Final Project

透過神經網路模型進行文本分類

108034557 陳宣任

Agenda

- 背景介紹
- 問題定義
- 模型建構
- 模型改善
- 結論

Agenda

- 背景介紹
- 問題定義
- 模型建構
- 模型改善
- 結論

背景介紹

- 隨著網路的普及，許多人會透過google、社群論壇等平台進行提問。
- 但也造成許多人利用提問功能濫發電子訊息，作為廣告、評論之用，造成論壇的使用者的負面觀感。
- 透過大量人力刪除這些訊息，既沒有效率，也無法解決問題。

背景介紹

- 目的：
  建立一個神經網路模型，幫助我們將分類為有用與垃圾訊息，能夠在使用者提問時偵測是否為垃圾訊息。可以提高處理的效率及平台使用者的滿意度。

以下是可能為垃圾訊息的特徵：

- 用誇張的語氣過度強調他人的觀點
- 惡意歧視他人
- 攻擊 / 侮辱內容
- 廣告 / 宣傳

Agenda

背景介紹

問題定義

模型建構

模型改善

結論

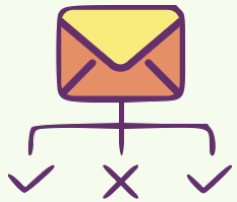
問題定義——5W1H

What	解決垃圾訊息過多的問題
When	使用者在平台發布訊息時
Who	平台的管理者
Where	平台網站
Why	增加使用者的滿意度，從而提高對平台的黏著度
How	建構分類模型，幫助區分是否為垃圾訊息

問題定義——流程圖

平台使用者提問

神經網路模型
1.CNN
2.LSTM
3.CNN+LSTM



問題分類

正確內容

避免刪除

垃圾訊息

刪除、通知

Agenda

背景介紹

問題定義

模型建構

模型改善

結論

模型架構

資料輸入

資料前處理

模型建購

結果輸出

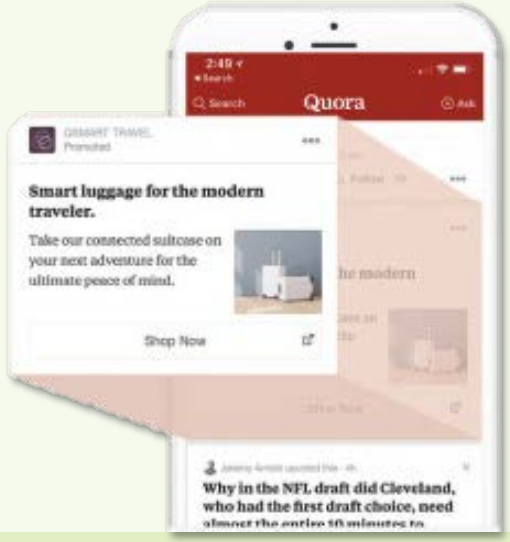
資料輸入

■使用Quora線上問答平台的提問資料

■共約137萬筆資料

✓有效提問約為128萬筆

✓垃圾訊息約9萬筆



資料前處理

■將正常問答設為第0類，垃圾訊息為第1類

■平衡訓練資料

✓Resampling

➢有效訊息中抽取10萬筆數據(隨機抽取，不重複)

➢垃圾訊息中抽取10萬筆數據(隨機抽取，有可能重複)

```
#resampling
df_0=df_train.loc[df_train["target"]== 0].sample(n=100000,random_state=42)
df_1=df_train.loc[df_train["target"]== 1].sample(n=100000,replace=True,random_state=42)
df_train=pd.concat([df_0,df_1])
```

資料前處理

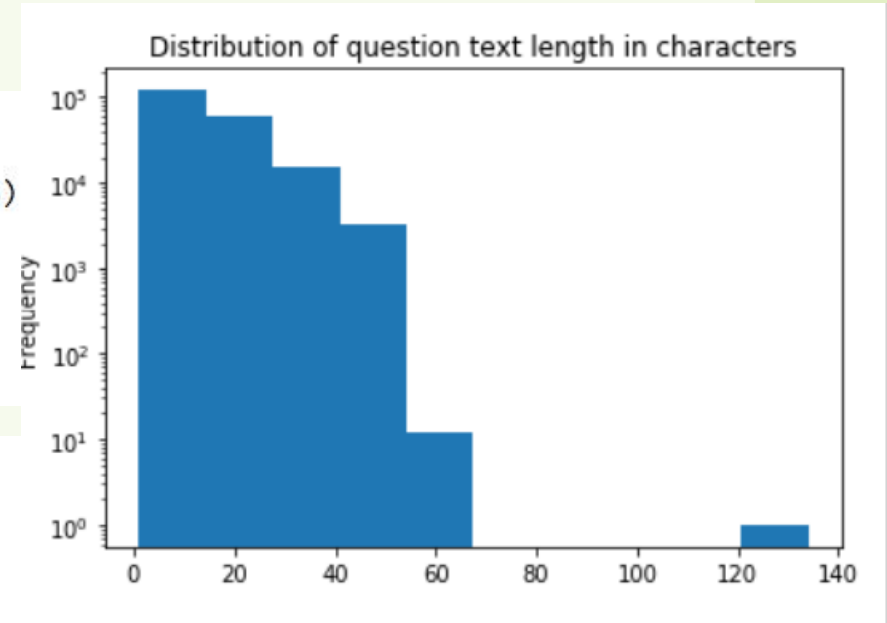
■從訓練集抽取10%作為驗證集，訓練集共180000張、驗證集20000張

■將提問文字轉換成序列向量(設定為9萬維)

