Question1

(a)

P(Intelligence|Artificial) = 6/6 = 1

P(nothing|is) = 0

P(is like) = 3/30 = 0.1

P(intelligence is) = 2/30 = 0.07

(b)

frequencies

	artificial	intelligence	is	what	i	like	nothing
artificial	0	6	0	0	0	0	0
intelligence	1	0	2	2	0	0	1
is	2	0	0	1	0	3	0
what	0	0	2	0	2	0	0
i	0	0	0	0	0	2	0
like	2	0	1	1	0	0	1
nothing	0	0	1	0	0	0	0

Conditional probabilities

	artificial	intelligence	is	what	i	like	nothing
artificial	0	1	0	0	0	0	0
intelligence	0.17	0	0.33	0.33	0	0	0.17
is	0.33	0	0	0.17	0	0.5	0
what	0	0	0.5	0	0.5	0	0
i	0	0	0	0	0	1	0
like	0.4	0	0.2	0.2	0	0	0.2
nothing	0	0	1	0	0	0	0

(c)
frequencies

	artificial	intelligence	is	what	i	like	nothing
artificial	0.5	6.5	0.5	0.5	0.5	0.5	0.5
intelligence	1.5	0.5	2.5	2.5	0.5	0.5	1.5
is	2.5	0.5	0.5	1.5	0.5	3.5	0.5
what	0.5	0.5	2.5	0.5	2.5	0.5	0.5
i	0.5	0.5	0.5	0.5	0.5	2.5	0.5
like	2.5	0.5	1.5	1.5	0.5	0.5	1.5
nothing	0.5	0.5	1.5	0.5	0.5	0.5	0.5

Conditional probabilities

	artificial	intelligence	is	what	i	like	nothing
artificial	0.05	0.68	0.05	0.05	0.05	0.05	0.05
intelligence	0.16	0.05	0.26	0.26	0.05	0.05	0.16
is	0.26	0.05	0.05	0.16	0.05	0.37	0.05
what	0.07	0.07	0.33	0.07	0.33	0.07	0.07
i	0.09	0.09	0.09	0.09	0.09	0.45	0.09
like	0.29	0.06	0.18	0.18	0.06	0.06	0.18
nothing	0.11	0.11	0.33	0.11	0.11	0.11	0.11

(d)
Sentence1: artificial intelligence is nothing
Bigrams: (artificial intelligence) (intelligence is) (is nothing)
Part b model: 1*0.33*0 = 0
Part c model: 0.68*0.26*0.05 =0.00884

Sentence2: i artificial like intelligence
Bigrams: (i artificial) (artificial like) (like intelligence)
Part b model: 0*0*0 = 0
Part c model: 0.09*0.05*0.06 = 0.00027

Sentence3: i like artificial intelligence
Bigrams: (i like) (like artificial) (artificial intelligence)
Part b model: 1*0.4*1 = 0.4
Part c model: 0.45*0.29*0.68 = 0.08874

Question2
Training set with stop-word removal

Sentence	Sense
orange very tasty	Sense1
fruit basket has orange apple banana pineapple	Sense1
maroon blue orange favorite colors	Sense2
do have these pants orange black	sense2
sun had just begun set casting warm orange glow over horizon signaling end another day	sense2
citrusy scent freshly squeezed orange juice wafted through air making mouth water with anticipation	sense1
i recommend try fresh orange juice do not like apple juice	sense1
glow orange shirt unreal	sense2
both orange juice apple juice tasty but i prefer apples	sense1
colours rainbow red orange yellow green blue indigo violet	sense2

V = 33
Smooth = 0.5
P(sense1) = 0.5
P(sense2) = 0.5

The sentence to disambiguate
(a)
orange favorite color because bright cheerful
P(favorite|sense1) = (0+0.5) /(26+33*0.5) = 0.01176
P(color|sense1) = (0+0.5) /(26+33*0.5) = 0.011764
P(because|sense1) = (0+0.5) /(26+33*0.5) = 0.011764

P(favorite|sense2) = (1+0.5) /(21+33*0.5) = 0.04
P(color|sense2) = (1+0.5) /(21+33*0.5) = 0.04
P(because|sense2) = (0+0.5) /(21+33*0.5) = 0.013333

score(sense1) = log(0.5) + log(0.011764) + log(0.011764) + log(0.011764) = -6.08936
score(sense2) = log(0.5) + log(0.04) + log(0.04) + log(0.013333) = -4.971982

Since score(sense2) > score(sense1). Sense2 is more probable

(b)
breakfast menu i recommend omelettes which come with orange juice
P(which|sense1) = (0+0.5) /(26+33*0.5) = 0.01176
P(come|sense1) = (0+0.5) /(26+33*0.5) = 0.01176
P(with|sense1) = (0+0.5) /(26+33*0.5) = 0.01176
P(juice|sense1) = (4+0.5) /(26+33*0.5) = 0.10588

P(which|sense2) = (0+0.5) /(21+33*0.5) = 0.013333
P(come|sense2) = (0+0.5) /(21+33*0.5) = 0.013333
P(with|sense2) = (0+0.5) /(21+33*0.5) = 0.013333
P(juice|sense2) = (0+0.5) /(21+33*0.5) = 0.013333

score(sense1) = log(0.5) + log(0.01176) + log(0.01176) + log(0.01176) + log(0.10588) = -7.06499
score(sense2) = log(0.5) + log(0.013333) + log(0.013333) + log(0.013333) + log(0.013333) = -7.80132

Since score(sense1) > score(sense2). Sense1 is more probable

(c)
tropical fruit salad filled with chunks pineapple mango orange apple all tossed tangy citrus dressing

P(chunks|sense1) = (0+0.5) /(26+33*0.5) = 0.01176
P(pineapple|sense1) = (1+0.5) /(26+33*0.5) = 0.03529
P(mango|sense1) = (0+0.5) /(26+33*0.5) = 0.01176
P(apple|sense1) = (2+0.5) /(26+33*0.5) = 0.05882
P(all|sense1) = (0+0.5) /(26+33*0.5) = 0.01176
P(tossed|sense1) = (0+0.5) /(26+33*0.5) = 0.01176

