遞迴神經網路與變形器

作業二

學號：313831025

姓名：俞博云

一、 題目敘述

在本次作業中，我們首先需對災難相關的文本數據進行前處理，接著分別使用 LSTM 與 GRU 模型來完成災難文本分類任務，判斷一則推文是否與真實災難有關（災難：1，非災難：0）。最後，比較 LSTM 與 GRU 在模型效能、訓練速度、記憶體使用量等方面的表現差異。

二、 數據預處理

本作業選用 Kaggke 上之災難文本資料集，大約包含 50 萬筆資料。首先需要對資料集進行預處理成使模型可以理解的數值形式，主要分為**數據讀取、建立詞彙表、拆分數據集、封裝自定義數據集以及建立 DataLoaders：**

2.1 數據讀取

數據集共約有 7613 筆資料，其中包含五種資料，為”id”、”keyword”、”location”、”text”，以及”target”，如表 2.1 所示，”id”為問題序號，”text”為內文，而”target”包含”0”及”1”，表示是否與災難有關。

表 2.1、數據集資料

id	text	keyword	location	target
1	Our Deeds are the Reason of this #earthquake May ALLAH Forgive us all	NaN	NaN	1
53	On plus side LOOK AT THE SKY LAST NIGHT IT WAS ABLAZE http://t.co/qqsmshaJ3N	NaN	NaN	0

首先對於數據資料集(CSV 格式)使用 Pandas 讀取，詳細程式碼如圖 2.1 所示。

```
In [3]: train_data = pd.read_csv('/home/aicv/work/datasets/train.csv')
```

圖 2.1、讀取資料程式碼

2.2 清理文字串

由於資料集文本有大量冗餘之詞彙，如標點符號、網址及 Emoji 等等，因此需要對資料進行清理，以下針對每種冗餘詞彙進行處理：

首先，移除標點符號，如!?.,;:"等符號，讓模型更專注於單字本身，而非標點，詳細程式碼如圖 2.2 所示。

```
In [5]:
def toclean_text(text):

    clean_text = [char for char in text if char not in string.punctuation]

    clean_text = ''.join(clean_text)

    return clean_text

In [6]:
train_data['clean_text'] = train_data['text'].apply(toclean_text)
```

圖 2.2、移除標點符號程式碼

接著，為了防止網址影響模型學習語意，統一處理為一個 token 為 URL，並且移除 HTML 的標籤以及非 ASCII 字符的字，如無法顯示的字符，詳細程式碼如圖 2.3 所示。

```
# Remove all URLs, replace by URL
def remove_URL(text):
    url = re.compile(r'https?://\S+|www\.\S+')
    return url.sub(r'URL',text)

# Remove HTML beacon
def remove_HTML(text):
    html=re.compile(r'<.*?>')
    return html.sub(r'',text)

# Remove non printable characters
def remove_not_ASCII(text):
    text = ''.join([word for word in text if word in string.printable])
    return text
```

圖 2.3、替換網址程式碼

再針對各種縮寫展開成完整單字，如表 2.2 之對應表，詳細程式碼如圖 2.4 所示。

表 2.2、縮寫對應表

符號	完整字
$	doller
€	euro

```
# Change an abbreviation by its true meaning
def word_abbrev(word):
    return abbreviations[word.lower()] if word.lower() in abbreviations.keys() else word
```

圖 2.4、替換縮寫字符程式碼

並且再對有提及人物的部分均替換為 USER，而由於數字通常與分類無直接語意關聯，因此也替換成 NUMBER，詳細程式碼如圖 2.5 所示。

```
# Remove @ and mention, replace by USER
def remove_mention(text):
    at=re.compile(r'@\S+')
    return at.sub(r'USER',text)

# Remove numbers, replace it by NUMBER
def remove_number(text):
    num = re.compile(r'[-+]?[.\d]*[\d]+[:,.\d]*')
    return num.sub(r'NUMBER', text)
```

圖 2.5、替換 USER 以及 NUMBER 程式碼

接著再針對 EMOJI 的部分替換，以避免 EMOJI 編碼影響模型結果，詳細程式碼如圖 2.6 所示。

```
# Remove all emojis, replace by EMOJI
def remove_emoji(text):
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'EMOJI', text)

# Replace some others smileys with SADFACE
def transcription_sad(text):
    eyes = "[8:=;]"
    nose = "['`\-]"
    smiley = re.compile(r'[8:=;][\'\-]?[(\\/]')
    return smiley.sub(r'SADFACE', text)

# Replace some smileys with SMILE
def transcription_smile(text):
    eyes = "[8:=;]"
    nose = "['`\-]"
    smiley = re.compile(r'[8:=;][\'\-]?[)dDp]')
    #smiley = re.compile(r'#{eyes}#{nose}[)d]+|[)d]+#{nose}#{eyes}/i')
    return smiley.sub(r'SMILE', text)

# Replace <3 with HEART
def transcription_heart(text):
    heart = re.compile(r'<3')
    return heart.sub(r'HEART', text)
```