■將序列擷取為相同長度(設定為70)

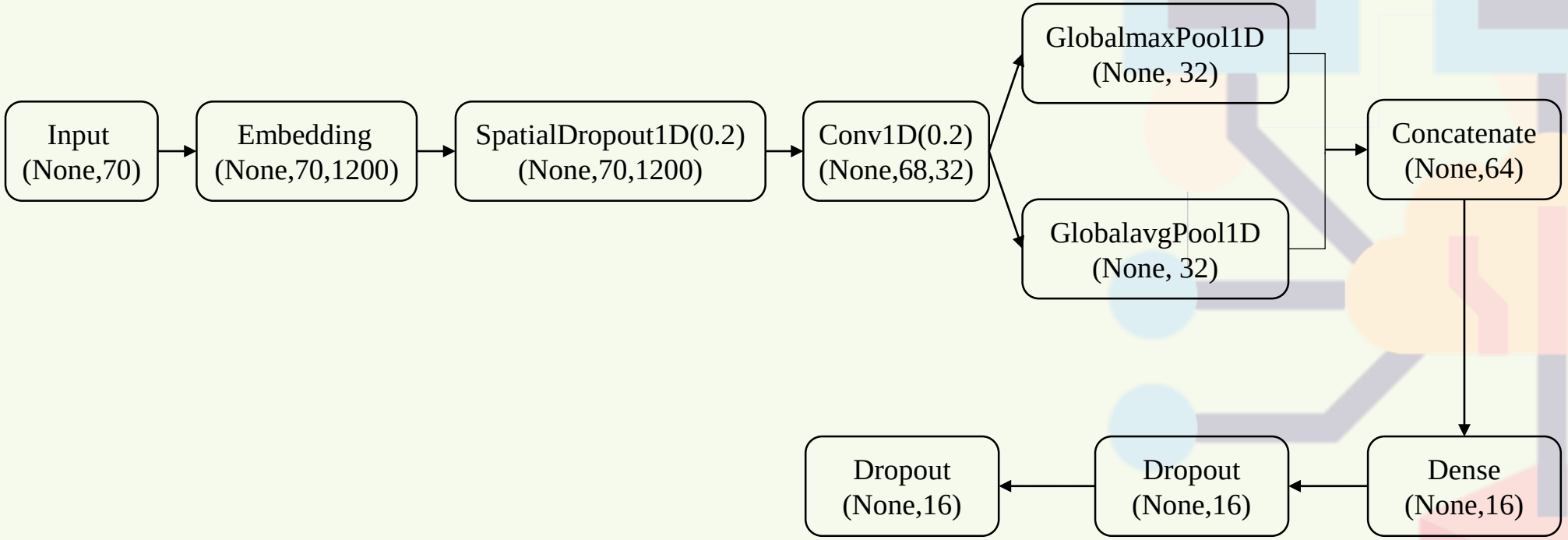
```
max_features = 100000
tk = Tokenizer(lower = True, filters='', num_words=max_features+1)
full_text = list(df_train['question_text'].values) + list(df_test['question_text'].values)
#使用一系列文档来生成token词典，texts为list类，每个元素为一个文档
tk.fit_on_texts(full_text)

train_tokenized = tk.texts_to_sequences(df_train['question_text'].fillna('missing'))
test_tokenized = tk.texts_to_sequences(df_test['question_text'].fillna('missing'))
```

```
#將序列整理成相同長度(剪斷或補齊)，最大長度70
x_train = pad_sequences(train_tokenized, maxlen = max_len)
x_test = pad_sequences(test_tokenized, maxlen = max_len)
```