P(chunks|sense2) = (0+0.5) /(21+33*0.5) = 0.01333
P(pineapple|sense2) = (0+0.5) /(21+33*0.5) = 0.01333
P(mango|sense2) = (0+0.5) /(21+33*0.5) = 0.01333
P(apple|sense2) = (0+0.5) /(21+33*0.5) = 0.01333
P(all|sense2) = (0+0.5) /(21+33*0.5) = 0.01333
P(tossed|sense2) = 0+0.5) /(21+33*0.5) = 0.01333

score(sense1) = log(0.5) + log(0.01176) + log(0.01176) + log(0.01176) + log(0.01176) + log(0.03529) + log(0.05882) = -10.70222
score(sense2) = log(0.5) + 6*log(0.01333) + log(0.04) = -11.07482

Since score(sense1) > score(sense2). Sense1 is more probable

Question3

Training sentence
Artificial Intelligence can replace humans in several jobs and applications
Dimension = 3, window size = 4
(a)

instance	Context word -2	Context word -1	Context word +1	Context word+2	To predict
1	artificial	intelligence	replace	humans	can
2	intelligence	can	humans	in	replace
3	can	replace	in	several	humans
4	replace	humans	several	jobs	in
5	humans	in	jobs	and	several
6	in	several	and	applications	jobs

(b) defline the vocabulary, assume alphabetical ordering

word	Hot vector									
and	1	0	0	0	0	0	0	0	0	0
applications	0	1	0	0	0	0	0	0	0	0
artificial	0	0	1	0	0	0	0	0	0	0
can	0	0	0	1	0	0	0	0	0	0
humans	0	0	0	0	1	0	0	0	0	0
in	0	0	0	0	0	1	0	0	0	0
intelligence	0	0	0	0	0	0	1	0	0	0
jobs	0	0	0	0	0	0	0	1	0	0
replace	0	0	0	0	0	0	0	0	1	0
several	0	0	0	0	0	0	0	0	0	1

Represent the training data using the one hot encoded words

instance	context	word	Hot vector									
1	Context word -2	artificial	0	1	0	0	0	0	0	0	0	0
	Context word –1	intelligence	0	0	0	0	0	0	1	0	0	0
	Context word +1	replace	0	0	0	0	0	0	0	0	1	0
	Context word +2	humans	0	0	0	0	1	0	0	0	0	0
2	Context word -2	intelligence	0	0	0	0	0	0	1	0	0	0
	Context word –1	can	0	0	0	1	0	0	0	0	0	0
	Context word +1	humans	0	0	0	0	1	0	0	0	0	0
	Context word +2	in	0	0	0	0	0	1	0	0	0	0
3	Context word -2	can	0	0	0	1	0	0	0	0	0	0
	Context word –1	replace	0	0	0	0	0	0	0	0	1	0
	Context word +1	in	0	0	0	0	0	1	0	0	0	0
	Context word +2	several	0	0	0	0	0	0	0	0	0	1
4	Context word -2	replace	0	0	0	0	0	0	0	0	1	0
	Context word –1	humans	0	0	0	0	1	0	0	0	0	0
	Context word +1	several	0	0	0	0	0	0	0	0	0	1
	Context word +2	jobs	0	0	0	0	0	0	0	1	0	0
5	Context word -2	humans	0	0	0	0	1	0	0	0	0	0
	Context word –1	in	0	0	0	0	0	1	0	0	0	0
	Context word +1	jobs	0	0	0	0	0	0	0	1	0	0
	Context word +2	and	1	0	0	0	0	0	0	0	0	0
6	Context word -2	in	0	0	0	0	0	1	0	0	0	0
	Context word –1	several	0	0	0	0	0	0	0	0	0	1
	Context word +1	and	1	0	0	0	0	0	0	0	0	0
	Context word +2	applications	0	1	0	0	0	0	0	0	0	0

Represent labels using the one hot encoded words

instance	To predict	Hot vector									
1	can	0	0	0	1	0	0	0	0	0	0
2	replace	0	0	0	0	0	0	0	0	1	0
3	humans	0	0	0	0	1	0	0	0	0	0
4	in	0	0	0	0	0	1	0	0	0	0
5	several	0	0	0	0	0	0	0	0	0	1
6	jobs	0	0	0	0	0	0	0	1	0	0

(c)
n = 4
m = 3
sizes for Ii , Wi (for each 1 ≤ i ≤ n), W', and O
$I_1 = I_2 = I_3 = I_4$ = 1x10, I = 4x10
$W_1 = W_2 = W_3 = W_4$ = 10x3
W' = 3x10
O = 1x10

(d)(e)
First feed instance 1
Calculate the output of each hidden node for each context word
I = [[0,1,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,1,0,0,0], [0,0,0,0,0,0,0,0,1,0], [0,0,0,0,1,0,0,0,0,0]]
W = [[1,3,2],[2,2,1],[3,1,2],[3,1,2],[2,2,2],[1,3,2],[1,3,2],[2,2,1],[2,1,3],[1,1,3]]
H = IxW = [[2, 2, 1], [1, 3, 2], [2, 1, 3], [2, 2, 2]]

Take the average
H_avg = [1.75, 2.0, 2.0]

Calculate output
O = H_avg x W' = [11.75, 17.5, 17.0, 19.25, 13.75, 19.5, 27.25, 33.25, 35.5, 11.25]

Calculate softmax probabilities for the output
softmax(O) = [4.384137620446313e-11, 1.3774550937513794e-08, 8.354687467375514e-09, 7.926706768579412e-08, 3.239463882291014e-10, 1.0178092961486718e-07, 0.00023629179852734526, 0.09532691519211534, 0.9044365894377426, 2.6591138832002768e-11]
Calculate error
E = O - T = [4.384137620446313e-11, 1.3774550937513794e-08, 8.354687467375514e-09, -0.9999999207329323, 3.239463882291014e-10, 1.0178092961486718e-07, 0.00023629179852734526, 0.09532691519211534, 0.9044365894377426, 2.6591138832002768e-11]