圖 2.6、替換 EMOJI 程式碼

因為英文的 stopword 在分類任務中多為冗餘字，如 is、the 及 and 等等，因此再將英文的 stopword 移除，詳細程式碼如圖 2.7 所示。

```
In [11]:
def toremove_stopword(text):
    remove_stopword = [word for word in text.split() if word.lower() not in stopwords.words('english')]

    return remove_stopword

train_data['clean_text'] = train_data['clean_text'].apply(toremove_stopword)
```

圖 2.7、移除 stopword 程式碼

2.3 定義詞彙表

下一步建立一個最多保留 3000 詞的詞彙表，將文字透過 Tokenizer 轉成數字 ID，並且使用 pad_sequences 把序列補齊成固定長度，詳細程式碼如圖 2.8 所示。

```
In [12]: max_features=3000
         tokenizer=Tokenizer(num_words=max_features,split=' ')
         tokenizer.fit_on_texts(train_data['clean_text'].values)
         X = tokenizer.texts_to_sequences(train_data['clean_text'].values)
         X = pad_sequences(X)
```

圖 2.8、定義詞彙表程式碼

三、 模型架構與訓練細節

在本次作業中，嘗試了各三種基於 LSTM 以及 GRU 的模型，並在架構上進行了一些變化，以探索不同結構的影響。下面分為 LSTM 模型架構、GRU 模型架構及模型超參數兩個部分進行說明：

3.1 LSTM 模型架構

3.1.1 基本 LSTM 架構

此為基本的 LSTM 架構，分為三個部分，如圖 3.1 所示，分別為**嵌入層 (Embedding Layer)**、**LSTM 層以及全連接層 (Fully Connected Layer)**。

Layer (type)	Output Shape	Param #	Trai…
embedding (Embedding)	(512, 20, 1024)	3,072,000	Y
lstm (LSTM)	(512, 512)	3,147,776	Y
dense (Dense)	(512, 1)	513	Y

圖 3.1、基本 LSTM 架構

程式碼如圖 3.2 所示，建立完三層基本架構後，模型首先會檢查輸入序列的長度是否為零，若為零，則返回一個全為零的張量以避免計算錯誤。接著，將輸入的單詞索引轉換為對應的嵌入向量，形成詞向量序列。這個序列隨後會被傳入設有 dropout=0.5 與 recurrent_dropout=0.5 的 LSTM 層，以進行時間序列建模並防止過擬合。最後，LSTM 的輸出結果會送入全連接層，映射為一個 logit 值，作為二分類任務（是否為災難相關推文）的最終預測輸出。

```
# Hyperparameters
embed_dim = 1024
lstm_out = 512

# Improved model architecture
model = Sequential()
model.add(Embedding(max_features, embed_dim, input_length=X.shape[1]))
model.add(LSTM(lstm_out, dropout=0.5, recurrent_dropout=0.5))
model.add(Dense(1))
```

圖 3.2、基本 LSTM 架構程式碼

3.1.2 基本 LSTM 架構結合 Dropout 層

此為基本的 LSTM 架構結合 Dropout 層避免過擬合，分為五個部分，如圖 3.3 所示，分別為**嵌入層(Embedding Layer)、Dropout 層、LSTM 層、Dropout 層以及全連接層(Fully Connected Layer)**。

Layer (type)	Output Shape	Param #	Trai…
embedding (Embedding)	(512, 20, 1024)	3,072,000	Y
dropout (Dropout)	(512, 20, 1024)	0	-
lstm (LSTM)	(512, 512)	3,147,776	Y
dropout_1 (Dropout)	(512, 512)	0	-
dense (Dense)	(512, 1)	513	Y

圖 3.3、基本 LSTM 架構結合 Dropout 架構

程式碼如圖 3.4 所示，建立完五層基本架構後，模型首先會檢查輸入序列的長度是否為零，若為零，則返回一個全為零的張量以避免計算錯誤。接著，將輸入的單詞索引轉換為對應的嵌入向量，形成詞向量序列。此詞向量序列會先經過一層 Dropout（rate = 0.3）以進行正則化，再傳入設有 dropout=0.5 與 recurrent_dropout=0.5 的 LSTM 層進行時間序列建模，藉此捕捉上下文語意並防止過擬合。LSTM 輸出後，會再經過一層 Dropout（rate＝0.3）以進一步降低過擬合風險，最後送入全連接層，映射為一個 logit 值，作為二分類任務（是否為災難相關推文）的最終預測輸出。

```
# Hyperparameters
embed_dim = 1024
lstm_out = 512

# Improved model architecture
model = Sequential()
model.add(Embedding(max_features, embed_dim, input_length=X.shape[1]))
model.add(Dropout(0.3))
model.add(LSTM(lstm_out, dropout=0.5, recurrent_dropout=0.5))
model.add(Dropout(0.3))
model.add(Dense(1))
```

