Keras 秘笈 ◎從入門到精通◎

1. Keras 深度學習：Keras 入門

Keras 是 Python 中以 CNTK、Tensorflow 或者 Theano 為計算後臺的一個深度學習建模環境。相對於其他深度學習的計算軟體，如：Tensorflow、Theano、Caffe 等，Keras 在實際應用中有一些顯著的優點，其中最主要的優點就是 Keras 已經高度模組化了，支援現有的常見模型（CNN、RNN 等），更重要的是建模過程相當方便快速，加快了開發速度。

筆者使用的是基於 Tensorflow 為計算後臺。接下來將介紹一些建模過程的常用層、搭建模型和訓練過程，而 Keras 中的文字、序列和圖像資料預處理，我們將在相應的實踐項目中進行講解。

1. 核心層（函數介紹一些常用參數，詳細參數可查 Keras 文檔）

1.1 全連接層：神經網路中最常用到的，實現對神經網路裡的神經元啟動。

Dense（units, activation='relu', use_bias=True）

參數說明：
units: 全連接層輸出的維度，即下一層神經元的個數。
activation：啟動函數，預設使用 Relu。
use_bias：是否使用 bias 偏置項。

1.2 啟動層：對上一層的輸出應用啟動函數。

Activation(activation)

參數說明：
Activation：想要使用的啟動函數，如：'relu'、'tanh'、'sigmoid'等。

1.3 Dropout 層：對上一層的神經元隨機選取一定比例的失活，不更新，但是權重仍然保留，防止過擬合。

Dropout(rate)

參數說明:

rate：失活的比例，0-1 的浮點數。

1.4 Flatten 層：將一個維度大於或等於 3 的高維矩陣，"壓扁"為一個二維矩陣。即保留第一個維度（如：batch 的個數），然後將剩下維度的值相乘作為"壓扁"矩陣的第二個維度。

Flatten()

1.5 Reshape 層：該層的作用和 reshape 一樣，就是將輸入的維度重構成特定的 shape。

Reshape(target_shape)

參數說明：
target_shape：目標矩陣的維度，不包含 batch 樣本數。
如我們想要一個 9 個元素的輸入向量重構成一個(None, 3, 3)的二維矩陣：
Reshape((3,3), input_length=(16,))

1.6 卷積層：卷積操作分為一維、二維、三維，分別為 Conv1D、Conv2D、Conv3D。一維卷積主要應用於以時間序列資料或文本資料，二維卷積通常應用於圖像資料。由於這三種的使用和參數都基本相同，所以主要以處理圖像資料的 Conv2D 進行說明。

Conv2D(filters, kernel_size, strides=(1, 1), padding='valid')

參數說明：
filters：卷積核的個數。
kernel_size：卷積核的大小。
strdes：步長，二維中默認為(1, 1)，一維默認為 1。
Padding：補"0"策略，'valid'指卷積後的大小與原來的大小可以不同，'same'則卷積後大小與原來大小一致。

1.7 池化層：與卷積層一樣，最大統計量池化和平均統計量池化也有三種，分別為 MaxPooling1D、MaxPooling2D、MaxPooling3D 和 AveragePooling1D、AveragePooling2D、AveragePooling3D，由於使用和參數基本相同，所以主要以 MaxPooling2D 進行說明。

MaxPooling(pool_size=(2,2), strides=None, padding='valid')

參數說明：
pool_size：長度為 2 的整數 tuple，表示在橫向和縱向的下採樣樣子，一維則為縱向的下採樣因數。
padding：和卷積層的 padding 一樣。

1.8 迴圈層：迴圈神經網路中的 RNN、LSTM 和 GRU 都繼承本層，所以該父類的參數同樣使用於對應的子類 SimpleRNN、LSTM 和 GRU。

Recurrent(return_sequences=False)

參數說明：
return_sequences：控制返回的類型，"False"返回輸出序列的最後一個輸出，"True"則返回整個序列。當我們要搭建多層神經網路（如深層 LSTM）時，若不是最後一層，則需要將該參數設為 True。

1.9 嵌入層：該層只能用在模型的第一層，是將所有索引標號的疏鬆陣列映射到緻密的低維矩陣。如我們對文本資料進行處理時，我們對每個詞編號後，我們希望將詞編號變成詞向量就可以使用嵌入層。

Embedding(input_dim, output_dim, input_length)

參數說明：
Input_dim：大於或等於 0 的整數，字典的長度即輸入資料的個數。
output_dim：輸出的維度，如詞向量的維度。
input_length：當輸入序列的長度為固定時為該長度，然後要在該層後加上 Flatten 層，然後再加上 Dense 層，則必須指定該參數，否則 Dense 層無法自動推斷輸出的維度。
該層可能有點費解，舉個例子，當我們有一個文本，該文本有 100 句話，我們已經通過一系列操作，使得文本變成一個(100,32)矩陣，每行代表一句話，每個元素代表一個詞，我們希望將該詞變為 64 維的詞向量：
Embedding(100, 64, input_length=32)
則輸出的矩陣的 shape 變為(100, 32, 64)：即每個詞已經變成一個 64 維的詞向量。

2. Keras 模型搭建

講完了一些常用層的語法後，通過模型搭建來說明 Keras 的方便性。Keras 中設定了兩類深度學習的模型，一類是序列模型（Sequential 類）；一類是通用模型（Model 類），接下來我們通過搭建下圖模型進行講解。

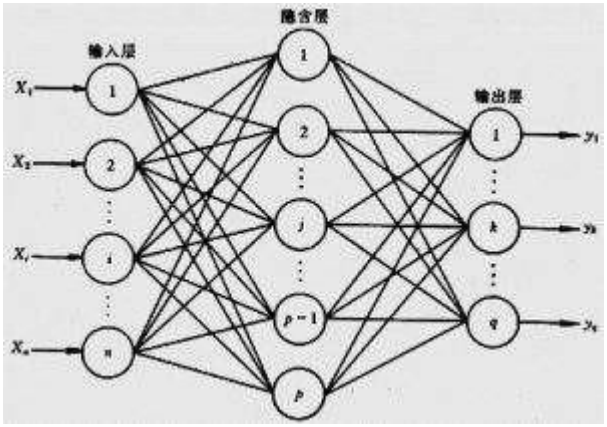


圖 1：兩層神經網路

假設我們有一個兩層神經網路，其中輸入層為 784 個神經元，隱藏層為 32 個神經元，輸出層為 10 個神經元，隱藏層使用 relu 啟動函數，輸出層使用 softmax 啟動函數。

分別使用序列模型和通用模型實現如下：

```
from keras.models import Sequential, Model
from keras.layers import Input, Dense, Activation, Conv2D, MaxPooling2D, Flatten
```

圖 2：導入相關庫

```
model = Sequential()
model.add(Dense(units=32, input_shape=(784,))) # 模型第一层要说明输入向量的维度，样本数就不用
model.add(Activation('relu'))
model.add(Dense(units=10))
model.add(Activation('softmax'))
```

圖 3：序列模型實現

使用序列模型，首先我們要產生實體 Sequential 類，之後就是使用該類的 add 函數加入我們想要的每一層，從而實現我們的模型。

```
x_input = Input(shape=(784,)) # Input将输入变成张量
dense_1 = Dense(units=32)(x_input)
act_1 = Activation('relu')(dense_1)
output = Dense(units=10, activation='softmax')(act_1)
model = Model(inputs=x_input, outputs=output)
```

圖 4：通用模型實現

使用通用模型，首先要使用 Input 函數將輸入轉化為一個 tensor，然後將每一層用變數存儲後，作為下一層的參數，最後使用 Model 類將輸入和輸出作為參數即可搭建模型。

從以上兩類模型的簡單搭建，都可以發現 Keras 在搭建模型比起 Tensorflow 等簡單太多了，如 Tensorflow 需要定義每一層的權重矩陣，輸入用預留位置等，這些在 Keras 中都不需要，我們只要在第一層定義輸入維度，其他層定義輸出維度就可以搭建起模型，通俗易懂，方便高效，這是 Keras 的一個顯著的優勢。

3. 模型優化和訓練

3.1 compile(optimizer, loss, metrics=None)

參數說明：
optimizer：優化器，如：'SGD'，'Adam'等。
loss：定義模型的損失函數，如：'mse'，'mae'等。
metric：模型的評價指標，如：'accuracy'等。

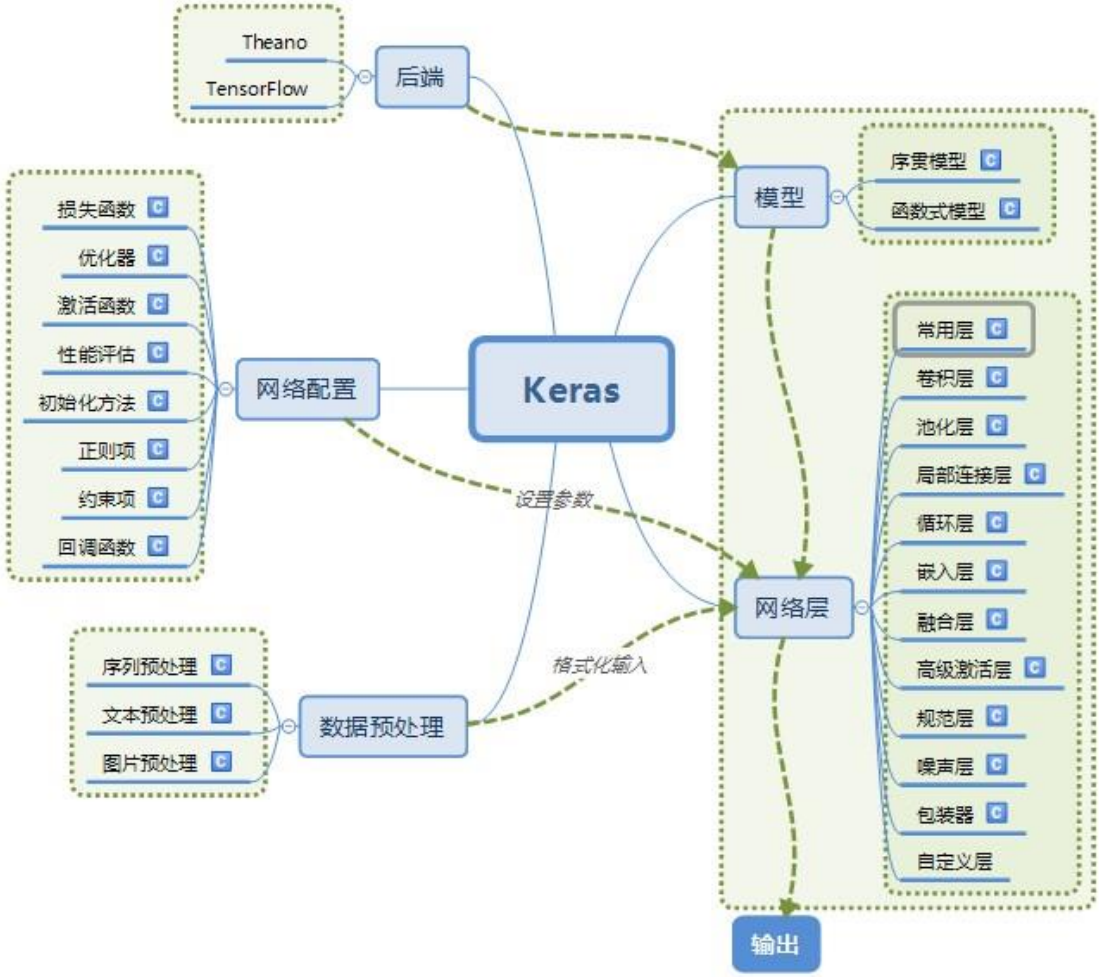
3.2 fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, validation_split=0.0)

參數說明：
x：輸入資料。
y：標籤。
batch_size：梯度下降時每個 batch 包含的樣本數。
epochs：整數，所有樣本的訓練次數。
verbose：日誌顯示，0 為不顯示，1 為顯示進度條記錄，2 為每個 epochs 輸出一行記錄。
validation_split：0-1 的浮點數，切割輸入資料的一定比例作為驗證集。

```
model.compile(optimizer='SGD', loss='mse', metrics=['accuracy'])
model.fit(x, y, batch_size=32, epochs=3, validation_split=0.3)
```

圖 5：優化和訓練實現

最後用以下圖片總結 keras 的模組，下一篇文章我們將會使用 keras 來進行項目實踐，從而更好的體會 Keras 的魅力。


Keras
后端
Theano
TensorFlow
模型
序贯模型
函数式模型
网络配置
损失函数
优化器
激活函数
性能评估
初始化方法
正则项
约束项
回调函数
设置参数
网络层
常用层
卷积层
池化层
局部连接层
循环层
嵌入层
融合层
高级激活层
规范层
噪声层
包装器
自定义层
数据预处理
序列预处理
文本预处理
图片预处理
格式化输入
输出

2. Keras 深度學習：CNN 講解及實踐

現今最主流的處理圖像資料的技術當屬深度神經網路了，尤其是卷積神經網路 CNN 尤為出名。本文將通過講解 CNN 的介紹以及使用 keras 搭建 CNN 常用模型 LeNet-5 實現對 MNist 資料集分類，從而使得讀者更好的理解 CNN。

1. CNN 的介紹

CNN 是一種自動化提取特徵的機器學習模型。首先我們介紹 CNN 所用到一些基本結構單元：

1.1 卷積層：在卷積層中，有一個重要的概念：權值共用。我們通過卷積核與輸入進行卷積運算。通過下圖可以理解如何進行卷積運算。卷積核從左到右對輸入進行掃描，每次滑動 1 格（步長為 1），下圖為滑動一次後，卷積核每個元素和輸入中綠色框相應位置的元素相乘後累加，得到輸出中綠色框中的 0。一般會使用多個卷積核對輸入資料進行卷積，得到多個特徵圖。

