SDML HW1 Report

成員：
B06902019 洪佳生
B06902103 尹聖翔
B06902125 黃柏瑋

0. Introduction

Problem Description

根據論文的資訊，判斷該論文是屬於Theoretical, Engineering, Empirical, or Others。論文可同時被分類至不同類型。

Data Description

有7000筆labeled的資料，資料的feature有標題、摘要、作者、類型、日期，另有20000筆unlabeled的資料作為public testset。我們還有額外的論文citation network，有770k個節點，還有1372k個邊。

1. Data Preprocessing

提出Title、Abstract當作training data，並以Task2作為label data。

Truncation Method

1. 一開始我們採用的是最簡單的方法：
   當Abstract字數超過sequence length，則從頭取sequence length個字。
2. 在[1]提及的三種truncating skills: 從開頭取值、從結尾取值、從開頭和結尾取值中，經過一些實驗後我們發現第三種方法的valid_acc可以達到平均80.3%的結果，其他兩種方法大約只有78~79%的成績，於是這裡選用以下方法作為truncating strategy：
   當Abstract字數超過sequence length，就從開頭取1/4 * sequence length個字接上結尾3/4 * sequence length個字作為training data；如果Abstract不足sequence length，便將Title補在前面。
3. 我們擔心**方法2**或許會丟失一些和分類有關的重要資訊，因此嘗試不會刪除整段文字的strategy：
   當Abstract字數超過sequence length，則每隔5個字便刪除1個，希望不連續的刪除方式可以避免過多重要資訊被刪除。

2. Our Approach

2-1 Bert-base

使用套件：**keras-bert**[2]

Simple fine-tuning

將Bert-base最後一個encoder的[CLS]結果取出，接上一層dropout(0.5)、full-connected layer(units=128, relu)與fully-connected layer(4, sigmoid)直接fine-tune。
這裡的Bert model是google提供的pretrained model，沒有額外的within-task pretraining。

Sequence length	Truncation	Loss	Batch	LR	Val_acc	Epoch
256	2	BCE	10	2e-5	0.7994	2

接著我們把模型再簡化，只將[CLS]的結果接上fully-connected layer(4, sigmoid)直接fine-tune。

Sequence length	Truncation	Loss	Batch	LR	Val_acc	Epoch
256	2	BCE	10	2e-5	0.8036	3
512	1	BCE	8	1e-5	0.8042	5

Simple fine-tuning with Google pretraining

利用Google提供的程式碼pretrain Bert-base[3]，首先先訓練seq_len=128的模型(steps=10000)，接著以其為基礎訓練seq_len=512的模型(steps=13000)作為initial model。

- 12th encoder's [CLS] + fc(128, relu) + fc(4, sigmoid)

Sequence length	Truncation	Loss	Batch	LR	Val_acc	Epoch
256	2	BCE	10	2e-5	0.8010	2

- 12th encoder's [CLS] + fc(4, sigmoid)

Sequence length	Truncation	Loss	Batch	LR	Val_acc	Epoch
256	2	BCE	10	2e-5	0.8022	2
512	1	BCE	8	1e-5	0.8053	5

Simple fine-tuning with keras-bert pretraining

利用Keras-bert提供的模板pretrain Bert-base，直到大約loss = 0.17 (MLM loss=0.07、NSP loss = 0.10)，並將其作為initial model。

- 12th encoder's [CLS] + fc(128, relu) + fc(4, sigmoid)

Sequence length	Truncation	Loss	Batch	LR	Val_acc	Epoch
256	2	BCE	10	2e-5	0.8025	2

- 12th encoder's [CLS] + fc(4, sigmoid)

Sequence length	Truncation	Loss	Batch	LR	Val_acc	Epoch
256	2	BCE	10	2e-5	0.8073	2

以上可知，用keras-bert pretrain的效果比較好，於是2-1的後半部將以keras-bert pretrained model作為initial model，繼續其他任務。

