Machine Learning 2019 Fall
HW5 Report

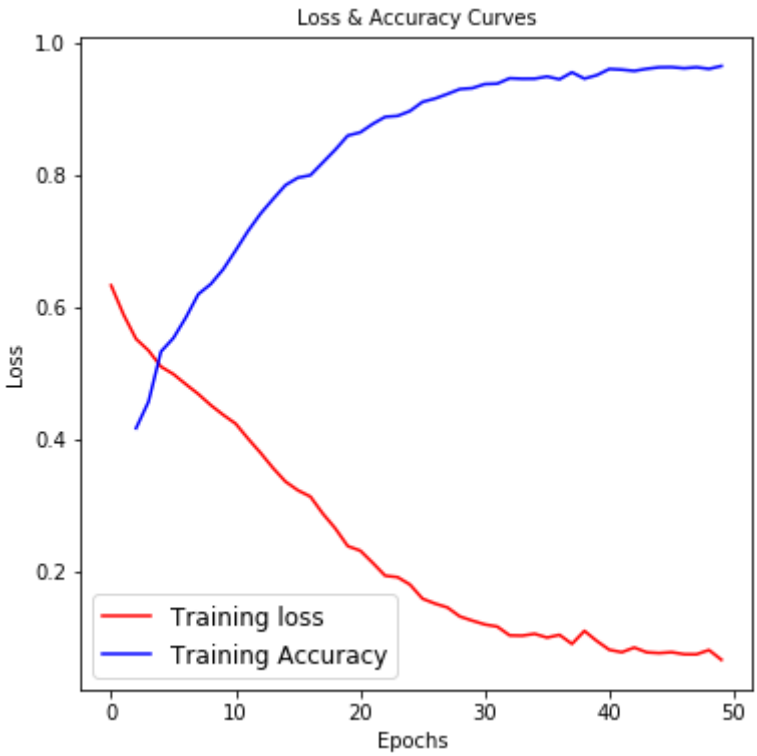
b05902105 資工四 余友竹

1. 請說明你實作之 RNN 模型架構及使用的 word embedding 方法

Architecture

- Word Embedding: 利用Gensim的word2vec將每個字轉成word vector，維度是256維。
- LSTM: 共有4層，hidden dimension 128維，dropout rate = 0.3
- Linear: 3層fully connected layer，每層都做：
    - Batch Normalization
    - ReLU
    - Dropout rate = 0.3

Training Loss/Accuracy Curves



F1 Score

	PUBLIC SCORE	PRIVATE SCORE
Word2vec + LSTM	0.76511	0.75813

2. 請實作 BOW+DNN 模型，敘述你的模型架構

Architecture

1. Preprocessing

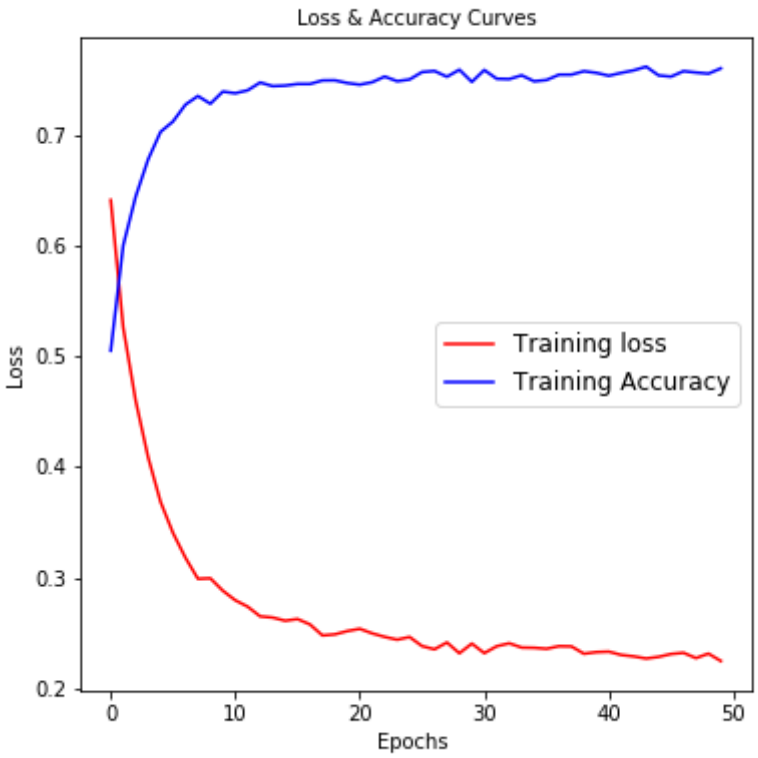
   使用One-Hot Encoding轉換，值則是改成某個句子裡，那個詞的出現次數。