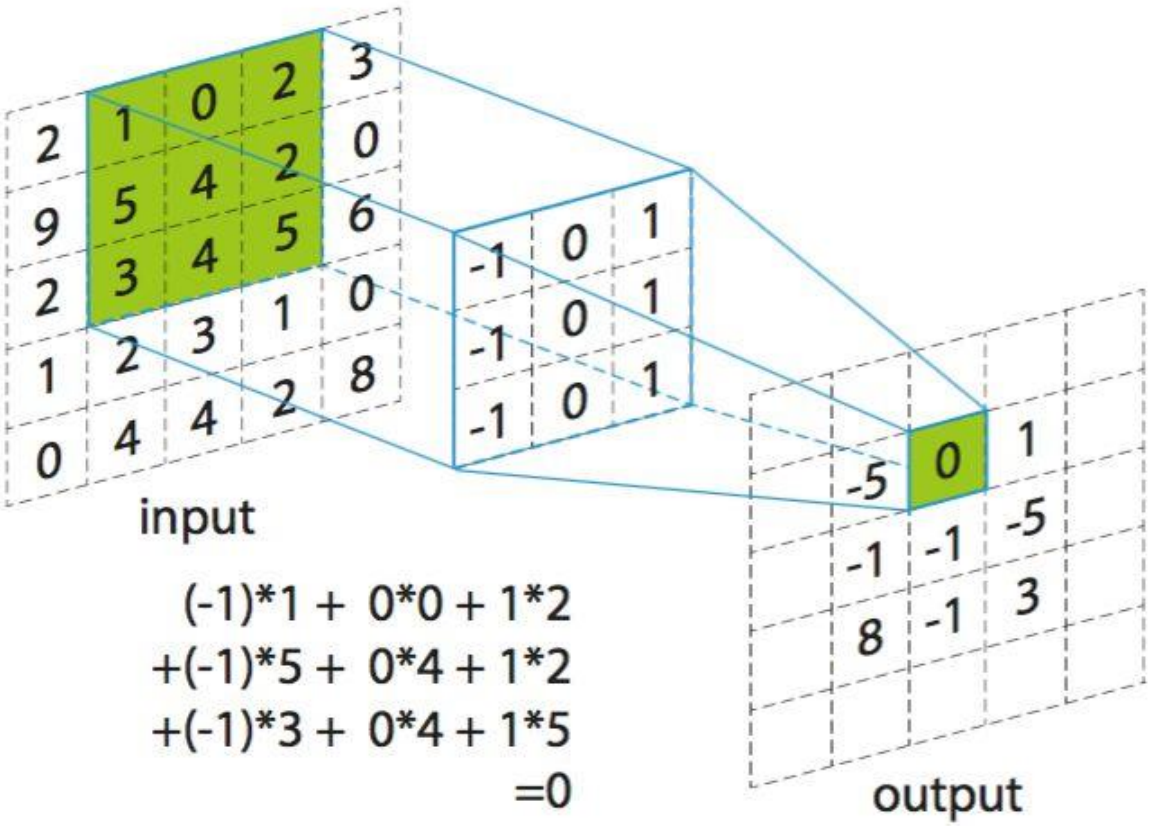


圖 1：卷積運算

1.2 啟動層：對卷積層的輸出進行一個非線性映射，因為卷積計算是一種線性計算。常見的啟動函數有 relu、tanh、sigmoid 等，一般使用 relu。

1.2.1 引入非線性啟動函數的原因：

如果不引入非線性映射的話，無論有多少層神經網路，輸出都是輸入的線性組合，這與一層隱藏層的效果相當。

1.2.2 一般使用 relu 的原因：

在反向傳導計算梯度中，使用 relu 求解明顯會比 tanh 和 sigmoid 簡單，可以**減少計算量**。

同時，**使用 tanh 和 sigmoid，當層數較多時容易導致梯度消失**，因為 tanh 和 sigmoid 的導數均小於 1（可參考啟動函數的導數公式），當我們神經網路有多層的時候，每層都要乘以這個小於 1 的導數，就有可能接近於 0，這就是所謂的梯度消失。而使用 relu 求解，若輸出不為 0 時，導數均為 1，可以有效避免梯度消失問題。

另外，relu 還會將小於 0 的映射為 0，使得網路較為稀疏，減少神經元之間的依賴，**避免過擬合**。

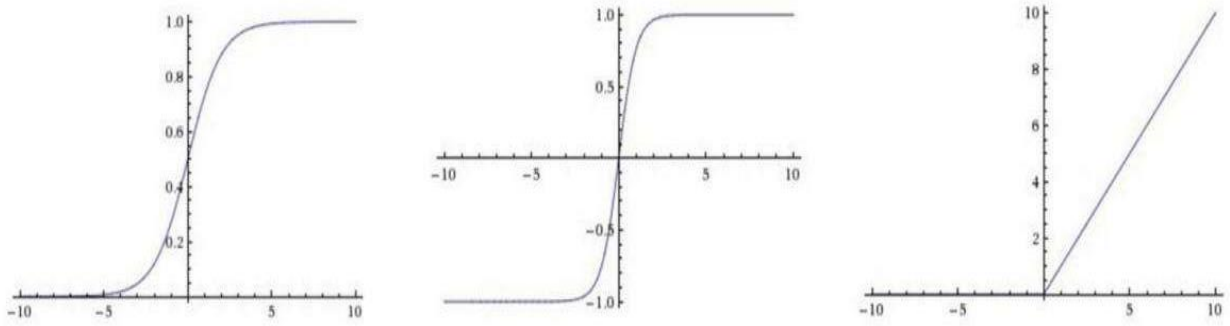
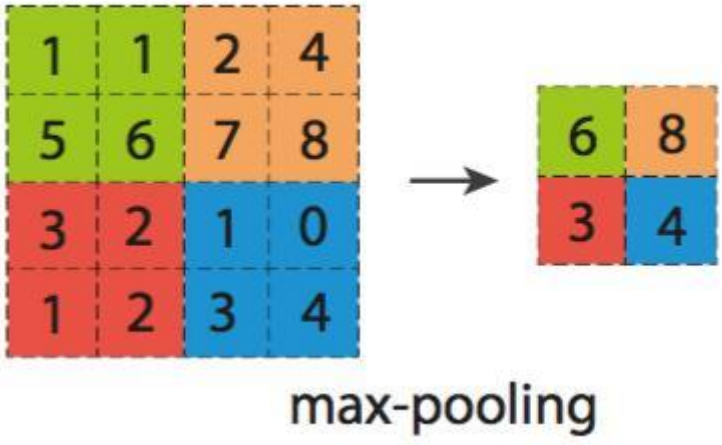


圖 2：從左到右依次為 sigmoid、tanh、relu 啟動函數

池化層：池化的目的就是減少特徵圖的維度，減少資料的運算量。池化層是在卷積層之後，對卷積的輸出，進行池化運算。池化運算，一般有兩種 MaxPooling 和 MeanPooling。選取一個池化視窗（一般為 2*2），然後從左往右進行掃描，步長一般為 2。如下圖 MaxPooling 操作，選取池化視窗中最大值作為該位置的輸出。如：左邊綠色方框中四個特徵值中，選取最大的 6 作

為輸出相應位置的特徵值。而 MeanPooling 則是對於池化視窗中的特徵值求平均。



全連接層：主要是對特徵進行重新的擬合，減少特徵資訊的丟失。通過卷積池化操作後得到的是多個特徵矩陣，而全連接層的輸入為向量，所以在進行全連接層之前，要將多個特徵矩陣"壓平"為一個向量。

對於 CNN 的卷積、池化操作，其實很多文章都會詳細的介紹，但卷積和池化的意義是什麼，很多文章都沒有明確給出解釋。可能會有人認為卷積和池化可以很大程度的減少權重參數，但只是因為這個原因嗎？顯然不是的，接下來將講解 CNN 是如何實現有效的分類從而理解卷積和池化的意義。

用深度學習解決圖像識別問題，從直觀上講是一個從**細節到抽象**的過程。所謂細節，就是指輸入圖像的每個圖元點，甚至圖元點構成的邊也可以理解為是細節。假設我們大腦接收到一張動物圖，大腦最先反應的是該圖的點和邊。然後由點和邊抽象成各種形狀，比如三角形或者圓形等，然後再抽象成耳朵和臉等特徵。最後由這些特徵決定該圖屬於哪種動物。深度學習識別圖像也是同樣的道理。這裡關鍵的就是抽象。何為抽象呢？抽象就是把圖像中的各種零散的特徵通過某種方式匯總起來，形成新的特徵。而利用這些新的特徵可更好區分圖像類別。如剛才這個例子，點和邊就是零散的特徵，通過將邊進行匯總我們就得到了三角形或圓形等新的特徵，同理，將三角形這個特徵和一些其他零散的特徵匯總成耳朵這個新特徵。顯而易見，耳朵這個新特徵會比三角形特徵更利於識別圖像。

深度學習正是通過**卷積操作實現從細節到抽象**的過程。因為**卷積的目的就是為了從輸入圖像中提取特徵，並保留圖元間的空間關係。**何以理解這句話？我們輸入的圖像其實就是一些紋理，此時，可以將卷積核的參數也理解為紋理，

我們目的是使得卷積核的紋理和圖像相應位置的紋理盡可能一致。當把圖像資料和卷積核的數值放在高維空間中，紋理等價于向量，卷積操作等價于向量的相乘，相乘的結果越大，說明兩個向量方向越近，也即卷積核的紋理就更貼近於圖像的紋理。因此，卷積後的新圖像在具有卷積核紋理的區域信號會更強，其他區域則會較弱。這樣，就可以實現從細節（圖元點）抽象成更好區分的新特徵（紋理）。每一層的卷積都會得到比上一次卷積更易區分的新特徵。

而池化目的主要就是為了減少權重參數，但為什麼可以以 Maxpooling 或者 MeanPooling 代表這個區域的特徵呢？這樣不會有可能損失了一些重要特徵嗎？這是因為**圖像資料在連續區域具有相關性，一般局部區域的圖元值差別不大**。比如眼睛的局部區域的圖元點的值差別並不大，故我們使用 Maxpooling 或者 MeanPooling 並不會損失很多特徵。

2. 專案實例

2.1 模型介紹

有了上文對 CNN 的講解後，讀者對 CNN 應該有了一定的理解，接下來我們將基於此搭建 CNN 常見模型 LeNet-5 模型，並對 Mnist 資料集進行預測。

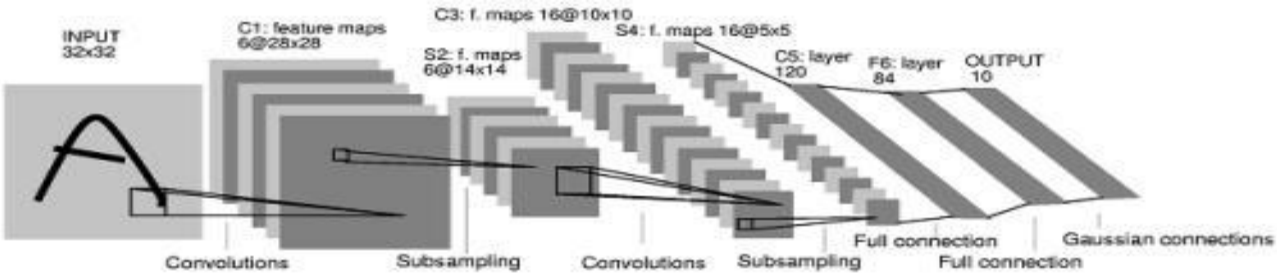


圖 3：LeNet-5 模型

從上圖 LeNet-5 模型中，可以瞭解到該模型由以下結構組成：

第一層：卷積層

這一層的輸入的原始的圖像圖元，該模型接受的圖像為 32321，6 個 5*5 卷積核，步長為 1，不使用全 0 填充。所以這層輸出的尺寸為 32-5+1=28，深度為 6。

第二層：池化層

該層的輸入為第一層的輸出，是一個 28286 的節點矩陣。*本層採用的篩檢程式大小為 22，長和寬的步長均為 2，所以本層的輸出矩陣大小為 1414*6*。

第三層：卷積層

本層的輸入矩陣大小為 14146，*16 個 55 卷積核，同樣不使用全 0 填充，步長為 1，則本層的輸出為 1010*16*。

第四層：池化層

*該層使用 22 的篩檢程式，步長為 2，故本層的輸出矩陣為 55*16*。

第五層：全連接層

如上文所說，在全連接層之前，需要將 5516 的矩陣"壓扁"為一個向量。本層的輸出節點個數為 120。

第六層：全連接層

該層輸出節點個數為 84。

第七層：全連接層

最後一層，輸出節點個數為 10，樣本的標籤個數。

2.2 代碼實現

2.2.1 資料導入及處理

Mnist 資料集為手寫字體，訓練集有 60000 張圖片，測試集中有 10000 張圖片，標籤為 0-9。由於 Mnist 資料集為 IDX 檔案格式，是一種用來存儲向量與多維度矩陣的檔案格式，不能直接讀取。有兩種方式可以進行讀取。第一種是 Keras.datasets 庫中有 mnist 資料集，直接調用即可，但是由於需要 Keras 指定位元址下載資料集，速度較慢，最好先下載；第二種是使用 struct 庫函數解析資料集，比較麻煩，但是也可以試試。

```
(x_train, y_train), (x_test, y_test) = mnist.load_data() # path参数可以指定在本地加载mnist
```

圖 4：導入 Mnist 資料集

對於 mnist 資料集只是做了一些簡單的預處理，將輸入資料的資料類型轉換為 float32，並進行歸一化。對標籤進行獨熱編碼，因為最後輸出節點個數為 10，而標籤只有 1 維。

```
x_train = x_train.astype('float32')
y_train = y_train.astype('float32')
x_test = x_test.astype('float32')
y_test = y_test.astype('float32')
```

```
# 归一化
x_train /= 255
x_test /= 255
```

```
# 独热编码
from keras.utils import np_utils
y_train_new = np_utils.to_categorical(num_classes=10, y=y_train)
y_test_new = np_utils.to_categorical(num_classes=10, y=y_test)
```

圖 5：數據預處理

2.2.2 LeNet-5 模型的搭建

```
def LeNet5():
    model = Sequential()
    # 模型第一层要指定input_shape，即输入的维度，输入和上述模型有点出入，但只需设定正确的输入维度即可
    model.add(Conv2D(filters=6, kernel_size=(5,5), padding='valid', activation='tanh', input_shape=(1,28,28)))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Conv2D(filters=16, kernel_size=(5,5), padding='valid', activation='tanh'))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Flatten())
    model.add(Dense(120, activation='tanh'))
    model.add(Dense(84, activation='tanh'))
    # softmax激活函数是用于计算该输入图片属于0-9数字的概率
    model.add(Dense(10, activation='softmax'))
    return model
```

圖 6： Keras 搭建 LeNet-5 模型

2.2.3 訓練模型

```
def train_model():
    model = LeNet5()
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    model.fit(x_train, y_train,
              batch_size=64,
              epochs=20,
              verbose=1,
              validation_split=0.2,
              shuffle=True)
    return model
```