Second feed instance 2

Calculate the output of each hidden node for each context word

I = [[0,0,0,0,0,0,1,0,0,0], [0,0,0,1,0,0,0,0,0,0], [0,0,0,0,1,0,0,0,0,0], [0,0,0,0,0,1,0,0,0,0]]

W = [[1,3,2],[2,2,1],[3,1,2],[3,1,2],[2,2,2],[1,3,2],[1,3,2],[2,2,1],[2,1,3],[1,1,3]]

H = IxW = [[1, 3, 2], [3, 1, 2], [2, 2, 2], [1, 3, 2]]

Take the average

H_avg = [1.75, 2.25, 2.0]

Calculate output

O = H_avg x W' = [12.75, 18.75, 17.5, 20.0, 14.0, 20.0, 28.25, 34.75, 37.25, 11.75]

Calculate softmax probabilities for the output

softmax(O) = [2.1157982750822326e-11, 8.535739453904352e-09, 2.4455302982923762e-09, 2.979265808966424e-08, 7.384861608843182e-11, 2.979265808966424e-08, 0.00011403514739332934, 0.07584952416165358, 0.9240363700215769, 7.783586870687532e-12]

Calculate error

E = O - T = [2.1157982750822326e-11, 8.535739453904352e-09, 2.4455302982923762e-09, -0.9999999702073419, 7.384861608843182e-11, 2.979265808966424e-08, 0.00011403514739332934, 0.07584952416165358, 0.9240363700215769, 7.783586870687532e-12] comp6721_A4

April 23, 2023

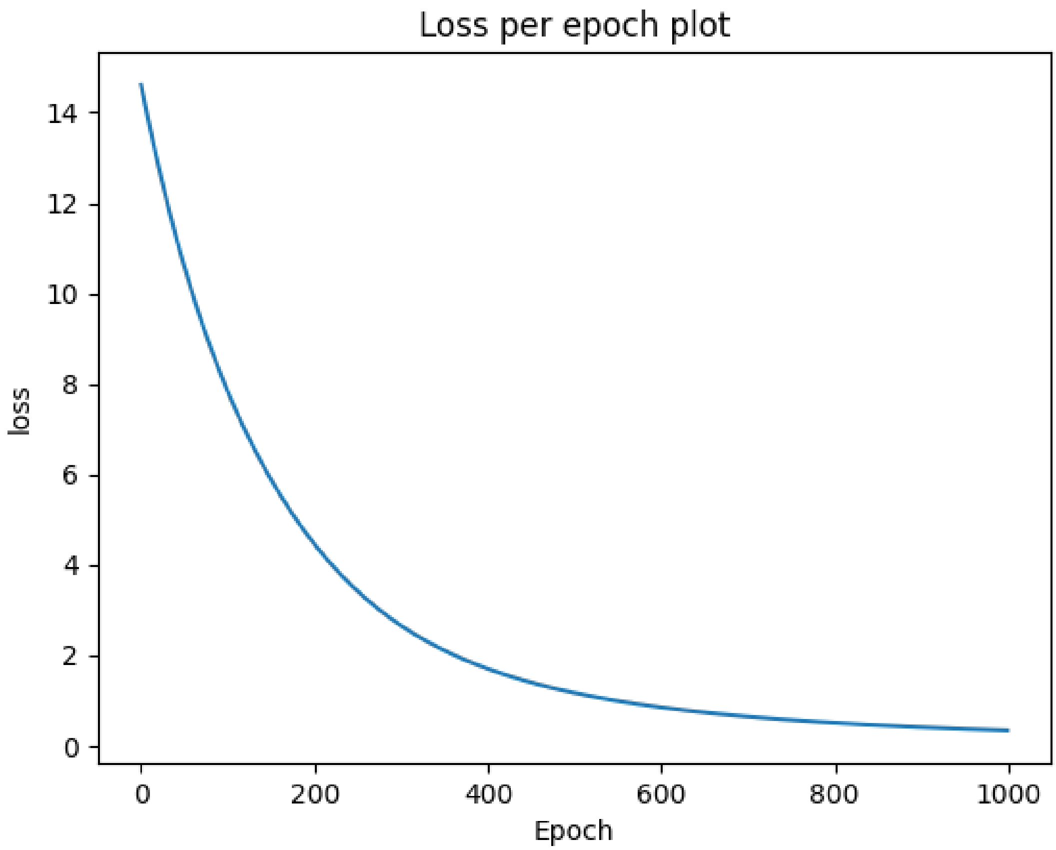
0.1 Implementation Questions

Question 1

[18]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt


CONTEXT_SIZE = 4
EMBEDDING_DIM = 3

test_sentence = '''Artificial Intelligence can replace humans in several jobs␣
↪and applications'''.split()
vocab = set(test_sentence)
word_to_ix = {word: i for i, word in enumerate(vocab)}
print(word_to_ix)
ngrams = [([test_sentence[i], test_sentence[i + 1], test_sentence[i + 3],␣
↪test_sentence[i + 4]], test_sentence[i + 2]) for i in␣
↪range(len(test_sentence) - CONTEXT_SIZE)]
print(ngrams)

class CBOWModeler(nn.Module):
    def __init__(self, vocab_size, embedding_dim, context_size):
        super(CBOWModeler, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.linear1 = nn.Linear(context_size * embedding_dim, 128)
        self.linear2 = nn.Linear(128, vocab_size)

    def forward(self, inputs):
        embeds = self.embeddings(inputs).view((1, -1))
        out = F.relu(self.linear1(embeds))
        out = self.linear2(out)
        log_probs = F.log_softmax(out, dim=1)
        return log_probs
```

