AI6127 Deep Neural Networks for Natural Language Processing
Assignment 3

Ron Kow Kheng Hui
s190103@e.ntu.edu.sg
Matric Number G1903451J

1 Results

Character encoding (Tutorial)

Vocabulary Size	Test Loss	Test Accuracy
31 = 26 + <PAD> + punctuation (, . ! ?)	0.5548	82.57

(a) Wordpiece BPE

- Uses the same model as the character encoding model in the tutorial
- Vocabulary trained using the news_ag data

Vocabulary Size (including all 26 characters of the alphabet)	Test Loss	Test Accuracy
1000 + <PAD> + <UNK>	0.5662	83.76
3000 + <PAD> + <UNK>	0.6200	83.80
10,000 + <PAD> + <UNK>	0.5868	84.33

Test accuracy is slightly better with wordpiece BPE than with character encoding.

Punctuation does provide any information in classification tasks. Thus I used <UNK> to encode punctuation.
Using BPE allows us to encode both single characters and whole words in each training example.
Whole words in the BPE vocabulary such as ‘hostage’ (likely to be related to world news), ‘sony’ (science/tech), ‘euro’ (business), and ‘games’ (sports) help in classification.

(b) Sentencepiece BPE

- Uses the same model as word encoding (original model)
- Vocabulary trained using the news_ag data

Vocabulary Size (including all 26 characters of the alphabet)	Test Loss	Test Accuracy
1000 + <PAD> + <UNK>	0.8344	78.70
3000 + <PAD> + <UNK>	0.8283	78.90
10,000 + <PAD> + <UNK>	0.9589	80.63

Results for sentencepiece BPE are not as good as for wordpiece BPE and for character encoding.

This might be due to the encoding of whitespaces. For classification tasks, whitespace does not convey important information. Whitespaces encoding adds noise to the input data, leading to poorer performance.

Training Hyperparameters

	Character Encoding	Wordpiece BPE (1k, 3k, 10k)	Sentencepiece BPE (1k, 3k, 10k)
seed	1337	1337	1337
learning_rate	0.001	0.001	0.001
dropout_p	0.1	0.1	0.1
early_stopping_criteria	5	5	5
batch_size	128	256	256
num_epochs	100	100, 100, 50	25, 25, 25

Observations for wordpiece BPE:

For vocabulary size 1000, validation loss and accuracy in the first 20 epochs:

```
Epoch 0: Val Loss: 0.5789737897259847, Val Acc: 78.78906249999996
Epoch 1: Val Loss: 0.5586280111755645, Val Acc: 80.32366071428571
Epoch 2: Val Loss: 0.53950383748327, Val Acc: 80.91517857142853
Epoch 3: Val Loss: 0.5132849889142174, Val Acc: 81.81919642857146
Epoch 4: Val Loss: 0.5177072571856635, Val Acc: 81.83593749999999
Epoch 5: Val Loss: 0.5111595426286969, Val Acc: 82.578125
Epoch 6: Val Loss: 0.4890825271606445, Val Acc: 83.05803571428572
Epoch 7: Val Loss: 0.506739050575665, Val Acc: 82.82924107142856
Epoch 8: Val Loss: 0.5009687019245965, Val Acc: 82.88504464285717
Epoch 9: Val Loss: 0.49490621728556494, Val Acc: 83.50446428571426
Epoch 10: Val Loss: 0.5059846171310969, Val Acc: 83.41517857142856
Epoch 11: Val Loss: 0.5118581533432007, Val Acc: 83.74441964285718
Epoch 12: Val Loss: 0.5222072239433018, Val Acc: 83.54352678571429
Epoch 13: Val Loss: 0.5344772670950207, Val Acc: 83.49330357142857
Epoch 14: Val Loss: 0.5389900752476282, Val Acc: 83.45982142857144
Epoch 15: Val Loss: 0.5493584658418382, Val Acc: 83.36495535714285
Epoch 16: Val Loss: 0.5500861150877815, Val Acc: 83.48772321428572
Epoch 17: Val Loss: 0.5574522938047138, Val Acc: 83.4654017857143
Epoch 18: Val Loss: 0.5683076143264771, Val Acc: 83.41517857142856
Epoch 19: Val Loss: 0.5660601079463962, Val Acc: 83.44308035714283
```

Loss decreased to 0.4891 in epoch 6, then increased after that.
Accuracy was around the same level from epoch 6 onwards.