圖 3.4、基本 LSTM 架構結合 Dropout 架構程式碼

3.1.3 雙層 LSTM 架構結合 Dropout 層

此架構與基本的 LSTM 架構相同，分為三個部分，如圖 3.5 所示，**分別為嵌入層(Embedding Layer)、LSTM 層以及全連接層(Fully Connected Layer)**。

Layer (type)	Output Shape	Param #	Trai…
embedding (Embedding)	(512, 20, 1024)	3,072,000	Y
dropout (Dropout)	(512, 20, 1024)	0	-
lstm (LSTM)	(512, 20, 512)	3,147,776	Y
lstm_1 (LSTM)	(512, 512)	2,099,200	Y
dropout_1 (Dropout)	(512, 512)	0	-
dense (Dense)	(512, 1)	513	Y

圖 3.5、雙層 LSTM 架構結合 Dropout 層架構

程式碼如圖 3.6 所示，建立完六層基本架構後，模型首先會檢查輸入序列的長度是否為零，若為零，則返回一個全為零的張量以避免計算錯誤。接著，將輸入的單詞索引轉換為對應的嵌入向量，形成詞向量序列。此詞向量序列會先經過一層 Dropout（rate＝0.3）以進行正則化，再依序傳入兩層皆設有 dropout=0.5 與 recurrent_dropout=0.5 的 LSTM 層，進行時間序列建模。透過堆疊式 LSTM 層，模型能更深入捕捉輸入文本中的長期與短期依賴關係。第二層 LSTM 輸出後，會再經過一層 Dropout（rate＝0.3）以進一步降低過擬合風險，最後送入全連接層，映射為一個 logit 值，作為二分類任務（是否為災難相關推文）的最終預測輸出。

```
# Hyperparameters
embed_dim = 1024
lstm_out = 512

# Improved model architecture
model = Sequential()
model.add(Embedding(max_features, embed_dim, input_length=X.shape[1]))
model.add(Dropout(0.3))
model.add(LSTM(lstm_out, dropout=0.5, recurrent_dropout=0.5, return_sequences=True))
model.add(LSTM(lstm_out, dropout=0.5, recurrent_dropout=0.5))
model.add(Dropout(0.3))
model.add(Dense(1))
```

圖 3.6、雙層 LSTM 架構結合 Dropout 層架構程式碼

3.2GRU 模型架構

3.2.1 基本 GRU 架構

此為基本的 GRU 架構，分為三個部分，如圖 3.7 所示，分別為**嵌入層(Embedding Layer)、GRU 層以及全連接層(Fully Connected Layer)**。

Layer (type)	Output Shape	Param #	Trai…
embedding (Embedding)	(512, 20, 1024)	3,072,000	Y
lstm (LSTM)	(512, 512)	3,147,776	Y
dense (Dense)	(512, 1)	513	Y

圖 3.7、基本 GRU 架構

程式碼如圖 3.8 所示，建立完三層基本架構後，模型首先會檢查輸入序列的長度是否為零，若為零，則返回一個全為零的張量以避免計算錯誤。接著，將輸入的單詞索引轉換為對應的嵌入向量，形成詞向量序列。這個序列隨後會被傳入設有 dropout=0.5 與 recurrent_dropout=0.5 的 GRU 層，以進行時間序列建模並防止過擬合。最後，GRU 的輸出結果會送入全連接層，映射為一個 logit 值，作為二分類任務（是否為災難相關推文）的最終預測輸出。

```
# Hyperparameters
embed_dim = 1024
lstm_out = 512

# Improved model architecture
model = Sequential()
model.add(Embedding(max_features, embed_dim, input_length=X.shape[1]))
model.add(LSTM(lstm_out, dropout=0.5, recurrent_dropout=0.5))
model.add(Dense(1))
```

圖 3.8、基本 GRU 架構程式碼

3.2.2 基本 GRU 架構結合 Dropout 層

此為基本的 GRU 架構結合 Dropout 層避免過擬合，分為五個部分，如圖 3.9 所示，分別為**嵌入層(Embedding Layer)、Dropout 層、GRU 層、Dropout 層以及全連接層(Fully Connected Layer)**。

Layer (type)	Output Shape	Param #	Trai…
embedding (Embedding)	(512, 20, 1024)	3,072,000	Y
dropout (Dropout)	(512, 20, 1024)	0	-
lstm (LSTM)	(512, 512)	3,147,776	Y
dropout_1 (Dropout)	(512, 512)	0	-
dense (Dense)	(512, 1)	513	Y