```
losses = []
loss_function = nn.NLLLoss()
model = CBOWModeler(len(vocab), EMBEDDING_DIM, CONTEXT_SIZE)
optimizer = optim.SGD(model.parameters(), lr=0.001)

for epoch in range(1000):
    total_loss = 0
    for context, target in ngrams:
        context_idxs = torch.tensor([word_to_ix[w] for w in context],␣
 ↪dtype=torch.long)
        model.zero_grad()
        log_probs = model(context_idxs)
        loss = loss_function(log_probs, torch.tensor([word_to_ix[target]],␣
 ↪dtype=torch.long))
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    losses.append(total_loss)

plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('loss')
plt.title('Loss per epoch plot')
plt.show()
```

```
{'Artificial': 0, 'can': 1, 'in': 2, 'several': 3, 'and': 4, 'humans': 5,
'jobs': 6, 'applications': 7, 'Intelligence': 8, 'replace': 9}
[(['Artificial', 'Intelligence', 'replace', 'humans'], 'can'), (['Intelligence',
'can', 'humans', 'in'], 'replace'), (['can', 'replace', 'in', 'several'],
'humans'), (['replace', 'humans', 'several', 'jobs'], 'in'), (['humans', 'in',
'jobs', 'and'], 'several'), (['in', 'several', 'and', 'applications'], 'jobs')]
```



Question 2

(a) Using the spacy library, extract and print named entities from the given text

[2]:
```
import spacy

text = '''
Who are you talking to right now? Who is it you think you see? Do you know
how much I make a year? I mean, even if I told you, you wouldn't believe
it. Do you know what would happen if I suddenly decided to stop going into
work? A business big enough that it could be listed on the NASDAQ goes belly
up. Disappears! It ceases to exist without me. No, you clearly don't know who
you're talking to, so let me clue you in. I am not in danger, Skyler. I am the
danger. A guy opens his door and gets shot and you think that of me? No. I
am the one who knocks!
'''

nlp = spacy.load("en_core_web_sm")
doc = nlp(text)
for ent in doc.ents:
```

```
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

```
a year 92 98 DATE
NASDAQ 283 289 ORG
Skyler 441 447 PERSON
```

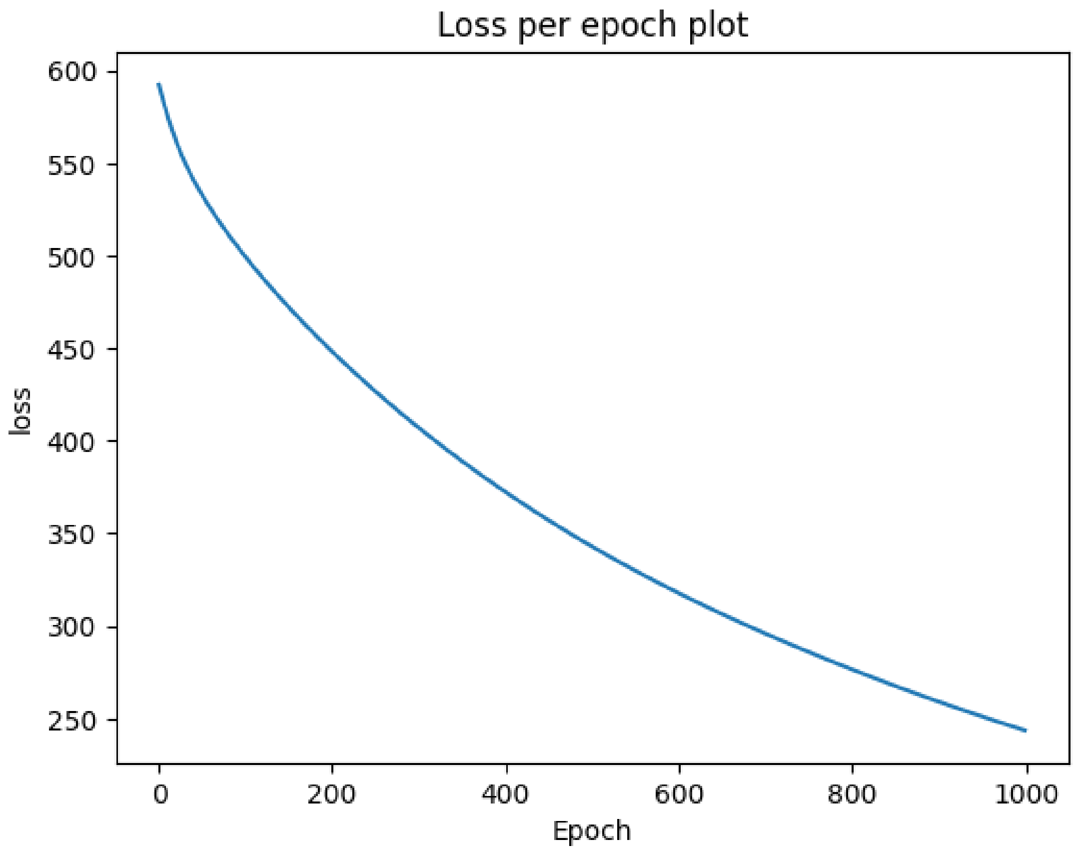
(b)