2. DNN

   使用4層Fully Connected Neural Network，每層都做：

   - Batch Normalization
   - ReLU
   - Dropout rate = 0.3

Training Loss/Accuracy Curves



F1 Score

	PUBLIC SCORE	PRIVATE SCORE
BOW + DNN	0.70930	0.73255

3. 請敘述你如何 improve performance（preprocess, embedding, 架構等）

以Word2Vec為例，我認為有兩個部分可以加強Performance

1. Data Preprocessing
2. Architecture

Data Preprocessing的部分，我使用了Spacy做分詞，Spacy的功能大致上就是將空白切開，只有少數英文的慣用用法如I've，會被拆解成兩個字I, ve。

此外，為了避免程式把不同大小寫的詞認成不同字，我先將所有的詞都轉成小寫，再存起來做為Corpus。

接著，就是將詞轉換成Word Embedding Vector，這裡我使用套件Gensim的word2vec轉換。訓練參數如下

- min_count = 1
- iter = 20
- size = 256

另外，我將training data跟testing data合在一起train。我認為在Word Embedding上是合理的，因為word embedding主要就是想找出字跟字之間的前後文關係，不需要知道training data 的 label。

Architecture的部分我的改進方式則是參考了Attention is All You Need當中提到的Multi-head Attention Mechanism。

Attention-based Model的好處是考慮字跟字之間的dependency關係時，不會受到距離限制。而Multi-head Attention則是做很多次的attention，concatenate後，再通過Linear Layer。Attention Layer主要有三個input: Query, Key, Value，概念上有點像是：根據Query，找出不同Key的加權比例，並根據這些加權比例，將Value做加權平均。

我根據這個概念，跟原本的LSTM結合在一起，將LSTM的output當作Query，把原本的word2vec當成Key跟Value。

但這樣的用法比較粗糙，最後做出來的效果相似。如果要完整參考原本論文的用法，我想，應該要將原本的Input分別通過一層Linear Layer，得出學來的Key跟Value，再做接下來的運算。

4. 請比較不做斷詞 (e.g.,用空白分開) 與有做斷詞

不做斷詞得到的F1 Score:

	PUBLIC SCORE	PRIVATE SCORE
Without Segmentation	0.72558	0.73720

做了斷詞後得到的F1 Score:

	PUBLIC SCORE	PRIVATE SCORE
With Segmentation	0.76511	0.75813

我認為會有這樣的差異，可以用下面的例子舉例：

With Segmentation

> *['trump', '...', 'an', 'i', 've', 'league', 'billionaire', 'and', 'negotiator', 'extraordinaire', '!', 'jan', '...', '#', 'resister', '!', '🤔', 'i', 'll', 'take', 'trump', '!', '#', 'maga', '#', 'trump2020', 'url']*

Without Segmentation

> *['trump...', 'an', 'ive', 'league', 'billionaire', 'and', 'negotiator', 'extraordinaire!!', 'jan...', '#resister', '!!', '🤔', 'i'll', 'take', 'trump!!!', '#maga', '#trump2020', 'url']*

從這個例子很明顯可以發現，單純利用空白做斷詞，就會有一個問題是常常我們會在單字後面加上標點符號如逗號、刪節號等。這些詞如***"trump..."***跟***"trump"***在Word Embedding Model learning時，就會被當成不同的詞。

但他們應該要是一樣的，而在使用套件做斷詞後，***"trump"***跟***"..."***就被清楚的分隔開了，這樣在學習時就能正確學習到這個單字的word embedding。

5. 請比較 RNN 與 BOW 兩種不同 model 對於 "Today is hot, but I am happy."與"I am happy, but today is hot." 這兩句話的分數

RNN Model

根據我的RNN Model，第一句話的正向機率是99.956%；第二句話的正向機率是88.40%。

感覺得出來，這兩句話其實都不太帶有攻擊意味，但第二句話稍微有點抱怨的成分在。我的RNN Model也給了第二句話稍低的分數，正好反映了RNN判斷出字詞先後順序的能力。

BOW Model

根據BOW Model，兩句話的正向機率都是95.92%；反映出來BOW Model沒有判斷字詞順序關係的能力。

Math Problem

1. LSTM

根據題目條件，先列出所有會用到的公式(使用numpy做計算)：

```
import numpy as np

def f(x):
    return 1/(1 + np.exp(-x))
def g(x):
    return x
def h(x):
    return x

def z(w, b, x):
    return w.dot(x) + b
```

將初始值設定好：

```
w = np.array([0, 0, 0, 1])
wi = np.array([100, 100, 0, 0])
wf = np.array([-100, -100, 0, 0])
wo = np.array([0, 0, 100, 0])
b, bi, bf, bo = 0, -10, 110, -10
X = np.array([[0, 1, 0, 3],
              [1, 0, 1, -2],
              [1, 1, 1, 4],
              [0, 1, 1, 0],
              [0, 1, 0, 2],
              [0, 0, 1, -4],
              [1, 1, 1, 1],
              [1, 0, 1, 2]])
```

c代表初始memory值，使用一個for loop做8次運算，並使用`np.around()`將運算結果四捨五入至整數：

```
c = 0
for t, x in enumerate(X):
    print('=============== time = %d ===============' %
(t+1))
    print('memory: %d\n' % c)
    z = np.around(z(w, b, x))
    zi = np.around(z(wi, bi, x))
    zf = np.around(z(wf, bf, x))
    zo = np.around(z(wo, bo, x))
    print('z: %d, zi = %d, zf = %d, zo = %d' % (z, zi, zf,
zo))
    c = np.around(f(zi)*g(z) + c*f(zf))
    print('new memory = %d' % c)
    y = np.around(f(zo)* h(c))
    print('output: %d\n' % y)
```