圖 3.9、基本 GRU 架構結合 Dropout 架構

程式碼如圖 3.10 所示，建立完五層基本架構後，模型首先會檢查輸入序列的長度是否為零，若為零，則返回一個全為零的張量以避免計算錯誤。接著，將輸入的單詞索引轉換為對應的嵌入向量，形成詞向量序列。此詞向量序列會先經過一層 Dropout（rate = 0.3）以進行正則化，再傳入設有 dropout=0.5 與 recurrent_dropout=0.5 的 GRU 層進行時間序列建模，藉此捕捉上下文語意並防止過擬合。GRU 輸出後，會再經過一層 Dropout（rate＝0.3）以進一步降低過擬合風險，最後送入全連接層，映射為一個 logit 值，作為二分類任務（是否為災難相關推文）的最終預測輸出。

```
# Hyperparameters
embed_dim = 1024
lstm_out = 512

# Improved model architecture
model = Sequential()
model.add(Embedding(max_features, embed_dim, input_length=X.shape[1]))
model.add(Dropout(0.3))
model.add(LSTM(lstm_out, dropout=0.5, recurrent_dropout=0.5))
model.add(Dropout(0.3))
model.add(Dense(1))
```

圖 3.10、基本 GRU 架構結合 Dropout 架構程式碼

3.2.3 雙層 GRU 架構結合 Dropout 層

此架構與基本的 GRU 架構相同，分為三個部分，如圖 3.11 所示，**分別為嵌入層(Embedding Layer)、GRU 層以及全連接層(Fully Connected Layer)**。

Layer (type)	Output Shape	Param #	Trai…
embedding (Embedding)	(512, 20, 1024)	3,072,000	Y
dropout (Dropout)	(512, 20, 1024)	0	-
lstm (LSTM)	(512, 20, 512)	3,147,776	Y
lstm_1 (LSTM)	(512, 512)	2,099,200	Y
dropout_1 (Dropout)	(512, 512)	0	-
dense (Dense)	(512, 1)	513	Y

圖 3.11、雙層 GRU 架構結合 Dropout 層架構

程式碼如圖 3.12 所示，建立完六層基本架構後，模型首先會檢查輸入序列的長度是否為零，若為零，則返回一個全為零的張量以避免計算錯誤。接著，將輸入的單詞索引轉換為對應的嵌入向量，形成詞向量序列。此詞向量序列會先經過一層 Dropout（rate＝0.3）以進行正則化，再依序傳入兩層皆設有 dropout=0.5 與 recurrent_dropout=0.5 的 GRU 層，進行時間序列建模。透過堆疊式 GRU 層，模型能更深入捕捉輸入文本中的長期與短期依賴關係。第二層 LSTM 輸出後，會再經過一層 Dropout（rate＝0.3）以進一步降低過擬合風險，最後送入全連接層，映射為一個 logit 值，作為二分類任務（是否為災難相關推文）的最終預測輸出。

```
# Hyperparameters
embed_dim = 1024
lstm_out = 512

# Improved model architecture
model = Sequential()
model.add(Embedding(max_features, embed_dim, input_length=X.shape[1]))
model.add(Dropout(0.3))
model.add(LSTM(lstm_out, dropout=0.5, recurrent_dropout=0.5, return_sequences=True))
model.add(LSTM(lstm_out, dropout=0.5, recurrent_dropout=0.5))
model.add(Dropout(0.3))
model.add(Dense(1))
```

圖 3.12、雙層 GRU 架構結合 Dropout 層架構程式碼

3.3模型超參數設置

所有的超參數都是透過命令列參數來設定的，其預設值如表 3.1 所示。每個單詞的嵌入向量維度(Embedding dim)設為 1024，LSTM/GRU 輸入層大小均設為 512，並且，這些向量會經過設定的 Layer 層數(num layers)，最終模型在二分類任務中輸出一個標量。此外，訓練和驗證的批次大小均設定為 512，以便於觀察每個樣本的預測結果，由於訓練時採用選取驗證集上準確率最高的模型作為最終模型的設定，因此每次實驗中實際訓練的 epoch 數可能略有不同，視模型表現而定。本次實驗中各模型的訓練 epoch 範圍大致落在 16 到 19 之間。這些參數共同決定了模型的結構、容量以及訓練時的數據處理方式，從而最終影響分類任務的效果。

表 3.1、超參數設定表

超參數名稱	數值
Embedding dim	1024
LSTM/GRU out dim	512
Output dim	1
Num layers	1~2
Train Batch size	512
Val Batch size	512
Epoch	16~19