模型建構

■CNN:擷取單一句提問的重要特徵



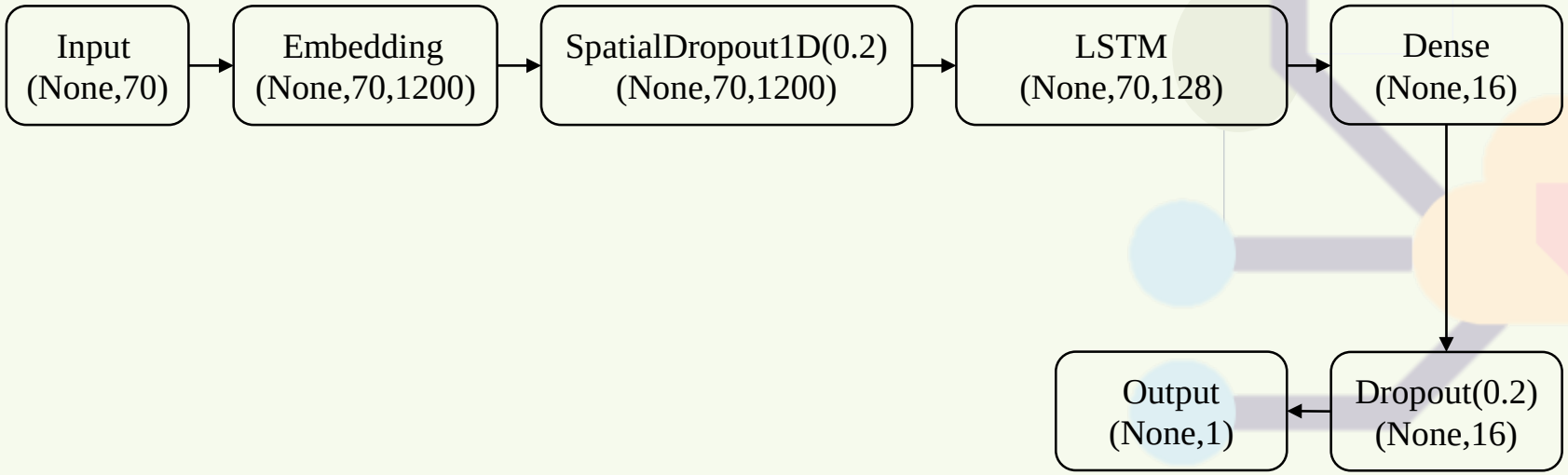
模型建構

■CNN:擷取單一句提問的重要特徵

```
inp = Input(shape=(max_len,))
x = Embedding(max_features, embed_size * 4, weights=[embedding_matrix])(inp)
x = SpatialDropout1D(0.2)(x)
x_conv1 = Conv1D(conv_size, kernel_size=3, padding='valid', kernel_initializer='he_uniform')
avg_pool = GlobalAveragePooling1D()(x_conv1)
max_pool = GlobalMaxPooling1D()(x_conv1)
conc = concatenate([avg_pool, max_pool])
x = Dense(16, activation="relu")(conc)
x = Dropout(0.2)(x)
x = Dense(1, activation="sigmoid")(x)
model = Model(inputs=inp, outputs=x)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

模型建構

■LSTM:考慮提問之間的序列關係



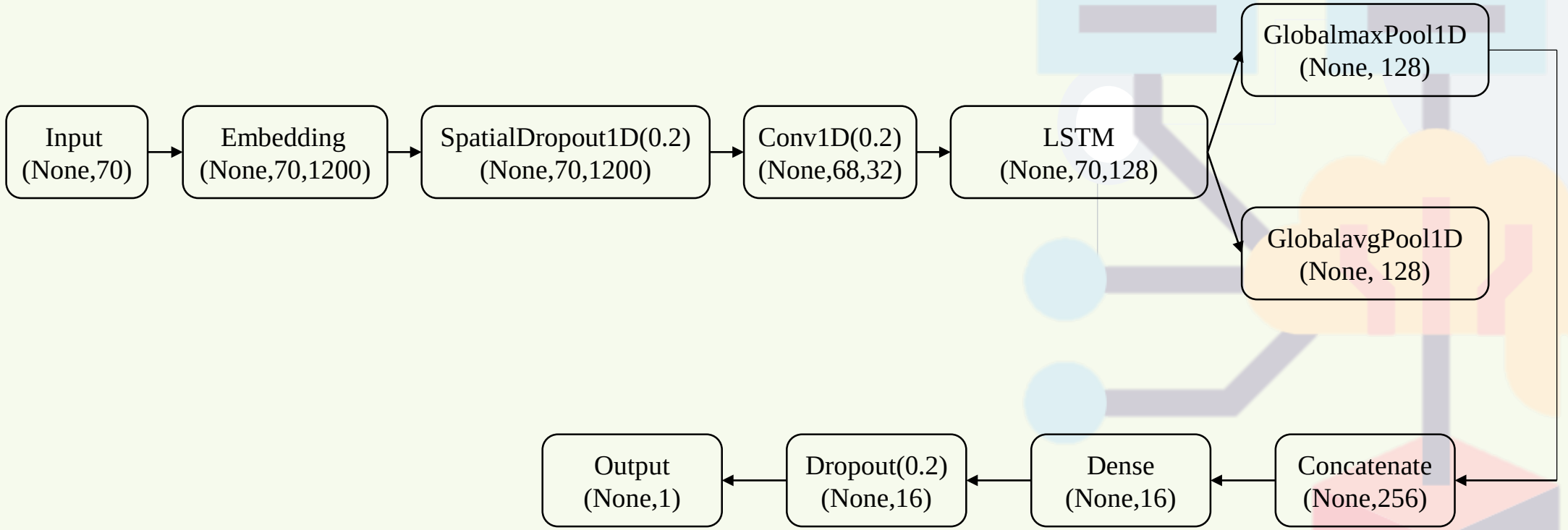
模型建構

■LSTM:考慮提問之間的序列關係

```
inp = Input(shape=(max_len,))
x = Embedding(max_features+1, embed_size * 4, weights=[embedding_matrix])(inp)
x = SpatialDropout1D(0.2)(x)
x = LSTM(128,return_sequences=True)(x)
x = Dense(16, activation="relu")(x)
x = Dropout(0.2)(x)
x = Dense(1, activation="sigmoid")(x)
model = Model(inputs=inp, outputs=x)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