For vocabulary size 10,000, validation loss and accuracy in the first 20 epochs:

```
Epoch 0: Val Loss: 0.6466105120522635, Val Acc: 75.48549107142856
Epoch 1: Val Loss: 0.5636232520852771, Val Acc: 79.44754464285711
Epoch 2: Val Loss: 0.5251179618494848, Val Acc: 81.26116071428571
Epoch 3: Val Loss: 0.5028112862791334, Val Acc: 82.27120535714283
Epoch 4: Val Loss: 0.49718393427985075, Val Acc: 82.56138392857143
Epoch 5: Val Loss: 0.4743576875754765, Val Acc: 83.61049107142858
Epoch 6: Val Loss: 0.4851674016032899, Val Acc: 83.33147321428572
Epoch 7: Val Loss: 0.48063636337007803, Val Acc: 84.05691964285717
Epoch 8: Val Loss: 0.5138381528002877, Val Acc: 83.87276785714289
Epoch 9: Val Loss: 0.509999144077301, Val Acc: 84.01785714285715
Epoch 10: Val Loss: 0.5268235926117216, Val Acc: 84.1796875
Epoch 11: Val Loss: 0.5308556458779745, Val Acc: 84.05691964285711
Epoch 12: Val Loss: 0.5520153322390148, Val Acc: 84.17410714285714
Epoch 13: Val Loss: 0.5668307896171296, Val Acc: 84.02343749999999
Epoch 14: Val Loss: 0.5705058072294509, Val Acc: 84.11830357142858
Epoch 15: Val Loss: 0.5736244512455803, Val Acc: 84.22991071428571
Epoch 16: Val Loss: 0.5836460969277791, Val Acc: 84.0234375
Epoch 17: Val Loss: 0.5831469752958842, Val Acc: 84.21874999999999
Epoch 18: Val Loss: 0.5869853313480105, Val Acc: 84.04575892857146
Epoch 19: Val Loss: 0.5939649607454027, Val Acc: 83.87834821428572
```

Loss decreased to 0.4744 in epoch 5, then increased after that.
Accuracy was around the same level from epoch 5 onwards.

Observations for sentencepiece BPE:
For vocabulary size 1000, validation loss and accuracy in the first 20 epochs:

```
Epoch 0: Val Loss: 0.943820606810706, Val Acc: 61.87499999999998
Epoch 1: Val Loss: 0.7728732075010027, Val Acc: 70.35156250000001
Epoch 2: Val Loss: 0.7092193390641894, Val Acc: 73.43191964285712
Epoch 3: Val Loss: 0.6709229230880737, Val Acc: 75.27901785714286
Epoch 4: Val Loss: 0.6543399214744567, Val Acc: 75.76450892857144
Epoch 5: Val Loss: 0.6234502473047803, Val Acc: 77.55022321428574
Epoch 6: Val Loss: 0.6118793725967409, Val Acc: 77.92968749999997
Epoch 7: Val Loss: 0.6311793182577408, Val Acc: 78.00781249999999
Epoch 8: Val Loss: 0.6342359917504445, Val Acc: 78.35379464285718
Epoch 9: Val Loss: 0.6430682867765427, Val Acc: 79.05691964285714
Epoch 10: Val Loss: 0.6792250335216521, Val Acc: 78.54352678571429
Epoch 11: Val Loss: 0.7129932037421632, Val Acc: 78.42633928571426
Epoch 12: Val Loss: 0.7377261970724377, Val Acc: 78.17522321428572
Epoch 13: Val Loss: 0.7790042749473028, Val Acc: 78.28124999999999
Epoch 14: Val Loss: 0.7934795277459281, Val Acc: 78.23660714285711
Epoch 15: Val Loss: 0.8104753596442088, Val Acc: 78.23660714285712
Epoch 16: Val Loss: 0.8264610826969147, Val Acc: 78.28683035714289
Epoch 17: Val Loss: 0.834068935258048, Val Acc: 78.3705357142857
Epoch 18: Val Loss: 0.8454511974539076, Val Acc: 78.0357142857143
Epoch 19: Val Loss: 0.8521914209638324, Val Acc: 78.18638392857142
```

Loss decreased to 0.6119 in epoch 6, then increased after that.
Accuracy was around the same level from epoch **7** onwards.