此外在訓練設定中，損失函數使用 Binary CrossEntropy Loss 結合 Sigmoid 和二元交叉熵計算，因此能夠直接處理模型輸出 logits，提升數值穩定性。在 Optimizer 優化器使用 Admin 優化器，其自適應學習率調整機制有助於加快收斂並提高訓練效率，範例程式如圖 3.13 所示。

```
In [17]:
checkpoint = ModelCheckpoint('GRU_Models/GRU_best_model.h5',   # Save path and filename
                             monitor='val_accuracy',  # Monitor validation accuracy
                             mode='max',              # Higher is better
                             save_best_only=True,     # Save only the best model
                             verbose=1)

# Set target accuracy to 0.90
custom_early_stop = CustomEarlyStopping(threshold=0.90, patience=3)

history = model.fit(X_train, y_train, epochs=50, batch_size=512, validation_data=(X_test, y_test), callbacks=[custom_early_
model.save('GRU_Models/GRU_final_model.h5')
```

圖 3.13、訓練示意程式碼

四、 實驗介紹與結果

4.1 實驗介紹

本章節中對於第三章提到的六種模型進行六種實驗，分別對不同LSTM/GRU 結構以及層數進行比較，詳細如表 4.1 所示。

表 4.1、模型實驗詳細內容表

Model_Name	Configuration
LSTM	基本 LSTM 架構 (層數：1)
LSTM+Dropout	基本 LSTM 架構+Dropout 層 (層數：2)
2LSTM+Dropout	基本 LSTM 架構+Dropout 層 (層數：2)
GRU	基本 GRU 架構 (層數：1)
GRU +Dropout	基本 GRU 架構+Dropout 層 (層數：2)
2 GRU +Dropout	基本 GRU 架構+Dropout 層 (層數：2)

4.2 實驗結果

首先在基本 LSTM 架構中，從模型的損失函數變化圖，如圖 4.1 所示，可觀察出，訓練初期（約第 0 至第 6 個 epoch）Train Loss 與 Val Loss 皆同步下降，顯示模型在有效學習並具備良好的泛化能力。然而在約第 8 個 epoch 之後，雖然 Train Loss 持續下降，但 Val Loss 開始趨於平緩甚至略微上升，顯示模型逐漸產生過擬合現象。整體而言，模型在第 6 至第 8 個 epoch 的表現最佳，此時訓練與驗證損失皆處於相對低點，差距也較小。本次實驗採用 Val Loss 最低的模型作為最終輸出，因此各模型訓練 epoch 數會依據 EarlyStopping 機制略有不同，範圍約為 16 至 19 個 epoch。

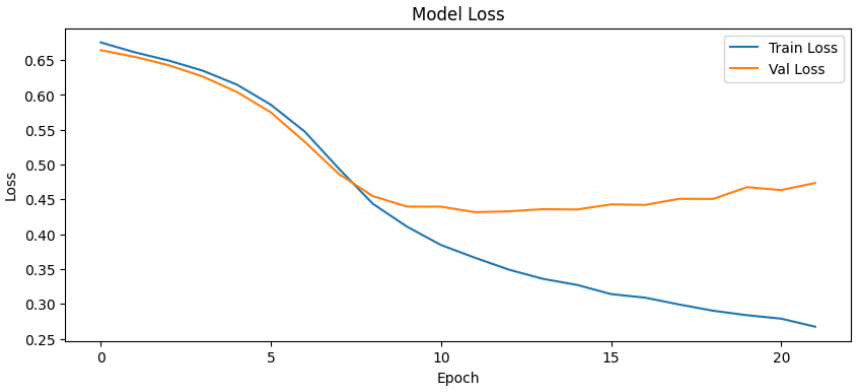

圖 4.1、基本 LSTM 架構 Train/Val loss 圖

相對地，從圖 4.2 可看出準確率曲線也呈現類似趨勢。在前段訓練期間，Train Accuracy 與 Val Accuracy 皆穩定上升，顯示模型學習效果良好。大約在第 8 個 epoch 後，Val Accuracy 開始趨於飽和，而 Train Accuracy 則持續上升，顯示模型雖對訓練資料學習得更好，但對驗證資料的泛化能力並未提升，進一步印證了過擬合的現象。

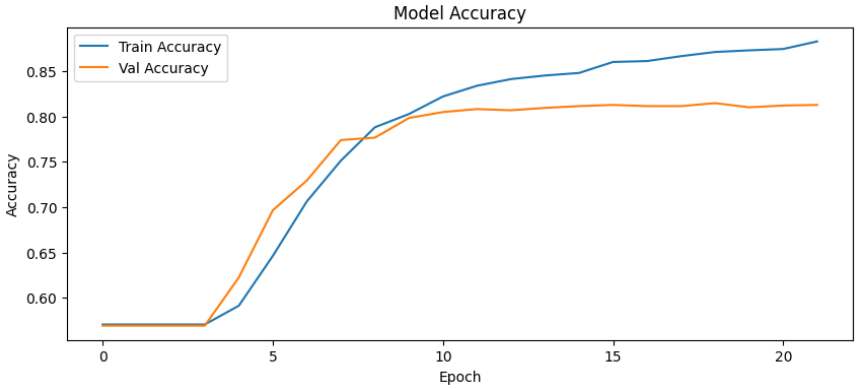

圖 4.2、基本 LSTM 架構 Accuracy 圖

由於基本的 LSTM 架構出現過擬合情況，因此在模型中加入額外的 Dropout 層以嘗試減緩過擬合，同時也透過增加一層 LSTM 以提升模型複雜度，觀察是否能改善該現象。同理，在 GRU 架構中亦進行相同調整。然而，從圖 4.3 至圖 4.12 的結果可看出，這些改動並未有效減少過擬合，顯著改善的效果並不明顯。

