ADLxMLDS HW1 B03705006 侯舜元

Model Description： RNN

資料前處理：

只取 MFCC 39 維的 feature 拿來當 training feature，並將 label 資料先 map 到對應的 39 個英文字母並給予編號。對於每一個聲音 sequence，將其 frame 的個數 padding 到一樣長度（777），並給予 padding 的 frame 一個自己的 label。最後得到的 input shape 會是（3696, 777, 39），output shape 為 （3696,777,40)。

Model 架構圖 （rnn_model27_d.h5）：

```
model = Sequential()
model.add(Bidirectional(GRU(512, return_sequences=True, activation='relu', dropout=0.4), input_shape=input_shape,))
model.add(Bidirectional(GRU(512, return_sequences=True, activation='relu', dropout=0.4)))
model.add(Bidirectional(GRU(256, return_sequences=True, activation='relu', dropout=0.4)))
model.add(Bidirectional(GRU(256, return_sequences=True, activation='relu', dropout=0.4)))
model.add(TimeDistributed(Dense(40, activation='softmax')))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_1 (Bidirection (None, 777, 1024)         1695744
_________________________________________________________________
bidirectional_2 (Bidirection (None, 777, 1024)         4721664
_________________________________________________________________
bidirectional_3 (Bidirection (None, 777, 512)          1967616
_________________________________________________________________
bidirectional_4 (Bidirection (None, 777, 512)          1181184
_________________________________________________________________
time_distributed_1 (TimeDist (None, 777, 40)           20520
=================================================================
```

Model：

使用 4 層 GRU 接上 1 層 Dense layer output，GRU layer 的大小每層略有不同，但 activation function 皆選擇 relu。最後的而有使用到的 wrapper 在下面說明：

Bidirectional GRU

將 GRU layer 包上 Bidirectional wrapper，其作用使兩個 RNN stack 在一起，並將 input 正反輸入一次，故每個 time step 的 output 會依據兩個 RNN 的 hidden state（參數也會變成 2 倍）。經嘗試後有包上 Bidirectional wrapper 的 performance 較好。

TimeDistributed Dense

包上 TimeDistributed wrapper，讓 Dense layer 獨立 apply 到每個 input time step。適合用來處理本次作業 many-to-many 的問題。

Performance:

用以上的 model 可以在 10% validation set 的 categorical crossentropy loss 得到 0.1957 的分數。上傳的 Kaggle 的分數 public set 最佳為 **7.6666**。

Model Description： CNN

資料前處理：

與 RNN 相同。

Model 架構圖（cnn_model3best.h5）：

```
def genModel(input_shape):
    model = Sequential()
    model.add(Conv1D(256,  30, input_shape=input_shape, padding = 'same', activation='relu'))
    model.add(Bidirectional(GRU(512, return_sequences=True, activation='relu', dropout=0.4)))
    model.add(Bidirectional(GRU(512, return_sequences=True, activation='relu', dropout=0.4)))
    model.add(TimeDistributed(Dense(512, activation='relu')))
    model.add(Dropout(0.4))
    model.add(TimeDistributed(Dense(512, activation='relu')))
    model.add(Dropout(0.4))
    model.add(TimeDistributed(Dense(40, activation='softmax')))
    model.summary()
    return model
```

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 777, 256)	299776
bidirectional_1 (Bidirection	(None, 777, 1024)	2362368
bidirectional_2 (Bidirection	(None, 777, 1024)	4721664
time_distributed_1 (TimeDist	(None, 777, 512)	524800
dropout_1 (Dropout)	(None, 777, 512)	0
time_distributed_2 (TimeDist	(None, 777, 512)	262656
dropout_2 (Dropout)	(None, 777, 512)	0
time_distributed_3 (TimeDist	(None, 777, 40)	20520

Model：

先接一層 kernel_size 為 30（每次掃過 30 個 frame），filter 數為 256 的 Conv1D，接上兩層 Bidirectional GRU 後接三層 TimeDistributed Dense。

Performance:

用以上的 model 可以在 10% validation set 的 categorical crossentropy loss 得到 0.2201 的分數。上傳的 Kaggle 的分數 public set 最佳為 **9.23163**

How to improve your performance

1. 資料處理

在 feature 的選擇上一開始嘗試 MFCC 和 fbank，performance 差不多但 MFCC feature 較少 train 起來較為迅速，故選擇使用 MFCC。