For vocabulary size 10,000, validation loss and accuracy in the first 20 epochs:

```
Epoch 0: Val Loss: 0.9592842817306519, Val Acc: 61.18861607142857
Epoch 1: Val Loss: 0.724089473485947, Val Acc: 72.73437499999999
Epoch 2: Val Loss: 0.6267269917896815, Val Acc: 76.796875
Epoch 3: Val Loss: 0.5880178783621104, Val Acc: 78.78906249999999
Epoch 4: Val Loss: 0.5727346343653543, Val Acc: 79.98883928571429
Epoch 5: Val Loss: 0.5673658094235826, Val Acc: 80.5970982142857
Epoch 6: Val Loss: 0.5633384082998552, Val Acc: 80.77566964285715
Epoch 7: Val Loss: 0.5795492104121616, Val Acc: 80.89843749999996
Epoch 8: Val Loss: 0.5970916599035262, Val Acc: 81.07142857142856
Epoch 9: Val Loss: 0.6512503313166755, Val Acc: 80.61383928571429
Epoch 10: Val Loss: 0.6982698325599943, Val Acc: 80.66964285714286
Epoch 11: Val Loss: 0.7608878361327308, Val Acc: 80.54687499999999
Epoch 12: Val Loss: 0.8000933238438198, Val Acc: 80.32924107142858
Epoch 13: Val Loss: 0.83136140022959, Val Acc: 80.15066964285712
Epoch 14: Val Loss: 0.8788521545273915, Val Acc: 80.01116071428575
Epoch 15: Val Loss: 0.8973643149648395, Val Acc: 79.9776785714286
Epoch 16: Val Loss: 0.9113477102347785, Val Acc: 79.94419642857143
Epoch 17: Val Loss: 0.9332726342337472, Val Acc: 79.78236607142856
Epoch 18: Val Loss: 0.9479768982955387, Val Acc: 79.8158482142857
Epoch 19: Val Loss: 0.9507594781262532, Val Acc: 79.82700892857143
```

Loss decreased to 0.5633 in epoch 8, then increased after that.
Accuracy was around the same level from epoch 4 onwards.

2 Loss and Accuracy Output

Please see ipython notebooks.

3 Vocabulary

Please see ipython notebooks.

4 Implementation of BPE

4.1 Wordpiece BPE

To create the vocabulary
Example: `text1 = 'low lower low lower low'`
`text2 = 'newest widest widest'`

Step 1:
Concatenate all texts into a string and remove punctuation. Break up the string into character pieces.
Create a dictionary of word tokens `tdict`:
`{'l o w': 3, 'l o w e r': 2, 'w i d e s t': 2, 'n e w e s t': 1}`

Step 2:
Form character pairs.
```
{('l', 'o'): 5, ('o', 'w'): 5, ('w', 'e'): 3, ('e', 's'): 3,
('s', 't'): 3, ('e', 'r'): 2, ('w', 'i'): 2, ('i', 'd'): 2, ('d', 'e'): 2,
('n', 'e'): 1, ('e', 'w'): 1}
```

Select the best pair (the pair with the largest count greater than 1).
The best pair is `('l', 'o')` with count of 5. Add `lo` to the vocabulary. Merge `l` and `o` and update `tdict`:
`{'lo w': 3, 'lo w e r': 2, 'w i d e s t': 2, 'n e w e s t': 1}`

Form pairs again:
```
{('lo', 'w'): 5, ('w', 'e'): 3, ('e', 's'): 3, ('s', 't'): 3, ('e', 'r'): 2, \
('w', 'i'): 2, ('i', 'd'): 2, ('d', 'e'): 2, ('n', 'e'): 1, ('e', 'w'): 1}
```

The best pair is `('lo', 'w')` with count of 5. Add `low` to the vocabulary. Merge `lo` and `w` and update `tdict`:

`{'low': 3, 'low e r': 2, 'w i d e s t': 2, 'n e w e s t': 1}`

Step 3:
Repeat Step 2 until the target vocabulary size is achieved, or if the best pair has count of 1. Final `tdict` after 9 merges:
`{'low': 3, 'lower': 2, 'widest': 2, 'n e w est': 1}`

The wordpiece vocabulary (by default containing all the letters of the alphabet):
```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q',
'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'lo', 'low', 'es', 'est', 'lowe', 'lower',
'wi', 'wid', 'widest']
```

Counts for the vocabulary:
```
{'lo': 5, 'low': 5, 'es': 3, 'est': 3, 'lowe': 2, 'lower': 2, 'wi': 2, 'wid': 2,
'widest': 2}
```

To encode text using the vocabulary

Example: `text = 'i am below'`

Word 1: Single character `i` is encoded using its character index.

Word 2: `am` is not in the vocabulary. So `am` is encoded using character indexes for `a` and `m`.