圖 7：訓練模型

2.2.4 評估模型

```
#返回测试集损失函数值和准确率
loss, accuracy = model.evaluate(x_test, y_test_new)
```

```
10000/10000 [==============================] - 4s 380us/step
```

圖 8：評估模型

最終在測試集的準確率可以達到 **99.7%**。

通過一個簡單項目的實現，既可以幫助我們進一步瞭解 CNN，又可以熟悉 Keras 應用。最終模型還可以保存到本地，便於下次使用。

```
# 保存和读取模型
model.save('lenet5.h5')
model_lenet5 = model.load('lenet5.h5')
```

圖 9：保存和讀取模型

3．遷移學習

遷移學習就是把**已訓練好的模型參數遷移到新模型來說明新模型訓練**。考慮到大部分數據或任務存在相關性的，所以通過遷移學習我們可以將已經學到的模型參數通過某種方式來分享給模型從而加快訓練模型。

keras.applications 庫中有許多已經訓練好的模型，我們可以對已有的模型進行一些修改得到我們想要的模型，從而**提高模型搭建和訓練的效率**。另外，當我們的資料不足的時候，使用遷移學習思想也是一個很好的想法。

在下圖，將簡單的通過遷移學習實現 VGG16。但是由於 VGG16 模型要求輸入為 RGB 圖像，所以需要使用 opencv 模組對圖像進行處理。

```
# 使用迁移学习思想，利用keras中的vgg16模型
#inclue_top参数表示去掉模型顶层，weights指定使用imagenet训练出来的参数集
model_vgg = VGG16(include_top=False, weights='imagenet', input_shape=(48,48,3))
# 若不想再训练权重可以将vgg16每层的权重训练设置为False
for layer in model_vgg.layers:
    layer.trainable = False
model = Flatten()(model_vgg.output)
model = Dense(4096, activation='relu')(model)
model = Dense(4096, activation='relu')(model)
model = Dropout(0.5)(model)
model = Dense(10, activation='softmax')(model)
model_agg_mnist = Model(model_vgg.input, model)
```

圖 10：通過遷移學習高效搭建 vgg16 模型

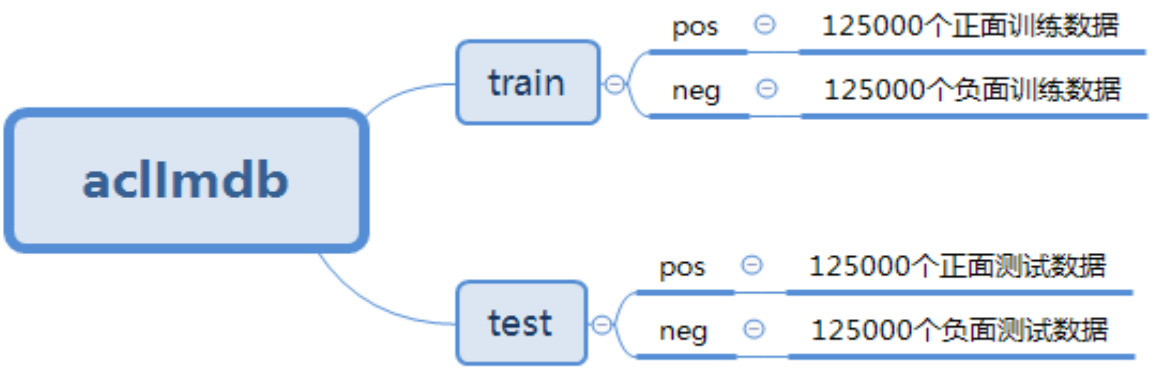
通過上圖，可以看出通過遷移學習我們可以省去搭建多個卷積和池化層，並且可以省去訓練參數的時間，vgg16 有 3364 萬個網路權重，如果全部重新訓練將需要一段較長的時間。是否重新訓練網路權重參數，要取決於**我們要所用的資料集的分佈與原模型所使用的資料集的分佈是否具有相關性**。因為模型訓練是讓模型學習資料的分佈，如果不具有相關性，已有的網路權重並不適合於我們的資料集。

3. Keras 深度學習：使用 text-CNN 處理自然語言（上）

上一篇文章中一直圍繞著 CNN 處理圖像資料進行講解，而 CNN 除了處理圖像資料之外，還適用于文本分類。CNN 模型首次使用在文本分類，是 Yoon Kim 發表的"Convolutional Neural Networks for Sentence Classification"論文中。在講解 text-CNN 之前，先介紹自然語言處理和 Keras 對自然語言的預處理。

自然語言處理就是通過對文本進行分析，從文本中提取關鍵字來讓電腦處理或理解自然語言，完成一些有用的應用，如：情感分析，問答系統等。比如在情感分析中，其本質就是根據已知的文字和情感符號（如評論等）推測這段文字是正面還是負面的。想像一下，如果我們能夠更加精確地進行情感分析，可以大大提升人們對於事物的理解效率。比如不少基金公司利用人們對於某家公司的看法態度來預測未來股票的漲跌。

接下來將使用 imdb 影評資料集簡單介紹 Keras 如何預處理文本資料。該資料集在這裡下載。由於下載得的是 tar.gz 壓縮檔，可以使用 python 的 tarfile 模組解壓。解壓後的目錄為：



1. 讀取 imdb 資料集

我們通過以下函數分別讀取 train 和 test 中的所有影評

```
def read_files(filetype):
    """
    filetype: 'train' or 'test'
    return:
    all_texts: filetype数据集文本
    all_labels: filetype数据集标签
    """
    # 标签1表示正面，0表示负面
    all_labels = [1]*12500 + [0]*12500
    all_texts = []
    file_list = []
    path = r'./aclImdb/'
    # 读取正面文本名
    pos_path = path + filetype + '/pos/'
    for file in os.listdir(pos_path):
        file_list.append(pos_path+file)
    # 读取负面文本名
    neg_path = path + filetype + '/neg/'
    for file in os.listdir(neg_path):
        file_list.append(neg_path+file)
    # 将所有文本内容加到all_texts
    for file_name in file_list:
        with open(file_name, encoding='utf-8') as f:
            all_texts.append(rm_tags(" ".join(f.readlines())))
    return all_texts, all_labels
```

得到的影評如下圖，每條影評用雙引號包住。

```
'Bromwell High is a cartoon comedy. It ran at the same time as some other programs about school life, such as
"Teachers". My 35 years in the teaching profession lead me to believe that Bromwell High\'s satire is much clo
ser to reality than is "Teachers". The scramble to survive financially, the insightful students who can see ri
ght through their pathetic teachers\' pomp, the pettiness of the whole situation, all remind me of the schools
I knew and their students. When I saw the episode in which a student repeatedly tried to burn down the school,
I immediately recalled ..... at ..... High. A classic line: INSPECTOR: I\'m here to sack one of your
teachers. STUDENT: Welcome to Bromwell High. I expect that many adults of my age think that Bromwell High is f
ar fetched. What a pity that it isn\'t!'
```

2. 使用 Tokenizer 將影評文字轉換成數位特徵

在上文中已經得到了每條影評文字了，但是 text-CNN 的輸入應該是數字矩陣。可以使用 Keras 的 Tokenizer 模組實現轉換。

簡單講解 Tokenizer 如何實現轉換。當我們創建了一個 Tokenizer 物件後，使用該物件的 fit_on_texts()函數，可以將輸入的文本中的每個詞編號，編號是根據詞頻的，詞頻越大，編號越小。可能這時會有疑問：Tokenizer 是

如何判斷文本的一個詞呢？其實它是以空格去識別每個詞。因為英文的詞與詞之間是以空格分隔，所以我們可以直接將文本作為函數的參數，但是當我們處理中文文本時，我們需要使用分詞工具將詞與詞分開，並且詞間使用空格分開。具體實現如下：

```
# 英文文本不用切词，因为默认会将空格将词切开，所以如果是中文文本的话，就需要切词，且每个词用空格隔开
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=2000)  # 建立一个2000个单词的字典
tokenizer.fit_on_texts(train_texts)
```

使用 word_index 屬性可以看到每次詞對應的編碼，可以發現類似"the"、"a"等詞的詞頻很高，但是這些詞並不能表達文本的主題，我們稱之為停用詞。對文本預處理的過程中，我們希望能夠盡可能提取到更多關鍵字去表達這句話或文本的中心思想，因此我們可以將這些停用詞去掉後再編碼。網上有許多歸納好的停用詞，大家可以下載了之後，去除該文本中的停用詞。

```
print(tokenizer.word_index)  # 得到每个词的编号
```

```
{'the': 1, 'and': 2, 'a': 3, 'of': 4, 'to': 5, 'is': 6, 'in': 7, 'it': 8, 'i': 9, 'this': 10, 'that': 11, 'was':
12, 'as': 13, 'for': 14, 'with': 15, 'movie': 16, 'but': 17, 'film': 18, 'on': 19, 'not': 20, 'you': 21, 'are':
22, 'his': 23, 'have': 24, 'be': 25, 'he': 26, 'one': 27, 'all': 28, 'at': 29, 'by': 30, 'an': 31, 'they': 32,
'who': 33, 'so': 34, 'from': 35, 'like': 36, 'her': 37, 'or': 38, 'just': 39, 'about': 40, "it's": 41, 'out': 4
2, 'has': 43, 'if': 44, 'some': 45, 'there': 46, 'what': 47, 'good': 48, 'more': 49, 'when': 50, 'very': 51, 'u
p': 52, 'no': 53, 'time': 54, 'she': 55, 'even': 56, 'my': 57, 'would': 58, 'which': 59, 'only': 60, 'story': 6
1, 'really': 62, 'see': 63, 'their': 64, 'had': 65, 'can': 66, 'were': 67, 'me': 68, 'well': 69, 'than': 70, 'w
e': 71, 'much': 72, 'been': 73, 'get': 74, 'bad': 75, 'will': 76, 'also': 77, 'do': 78, 'into': 79, 'people': 8
```

對每個詞編碼之後，每句影評中的每個詞就可以用對應的編碼表示，即每條影評已經轉變成一個向量了：

```
# 对每一句影评文字转换为数字列表，使用每个词的编号进行编号
x_train_seq = tokenizer.texts_to_sequences(train_texts)
x_test_seq = tokenizer.texts_to_sequences(test_texts)
```

3.讓每句數位影評長度相同

對每個詞編碼之後，每句影評中的每個詞就可以用對應的編碼表示，即每條影評已經轉變成一個向量。但是，由於影評的長度不唯一，需要將每條影評的長度設置一個固定值。

```
from keras.preprocessing import sequence
x_train = sequence.pad_sequences(x_train_seq, maxlen=150)
x_test = sequence.pad_sequences(x_test_seq, maxlen=150)
```

每個句子的長度都固定為 150，如果長度大於 150，則將超過的部分截掉；如果小於 150，則在最前面用 0 填充。每個句子如下：

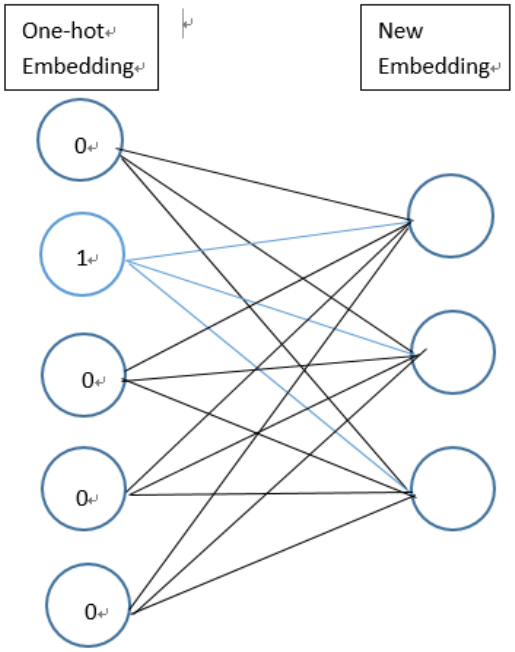
```
: x_train[0]

: array([   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
         308,    6,    3, 1068,  208,    8,   29,    1,  168,   54,   13,
          45,   81,   40,  391,  109,  137,   13,   57,  149,    7,    1,
         481,   68,    5,  260,   11,    6,   72,    5,  631,   70,    6,
           1,    5,    1, 1530,   33,   66,   63,  204,  139,   64, 1229,
           1,    4,    1,  222,  899,   28,   68,    4,    1,    9,  693,
           2,   64, 1530,   50,    9,  215,    1,  386,    7,   59,    3,
        1470,  798,    5,  176,    1,  391,    9, 1235,   29,  308,    3,
         352,  343,  142,  129,    5,   27,    4,  125, 1470,    5,  308,
           9,  532,   11,  107, 1466,    4,   57,  554,  100,   11,  308,
           6,  226,   47,    3,   11,    8,  214])
```

4. 使用 Embedding 層將每個詞編碼轉換為詞向量

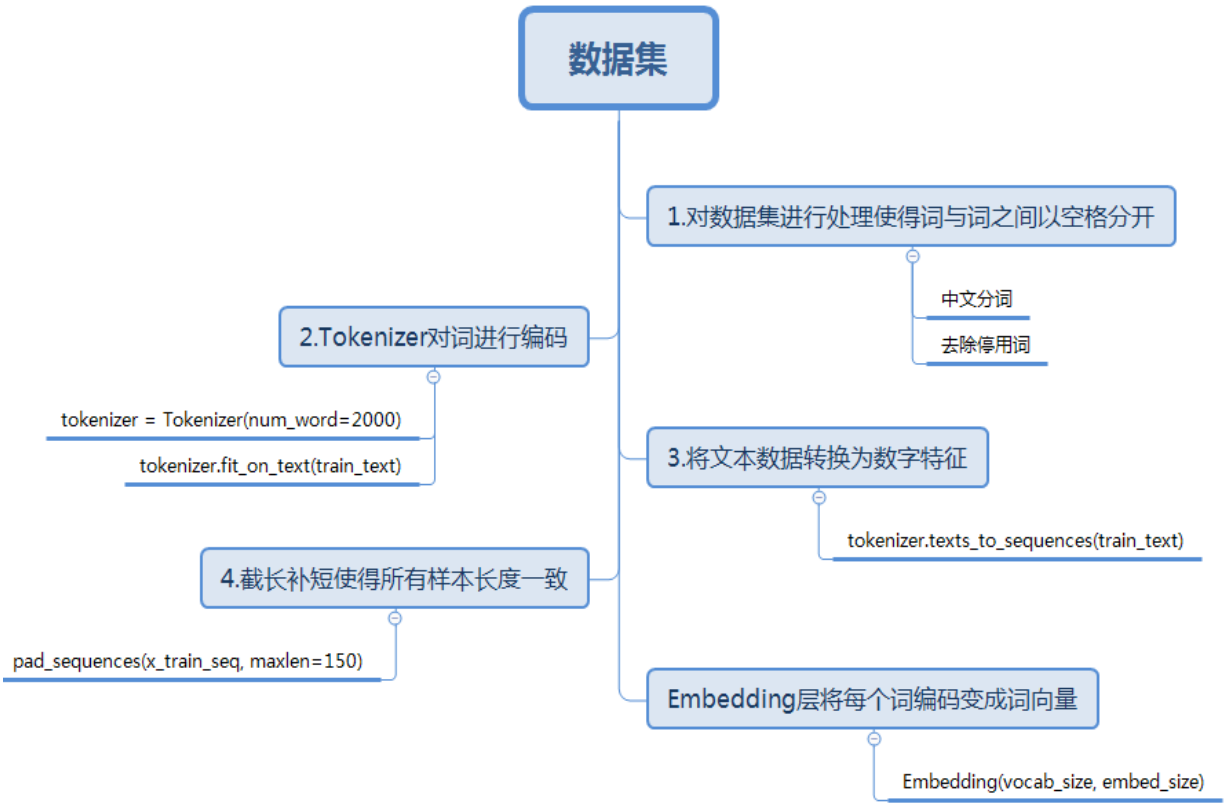
通過以上操作，已經將每個句子變成一個向量，但上文已經提及 text-CNN 的輸入是一個數字矩陣，即每個影評樣本應該是以一個矩陣，每一行代表一個詞，因此，需要將詞編碼轉換成詞向量。