Fine-tuning with frozen layers

接著，我們著手改進fine-tune技巧。一開始我們先freeze所有的Bert layers，用較大的learning rate訓練後方的fully-connected layers。接著再解放Bert layers，用較小的learning rate fine-tune Bert-base模型。
此外，由於我們使用的Loss function是BCE，而非F1 score，所以結果的threshold明顯會影響到我們的F1 score：當我們將所有類型的threshold都調到0.4，發現結果會比不做調整時(threshold=0.5)更好。可是由於每個類型的數據有多有寡，可能導致模型在學習不同類型時效果不太一致，因此我們最後決定對每個類型取出不同的threshold，使得val_F1越高越好。

- 12th encoder's [CLS] + fc(128, relu) + fc(4, sigmoid)

Sequence length	Truncation	Loss	Batch	LR	Val_acc	Epoch
256	2	BCE	25	1e-4	0.8000	47
256	2	BCE	10	2e-6	0.8125	4

Threshold	Val_F1	Test_F1
-	-	-
[0.45, 0.54, 0.3, 0.39]	0.7121	0.7048

- 12th encoder's [CLS] + fc(4, sigmoid)

Sequence length	Truncation	Loss	Batch	LR	Val_acc	Epoch
256	2	BCE	25	1e-4	0.7889	63
256	2	BCE	10	2e-6	0.8140	5

Threshold	Val_F1	Test_F1
-	-	-
[0.39, 0.24, 0.24, 0.45]	0.7117	0.7087

由此可知，先訓練上層再fine-tune的效果明顯提升，或許之後也能像前幾名的組別嘗試在fine-tune時分段frozen，結果應該會更加漂亮。然而，我們取threshold的方式僅是簡易的暴搜法，多做幾次實驗之後發現這樣其實很容易overfit(導致Val_F1有時不準)，也許能有更好的threshold classifier值得我們試試。

BiLSTM with Bert's embeddings

不同於fine-tune方法，在取出每個字的embedding之後，直接丟入一個128層的bidirectional LSTM(dropout=0.5)中訓練，再接出一個fully connected layer(units=4, sigmoid)求出結果。

Sequence length	Truncation	Loss	Batch	LR	Val_acc	Epoch
256	2	BCE	25	2e-5	0.7977	35

這時，當我們想要像上面一樣在訓練好BiLSTM之後，解放所有的Bert Layers和BiLSTM一起fine-tune，卻發現無論learning rate有多小都train不起來(loss = nan)。而當我們將sequence length縮短至128時，會發現fine-tune是有結果的。

Sequence length	Truncation	Loss	Batch	LR	Val_acc	Epoch
128	2	BCE	25	1e-4	0.7971	21
128	2	BCE	25	5e-6	0.8014	2

目前雖然不太明白在sequence length=256時失敗的原因，但說不定只解放後面幾層與biLSTM一起fine-tune能解決這樣的問題，算是future mission中值得討論的一項研究。

2-2 Bert-large

Sliding Window

因為GPU的記憶體容量限制，我們發現使用bert-large作為fine-tune的基礎模型時，sequence length和training batch size都不能設太大。因此我們使用sliding windows的技巧來訓練模型。舉例來說：若文字為a,b,c且sequence length為2，則分別將a,b和b,c以同樣的label輸入bert model訓練。
除了訓練以外，在預測時也要針對超過sequence length的輸入進行處理。我們同樣採取sliding window，並且對出來的複數預測應用以下兩種方法：

- 將預測出的機率取平均值

Sequence length	Loss	Batch	LR	Val_acc	Epoch	Threshold	Test F1
128	BCE	4	1e-5	0.6428	5	0.5	-
64	BCE	8	1e-5	0.5791	5	0.5	0.3916

- 將embedding的向量取平均值

Sequence length	Loss	Batch	LR	Val_acc	Epoch	Threshold	Test F1
128	BCE	4	1e-5	0.6002	5	0.5	-
64	BCE	8	1e-5	0.5801	5	0.5	0.3833

Truncation Method

由以上方法可以看出，兩種方法的表現都不如預期，因此我們改回使用前述的Truncation method，成果如下：

Sequence length	Truncation	Loss	Batch	LR	Val_acc	Epoch	Threshold	T F
450	1	BCE	1	2e-6	0.7915	5	0.4	0.
450	2	BCE	1	2e-6	0.6867	5	0.4	-