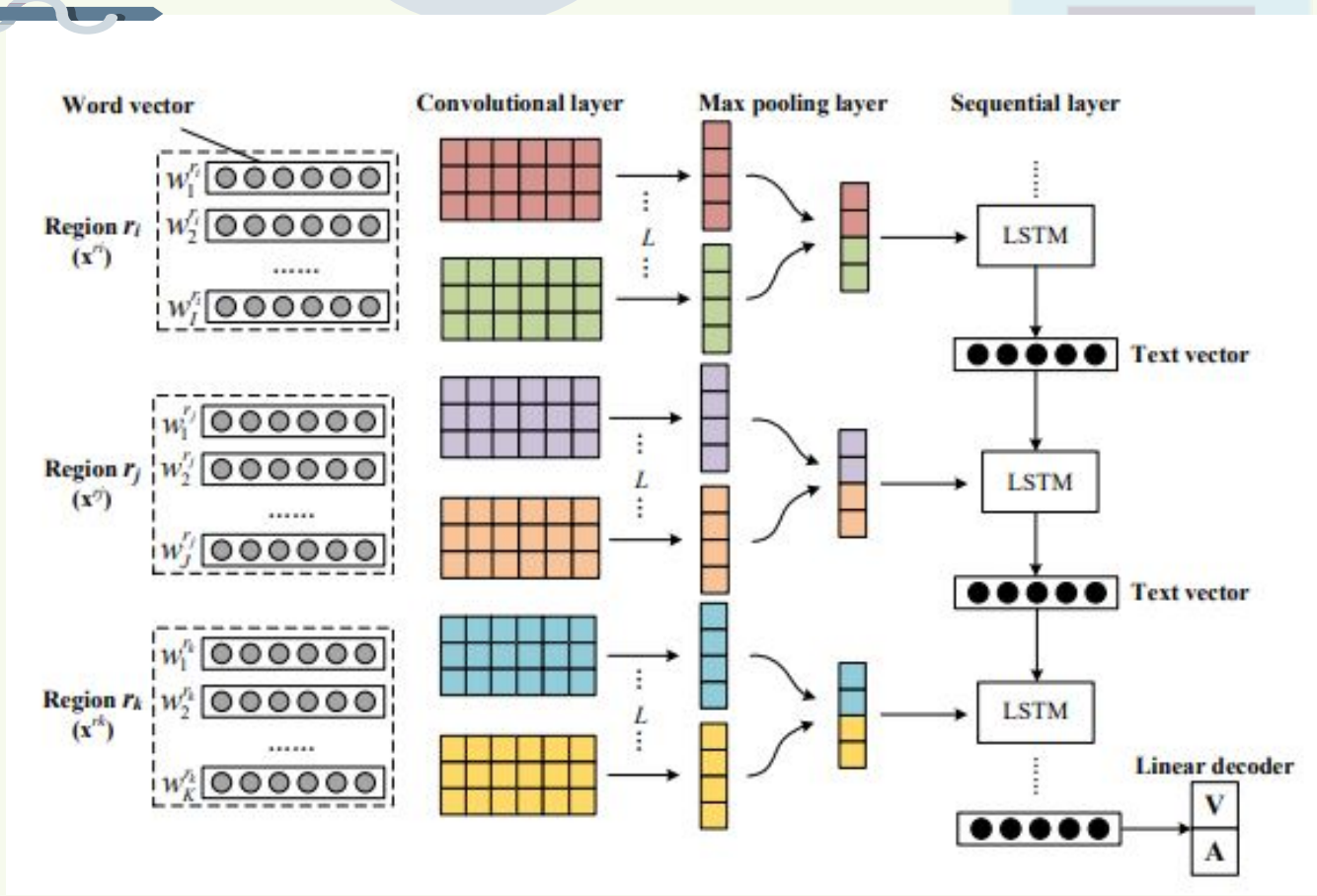
模型建構

■CNN+LSTM



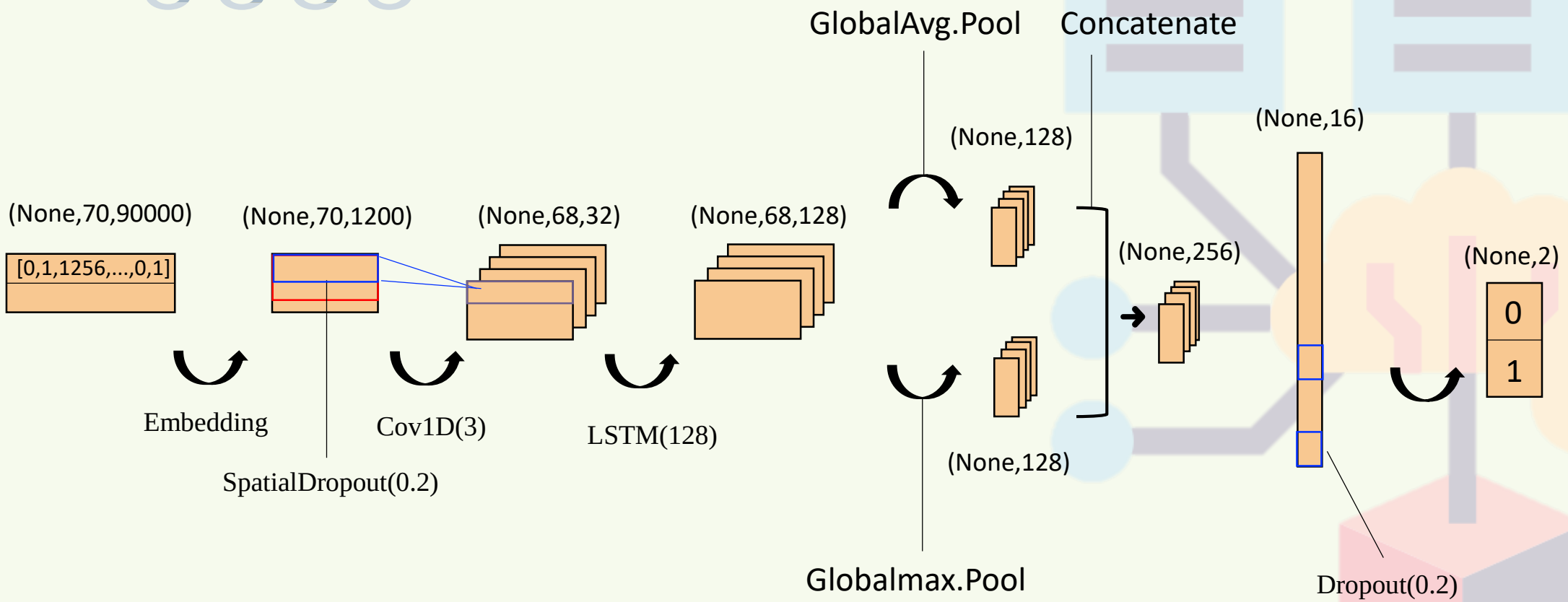
模型建構

■CNN+LSTM



模型建構

■CNN+LSTM



模型建構

■CNN+LSTM

```
#連模
early_stop=EarlyStopping(monitor="val_loss",mode="min",patience=3)
conv_size=32
inp = Input(shape=(max_len,))
x = Embedding(max_features+1, embed_size * 4, weights=[embedding_matrix])(inp)
x = SpatialDropout1D(0.2)(x)
#x = Reshape((maxlen, embed_size * 4, 1))(x)
x_conv1 = Conv1D(conv_size, kernel_size=1, padding='valid', kernel_initializer='he_uniform')(x)
x_conv2 = Conv1D(conv_size, kernel_size=2, padding='valid', kernel_initializer='he_uniform')(x_conv1)
x_conv3 = Conv1D(conv_size, kernel_size=3, padding='valid', kernel_initializer='he_uniform')(x_conv2)
x_conv4 = Conv1D(conv_size, kernel_size=5, padding='valid', kernel_initializer='he_uniform')(x_conv3)
x =  Bidirectional(LSTM(128, return_sequences=True))(x_conv4)
avg_pool = GlobalAveragePooling1D()(x)
max_pool = GlobalMaxPooling1D()(x)
conc = concatenate([avg_pool, max_pool])
x = Dense(16, activation="relu")(conc)
x = Dropout(0.2)(x)
x = Dense(1, activation="sigmoid")(x)
model = Model(inputs=inp, outputs=x)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

模型建構

■提前終止(Early Stopping)

- ✓ 為了避免在訓練過程中出現過擬合的狀況，透過監控驗證集的準確率，來判斷是否需要提前終止訓練。