使用 Keras 的 Embedding 層可以實現轉換。Embedding 層基於上文所得的詞編碼，對每個詞進行 one-hot 編碼，每個詞都會以一個 vocabulary_size(如上文的 2000)維的向量；然後通過神經網路的訓練反覆運算更新得到一個合適的權重矩陣（具體實現過程可以參考 skip-gram 模型），行大小為 vocabulary_size，列大小為詞向量的維度，將本來以 one-hot 編碼的詞向量映射到低維空間，得到低維詞向量。比如 the 的編號為 1，則對應的詞向量為權重矩陣的第一行向量。

如下圖，藍色線對應權重值組成了該詞的詞向量。需要聲明一點的是 Embedding 層是作為模型的第一層，在訓練模型的同時，得到該語料庫的詞向量。當然，也可以使用已經預訓練好的詞向量表示現有語料庫中的詞。



至此已經將文本資料預處理完畢，將每個影評樣本轉換為一個數位矩陣，矩陣的每一行表示一個詞向量。下圖梳理了處理文本資料的一般步驟。在此基礎上，可以針對相應資料集的特點對資料集進行特定的處理。比如：在該資料集中影評可能含有一些 html 標籤，我們可以使用規則運算式將這些標籤去除。



下一篇文章，我們將介紹 text-CNN 模型，利用該模型對 imdb 資料集進行情感分析，並在文末給出整個專案的完整代碼連結。

4. Keras 深度學習：使用 text-CNN 處理自然語言（下）

在上一篇文章中，已經介紹了 Keras 對文本資料進行預處理的一般步驟。預處理完之後，就可以使用深度學習中的一些模型進行文本分類。在這篇文章中，將介紹 text-CNN 模型以及使用該模型對 imdb 影評資料集進行情感分析。

正如上篇文章所說，文本分類的關鍵在於準確提煉文檔或者句子的中心思想，而提煉中心思想的方法是抽取文檔或句子的關鍵字作為特徵，基於這些特徵去訓練分類器並分類。每個類別可以理解為一種中心思想，如情感分析中，分類器將樣本分為兩類，一類為正面評論，另一類為負面評論，而正面和負面評論正是該文本或句子的中心思想。對於思維敏銳的讀者來說，當說到提取特徵的時候，可能就能想到為什麼卷積神經網路可以很好進行自然語言處理。沒錯，就是因為卷積和池化過程就是一個抽取特徵的過程，當我們可以準確抽取關鍵字的特徵時，就能準確的提煉出文檔或句子的中心思想。

卷積神經網路首次應用于文本分類可以說是在 2004 年 Yoon Kim 在 “Convolutional Neural Networks for Sentence Classification” 一文中提出（雖然第一個用的並不是他，但是在這篇文章中提出了 4 種 Model Variations，並有詳細的調參），本文也是基於對這篇文章的理解。接下來將介紹 text-CNN 模型，並使用 Keras 搭建該模型對 imdb 資料集進行情感分析。

text-CNN 模型

由於上篇文章已經將 Embedding 層講過了，在這裡就不再敘述。主要講解卷積層、池化層和全連接層。

1.卷積層

在處理圖像資料時，CNN 使用的卷積核的寬度和高度的一樣的，但是在 text-CNN 中，**卷積核的寬度是與詞向量的維度一致！！！**這是因為我們輸入的每一行向量代表一個詞，在抽取特徵的過程中，**詞做為文本的最小細微性**，如果我們使用卷積核的寬度小於詞向量的維度就已經不是以詞作為最小細微性了。而高度和 CNN 一樣，可以自行設置（通常取值 2,3,4,5）。由於我們的輸入是一個句子，句子中相鄰的詞之間關聯性很高，因此，當**我們用卷積核進行卷積時，不僅考慮了詞義而且考慮了詞序及其上下文**。（類似於 skip-gram 和 CBOW 模型的思想）。

詳細講解卷積的過程：卷積層輸入的是一個表示句子的矩陣，維度為 n*d，即每句話共有 n 個詞，每個詞有一個 d 維的詞向量表示。假設 $X_{i:i+j}$ 表示 X_i 到 X_{i+j} 個詞，使用一個寬度為 d，高度為 h 的卷積核 W 與 $X_{i:i+h-1}$(h 個詞)進行卷積操作後再使用啟動函數啟動得到相應的特徵 c_i，則卷積操作可以表示為：（使用點乘來表示卷積操作）

$$c_i = f(W .* X_{i:i+h-1} + b) \qquad \text{其中 f 为激活函数}$$

因此經過卷積操作之後，可以得到一個 n-h+1 維的向量 c 形如：

$$c = [c_1, c_2, ..., c_{n-h+1}]$$

以上是一個卷積核與輸入句子的卷積操作，同樣的，**我們也可以使用更多高度不同的卷積核，且每個高度的卷積核多個，得到更多不同特徵**。

2.池化層

因為在卷積層過程中我們使用了不同高度的卷積核，使得我們通過卷積層後得到的向量維度會不一致，所以在池化層中，我們使用 1-Max-pooling 對每個特徵向量池化成一個值，即抽取每個特徵向量的最大值表示該特徵，而且認為**這個最大值表示的是最重要的特徵**。當我們對所有特徵向量進行 1-Max-Pooling 之後，還**需要將每個值給拼接起來**。得到池化層最終的特徵向量。**在池化層到全連接層之前可以加上 dropout 防止過擬合。**

3.全連接層

全連接層跟其他模型一樣，假設有兩層全連接層，第一層可以加上'relu'作為啟動函數，第二層則使用 softmax 啟動函數得到屬於每個類的概率。如果處理的資料集為二分類問題，如情感分析的正負面時，第二層也可以使用 sigmoid 作為啟動函數，然後損失函數使用對數損失函數'binary_crossentropy'。

4.text-CNN 的小變種

在詞向量構造方面可以有以下不同的方式：

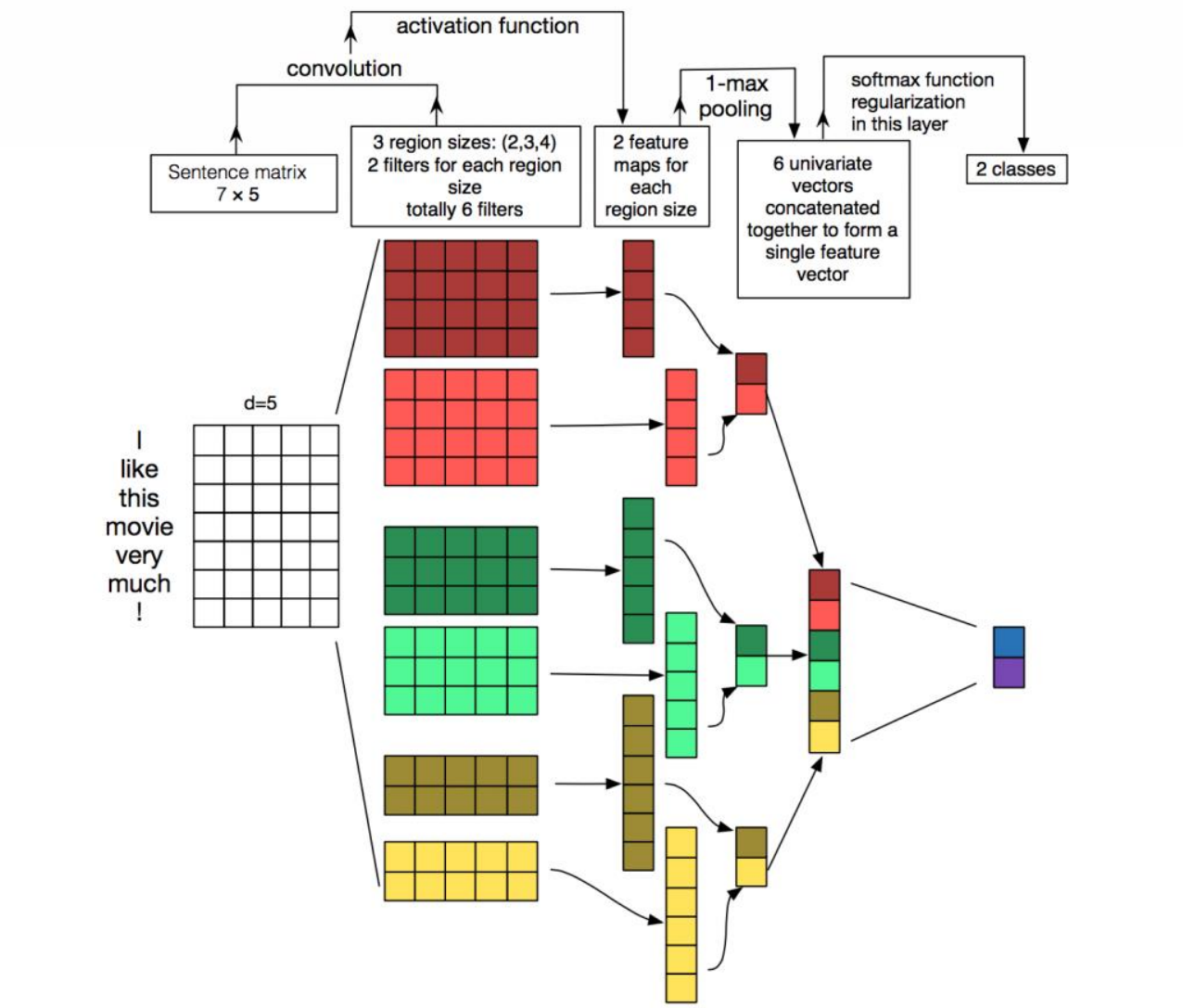
CNN-rand: 隨機初始化每個單詞的詞向量通過後續的訓練去調整。

CNN-static: 使用預先訓練好的詞向量，如 word2vec 訓練出來的詞向量，在訓練過程中不再調整該詞向量。

CNN-non-static: 使用預先訓練好的詞向量，並在訓練過程進一步進行調整。

CNN-multichannel: 將 static 與 non-static 作為兩通道的詞向量。

使用網上的一張經典圖進一步講解 text-CNN



在上圖中，輸入了一句話"I like this movie very much!"，其對應的句子矩陣維度為 7*5，每個詞用維度為 5 的詞向量表示。在卷積層中，分別使用高度為 4,3,2 的卷積核，且每種卷積核有 2 個。卷積之後得到 6 個對應的特徵向

量，維度從上往下分別為 4,4,5,5,6,6，然後對每個向量進行 1-Max-pooling，再拼接起來一個維度為 6 的特徵向量。最後通過全連接層，啟動函數為 softmax 得到 2 個類別的概率。

使用 text-CNN 模型對 imdb 資料集進行情感分析

從上文對 text-cnn 模型的介紹，想必讀者對該模型已經有了初步的理解了。趁熱打鐵，我們將利用 Keras 搭建該模型並對 imdb 資料集進行情感分析。由於資料集預處理部分上一篇文章已經講解，在此將不再敘述。在搭建模型之前，先講解用到的一些主要函數：

卷積過程由於只是沿著高度方向進行卷積，即只在一個維度卷積所以使用 Conv1d。

Conv1d(filters, kernel_size, activation):

filters: 卷積核的個數

kernel_size: 卷積核的寬度

activation: 卷積層使用的啟動函數

池化過程使用的在一個維度上的池化，使用 MaxPooling1D

MaxPooling1D(pool_size):

pool_size: 池化視窗的大小，由於我們要將一個卷積核得到特徵向量池化為 1 個值，所以池化視窗可以設為(句子長度-卷積核寬度+1)

池化過程最後還需要對每個值拼接起來，可以使用 concatenate 函數實現。

concatenate(inputs, axis):

inputs: inputs 為一個 tensor 的 list，所以需要將得到 1-MaxPooling 得到每個值 append 到 list 中，並把該 list 作為 inputs 參數的輸入。

axis: 指定拼接的方向。

瞭解了這幾個函數的用法之後，我們就可以搭建我們的模型如下圖。

```
def text_cnn(maxlen = 150, max_features = 2000, embed_size = 32):
    #Inputs
    comment_seq = Input(shape=[maxlen], name='x_seq')

    #Embeddings layers
    emb_comment =Embedding(max_features, embed_size)(comment_seq)

    # conv layers
    convs = []
    filter_sizes = [2,3,4,5]
    for fsz in filter_sizes:
        l_conv = Conv1D(filters=100,kernel_size=fsz,activation='tanh')(emb_comment)
        l_pool = MaxPooling1D(maxlen-fsz+1)(l_conv)
        l_pool = Flatten()(l_pool)
        convs.append(l_pool)
    merge =concatenate(convs,axis=1)

    out = Dropout(0.5)(merge)
    output  = Dense(32,activation='relu')(out)

    output = Dense(units=1,activation='sigmoid')(output)

    model = Model([comment_seq],output)
#     adam = optimizers.Adam(lr=0.005, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
    model.compile(loss="binary_crossentropy", optimizer="adam", metrics=['accuracy'])
    return model
```