以下是執行結果，其中 `new memory` 表示 c'，`output` 表示 y：

```
=============== time = 1 ===============
```

```
memory: 0

z: 3, zi = 90, zf = 10, zo = -10
new memory = 3
output: 0

=============== time = 2 ===============
memory: 3

z: -2, zi = 90, zf = 10, zo = 90
new memory = 1
output: 1

=============== time = 3 ===============
memory: 1

z: 4, zi = 190, zf = -90, zo = 90
new memory = 4
output: 4

=============== time = 4 ===============
memory: 4

z: 0, zi = 90, zf = 10, zo = 90
new memory = 4
output: 4

=============== time = 5 ===============
memory: 4

z: 2, zi = 90, zf = 10, zo = -10
new memory = 6
output: 0

=============== time = 6 ===============
memory: 6

z: -4, zi = -10, zf = 110, zo = 90
new memory = 6
output: 6

=============== time = 7 ===============
memory: 6

z: 1, zi = 190, zf = -90, zo = 90
new memory = 1
output: 1

=============== time = 8 ===============
memory: 1

z: 2, zi = 90, zf = 10, zo = 90
```

```
new memory = 3
output: 3
```

2. Word Embedding

為了避免標示混淆，以下令$W = [w_{ij}] \in \mathbb{R}^{V\times N}$, $W' = [w'_{ij}] \in \mathbb{R}^{N\times V}$ 。

$$\mathbf{h} = W^T\mathbf{x} = \begin{bmatrix} \sum_{p=1}^{V} w_{p1}x_i & \sum_{p=1}^{V} w_{p2}x_p & \cdots & \sum_{p=1}^{V} w_{pN}x_p \end{bmatrix}^T$$

$$\mathbf{u} = W'^T\mathbf{h} = \begin{bmatrix} \sum_{q=1}^{N} w'_{q1}(\sum_{p=1}^{V} w_{pq}x_p) & \cdots & \sum_{q=1}^{N} w'_{qV}(\sum_{p=1}^{V} w_{pq}x_p) \end{bmatrix}^T$$

$$\begin{aligned} L &= -\log(\prod_{c\in C}\frac{\exp(u_c)}{\sum_{r\in V}\exp(u_r)}) \\ &= -\sum_{c\in C}(\log(\exp(u_c)) - \log(\sum_{r\in V}\exp(u_r))) \\ &= -\sum_{c\in C}(u_c - \log(\sum_{i\in V}\exp(u_r))) \\ &= -\sum_{c\in C}\Big(\sum_{q=1}^{N} w'_{qc}(\sum_{p=1}^{V} w_{pq}x_p) - \log\big(\sum_{r\in V}\exp(\sum_{q=1}^{N} w'_{qr}(\sum_{p=1}^{V} w_{pq}x_p))\big)\Big) \end{aligned}$$

將L對w_{ij}^T取偏微分，等同於對w_{ji}取偏微分

$$\begin{aligned} \frac{\partial L}{\partial W_{ji}} &= -\sum_{c\in C}\Big(w'_{ic}x_j - \frac{\sum_{r\in V} w'_{ir}x_j\cdot\exp\big(\sum_{q=1}^{N} w'_{qr}(\sum_{p=1}^{V} w_{pq}x_p)\big)}{\sum_{r\in V}\exp\big(\sum_{q=1}^{N} w'_{qr}(\sum_{p=1}^{V} w_{pq}x_p)\big)}\Big) \\ &= -\sum_{c\in C}\Big(w'_{ic}x_j - \frac{\sum_{r\in V} w'_{ir}x_j\cdot\exp(u_r)}{\sum_{r\in V}\exp(u_r)}\Big) \end{aligned}$$

將L對$w_{ij}'^T$ 取偏微分，等同於對w'_{ji}取偏微分

$$\begin{aligned} \frac{\partial L}{\partial W'_{ji}} &= [i\in C](-\sum_{p=1}^{V} w_{pj}x_p) + \sum_{c\in C}\frac{(\sum_{p=1}^{V} w_{pj}x_p)\exp\big(\sum_{q=1}^{N} w'_{qi}(\sum_{p=1}^{V} w_{pq}x_p)\big)}{\sum_{r\in V}\exp\big(\sum_{q=1}^{N} w'_{qr}(\sum_{p=1}^{V} w_{pq}x_p)\big)} \\ &= [i\in C](-\sum_{p=1}^{V} w_{pj}x_p) + \sum_{c\in C}\frac{(\sum_{p=1}^{V} w_{pj}x_p)\exp(u_i)}{\sum_{r\in V}\exp(u_r)} \end{aligned}$$