```
early_stop = EarlyStopping(monitor = "val_loss", mode = "min", patience = 3)
```

模型建構

- 訓練集共20萬張
- 測試集共10萬張
- 激活函數:relu
- 輸出層激活函數:sigmoid

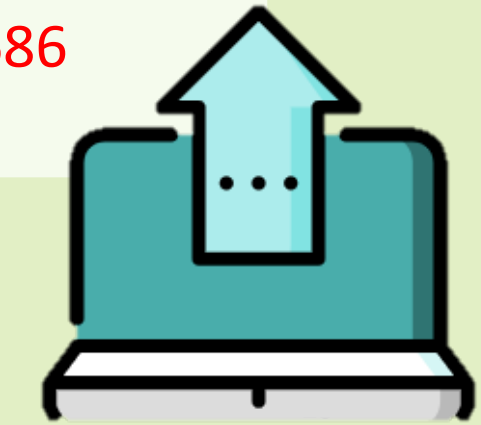
- 損失函數:二元交叉熵
- 優化器:adam
- Batrch_size:5000
- Epochs:10

結果輸出

Precision:預測為正確提問當中，實際也是正確提問的比率。

Recall:實際為正確提問當中，預測是正確提問的比率。

CNN	LSTM	CNN+LSTM
Accuracy: 0.9121	Accuracy: 0.8965	Accuracy: 0.9140
Precision: 0.6894	Precision: 0.6684	Precision: 0.69
Recall: 0.8651	Recall: 0.8648	Recall: 0.8686



Agenda

背景介紹

問題定義

模型建構

模型改善

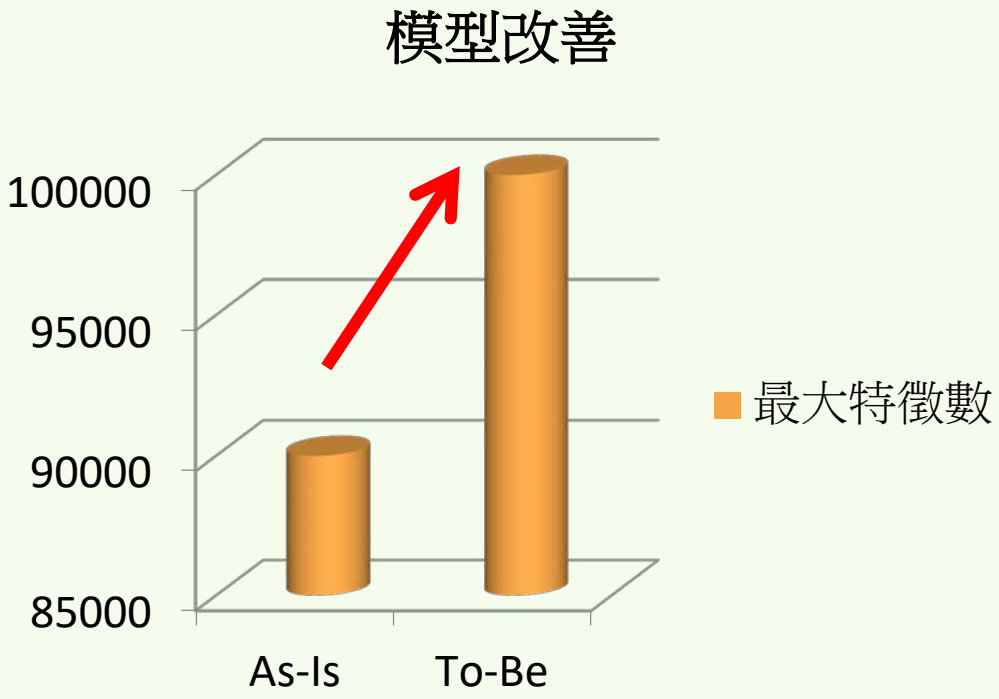
結論

模型改善

Model	Accuracy	Precision	Recall
原Model	0.9140	0.69	0.8686
最大特徵數改為100000	0.9132	0.701	0.8725

■增加最大特徵數

✓從資料集中選出出現頻率最高的字詞數量



AS-IS Model

Accuracy:0.9140
Precision:0.69
Recall:0.8686

To-Be Model

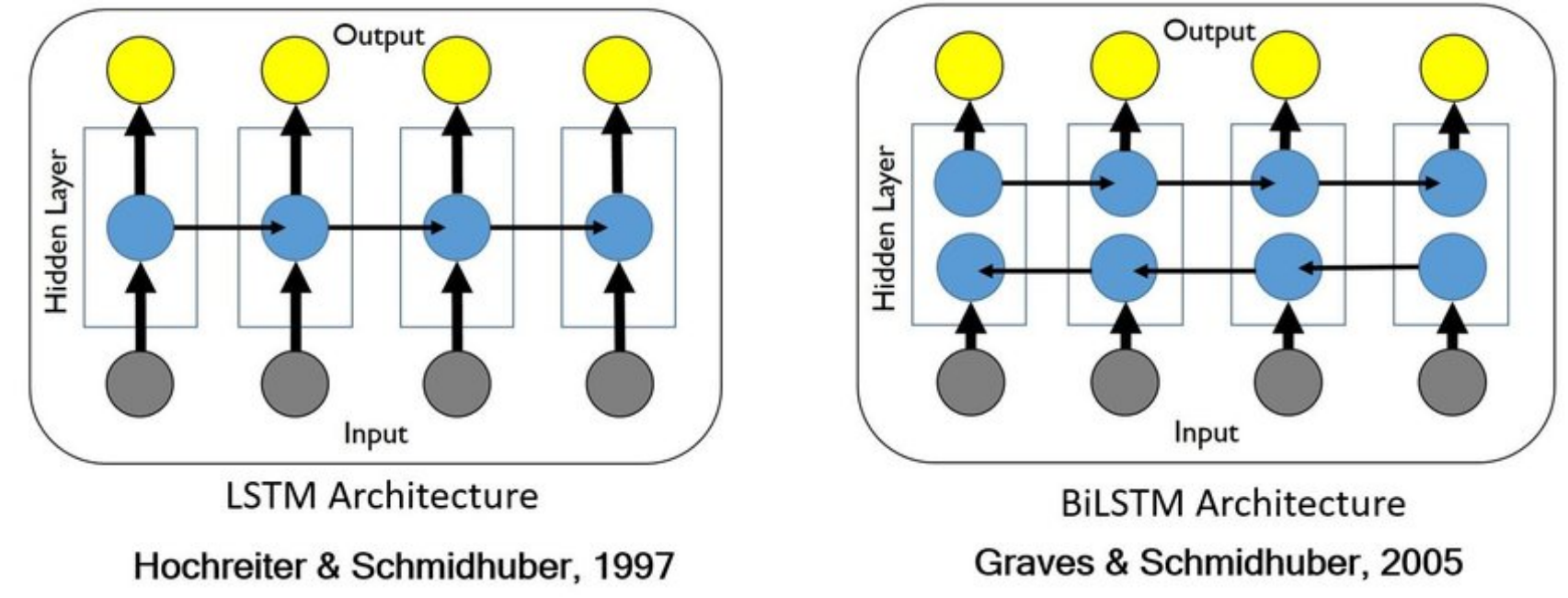
Accuracy:0.9132
Precision:0.701
Recall:0.8725

模型改善

Model	Accuracy	Precision	Recall
原Model	0.9140	0.69	0.8686
最大特徵數改為100000	0.9132	0.701	0.8725
LSTM改為雙向	0.9121	0.693	0.8874

■雙向LSTM

✓同時考慮該字詞的上文與下文



- Accuracy: 0.9132
- Precision: 0.701
- Recall: 0.8725

- Accuracy: 0.9121