基於上篇文章對 imdb 資料集中影評處理後，得到每個句子長度(maxlen)為 150，共有 2000 個詞(max_features)，詞向量維度為 32(embed_size)。在該模型中，使用了高度分別為 2,3,4,5 的四種卷積核，每種卷積核 100 個，最後使用 sigmoid 作為啟動函數，損失函數使用對數損失函數。

```
model = text_cnn()
batch_size = 64
epochs = 10
model.fit(x_train, y_train,
            validation_split=0.1,
            batch_size=batch_size,
            epochs=epochs,
            shuffle = True)
```

模型訓練過程 batch_size 設為 64，epochs 為 10，最終可以在驗證集可以得到 86.5%的準確率。

```
scores = model.evaluate(x_test, y_test)
```

```
25000/25000 [==============================] - 13s 515us/step
```

```
print('test_loss: %f, accuracy: %f'%(scores[0], scores[1]))
```

```
test_loss: 0.428225, accuracy: 0.865160
```

至此我們已經實現了使用 text-CNN 模型對 imdb 資料集進行情感分析，準確率還算可以，有興趣的讀者可以基於該模型進行改進，得到更高的準確率。

完整代碼可到以下連結下載：

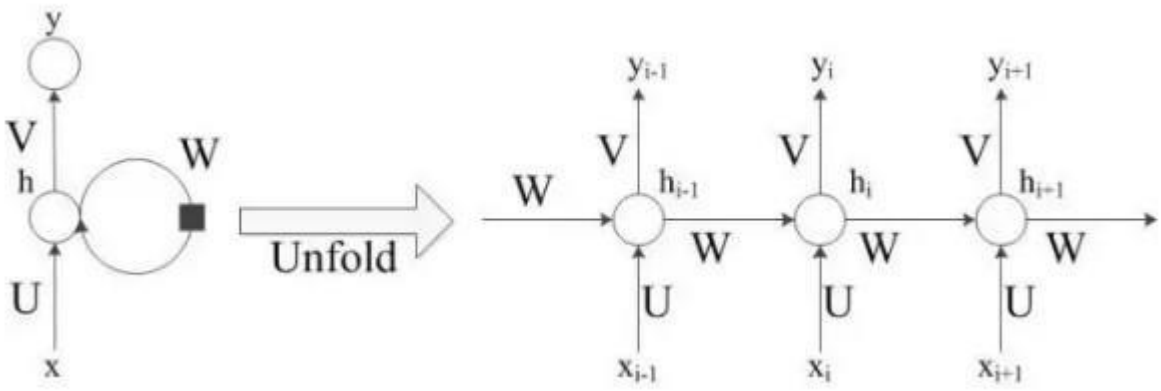
https://github.com/hongweijun811/wjgit/blob/master/text_cnn_demo.py

5. Keras 深度學習：RNN 和雙向 RNN 講解及實踐

通過對前面文章的學習，對深度神經網路(DNN)和卷積神經網路(CNN)有了一定的瞭解，也感受到了這些神經網路在各方面的應用都有不錯的效果。然而這些網路都有一個共同的特點：**每一層的神經元之間是相互獨立的**，如輸入層的神經元彼此之間是獨立的。然而，現實世界中很多元素之間都是有相互聯繫的。比如一部連續劇的內容，上一集和這一集的內容會有一定的聯繫；同樣的，一句話，如"天空很藍"，我們通過"天空"和"很"會認為接下來的詞為"藍"的概率會較高。正如這種時序資料問題，使用之前所學的模型(除了 text-CNN)可能很難做到準確的推斷，因此我們引入今天所講的迴圈神經網路(recurrent neural network)，其主要的用處就是**處理和預測序列資料**。

一、RNN 網路結構及原理講解

RNN 的網路結構如下圖：



圖中左側是未展開 RNN 模型，在模型中間有個大圓圈表示迴圈，正是這個迴圈允許資訊的持久化。但這個看起來可能不太好理解，因此將其展開為右側的模型，並做進一步詳細介紹如何實現資訊的持久化。

右圖中圓圈可以看作一個單元。定義 Xi 為第 i 時刻的輸入，hi 為第 i 時刻的記憶，yi 為第 i 時刻的輸出。

舉個例子說明 RNN 實現過程：假設有一個句子的輸入是"今天天空很"，要預測下個詞是什麼。通過分詞後可能得到三個詞作為輸入："今天"，"天空"，"很"，對應的就是上圖的 Xi-1，Xi，Xi+1，那麼輸出 yi-1 應該是"天空"，yi 應該是"很"，預測下個詞 yi+1 是什麼，根據這句話，"藍"的概率比較大。因此預測下個詞應該是"藍"。

通過上述淺顯易懂的例子讀者應該對 RNN 實現過程有個大概的瞭解，接下來將具體講解 RNN 實現的詳細過程。

輸入層到隱藏層：

從上圖的箭頭指示，讀者或許發現第 i 時刻的輸出是由上一時刻的記憶和當前時刻共同決定的。這個思想很符合對時序資料處理的思路。正如我們在看連續劇的時候如果直接看中間某一集，可能會對部分劇情不能理解，但是，當我們看過前幾集後會對劇情有所記憶，再加上該集劇情內容，我們就能更好的理解接下來劇情內容。因此用公式表示 RNN 當前時刻的記憶為：

$$h_i = f(U * X_i + W * h_{i-1})$$

其中 f()函數為啟動函數。在此處要加上啟動函數也很好理解，因為得到的資訊並不是所有的都是重要的資訊，而我們只需要記住重要的資訊即可。這個時候就可以使用啟動函數，如 tanh，去對一些不重要的資訊進行過濾，保留重要的資訊即可。

隱藏層到輸出層：

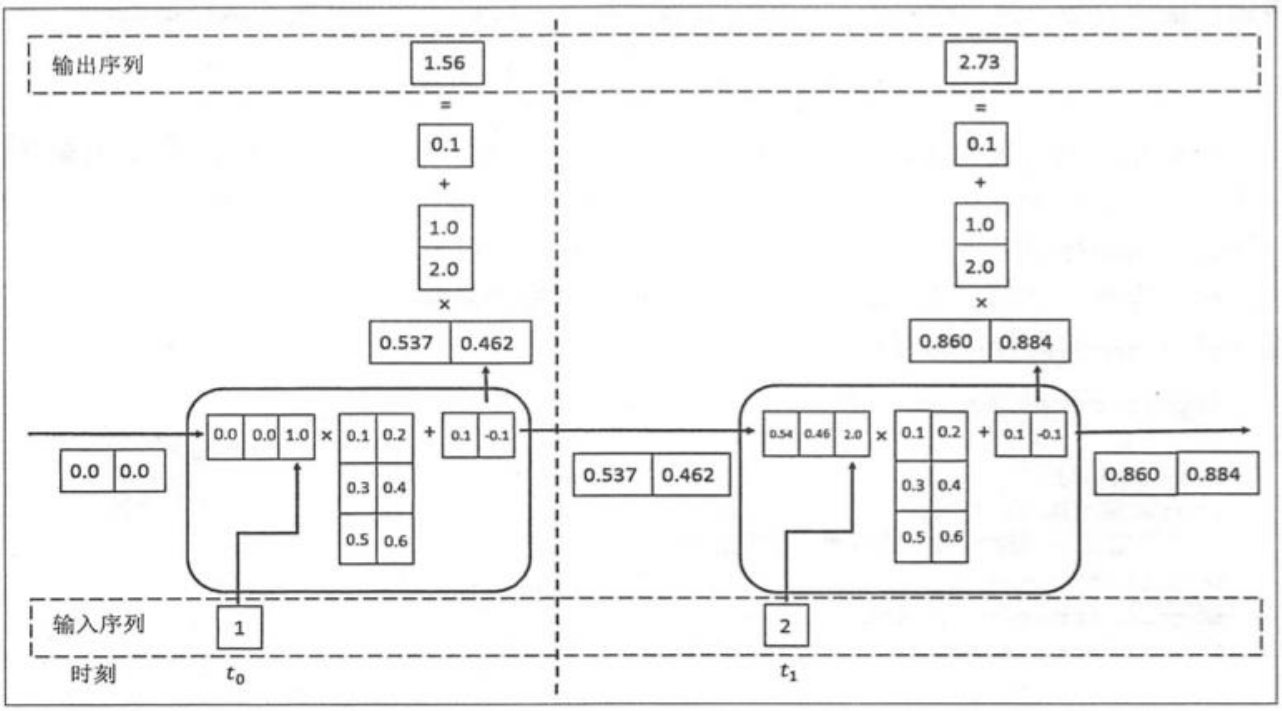
同樣使用電視劇的例子進行通俗解釋，當我們對上幾集和該集的劇情進行整理，留下一些重要資訊之後，我們會試圖去猜測下一集的內容大概會是怎麼樣的。同樣的，RNN 的思路也如此。當我們 hi 中保留了 i 時刻的重要資訊後，就試圖使用這些重要資訊進行預測下一個詞應該是什麼。用公式表示 RNN 當前時刻的輸出為：

$$y_i = softmax(V * h_i)$$

使用 softmax 函數是 RNN 希望預測每個詞出現的概率，然後概率最高的詞就是預測的下一個詞。

註：U、W、V 分別是對應的權重矩陣，通過反向傳播演算法調整相應的值使得預測的結果更加準確。與 CNN 一樣，網路中的每個單元都共用同一組(U、V、W)，可以極大的降低了計算量。

具體的前向傳播計算過程如下：

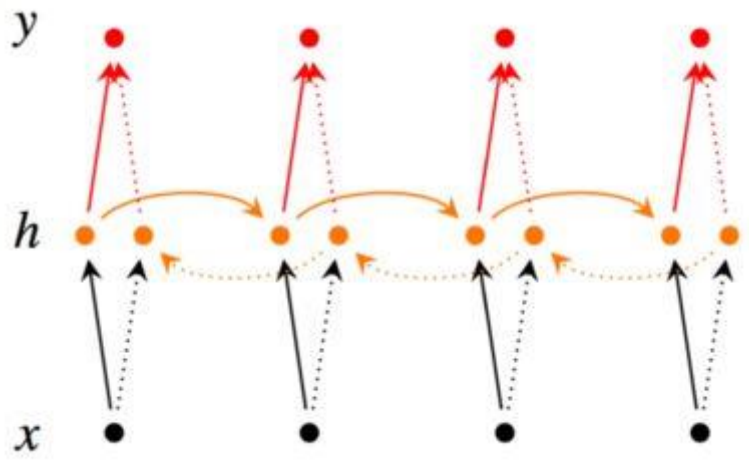


在 t1 時刻的輸入為 2，結合上一時刻的記憶(0.537,0.462)，得到(0.54,0.46,2.0)，然後與隱藏層的權重矩陣相乘得到該時刻的記憶(0.860,0.884)。通過該時刻的記憶與輸出層的權重矩陣相乘得到該時刻的預測值 2.73。這就是一個時刻 RNN 前向傳播的具體過程。

因此，通過上述思想，RNN 就能有效的處理時序資料，對每個輸入保留一些重要的資訊，理論上最後就能得到整個輸入的所有重要資訊，進而綜合考慮所有輸入去預測輸出。

二、雙向 RNN(BRNN)網路結構及原理講解

在 RNN 中只考慮了預測詞前面的詞，即只考慮了上下文中"上文"，並沒有考慮該詞後面的內容。這可能會錯過了一些重要的資訊，使得預測的內容不夠準確。正如電視劇的例子，當在該集新出現了一個人物，若要預測該人物的名字，單從前幾集的內容，並不能有效的進行預測。但如果我們看了後幾集的內容，可能就能更加有效的進行預測。雙向 RNN 也是基於這種思想，不僅從前往後(如下圖黃色實箭頭)保留該詞前面的詞的重要資訊，而且從後往前(如下圖黃色虛箭頭)去保留該詞後面的詞的重要資訊，然後基於這些重要資訊進行預測該詞。雙向 RNN 模型如下：



用公式表示雙向 RNN 過程如下：

从前往后： $h_i^1 = f(U^1 * X_i + W^1 * h_{i-1})$

从后往前： $h_i^2 = f(U^2 * X_i + W^2 * h_{i-1})$

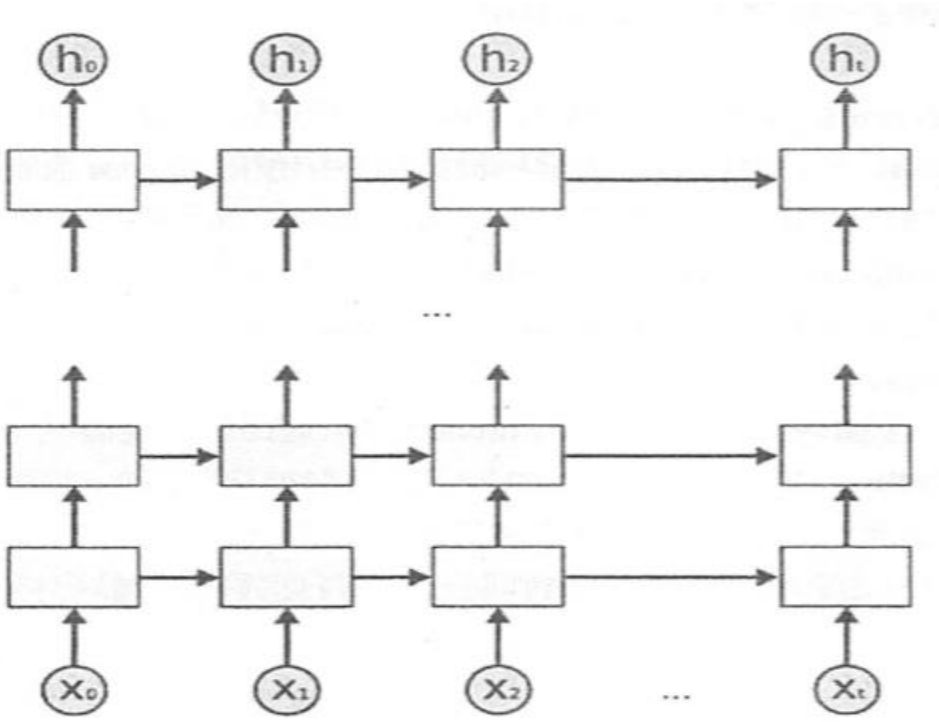
输出： $y_i = \text{softmax}(V * [h_i^1; h_i^2])$