[10]:
```
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


CONTEXT_SIZE = 2
EMBEDDING_DIM = 3

test_sentence = text = '''
Who are you talking to right now? Who is it you think you see? Do you know
how much I make a year? I mean, even if I told you, you wouldn't believe
it. Do you know what would happen if I suddenly decided to stop going into
work? A business big enough that it could be listed on the NASDAQ goes belly
up. Disappears! It ceases to exist without me. No, you clearly don't know who
you're talking to, so let me clue you in. I am not in danger, Skyler. I am the
danger. A guy opens his door and gets shot and you think that of me? No. I
am the one who knocks!
'''.split()
vocab = set(test_sentence)
word_to_ix = {word: i for i, word in enumerate(vocab)}
print(word_to_ix)
ngrams = []
for i in range(len(test_sentence) - CONTEXT_SIZE):
    tup = [test_sentence[j] for j in np.arange(i + 1 , i + CONTEXT_SIZE + 1) ]
    ngrams.append((test_sentence[i],tup))
print(ngrams)

class SkipgramModeler(nn.Module):
    def __init__(self, vocab_size, embedding_dim, context_size):
        super(SkipgramModeler, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.linear1 = nn.Linear(embedding_dim, 128)
        self.linear2 = nn.Linear(128, context_size * vocab_size)

    def forward(self, inputs):
        embeds = self.embeddings(inputs).view((1, -1))
        out1 = F.relu(self.linear1(embeds))
```

```python
        out2 = self.linear2(out1)
        log_probs = F.log_softmax(out2, dim=1).view(CONTEXT_SIZE,-1)
        return log_probs


losses = []
loss_function = nn.NLLLoss()
model = SkipgramModeler(len(vocab), EMBEDDING_DIM, CONTEXT_SIZE)
optimizer = optim.SGD(model.parameters(), lr=0.001)

for epoch in range(1000):
    total_loss = 0
    for context, target in ngrams:
        context_idxs = torch.tensor([word_to_ix[context]], dtype=torch.long)
        model.zero_grad()
        log_probs = model(context_idxs)
        target_list = torch.tensor([word_to_ix[w] for w in target], dtype=torch.
 ↪long)
        loss = loss_function(log_probs, target_list)
        loss.backward()
        optimizer.step()

        # Get the Python number from a 1-element Tensor by calling tensor.item()
        total_loss += loss.item()
    losses.append(total_loss)

plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('loss')
plt.title('Loss per epoch plot')
plt.show()
```

```
{'of': 0, 'decided': 1, 'A': 2, 'clearly': 3, 'now?': 4, 'me.': 5, 'danger,': 6,
'clue': 7, 'exist': 8, 'right': 9, 'mean,': 10, 'opens': 11, 'let': 12, 'be':
13, 'is': 14, 'on': 15, 'would': 16, 'you're': 17, 'who': 18, 'going': 19, 'to':
20, 'goes': 21, 'you': 22, 'the': 23, 'so': 24, 'knocks!': 25, 'and': 26, 'in':
27, 'could': 28, 'what': 29, 'me': 30, 'talking': 31, 'It': 32, 'even': 33,
'that': 34, 'think': 35, 'danger.': 36, 'am': 37, 'stop': 38, 'No.': 39,
'NASDAQ': 40, 'wouldn't': 41, 'if': 42, 'ceases': 43, 'listed': 44, 'not': 45,
'work?': 46, 'believe': 47, 'see?': 48, 'shot': 49, 'how': 50, 'don't': 51,
'much': 52, 'you,': 53, 'business': 54, 'told': 55, 'up.': 56, 'a': 57, 'know':
58, 'to,': 59, 'Skyler.': 60, 'guy': 61, 'one': 62, 'are': 63, 'me?': 64, 'I':
65, 'door': 66, 'gets': 67, 'year?': 68, 'suddenly': 69, 'it': 70, 'into': 71,
'big': 72, 'No,': 73, 'enough': 74, 'his': 75, 'happen': 76, 'make': 77,
'belly': 78, 'Disappears!': 79, 'without': 80, 'in.': 81, 'Who': 82, 'Do': 83,
'it.': 84}
[('Who', ['are', 'you']), ('are', ['you', 'talking']), ('you', ['talking',
'to']), ('talking', ['to', 'right']), ('to', ['right', 'now?']), ('right',
```

['now?', 'Who']), ('now?', ['Who', 'is']), ('Who', ['is', 'it']), ('is', ['it', 'you']), ('it', ['you', 'think']), ('you', ['think', 'you']), ('think', ['you', 'see?']), ('you', ['see?', 'Do']), ('see?', ['Do', 'you']), ('Do', ['you', 'know']), ('you', ['know', 'how']), ('know', ['how', 'much']), ('how', ['much', 'I']), ('much', ['I', 'make']), ('I', ['make', 'a']), ('make', ['a', 'year?']), ('a', ['year?', 'I']), ('year?', ['I', 'mean,']), ('I', ['mean,', 'even']), ('mean,', ['even', 'if']), ('even', ['if', 'I']), ('if', ['I', 'told']), ('I', ['told', 'you,']), ('told', ['you,', 'you']), ('you,', ['you', 'wouldn’t']), ('you', ['wouldn’t', 'believe']), ('wouldn’t', ['believe', 'it.']), ('believe', ['it.', 'Do']), ('it.', ['Do', 'you']), ('Do', ['you', 'know']), ('you', ['know', 'what']), ('know', ['what', 'would']), ('what', ['would', 'happen']), ('would', ['happen', 'if']), ('happen', ['if', 'I']), ('if', ['I', 'suddenly']), ('I', ['suddenly', 'decided']), ('suddenly', ['decided', 'to']), ('decided', ['to', 'stop']), ('to', ['stop', 'going']), ('stop', ['going', 'into']), ('going', ['into', 'work?']), ('into', ['work?', 'A']), ('work?', ['A', 'business']), ('A', ['business', 'big']), ('business', ['big', 'enough']), ('big', ['enough', 'that']), ('enough', ['that', 'it']), ('that', ['it', 'could']), ('it', ['could', 'be']), ('could', ['be', 'listed']), ('be', ['listed', 'on']), ('listed', ['on', 'the']), ('on', ['the', 'NASDAQ']), ('the', ['NASDAQ', 'goes']), ('NASDAQ', ['goes', 'belly']), ('goes', ['belly', 'up.']), ('belly', ['up.', 'Disappears!']), ('up.', ['Disappears!', 'It']), ('Disappears!', ['It', 'ceases']), ('It', ['ceases', 'to']), ('ceases', ['to', 'exist']), ('to', ['exist', 'without']), ('exist', ['without', 'me.']), ('without', ['me.', 'No,']), ('me.', ['No,', 'you']), ('No,', ['you', 'clearly']), ('you', ['clearly', 'don’t']), ('clearly', ['don’t', 'know']), ('don’t', ['know', 'who']), ('know', ['who', 'you’re']), ('who', ['you’re', 'talking']), ('you’re', ['talking', 'to,']), ('talking', ['to,', 'so']), ('to,', ['so', 'let']), ('so', ['let', 'me']), ('let', ['me', 'clue']), ('me', ['clue', 'you']), ('clue', ['you', 'in.']), ('you', ['in.', 'I']), ('in.', ['I', 'am']), ('I', ['am', 'not']), ('am', ['not', 'in']), ('not', ['in', 'danger,']), ('in', ['danger,', 'Skyler.']), ('danger,', ['Skyler.', 'I']), ('Skyler.', ['I', 'am']), ('I', ['am', 'the']), ('am', ['the', 'danger.']), ('the', ['danger.', 'A']), ('danger.', ['A', 'guy']), ('A', ['guy', 'opens']), ('guy', ['opens', 'his']), ('opens', ['his', 'door']), ('his', ['door', 'and']), ('door', ['and', 'gets']), ('and', ['gets', 'shot']), ('gets', ['shot', 'and']), ('shot', ['and', 'you']), ('and', ['you', 'think']), ('you', ['think', 'that']), ('think', ['that', 'of']), ('that', ['of', 'me?']), ('of', ['me?', 'No.']), ('me?', ['No.', 'I']), ('No.', ['I', 'am']), ('I', ['am', 'the']), ('am', ['the', 'one']), ('the', ['one', 'who']), ('one', ['who', 'knocks!'])]