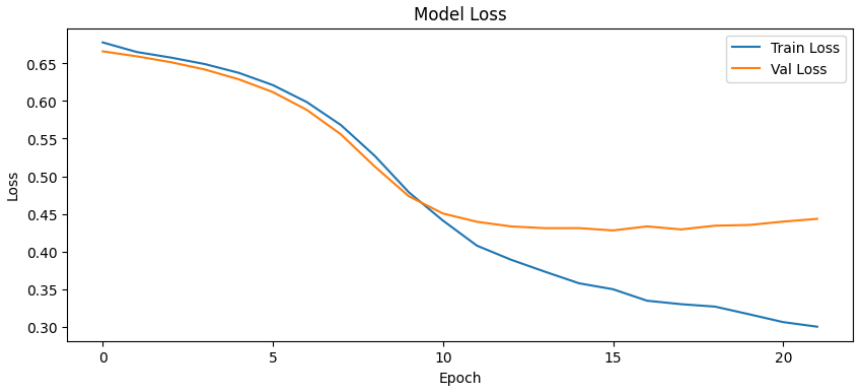

圖 4.3、基本 LSTM 架構+Dropout 層 Train/Val loss 圖

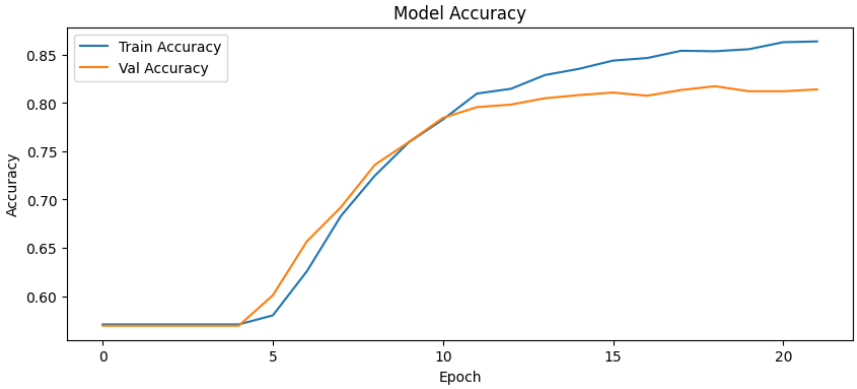

圖 4.4、基本 LSTM 架構+Dropout 層 Accuracy 圖

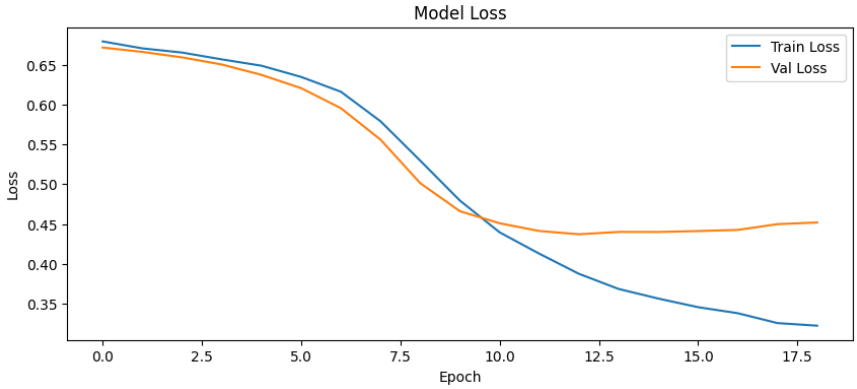

圖 4.5、2 層基本 LSTM 架構+Dropout 層 Train/Val loss 圖

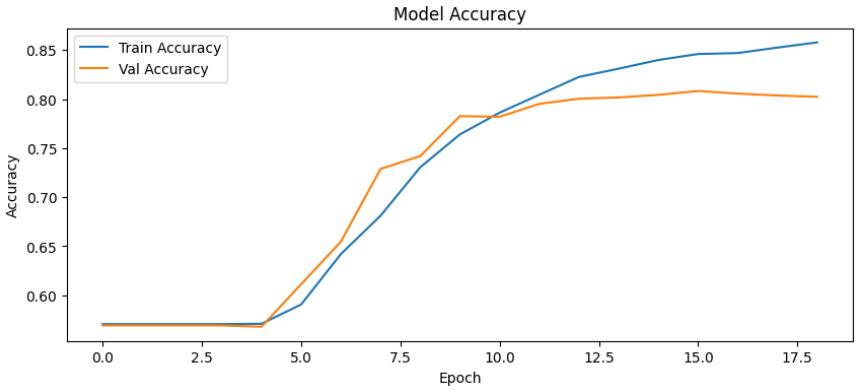

圖 4.6、2 層基本 LSTM 架構+Dropout 層 Accuracy 圖

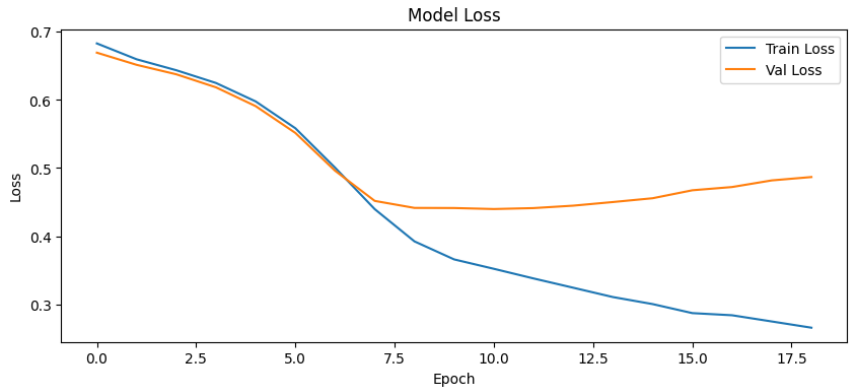

圖 4.7、基本 GRU 架構 Train/Val loss 圖

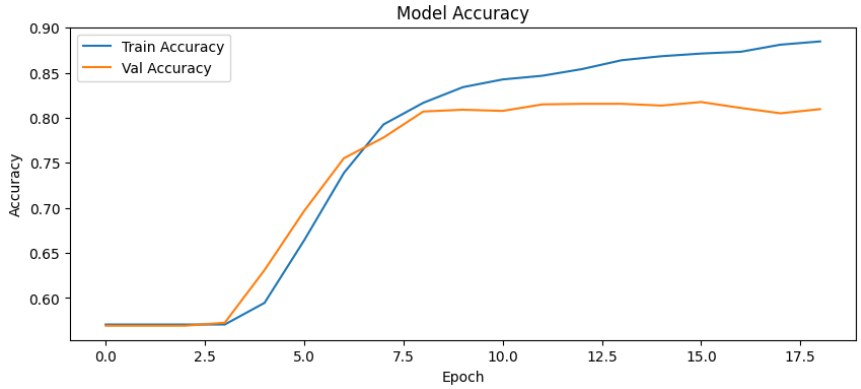

圖 4.8、基本 GRU 架構 Accuracy 圖

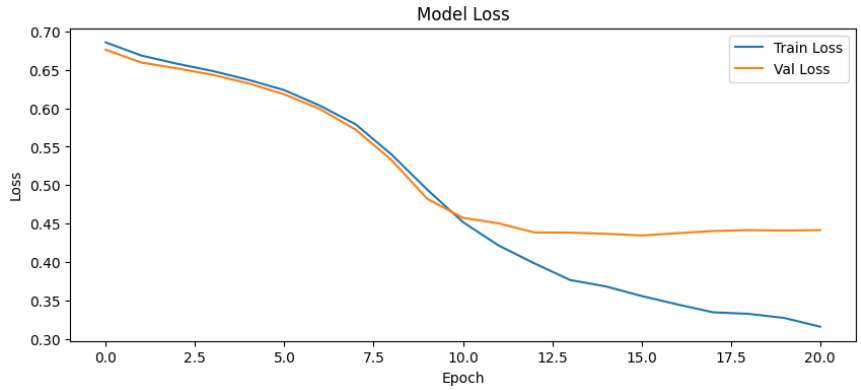

圖 4.9、基本 GRU 架構+Dropout 層 Train/Val loss 圖

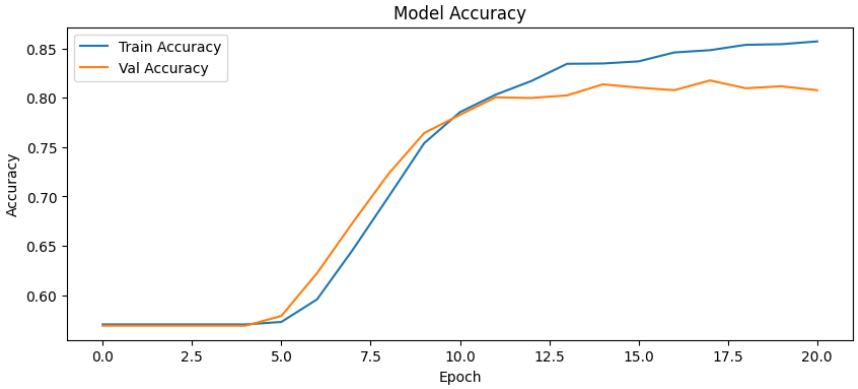