注：因为我们得到从前往后的重要信息和从后往前的重要信息，因此我们需要将得到这两本部分结果拼接起来，表示为$[h_i^1; h_i^2]$，如：他们都是 1000*1 维的，拼接起来就是 1000*2 维。

另外，雙向 RNN 需要保存兩個方向的權重矩陣，所以需要的記憶體約為 RNN 的兩倍。

三、深層 RNN(DRNN)網路結構

深層 RNN 網路是在 RNN 模型多了幾個隱藏層，是因為考慮到當信息量太大的時候一次性保存不下所有重要資訊，通過多個隱藏層可以保存更多的重要資訊，正如我們看電視劇的時候也可能重複看同一集記住更多關鍵劇情。同樣的，我們也可以在雙向 RNN 模型基礎上加多幾層隱藏層得到深層雙向 RNN 模型。



註：每一層循環體中參數是共用的，但是不同層之間的權重矩陣是不同的。

四、Keras 對 RNN 的支持

在 Keras 同樣對 RNN 模型進行了封裝，並且調用起來十分方便，我們將會在下一節搭建 RNN 模型來呈現使用 Keras 搭建是多麼方便。

Keras 在 layers 包的 Recurrent 模組中實現了 RNN 相關層模型的支援，並在 wrapper 模型中實現雙向 RNN 包裝器。

Recurrent 模組中的 RNN 模型包括 RNN、LSTM、GRU 等模型(後兩個模型將在後面 Keras 系列文章講解)：

1.RNN：全連接 RNN 模型

SimpleRNN(units,activation='tanh',dropout=0.0,recurrent_dropout=0.0, return_sequences=False)

2.LSTM：長短記憶模型

LSTM(units,activation='tanh',dropout=0.0,recurrent_dropout=0.0,return_sequences=False)

3.GRU：門限迴圈單元

GRU(units,activation='tanh',dropout=0.0,recurrent_dropout=0.0,return_sequences=False)

4.參數說明：

units: RNN 輸出的維度

activation: 啟動函數，預設為 tanh

dropout: 0~1 之間的浮點數，控制輸入線性變換的神經元失活的比例

recurrent_dropout：0~1 之間的浮點數，控制迴圈狀態的線性變換的神經元失活比例

return_sequences: True 返回整個序列,用於 stack 兩個層，False 返回輸出序列的最後一個輸出，若模型為深層模型時設為 True

input_dim: 當使用該層為模型首層時，應指定該值

input_length: 當輸入序列的長度固定時，該參數為輸入序列的長度。當需要在該層後連接 Flatten 層，然後又要連接 Dense 層時，需要指定該參數

wrapper 模組實現雙向 RNN 模型：

1. 雙向 RNN 包裝器

Bidirectional(layer, merge_mode='concat', weights=None)

參數說明：

layer: SimpleRNN、LSTM、GRU 等模型結構，確定是哪種 RNN 的雙向模型

Merge_mode: 前向和後向 RNN 輸出的結合方式，為 sum,mul,concat,ave 和 None 之一，若為 None，則不結合，以列表形式返回，若是上文說到的拼接則為 concat

五、使用 Keras RNN、BRNN 模型、DBRNN 模型進行實踐

本次實踐同樣使用上一篇文章中使用到的 Imdb 資料集進行情感分析。對於該資料集的預處理在本篇文章中就不再介紹，若想瞭解可閱讀上一篇文章。

Keras 在實現迴圈神經網路很方便，已經將其封裝好，只需要調用相應的層就可以搭建該模型，接下來簡單的搭建上述三種模型。

RNN模型

```
def RNN(maxlen = 380, max_features = 3800, embed_size = 32):
    model = Sequential()
    model.add(Embedding(max_features, embed_size, input_length=maxlen))
    model.add(Dropout(0.5))
    model.add(SimpleRNN(16))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))
    return model
```

搭建一層的 RNN 模型，只需要在模型中加入 SImpleRNN 層，並設置該層的輸出即可，其他模型的搭建都和上篇文章中講解的一樣，相當方便。

BRNN模型

```
def BRNN(maxlen = 380, max_features = 3800, embed_size = 32):
    model = Sequential()
    model.add(Embedding(max_features, embed_size, input_length=maxlen))
    model.add(Dropout(0.5))
    model.add(Bidirectional(SimpleRNN(16, return_sequences=True), merge_mode='concat'))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model
```

BRNN 模型需要使用 wrappers 包的 Bidirecitional 模組實現雙向 RNN 模型，並且要將 return_sequences 參數設置為 True，因為如上文所述需要將前、後向的重要資訊拼接起來，所以需要將整個序列返回，而不是只返回最後一個預測詞。

並且上文提到的是將前後向的進行拼接，所以使用的是'concat'，也可以使用 sum 對前後向結果求和或者其他對結果進行相應的操作。

DBRNN

```
def DBRNN(maxlen = 380, max_features = 3800, embed_size = 32):
    model = Sequential()
    model.add(Embedding(max_features, embed_size, input_length=maxlen))
    model.add(Dropout(0.5))
    model.add(Bidirectional(SimpleRNN(16, return_sequences=True), merge_mode='concat'))
    model.add(SimpleRNN(8))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))
    return model
```

DBRNN 模型的搭建也很方便，比如在這裡我們要搭建一個兩層的 DBRNN 模型，只需要再加一層 SimpleRNN 即可。要注意的是，如果要搭建多層 DBRNN 模型，除了最後一層 SimpleRNN 外，其他的 SimpleRNN 層都需要將 return_sequences 參數設置為 True。

可能讀者會認為雖然 Keras 搭建模型很方便，但是這會導致新手對於模型的輸入輸出欠缺理解。同樣的，Keras 也考慮到了這一點，因此 Keras 中有 model.summary()的內置函數，通過這個函數就可以知道我們搭建的模型的輸入輸出和參數等資訊，便於我們理解模型和 debug。下圖給出上圖搭建的 DBRNN 的 summary。

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_34 (Embedding)     (None, 380, 32)           121600
_________________________________________________________________
dropout_67 (Dropout)         (None, 380, 32)           0
_________________________________________________________________
bidirectional_30 (Bidirectio (None, 380, 32)           1568
_________________________________________________________________
simple_rnn_38 (SimpleRNN)    (None, 8)                 328
_________________________________________________________________
dropout_68 (Dropout)         (None, 8)                 0
_________________________________________________________________
dense_39 (Dense)             (None, 1)                 9
=================================================================
Total params: 123,505
Trainable params: 123,505
Non-trainable params: 0
_________________________________________________________________
```

模型的損失函數，優化器和評價指標如下：

```
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=['accuracy'])
```

在訓練模型之前，介紹 Keras 中一種優化模型效果且可以加快模型學習速度的方法：EarlyStopping。

EarlyStopping 介紹

EarlyStopping 是 Callbacks 的一種，callbacks 用於指定在每個 epoch 開始和結束的時候進行哪種特定操作，即用於提前停止訓練的 callbacks。之所以要提前停止訓練，是因為繼續訓練會導致測試集上的準確率下降。那繼續訓練導致測試準確率下降的原因筆者猜測可能是 1. 過擬合 2. 學習率過大導致不收斂 3. 使用正則項的時候，Loss 的減少可能不是因為準確率增加導致的，而是因為權重大小的降低。

EarlyStopping 的使用

一般是在 model.fit 函數中調用 callbacks，fit 函數中有一個參數為 callbacks。注意這裡需要輸入的是 list 類型的資料，所以通常情況只用 EarlyStopping 的話也要是[EarlyStopping()]

keras.callbacks.EarlyStopping(monitor='val_loss', patience=0, verbose=0, mode='auto')

參數說明：
monitor：需要監視的量，如'val_loss', 'val_acc', 'acc', 'loss'。
patience：能夠容忍多少個 epoch 內都沒有 improvement。
verbose：資訊展示模式
mode：'auto'，'min'，'max'之一，在 min 模式下，如果檢測值停止下降則中止訓練。在 max 模式下，當檢測值不再上升則停止訓練。例如，當監測值為 val_acc 時，模式應為 max，當檢測值為 val_loss 時，模式應為 min。在 auto 模式下，評價準則由被監測值的名字自動推斷。

引入EarlyStopping，当验证集准确率不再改善时停止训练

```
es = EarlyStopping(monitor='val_acc', patience=5)
```

训练模型

```
batch_size = 128
epochs = 20
model.fit(x_train, y_train,
          validation_split=0.1,
          batch_size=batch_size,
          epochs=epochs,
          callbacks=[es],
          shuffle=True)
```

```
Epoch 11/20
22500/22500 [==============================] - 37s 2ms/step - loss: 0.0594 - acc: 0.9777 - val_loss: 0.7208 -
val_acc: 0.8400
Epoch 12/20
22500/22500 [==============================] - 37s 2ms/step - loss: 0.0548 - acc: 0.9805 - val_loss: 0.6934 -
val_acc: 0.8508
Epoch 13/20
22500/22500 [==============================] - 36s 2ms/step - loss: 0.0565 - acc: 0.9781 - val_loss: 0.7393 -
val_acc: 0.8460
```

```
<keras.callbacks.History at 0x1db3fd7c978>
```

可以看到在第 13 次訓練完成後，驗證集的準確率下降後就停止了繼續訓練，這樣可以既可以加快訓練模型速度，也可以使得在驗證集的準確率不再下降。

最後我們使用三種訓練好的模型進行預測測試集，得到在 RNN 和 DBRNN 上模型的準確率在 0.85 左右，在 BRNN 模型在 0.87 左右。讀者可以通過調參進一步提高模型的準確率。

预测测试集