比較這三種方法，我們可以看到truncation method 1的表現最好，因此我們對Abstract在encode後的長度進行了調查：在trainset中長度超過450的僅有4組，由此可見大部分的資料並不會被捨棄的資料所影響。

2-3 Graph Embedding

在實作的方面上已經有數種想法可以實現：HOPE, SDNE, node2vec, LINE, DeepWalk等等。其中可以分成NN-based或者是傳統的隨機法

word2vec&skip-gram:

word2vec是一種將word轉換成embedding的方法，是graph embedding的基礎概念，類似於上述的bert，透過skip-gram network的方式取得代表word的vector

node2vec：

一開始我們首先嘗試建立在deepwalk algorithm基礎上的node2vec，發現在large data base的graph上random walk由於隨機性的關係，會高度重複遊走high level(相鄰鄰居與自身相連的數量)的vertex，造成data的提取產生了bias，並且在deepwalk的方法中並不存在有彈性的的loss function去解釋整個結構性，因此在龐大的graph下較難得到好的成效。[4]

Graph nodes	walk number	walk length	Val_acc	Epoch
200thousands	400	1000	0.5308	1
50thousands	400	1000	0.5431	1

LINE

LINE以與deepwalk相似的方式進行鄰居相似性的推算，deepwalk是基於DFS，LINE則是基於BFS，引入了first & second-order的觀念，分別是local與global的結構性審視，並加入negative sampling與edge sampling的方式進行優化

SDNE

SDNE的graph embedding是以Neural Network的方式進行embedding的學習，其中它混合的傳統的BFS與DFS方法並在其中取得某種平衡，在first-order & second-order的graph上表現的不錯，適用於各式各樣的需求。[5]

Graph nodes	Loss	Batch	LR	Val_acc	Epoch
200thousands	BCE	4	1e-7	0.7189	16
50thousands	BCE	4	1e-7	0.6773	16

Dilemma

基於本次作業graph的node＆edge數量龐大，我們沒有足夠的meomory去將整個graph讀入其中，因此我們只讀入了部分的graph，主要以和train data＆test data相連的vertex為主，但由於subgraph的node數量依然過多，因此我們給定它一定的機率捨去這個node。並且發現在過於高等的套件比如keras中，吃的額外記憶體比tensorflow的語言要多得多，在keras的模型中我們最多只能存到六萬個節點，但在tensorflow中我們卻能存到20萬個節點。

Conclusion

我們認為在記憶體有限的情況下graph embedding的結果應該與其他的embedding方法concatenate起來會比單獨進行DNN的預測要來得好。

2-4 Others

Adjustment toward Results

根據label的性質，我們可以得知'Others'和其他分類是不會共存的。針對這一點我們嘗試加入此判斷：**若'Others'和其餘標籤同時出現，則低機率將其他labels結果轉為0，高機率將Others結果轉為0**。並且根據我們對結果的觀察，Theoretical、Engineering的結果較多、Others的結果極少。因此**若有論文不屬於四種分類，則高機率將Theoretical和Engineering的結果轉為1，低機率將Empirical轉為1**。
以上兩種方法對結果的影響並不顯著，對第二種方法的推測原因是原本不屬於任何分類的論文就很少，而第一種方法則是因為F1 score較鼓勵True Positive的特性導致效果抵銷。

3. Future Mission

- 更好的threshold classifier
- 使用freezing技巧，分層fine-tune
- train出更好的pretrain model
- 利用pytorch或tensorflow實作以避免keras使用記憶體過大的問題
- 嘗試emsemble

Reference

1. https://arxiv.org/abs/1905.05583 (https://arxiv.org/abs/1905.05583) ↩
2. https://github.com/CyberZHG/keras-bert (https://github.com/CyberZHG/keras-bert) ↩
3. https://github.com/google-research/bert (https://github.com/google-research/bert) ↩
4. https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf (https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf) ↩
5. https://www.kdd.org/kdd2016/papers/files/rfp0191-wangAemb.pdf (https://www.kdd.org/kdd2016/papers/files/rfp0191-wangAemb.pdf) ↩