Bonus Question Text pre-processing

[1]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt


# Load the data from CSV files
model_path = F"/content/drive/My Drive/comp6721-assignment/comp6721_a4_bonus.
↪pth"
```

```python
fake_df = pd.read_csv('/content/drive/My Drive/comp6721-assignment/Fake.csv')
real_df = pd.read_csv('/content/drive/My Drive/comp6721-assignment/True.csv')

# Add a label column to the dataframes
fake_df['label'] = 0
real_df['label'] = 1

# Concatenate the dataframes and shuffle the rows
df = pd.concat([fake_df, real_df], ignore_index=True, sort=False)
df = df.sample(frac=1).reset_index(drop=True)

# Split the data into train, validation, and test sets, 70-10-20
train_df, test_df = train_test_split(df, test_size=0.2, stratify=df['label'], random_state=42)
train_df, val_df = train_test_split(train_df, test_size=0.125, stratify=train_df['label'], random_state=42)

# Tokenize and preprocess the input sentences
nltk.download('stopwords')
nltk.download('wordnet')

stop_words = stopwords.words('english')
lemmatizer = WordNetLemmatizer()

def preprocess(text):
    text = re.sub('[^a-zA-Z]', ' ', text)
    text = text.lower()
    words = text.split()
    words = [lemmatizer.lemmatize(word) for word in words if not word in stop_words]
    return ' '.join(words)

train_df['text'] = train_df['text'].apply(preprocess)
val_df['text'] = val_df['text'].apply(preprocess)
test_df['text'] = test_df['text'].apply(preprocess)

# Define a function to tokenize and pad the input sentences
def tokenize_and_pad(text, word_to_idx, max_len):
    tokens = text.split()
    token_ids = [word_to_idx.get(token, 0) for token in tokens]
    pad_len = max_len - len(token_ids)
    if pad_len > 0:
        token_ids = token_ids + [0] * pad_len
    else:
        token_ids = token_ids[:max_len]
    return token_ids
```

```python
# Define a custom dataset class for the news articles
class NewsDataset(Dataset):
    def __init__(self, df, word_to_idx, max_len):
        self.texts = df['text']
        self.labels = df['label']
        self.word_to_idx = word_to_idx
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts.iloc[idx]
        label = self.labels.iloc[idx]
        token_ids = tokenize_and_pad(text, self.word_to_idx, self.max_len)
        return np.array(token_ids), label
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
```

CNN model and training

[5]:
```python
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay



# Define a simple CNN model for text classification
class CNN(nn.Module):
    def __init__(self, vocab_size, embedding_size, hidden_size, num_classes, max_len):
        super(CNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.conv1 = nn.Conv1d(embedding_size, hidden_size, kernel_size=3)
        self.pool1 = nn.MaxPool1d(kernel_size=max_len-2)
        self.fc1 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        x = self.embedding(x)
        x = x.permute(0, 2, 1)
        x = nn.functional.relu(self.conv1(x))
        x = self.pool1(x)
        x = x.squeeze(-1)
        x = self.fc1(x)
        return x


# Create a list of all the words in the training data
```

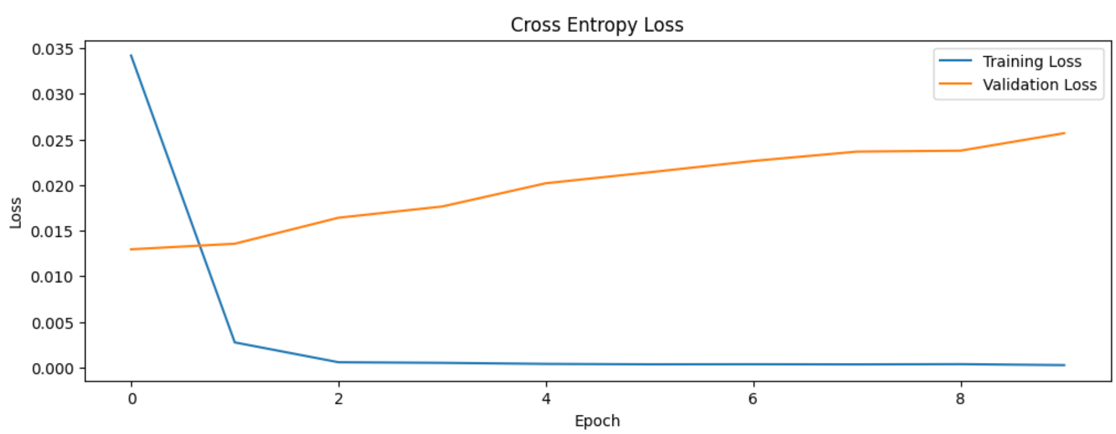
```python
words = []
for text in train_df['text']:
    words += text.split()