Word 3: `below` has 4 pairs `[('b', 'e'), ('e', 'l'), ('l', 'o'), ('o', 'w')]`
The best merged pair is the one with the largest count in the vocabulary: `lo`
After merging, it becomes `b e lo w` with 3 pairs `[('b', 'e'), ('e', 'lo'), ('lo', 'w')]`
The best merged pair is `low`
After merging, it becomes `b e low` with 2 pairs `[('b', 'e'), ('e', 'low')]`
`be` and `elow` are not found in the vocabulary.
Thus `below` is encoded in 3 pieces: `b e low`.

4.2 Sentencepiece BPE

To create the vocabulary
Example: `text1 = 'low lower low lower low'`
`text2 = 'newest widest widest'`

Step 1:
Concatenate all texts into a string and remove punctuation.
Replace all whitespaces with the underscore character `'_'`
Create a dictionary of word tokens `tdict`:
`{'l o w _ l o w e r _ l o w _ l o w e r _ l o w _ n e w e s t _ w i d e s t _ w i d e s t': 1}`

Step 2:
Form character pairs.
`{('l', 'o'): 5, ('o', 'w'): 5, ('_', 'l'): 4, ('w', '_'): 3, ('w', 'e'): 3, ('e', 's'): 3, ('s', 't'): 3, ('e', 'r'): 2, ('r', '_'): 2, ('t', '_'): 2, ('_', 'w'): 2, ('w', 'i'): 2, ('i', 'd'): 2, ('d', 'e'): 2, ('_', 'n'): 1, ('n', 'e'): 1, ('e', 'w'): 1}`

Select the best pair (the pair with the largest count greater than 1).
The best pair is `('l', 'o')` with count of 5. Add `lo` to the vocabulary. Merge `l` and `o` and update `tdict`:
`{'lo w _ lo w e r _ lo w _ lo w e r _ lo w _ n e w e s t _ w i d e s t _ w i d e s t': 1}`

Form pairs again:
`{('lo', 'w'): 5, ('_', 'lo'): 4, ('w', '_'): 3, ('w', 'e'): 3, ('e', 's'): 3, ('s', 't'): 3, ('e', 'r'): 2, ('r', '_'): 2, ('t', '_'): 2, ('_', 'w'): 2, ('w', 'i'): 2, ('i', 'd'): 2, ('d', 'e'): 2, ('_', 'n'): 1, ('n', 'e'): 1, ('e', 'w'): 1}`

The best pair is `('lo', 'w')` with count of 5. Add `low` to the vocabulary. Merge `lo` and `w` and update `tdict`:

`{'low _ low e r _ low _ low e r _ low _ n e w e s t _ w i d e s t _ w i d e s t': 1}`

Step 3:
Repeat Step 2 until target vocabulary size is achieved, or if the best pair has count of 1. Final `tdict` after 12 merges:
`{'low _lower_low _lower_low _ n e w est_wid est_wid est': 1}`

The wordpiece vocabulary (by default containing all the letters of the alphabet):
`['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'lo', 'low', '_low', 'es', 'est', '_lowe', '_lower', '_lower_low', 'est_', 'est_w', 'est_wi', 'est_wid']`

Counts for the vocabulary:
`{'lo': 5, 'low': 5, '_low': 4, 'es': 3, 'est': 3, '_lowe': 2, '_lower': 2, '_lower_low': 2, 'est_': 2, 'est_w': 2, 'est_wi': 2, 'est_wid': 2}`

To encode text using the vocabulary
Example: `text = 'i am below'`

Replace all whitespaces with the underscore character `'_'`
Split into character pieces: `i _ a m _ b e l o w`

Form pairs:
`[('i', '_'), ('_', 'a'), ('a', 'm'), ('m', '_'), ('_', 'b'), ('b', 'e'), ('e', 'l'), ('l', 'o'), ('o', 'w')]`

The best merged pair is `lo`, which has the largest count in the vocabulary.

After merging, sentence becomes `i _ a m _ b e lo w` with pairs:
`[('i', '_'), ('_', 'a'), ('a', 'm'), ('m', '_'), ('_', 'b'), ('b', 'e'), ('e', 'lo'), ('lo', 'w')]`

The best merged pair is `low`. After merging, sentence becomes `i _ a m _ b e low` with pairs:
`[('i', '_'), ('_', 'a'), ('a', 'm'), ('m', '_'), ('_', 'b'), ('b', 'e'), ('e', 'low')]`

All merged pairs are not found in the vocabulary.
Thus `i am below` is encoded in 8 pieces: `i _ a m b e low`