圖 4.10、基本 GRU 架構+Dropout 層 Accuracy 圖

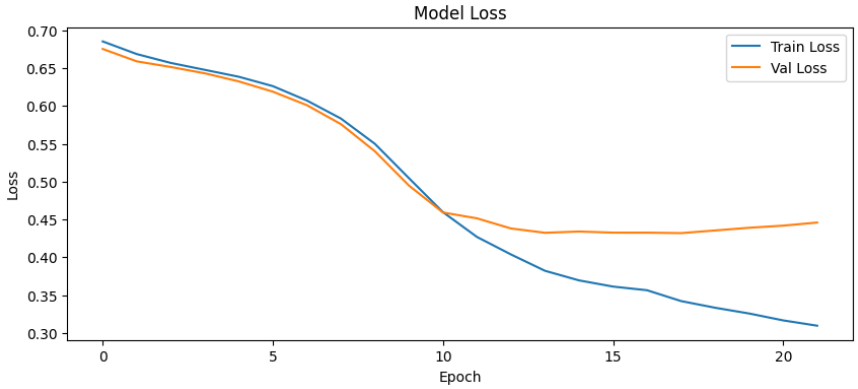

圖 4.11、2 層基本 GRU 架構+Dropout 層 Train/Val loss 圖

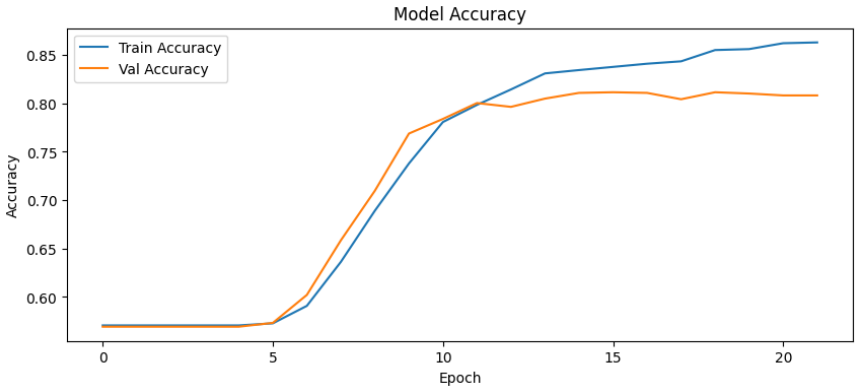

圖 4.12、2 層基本 GRU 架構+Dropout 層 Accuracy 圖

接著，針對各模型的模型參數量、模型大小、訓練集以及驗證集準確度等等作比較，如表 4.2 所示。

表 4.2、實驗結果數據表

Model_Name	Total params	Model Size	Train Accuracy	Val Accuracy	Val Recall	Val Precision
LSTM	6,220,289	23.73 MB	0.88670	0.81484	0.68140	0.85962
LSTM+Dropout	6,220,289	23.73 MB	0.87833	0.81747	0.69207	0.85660
2LSTM+Dropout	8,319,489	31.74 MB	0.86174	0.80827	0.65396	0.86842
GRU	5,434,881	20.73 MB	0.89113	0.81747	0.71951	0.83392
