i in range(5):
  print(sample_sequence(model, e2, temperature=1))
print("e3:")
for i in range(5):
  print(sample_sequence(model, e3, temperature=1))
```

```
e1:
['markets-asia', 'by', 'at', 'swap', 'hold', 'march', 'harper', '
                                  update']
['dems', 'update', 'options', 'than', 'students', 'march', 'between
                                  ', 'wall']
['permian', 'arabia', 'britain', 'employees', 'on', 'moscow', '
                                  usmca', 'top']
['dems', 'lawmakers', 'after', 'exports', 'failings', 'hopes', '
                                  peaceful', '777x']
['williams', 'asks', 'meet', 'after', 'minnesota', 'dead', '<pad>',
                                   'nyse']
e2:
['warren', 'update', 'after', 'guide', 'small', 'iran', '<unk>', '
                                  xinhua']
['non-binding', 'turkish', 'sales', 'tips', 'criticism', 'airliner'
                                  , 'reuters', 'abuse']
['markets-asia', 'government', 'moscow', 'increased', 'now', '
                                  detention', 'mlb', '<pad>']
['_num_-jpmorgan', 'squeeze', 'program', 'indicted', 'all', 'lavrov
                                  ', 'cars']
['searching', 'china', 'court', 'intimidation', 'attack', '
                                  laundering', '<pad>']
e3:
['extradition', 'long-awaited', 'at', 'declares', 'champ', 'nato',
                                  'sources']
['genetic', 'go', 'of', 'attacker', 'autos', ':', 'oilfields']
['<unk>', 'protection', 'indicts', 'as', 'petrobras', 'commander',
                                  'age']
['copa', 'botched', 'on', 'employees', 'pakistan', 'necessary', '-'
                                  ]
['copa', 'articles', 'college', 'quake', 'appear', '<unk>', '.']
```

6 Question 6 Work Allocation

```
#Ming worked on the assignment on Mar 15 - Apr 2, Hongyu worked on
                                  the assignment on Mar 15 - Apr 2
# We had meetings on Mar 15, 18, 25 and Apr 1. We also had several
                                  short chats during our working
                                  time.
# Ming did Question 2, 3 and Hongyu did question 1, 4, 5
# We built code structure, checked each other's python code
                                  function and discuss the math
                                  problems together.
```