- Precision: 0.693
- Recall: 0.8874

模型改善

Model	Accuracy	Precision	Recall
原Model	0.9140	0.69	0.8686
最大特徵數改為100000	0.9132	0.701	0.8725
LSTM改為雙向	0.9121	0.693	0.8874
增加卷積層數	0.9286	0.704	0.8912

■增加卷積層數

✓擷取不同數量(取決於卷積核大小)的字詞

卷積核大小為1的卷積層

卷積核大小為2的卷積層

卷積核大小為3的卷積層

卷積核大小為5的卷積層

AS-IS Model

Accuracy:0.9121
Precision:0.693
Recall:0.8874

To-Be Model

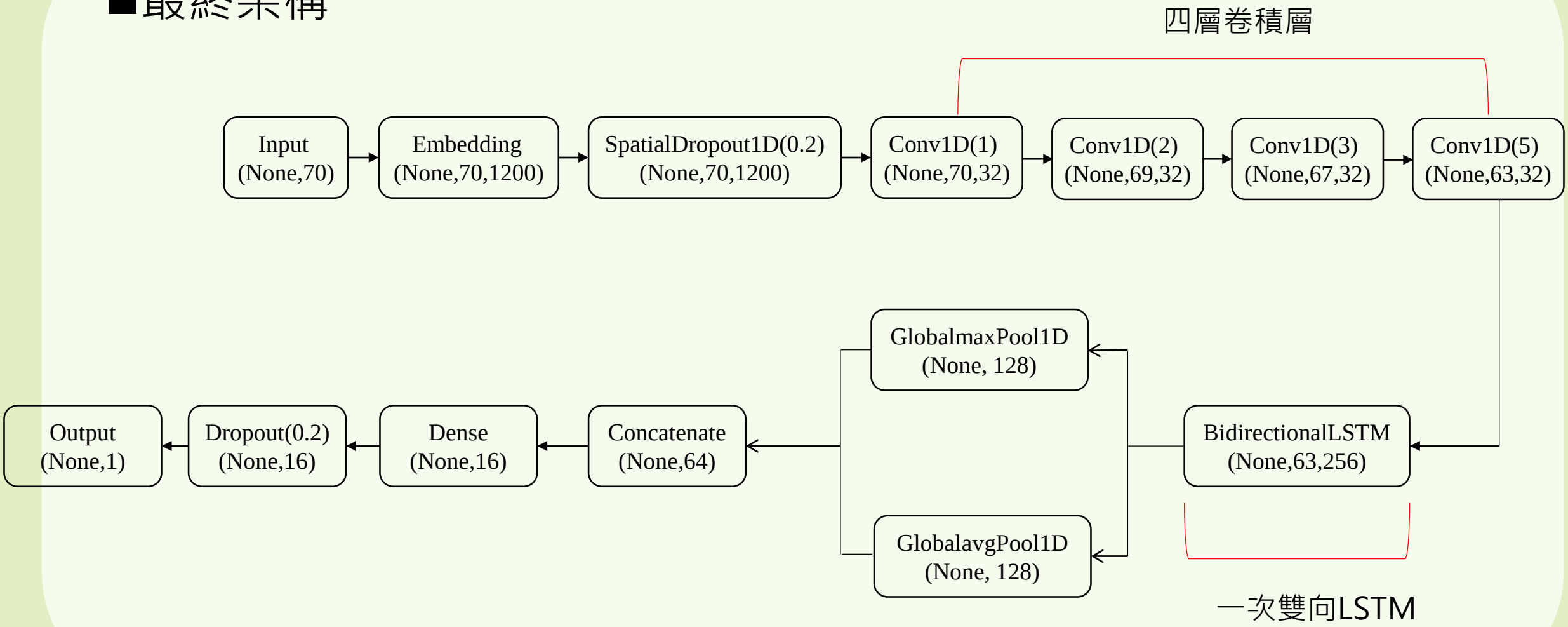
Accuracy:0.9286
Precision:0.704
Recall:0.8912

Agenda

- 背景介紹
- 問題定義
- 模型建構
- 模型改善
- 結論

結論

■最終架構


四層卷積層
Input
(None,70)
Embedding
(None,70,1200)
SpatialDropout1D(0.2)
(None,70,1200)
Conv1D(1)
(None,70,32)
Conv1D(2)
(None,69,32)
Conv1D(3)
(None,67,32)
Conv1D(5)
(None,63,32)
BidirectionalLSTM
(None,63,256)
GlobalmaxPool1D
(None, 128)
GlobalavgPool1D
(None, 128)
Concatenate
(None,64)
Dense
(None,16)
Dropout(0.2)
(None,16)
Output
(None,1)
一次雙向LSTM

結論

■最終結果

- ✓ 最終的模型準確率為0.9286，精確度0.704，召回率0.8912，皆比原模型有所提升。
- ✓ 此模型可以做為文本分類及CNNLSTM模型建置之參考。

結論

■小結

- ■ 透過比較CNN、LSTM以及CNNLSTM等三個模型，並針對CNNLSTM進行後續改善，得到以下結論:

  - ✓ 只使用CNN模型進行文本分類，表現也不差，且訓練時間很短
  - ✓ 最終模型準確率雖然到達9成以上，且各績效指標皆有提升，但精確度仍不高，只有約7成

- ■ 自行測試