GRU +Dropout	5,434,881	20.73 MB	0.87192	0.81747	0.68902	0.85931
2 GRU +Dropout	7,010,817	26.74 MB	0.86141	0.81418	0.67378	0.86497

從表 4.2 中的結果可以看出，雖然透過加入 Dropout 或堆疊更多層 LSTM/GRU 嘗試減緩過擬合現象，但整體驗證準確率並未顯著提升，甚至效果有限。特別是在 LSTM 架構中，儘管增加模型複雜度與正則化，驗證表現反而略為下降。相較之下，GRU 模型在參數較少的情況下仍能達到穩定且優異的驗證準確率（81.747%）與召回率（71.951%），整體平衡表現最佳。

另外在 Kaggle 網站上上傳全部的 test 資料，結果如表 4.3 所示，可以發現加入 Dropout 的 GRU 模型在實際預測表現上表現最佳達 0.79681。

表 4.3、Kaggle 實驗結果數據表

Lstm	0.79313
Lstm dropout	0.79589
2 Lstm dropout	0.79037
GRU	0.78700
GRU dropout	**0.79681**
2 GRU dropout	0.79129

綜合上述實驗結果與實際應用於 Kaggle 測試資料的表現來看，GRU 架構在本次災難文本分類任務中整體表現最為優異。