# Count the frequency of each word
word_counts = pd.Series(words).value_counts()

# Get the number of unique words in the training data
vocab_size = len(word_counts)
word_to_idx = {word: i + 1 for i, word in enumerate(train_df['text'].str.
 ↪split(expand=True).unstack().value_counts().index[:vocab_size])}
embedding_size = 128
hidden_size = 256
num_classes = 2
max_len = 50
batch_size = 32
lr = 1e-3
num_epochs = 10

train_dataset = NewsDataset(train_df, word_to_idx, max_len)
val_dataset = NewsDataset(val_df, word_to_idx, max_len)
test_dataset = NewsDataset(test_df, word_to_idx, max_len)

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN(vocab_size, embedding_size, hidden_size, num_classes, max_len)
# Move the model to the device (GPU or CPU)
model = model.to(device)

optimizer = optim.Adam(model.parameters(), lr=lr)
criterion = nn.CrossEntropyLoss()

train_acc_list = []
train_loss_list = []
val_acc_list = []
val_loss_list = []

for epoch in range(num_epochs):
    train_loss = 0.0
    train_acc = 0.0
    model.train()
    for batch_idx, (inputs, targets) in enumerate(train_loader):
        # Move the inputs and targets to the device
        inputs, targets = inputs.to(device), targets.to(device)
```

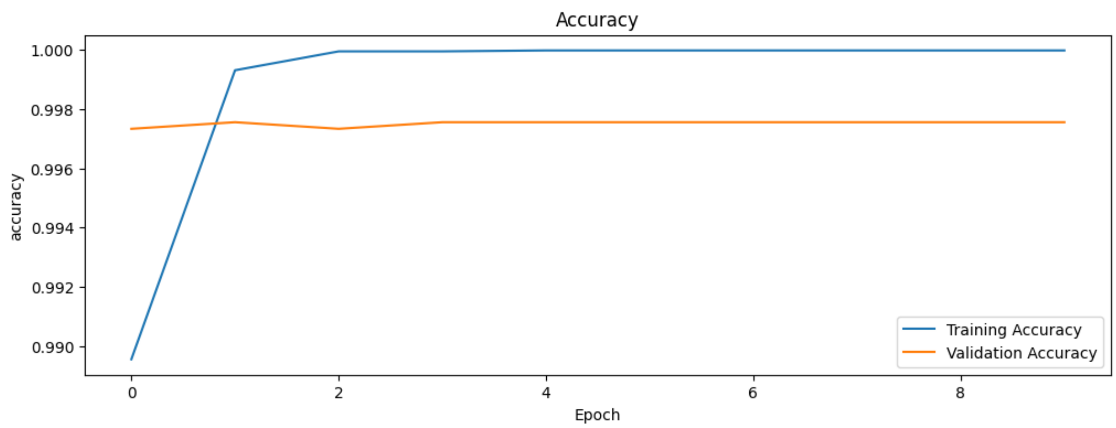
```
        optimizer.zero_grad()
        outputs = model(inputs.long())
        loss = criterion(outputs, targets.long())
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        train_acc += (predicted == targets).sum().item()

    train_loss /= len(train_loader)
    train_acc /= len(train_loader.dataset)

    # Evaluate the model on the validation set
    val_loss = 0.0
    val_acc = 0.0
    model.eval()

    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(val_loader):
            # Move the inputs and targets to the device
            inputs, targets = inputs.to(device), targets.to(device)

            outputs = model(inputs.long())
            loss = criterion(outputs, targets.long())

            val_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            val_acc += (predicted == targets).sum().item()

        val_loss /= len(val_loader)
        val_acc /= len(val_loader.dataset)

    train_acc_list.append(train_acc)
    train_loss_list.append(train_loss)
    val_acc_list.append(val_acc)
    val_loss_list.append(val_loss)
    print('Epoch [{}/{}], Train Loss: {:.4f}, Train Acc: {:.4f}, Val Loss: {:.
↪4f}, Val Acc: {:.4f}'
          .format(epoch+1, num_epochs, train_loss, train_acc, val_loss,␣
↪val_acc))

# Plot the training and validation loss
plt.figure(figsize=(12, 4))
plt.plot(train_loss_list, label='Training Loss')
plt.plot(val_loss_list, label='Validation Loss')
```

```python
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Cross Entropy Loss')
plt.legend()
plt.show()

plt.figure(figsize=(12, 4))
plt.plot(train_acc_list, label="Training Accuracy")
plt.plot(val_acc_list, label="Validation Accuracy")
plt.xlabel('Epoch')
plt.ylabel('accuracy')
plt.title('Accuracy')
plt.legend()
plt.show()

# Save model
torch.save(model.state_dict(), model_path)

# Evaluation
def evaluate_model(title, model, dataloader, device):
    model.eval()  # for batch normalization layers
    corrects = 0
    y_true = []
    y_pred = []

    with torch.no_grad():
        for inputs, targets in dataloader:
            inputs, targets = inputs.to(device), targets.to(device)

            outputs = model(inputs.long())
            _, preds = torch.max(outputs, 1)
            corrects += (preds == targets.data).sum()
            y_true.extend(targets.tolist())
            y_pred.extend(preds.tolist())