Baseline model 為使用『2 層 Bidirectional GRU，接上 2 層 TimeDistributed Dense』的 model。一開始在本機端 train 出來的 loss 都相當不錯(0.25 附近），但 Kaggle 上的分數相當的差，皆 **33 以上**。後來發現是在做 DataFrame 的 groupby 時不小心將 test data sort 過一遍，順序弄亂了。將順序恢復正常就順利過 public set baseline 了。

另外在輸出時，有塞選需要連續出現兩個以上的同樣 label 才會輸出，否則就拋棄該 label，如此可以提昇一些 performance。

2. Stack more layers
將 Model 建的更深且使用更寬廣參數更多的 Network，從 baseline model 發展到『2 層 Bidirectional GRU，接上 3 層 TimeDistributed Dense』，在 Kaggle 的 public set 上獲得 **8.7401**。

```
model = Sequential()
model.add(Bidirectional(GRU(512, return_sequences=True, activation='relu', dropout=0.4), input_shape=input_shape,))
model.add(Bidirectional(GRU(512, return_sequences=True, activation='relu', dropout=0.4)))
model.add(TimeDistributed(Dense(1024, activation='relu')))
model.add(Dropout(0.4))
model.add(TimeDistributed(Dense(1024, activation='relu')))
model.add(Dropout(0.4))

model.add(TimeDistributed(Dense(40, activation='softmax')))
```

後來在嘗試不同 RNN 層數後，model 發展到如 model description 所述的使用 4 層 GRU 接上 1 層 Dense layer。在 Kaggle public set 的分數 public set 最佳為 **7.6666**。

Why do I use this technique：更深的網路有更多參數，在沒有 over-fitting 的情況下往往可以更加的表現。而更多的 RNN 可以加強我們在做 sequence prediction 的 performance。

3. Ensemble
我將我在 kaggle public set 上最好的三個 model 拿來 ensemble，將 model 的 predict 的結果平均後再 argmax 取得對應的分類（label)。成功在 Kaggle public set 獲得在 **6.94915**，為所有模型最佳。

最好的三個 model 為 models 資料夾內的：
rnn_model21best.h5、rnn_model27_d.h5、rnn_model24_d.h5

Why do I use this technique：Ensemble 可以降第 model 的 variance，在不知道如何改善模型進步時是相當有效率的辦法。

Experimental results and settings

Compare and analyze the results between RNN and CNN
我嘗試了許多 CNN 的模型，Conv1D、Conv2D 搭配/不搭配 maxpooling，其中為 Conv1D 的 performace 最佳（在 Kaggle Public set 上獲得 **9.23163**）， Conv2D 使終於法得到很好的結果（最佳的 performace 在 Kaggle Public set 上獲得 **12.60451**）。
CNN+RNN 的 model 比起純粹使用 RNN layers 的 model，始終是差了一些。推測是 CNN 沿著時間軸 filter 一定數量的 frames，並沒有真的學到很多 feature。而若將 frame 在每一

個 time step 丟進 RNN 去 train，且在有包 Bidirectional Wrapper 的情下可以學到更多 sequence 的資訊。

Compare and analyze the results with other models

1. Bidirectional Wrappers
同樣的 RNN model 架構，在包上 Bidirectional Wrappers performance 都會比原先的 model performance 佳。將 input 正反輸入一次並共用兩個相鄰 RNN 的 hidden state 在這次問題上看來有顯著的幫助。

2. Different RNN layers, GRU vs LSTM
選擇使用不同層數的 RNN (1~5 層），performance 為 4 層最佳。而 RNN cell 的個數有選擇從 128~512，通常越多 performance 越佳。而除了 GRU 之外也有嘗試使用 LSTM。LSTM 的結構比 GRU 較複雜需要花更多的時間 train，且個人在訓練上發現 LSTM 較難 train，有時 loss 會變成 nan 失敗。而 GRU 結構雖較簡單但也有不錯的表現，train 起來須要比較少的記憶體也較快，故最後皆選擇使用 GRU。

Library

使用的 library 和版本如下
tensorflow-gpu==1.1.0
sklearn==0.0
Theano==0.9.0
Keras==2.0.4
numpy==1.12.1
pandas==0.20.1