```
scores = model.evaluate(x_test, y_test)
```

```
25000/25000 [==============================] - 19s 746us/step
```

```
print('RNN:test_loss: %f, accuracy: %f' % (scores[0], scores[1]))
```

```
RNN:test_loss: 0.594139, accuracy: 0.853720
```

```
print('BRNN:test_loss: %f, accuracy: %f' % (scores[0], scores[1]))
```

```
BRNN:test_loss: 0.371344, accuracy: 0.867480
```

```
print('DBRNN:test_loss: %f, accuracy: %f' % (scores[0], scores[1]))
```

```
DBRNN:test_loss: 0.392413, accuracy: 0.851440
```

完整代碼下載：

https://github.com/hongweijun811/wjgit

至此，我們應該對 RNN 模型以及 Keras 實現 RNN 模型有了一定的瞭解。下一篇文章我們將會對 RNN 模型的改進模型 LSTM 模型進行詳細講解。

6. Keras 深度學習：LSTM 和雙向 LSTM 講解及實踐

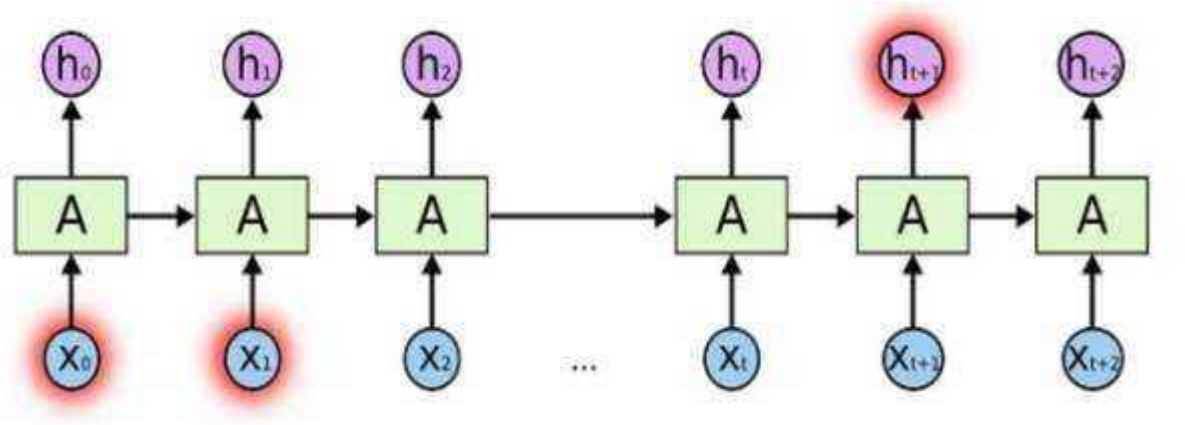
長短期記憶(Long Short Term Memory, LSTM)也是一種時間遞迴神經網路，最早由 Hochreiter & Schmidhuber 在 1997 年提出，設計初衷是希望能夠解決 RNN 中的長期依賴問題，讓記住長期資訊成為神經網路的預設行為，而不是需要很大力氣才能學會。

一、RNN 的長期依賴問題

在上篇文章中介紹的迴圈神經網路 RNN 在訓練的過程中會有長期依賴的問題，這是由於 RNN 模型在訓練時會遇到梯度消失(大部分情況)或者梯度爆炸(很少，但對優化過程影響很大)的問題。對於梯度爆炸是很好解決的，可以使用梯度修剪(Gradient Clipping)，即當梯度向量大於某個閾值，縮放梯度向量。但對於梯度消失是很難解決的。所謂的梯度消失或梯度爆炸是指訓練時計算和反向傳播，梯度傾向於在每一時刻遞減或遞增，經過一段時間後，梯度就會收斂到零(消失)或發散到無窮大(爆炸)。簡單來說，長期依賴的問題就是在每一個時間的間隔不斷增大時，RNN 會喪失到連接到遠處資訊的能力。

如下圖，隨著時間點 t 的不斷遞增，當 t 時刻和 0 時刻的時間間隔較大的時候，t 時刻的記憶 ht 可能已經喪失了學習連接到遠處 0 時刻的資訊的能力了。

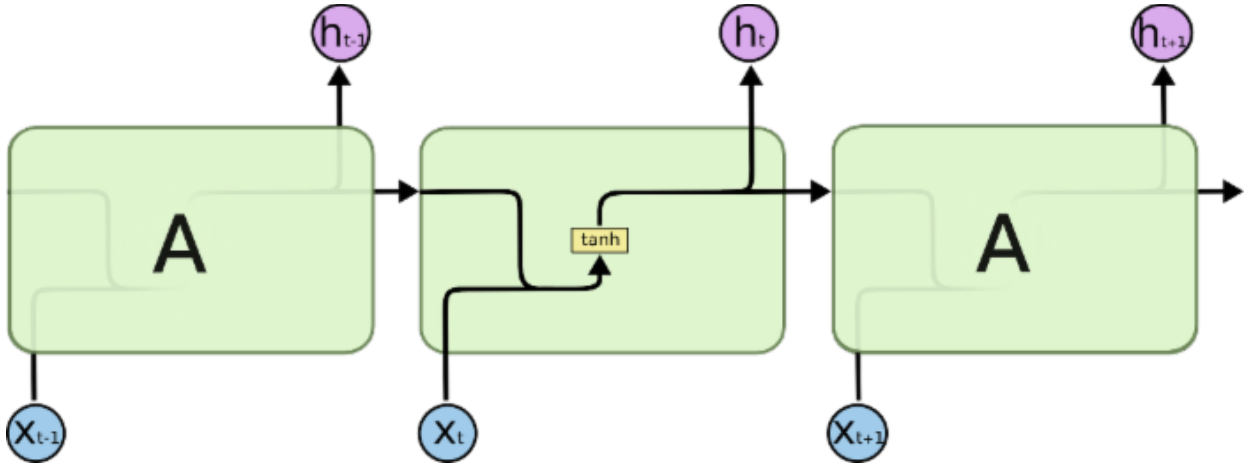
假設 X0 的輸入為"我住在深圳"，後面插入了很多其他的句子，然後在 Xt 輸入了"我在市政府上班"。由於 X0 與 Xt 相差很遠，當 RNN 輸入到 Xt 時，t 時刻的記憶 ht 已經喪失了 X0 時保存的資訊了。因此在 Xt 時刻神經網路無法理解到我是在哪一個城市的市政府上班了。



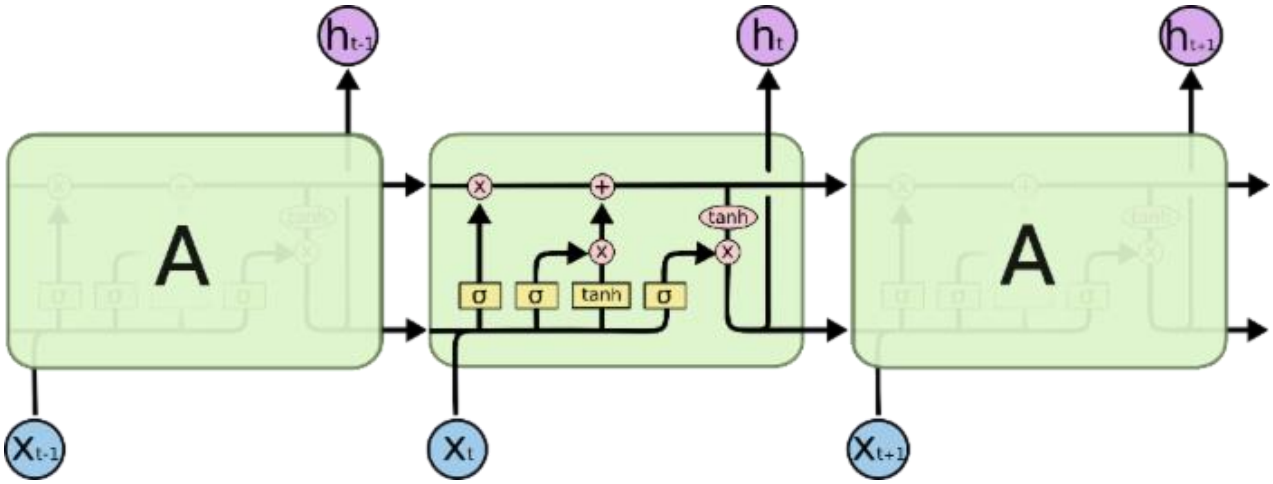
二、LSTM 原理講解

在理論上，RNN 絕對可以處理這樣的長期依賴問題。人們可以仔細挑選參數來解決這類問題中的最初級形式，但在實踐中，RNN 卻不能夠成功學習到這些知識。因此，LSTM 就是為了解決長期依賴問題而生的，LSTM 通過刻意的設計來避免長期依賴問題。記住長期的資訊在實踐中是 LSTM 的預設行為，而非需要付出很大代價才能獲得的能力！

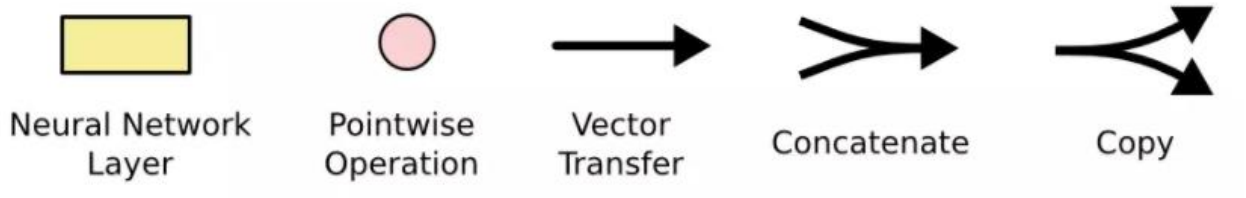
所有 RNN 都具有一種重複神經網路模組的鏈式的形式。在標準的 RNN 中，這個重複的模組只有一個非常簡單的結構，例如一個 tanh 層。



LSTM 同樣是這樣的結構，但是重複的模組擁有一個不同的結構。不同于 單一神經網路層，這裡是有四個，以一種非常特殊的方式進行交互。



先介紹上圖中的符號意義：

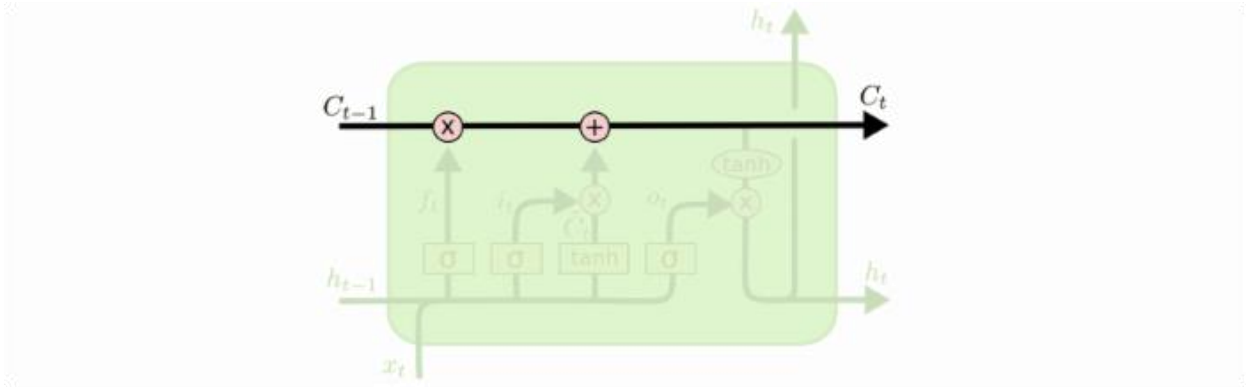


在上面的圖例中，每一條黑線傳輸著一整個向量，從一個節點的輸出到其他節點的輸入。粉色的圈代表 pointwise 的操作，諸如向量的和，而黃色的矩陣就是學習到的神經網路層。合在一起的線表示向量的連接，分開的線表示內容被複製，然後分發到不同的位置。

接下來將對 LSTM 進行逐步理解。在每個記憶單元(圖中 A)中包括細胞狀態(Ct)，遺忘門，輸入門和輸出門。這些門結構能讓資訊選擇性通過，用來去除或者增加資訊到細胞狀態。

1. 細胞狀態(Ct)

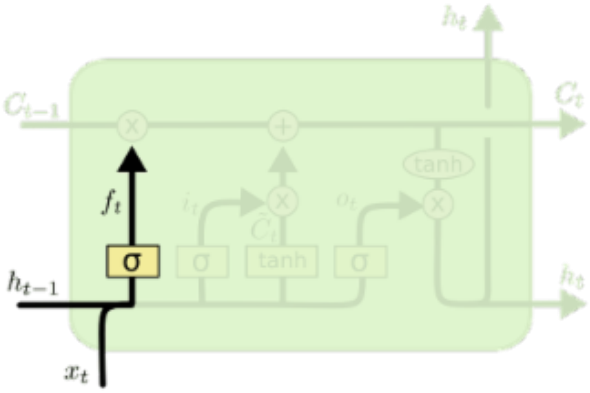
t 時刻的記憶資訊，用來保存重要資訊。就好像我們的筆記本一樣，保存了我們以前學過的知識點。如下圖的水平線從圖上方貫穿運行，直接在整個鏈上運行，使得資訊在上面流傳保持不變會很容易。



2. 遺忘門

控制遺忘上一層細胞狀態的內容，根據上一序列的 ht-1 和本序列的 Xt 為輸入，通過 sigmoid 啟動函數，得到上一層細胞狀態內容哪些需要去除，那些需要保留。值得注意的是，該輸入是以**向量**的形式，我們希望遺忘門輸出的值大多為 0 或 1，即**對向量中的每個值是完全忘記或者完全記住**，因此我們使用的是 sigmoid 函數作為啟動函數，因為該函數在許多取值範圍內的值都接近於 0 或 1(這裡不能用階躍函數作為啟動函數，因為它在所有位置的梯度都為 0，無

法作為啟動函數)。其他門使用 sigmoid 函數同理。因此，雖然在其他神經網路可以變換啟動函數，但並不建議變換 LSTM 的啟動函數。

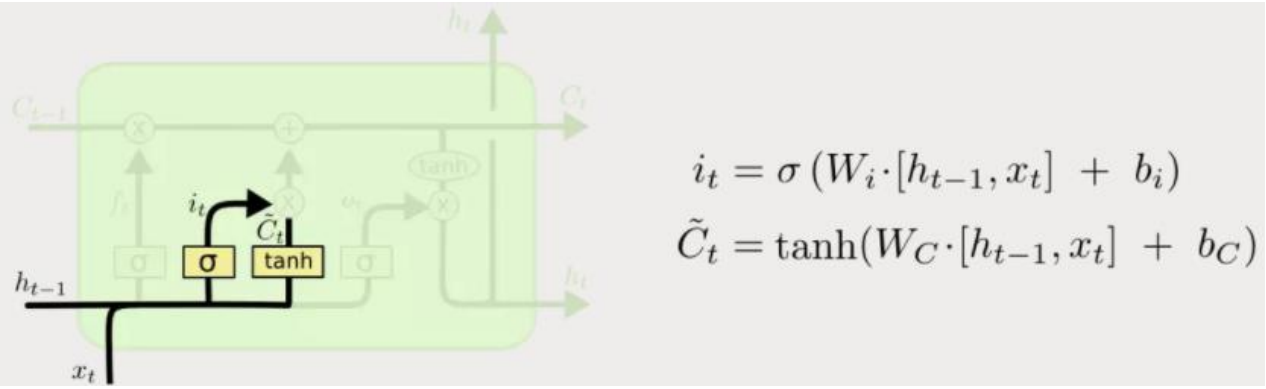


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

以一個例子來說明遺忘門的作用：在語言模型中，細胞狀態可能保存著這樣的重要資訊：當前主語為單數或者複數等。如當前的主語為"小明"，當輸入為"同學們"，此時遺傳門就要開始"幹活"了，將"小明"遺忘，主語為單數形式遺忘。

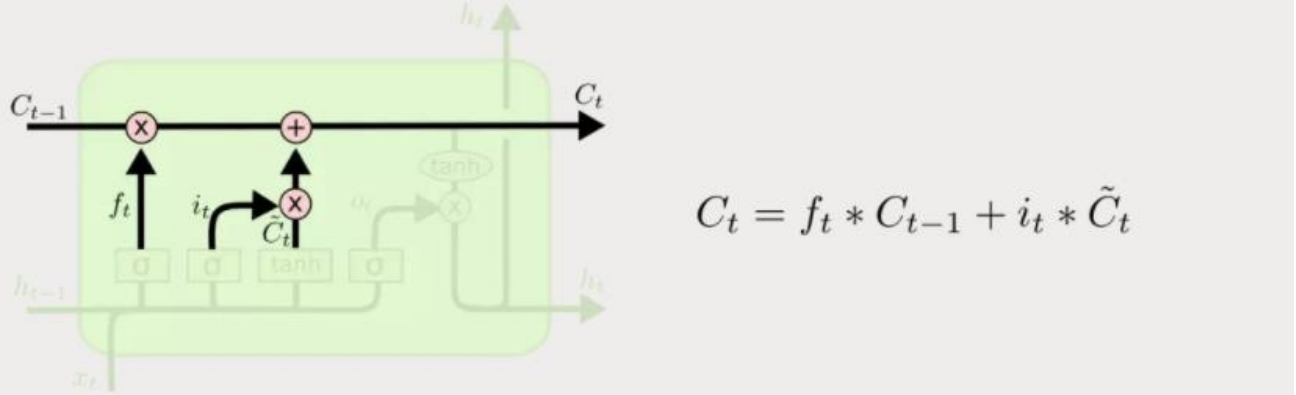
3. 輸入門

處理當前序列位置的輸入，確定需要更新的資訊，去更新細胞狀態。此過程分為兩部分，一部分是使用包含 sigmoid 層的輸入門決定哪些新資訊該被加入到細胞狀態；確定了哪些新資訊要加入後，需要將新資訊轉換成能夠加入到細胞狀態的形式。所以另一部分是使用 tanh 函數產生一個新的候選向量。(可以這麼理解，LSTM 的做法是對資訊都轉為能加入細胞狀態的形式，然後再通過第一部分得到的結果確定其中哪些新資訊加入到細胞狀態。)



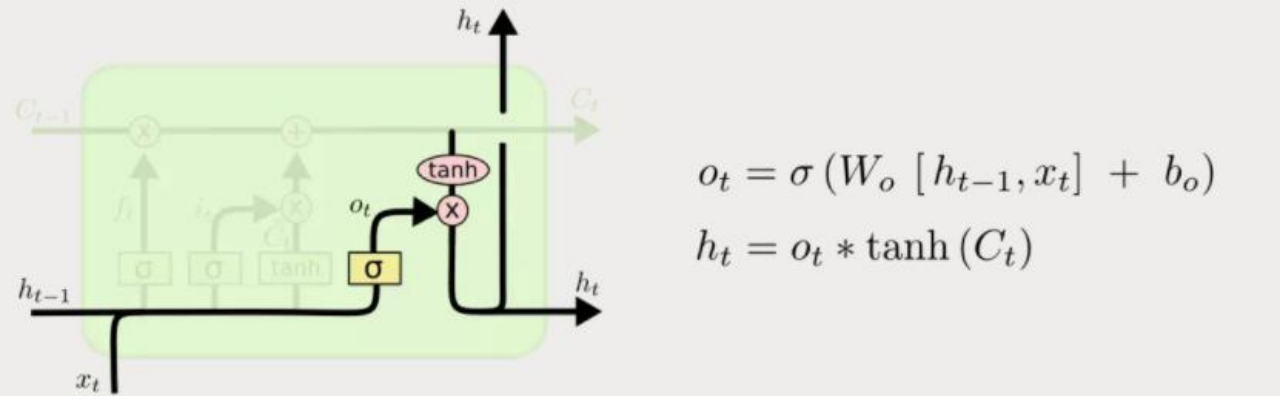
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] \; + \; b_C\right)$$

有了遺忘門和輸入門，現在我們就能把細胞狀態 Ct−1 更新為 Ct 了。如下圖所示，其中 ft×Ct−1 表示希望刪除的資訊，it×Ct 表示新增的資訊。



4. 輸出門

最後要基於細胞狀態保存的內容來確定輸出什麼內容。即選擇性的輸出細胞狀態保存的內容。類似於輸入門兩部分實現更新一樣，輸出門也是需要使用 sigmoid 啟動函數確定哪個部分的內容需要輸出，然後再使用 tanh 啟動函數對細胞狀態的內容進行處理(因為通過上面計算得到的 Ct 每個值不是在 tanh 的取值範圍-1~1 中，需要調整)，將這兩部分相乘就得到了我們希望輸出的那部分。

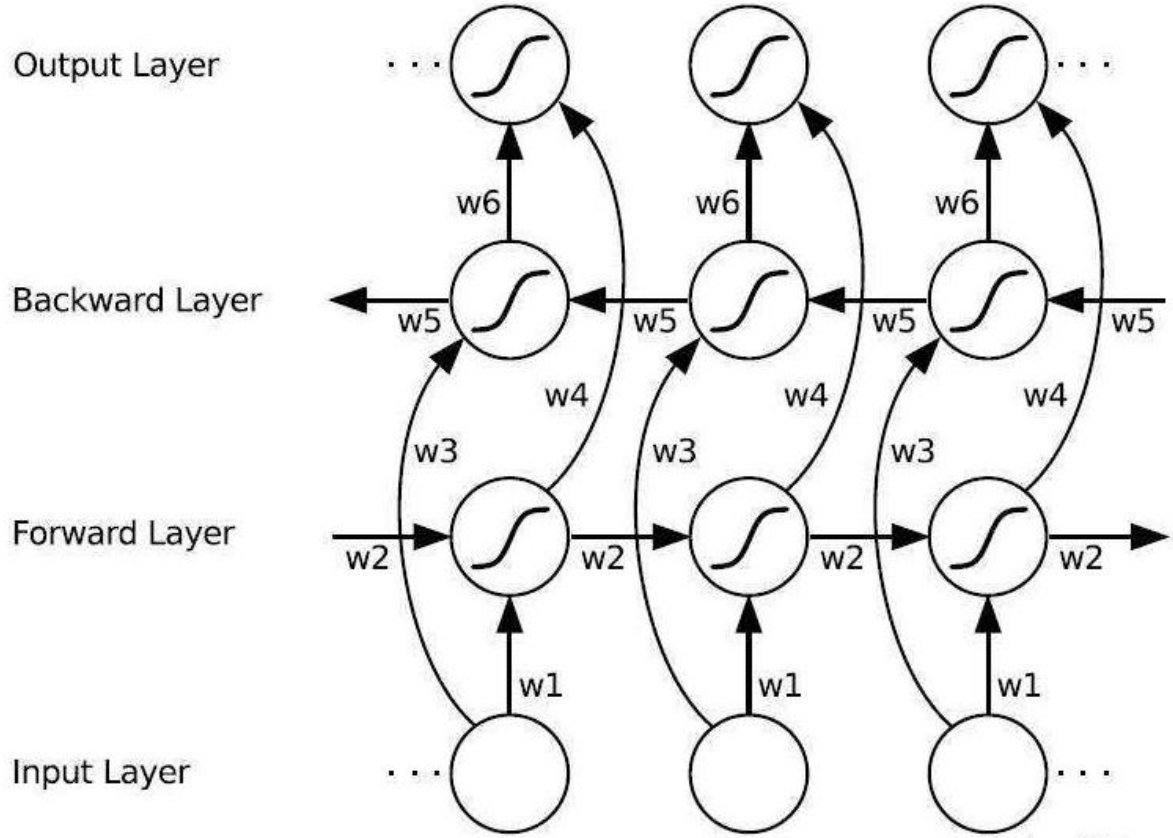


舉個例子，同樣在語言模型中，細胞狀態中此時包含很多重要資訊，比如：主語為單數形式，時態為過去時態，主語的性別為男性等，此時輸入為一個主語，可能需要輸出與動詞相關的資訊，這個時候只需要輸出是單數形式和時態為過程，而不需要輸出主語性別就可確定動詞詞性的變化。

三、雙向 LSTM（Bi-directional LSTM）

如上篇文章 BRNN 所述同理，有些時候預測可能需要由前面若干輸入和後面若干輸入共同決定，這樣會更加準確。因此提出了雙向迴圈神經網路，網路結構

如下圖。可以看到 Forward 層和 Backward 層共同連接著輸出層，其中包含了 6 個共用權值 w1-w6。



在 Forward 層從 1 時刻到 t 時刻正向計算一遍，得到並保存每個時刻向前隱含層的輸出。在 Backward 層沿著時刻 t 到時刻 1 反向計算一遍，得到並保存每個時刻向後隱含層的輸出。最後在每個時刻結合 Forward 層和 Backward 層的相應時刻輸出的結果得到最終的輸出，用數學運算式如下：

$$h_t = f(w_1 x_t + w_2 h_{t-1})$$

$$h'_t = f(w_3 x_t + w_5 h'_{t+1})$$

$$o_t = g(w_4 h_t + w_6 h'_t)$$

四、Keras 實現 LSTM 和雙向 LSTM

Keras 對迴圈神經網路的支援和封裝在上一篇文章已經講解了，在這裡僅介紹兩個模型的搭建，如有疑問請閱讀 keras 系列的上一篇文章。

LSTM模型