```
df_testself = pd.read_excel("D:/quora-insincere-questions-classification/testself.xlsx")
y_testself = df_testself["target"].values
testself_tokenized = tk.texts_to_sequences(df_testself['question_text'].fillna('missing'))
x_testself = pad_sequences(testself_tokenized, maxlen = max_len)
loss, accuracy = model.evaluate(x_testself, y_testself)
print("Test: accuracy = %f  ;  loss = %f" % (accuracy, loss))
```

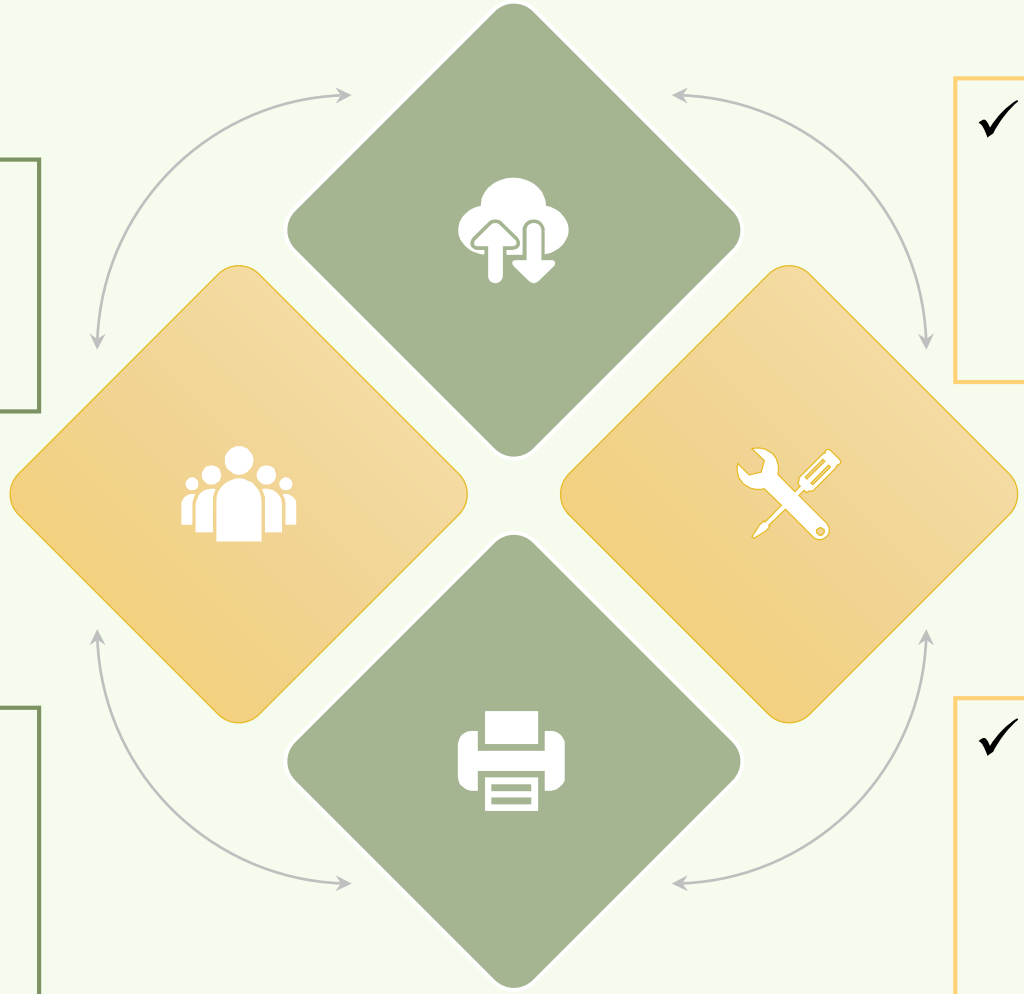
```
20/20 [==============================] - 0s 781us/step
Test: accuracy = 0.750000  ;  loss = 1.175951
```

  - ✓ 從網路上選擇了20個句子作為測試集，來測試最終模型的結果準確率為0.75

結論

■後續優化與應用

- ◆ 增加垃圾訊息的訓練樣本，提高精確度
- ◆ 改變其他超參數或是使用其他模型，來使分類結果更好。



- ✓ 可以用來解決不同的文本分類問題，幫助平台管理ex: 社群發文
- ✓ CNNLSTM模型還可以使用在生成一序列圖片或影像的文字描述,影片分類