    print('{title} accuracy: {:.2f}'.format(100. * corrects / len(dataloader.
 ↪dataset), title=title))
    if title.lower() == "test":
        # Compute the confusion matrix
        cm = confusion_matrix(y_true, y_pred)
        # Compute the precision, recall, and F1 score
        report = classification_report(y_true, y_pred)
        print('Confusion Matrix:')
        print(cm)
        print('Classification Report:')
        print(report)
        fig, ax = plt.subplots(figsize=(7, 7))
```

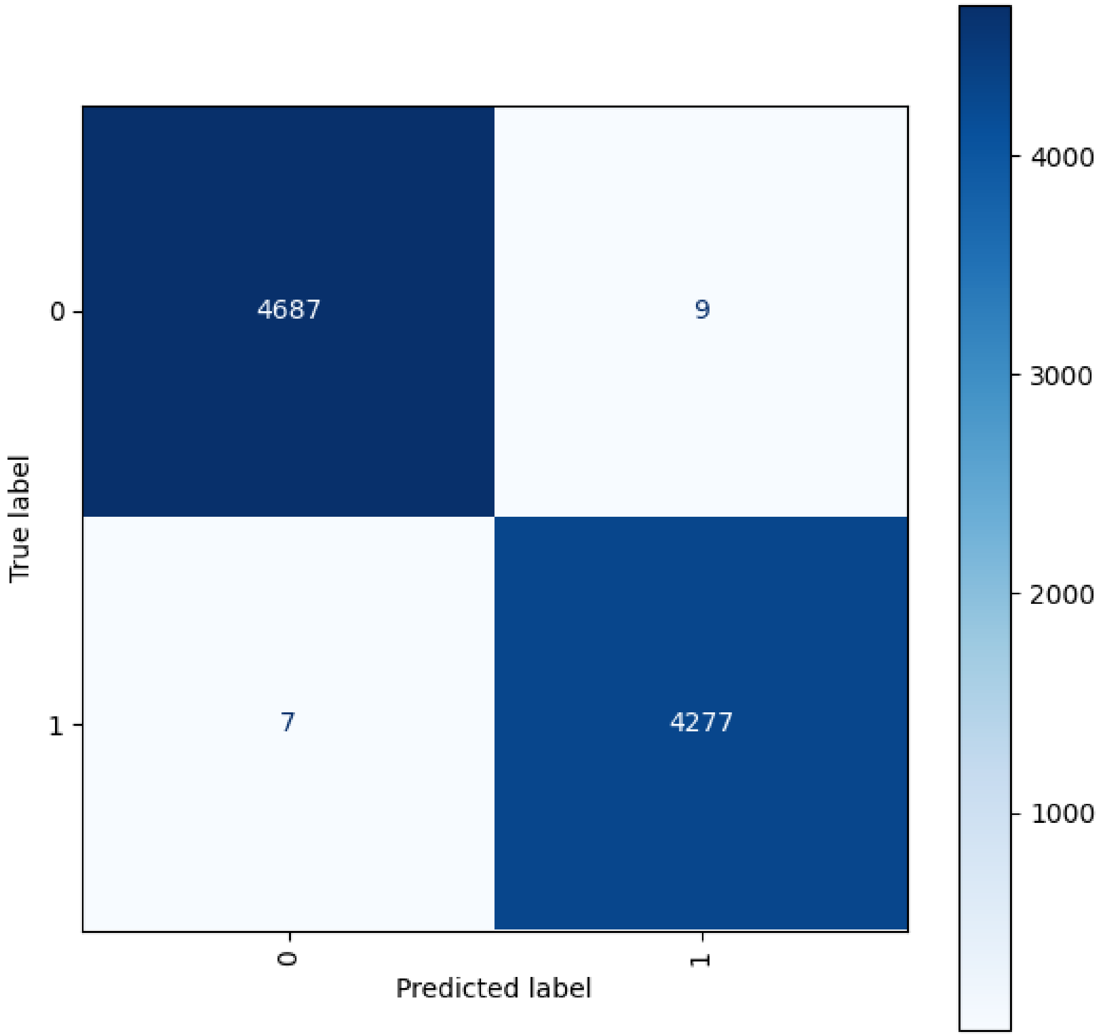
```
        ConfusionMatrixDisplay(cm).plot(ax=ax,cmap='Blues',␣
↪xticks_rotation='vertical', values_format='d')
        plt.show()


evaluate_model("Train", model, train_loader, device)
evaluate_model("Test", model, test_loader, device)
```

```
Epoch [1/10], Train Loss: 0.0342, Train Acc: 0.9896, Val Loss: 0.0129, Val Acc:
0.9973
Epoch [2/10], Train Loss: 0.0027, Train Acc: 0.9993, Val Loss: 0.0136, Val Acc:
0.9976
Epoch [3/10], Train Loss: 0.0006, Train Acc: 0.9999, Val Loss: 0.0164, Val Acc:
0.9973
Epoch [4/10], Train Loss: 0.0005, Train Acc: 0.9999, Val Loss: 0.0176, Val Acc:
0.9976
Epoch [5/10], Train Loss: 0.0004, Train Acc: 1.0000, Val Loss: 0.0202, Val Acc:
0.9976
Epoch [6/10], Train Loss: 0.0003, Train Acc: 1.0000, Val Loss: 0.0214, Val Acc:
0.9976
Epoch [7/10], Train Loss: 0.0003, Train Acc: 1.0000, Val Loss: 0.0226, Val Acc:
0.9976
Epoch [8/10], Train Loss: 0.0003, Train Acc: 1.0000, Val Loss: 0.0237, Val Acc:
0.9976
Epoch [9/10], Train Loss: 0.0004, Train Acc: 1.0000, Val Loss: 0.0238, Val Acc:
0.9976
Epoch [10/10], Train Loss: 0.0002, Train Acc: 1.0000, Val Loss: 0.0257, Val Acc:
0.9976
```






```
Train accuracy: 100.00
Test accuracy: 99.82
Confusion Matrix:
[[4687    9]
 [   7 4277]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4696
           1       1.00      1.00      1.00      4284

    accuracy                           1.00      8980
   macro avg       1.00      1.00      1.00      8980
weighted avg       1.00      1.00      1.00      8980
```


4000
3000
2000
1000
0
4687
9
True label
1
7
4277
0
1
Predicted label