```
model = Sequential()
model.add(Embedding(3800, 32, input_length=380))
model.add(Dropout(0.2))
model.add(LSTM(32))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
```

Bi-LSTM模型

```
model = Sequential()
model.add(Embedding(3800, 32, input_length=380))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(32, return_sequences=True), merge_mode='concat'))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
```

训练模型

```
es = EarlyStopping(monitor='val_acc', patience=5)
```

```
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=['accuracy'])
```

```
batch_size = 64
epochs = 20
model.fit(x_train, y_train,
          validation_split=0.1,
          batch_size=batch_size,
          epochs=epochs,
          callbacks=[es],
          shuffle=True)
```

预测

```
scores = model.evaluate(x_test, y_test)
```

```
25000/25000 [==============================] - 37s 1ms/step
```

```
print('LSTM:test_loss: %f, accuracy: %f' % (scores[0], scores[1]))
```

```
LSTM:test_loss: 0.458496, accuracy: 0.857640
```

```
print('Bi-LSTM:test_loss: %f, accuracy: %f' % (scores[0], scores[1]))
```

```
Bi-LSTM:test_loss: 0.333112, accuracy: 0.866520
```

參考文獻：https://colah.github.io/posts/2015-08-Understanding-LSTMs/

7. Keras 官方中文文檔發佈

Keras: 基於 Python 的深度學習庫

你恰好發現了 Keras。
Keras 是一個用 Python 編寫的高級神經網路 API，它能夠以 TensorFlow, CNTK, 或者 Theano 作為後端運行。Keras 的開發重點是支持快速的實驗。能夠以最小的時延把你的想法轉換為實驗結果，是做好研究的關鍵。

如果你有如下需求，請選擇 Keras：

允許簡單而快速的原型設計（用戶友好，高度模組化，可擴展性）。
同時支援卷積神經網路和迴圈神經網路，以及兩者的組合。
在 CPU 和 GPU 上無縫運行與切換。
查看文檔，請訪問 Keras.io。

Keras 相容的 Python 版本: Python 2.7-3.6。

指導原則
User friendliness. Keras 是為人類設計的 API，而不是機器。它把用戶體驗放在首要和中心位置。Keras 遵循減少認知困難的最佳實踐： 它提供一致和簡單的 API，它將常見用例所需的用戶運算元量降至最低，並且在使用者錯誤時提供清晰和可操作的回饋。

Modularity. 模型被理解為由獨立的，完全可配置的模組構成的序列或圖表。這些模組可以在盡可能少的限制下組裝在一起。特別是神經網路層、損失函數、優化器、初始化方法、啟動函數、正規化方法，它們都是可以結合起來構建新模型的模組。

Easy extensibility. 新的模組是很容易添加的（作為新的類和函數），現有的模組已經提供了充足的例子。為能夠輕鬆地創建可以提高表現力的新模組，使 Keras 更加適合高級研究。

Work with Python. Keras 沒有特定格式的單獨設定檔。模型定義在 Python 代碼中，這些代碼緊湊，易於調試，並且易於擴展。

快速開始：30 秒上手 Keras
https://keras.io/zh/

8. 使用 vgg16 模型進行圖片預測

前面我們學習了使用 cifra10 來判斷圖片的類別，今天我們使用更加強大的已經訓練好的模型來預測圖片的類別，那就是 vgg16,對應的供 keras 使用的模型人家已經幫我們訓練好，我可不想來買一個 gpu。

對應的模型在 **'vgg16'** 可以下載。附上連結 (http://pan.baidu.com/s/1qX0CJSC)

導入必要的庫

```
from keras.models import Sequential
from keras.layers.core import Flatten, Dense, Dropout
from keras.layers.convolutional import Convolution2D, MaxPooling2D, ZeroPadding2D
from keras.optimizers import SGD
import cv2, numpy as np
```

```
Using Theano backend.
D:\Anaconda\lib\site-packages\theano-0.8.0.dev0-py2.7.egg\theano\tensor\signal\downsample.py:5: UserWarning: downsample module has been moved to the pool module.
warnings.warn("downsample module has been moved to the pool module.")
```

使用 keras 建立 vgg16 模型

```
def VGG_16(weights_path=None):
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(3,224,224)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
```

```
14
15 model.add(ZeroPadding2D((1,1)))
16 model.add(Convolution2D(256, 3, 3, activation='relu'))
17 model.add(ZeroPadding2D((1,1)))
18 model.add(Convolution2D(256, 3, 3, activation='relu'))
19 model.add(ZeroPadding2D((1,1)))
20 model.add(Convolution2D(256, 3, 3, activation='relu'))
21 model.add(MaxPooling2D((2,2), strides=(2,2)))
22
23 model.add(ZeroPadding2D((1,1)))
24 model.add(Convolution2D(512, 3, 3, activation='relu'))
25 model.add(ZeroPadding2D((1,1)))
26 model.add(Convolution2D(512, 3, 3, activation='relu'))
27 model.add(ZeroPadding2D((1,1)))
28 model.add(Convolution2D(512, 3, 3, activation='relu'))
29 model.add(MaxPooling2D((2,2), strides=(2,2)))
30
31 model.add(ZeroPadding2D((1,1)))
32 model.add(Convolution2D(512, 3, 3, activation='relu'))
33 model.add(ZeroPadding2D((1,1)))
34 model.add(Convolution2D(512, 3, 3, activation='relu'))
35 model.add(ZeroPadding2D((1,1)))
36 model.add(Convolution2D(512, 3, 3, activation='relu'))
37 model.add(MaxPooling2D((2,2), strides=(2,2)))
38
39 model.add(Flatten())
40 model.add(Dense(4096, activation='relu'))
41 model.add(Dropout(0.5))
42 model.add(Dense(4096, activation='relu'))
43 model.add(Dropout(0.5))
44 model.add(Dense(1000, activation='softmax'))
45
46 if weights_path:
47 model.load_weights(weights_path)
48
49 return model
```

```
1 model = VGG_16('vgg16_weights.h5')
```

```
1
2 sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
  model.compile(optimizer=sgd, loss='categorical_crossentropy')
```

現在我們開始來預測了

首先寫一個方法來載入並處理圖片

```
def load_image(imageurl):
im = cv2.resize(cv2.imread(imageurl),(224,224)).astype(np.float32)
im[:,:,0] -= 103.939
im[:,:,1] -= 116.779
im[:,:,2] -= 123.68
im = im.transpose((2,0,1))
im = np.expand_dims(im,axis=0)
return im
```

讀取 vgg16 的類別文件

```
f = open('synset_words.txt','r')
lines = f.readlines()
f.close()
```

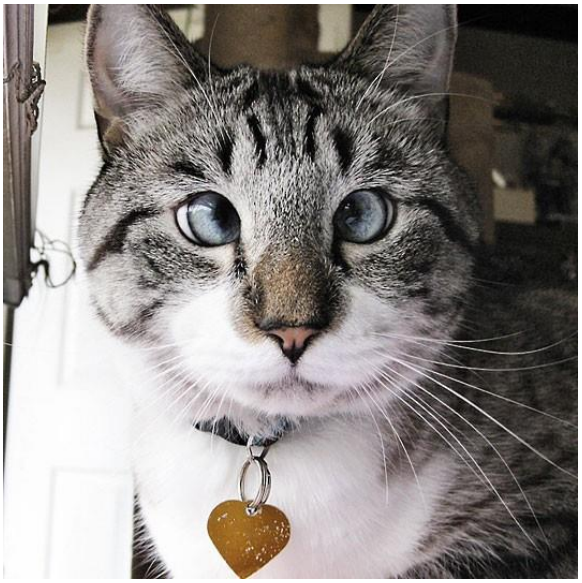
```
def predict(url):
im = load_image(url)
pre = np.argmax(model.predict(im))
print lines[pre]
```

```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
from IPython.display import Image
```

```
1 Image('cat1.jpg')
```



開始預測

```
1 predict('cat1.jpg')
```

n02123045 tabby, tabby cat

```
1 Image('zebra.jpg')
```

```
predict('zebra.jpg')
```

n02391049 zebra

```
Image('airplane.jpg')
```

```
predict('airplane.jpg')
```

n02690373 airliner

```
1 Image('pig.jpg')
```

```
1 predict('pig.jpg')
```

n02395406 hog, pig, grunter, squealer, Sus scrofa

可見，判斷率還是很高的。。。。

總結

通過這次學習，學會了使用 keras 來搭建模型，使用 vgg16 這個模型。

hadxu 授權 http://www.tensorflownews.com/ 轉載
原文連結：https://hadxu.github.io/