Machine Learning Report~

Semi-supervised Learning

1. Supervised learning:

Keras Model

```
Actinit = PReLU(init='zero', weights=None)
Act = LeakyReLU(alpha=0.3)

model = Sequential()
model.add(Convolution2D(32, 3, 3, border_mode='same',
                        input_shape=(32,32,3)))
#L1
model.add(Actinit)
model.add(Convolution2D(32, 3, 3, border_mode='same'))
model.add(BatchNormalization())
model.add(Act)
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
#L2
model.add(Convolution2D(64, 3, 3, border_mode='same'))
model.add(Act)
model.add(Convolution2D(64, 3, 3, border_mode='same'))
model.add(BatchNormalization())
model.add(Act)
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
#L3
model.add(Convolution2D(128, 3, 3, border_mode='same'))
model.add(Act)
model.add(Convolution2D(128, 3, 3, border_mode='same'))
model.add(BatchNormalization())
model.add(Act)
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
#L4
model.add(Convolution2D(256, 3, 3, border_mode='same'))
model.add(Act)
model.add(Convolution2D(256, 3, 3, border_mode='same'))
model.add(BatchNormalization())
model.add(Act)
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
#Dense Layer
model.add(Flatten())
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Act)
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))

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

接著使用 model.fit (Input, Label, batch_size, epoch) 即可得到最後模型。

*** Description on this method**

首先，我先介紹一下我使用的 Model，這個 model 在往後的 semi-supervised 部分也都會繼續適用，因此就一併介紹—

整個 model 有參考網路上針對這種 cifar 10 測資的模型，以及老師上課的投影片—結構上大致和範例一致，我增加了兩組 Convolution，因此總共是八層 convolution，中間四個 Max Pooling，經過 Flatten 以後，通過一層 Dense 神經元，再最後利用 Softmax 來判斷是屬於哪個 class。

我對這個 model 做了相當多的嘗試，而得到這個結果。我其他有改動的地方，像是把 activation 的方式改用 keras.io 上面 Advanced activation layer、dropout 值稍微下修、Optimizer 多方嘗試後決定用 Adam、以及增加 batch normalization 的功能等。

此外我又參考網路上 cifar 10 的範例，新增了 data generator 的功能，他會做一些 data augmentation，像是上下左右橫移、水平翻轉圖片等，去增加 training 的功能。這個手法會在下面兩個 semi-supervised learning 的手法當中繼續使用，因為著實令分數進步不少。

Model 講完以後，剩下的部分就相對 trivial，呼叫內建的 fit 和 predict 兩個函式，便能得到 test case 的預測結果了。而我 Supervised 這邊在 fit 的時候參數選擇為：batch size=100, epoch=50。

*** Performance (Validation accuracy and Kaggle public score)**

# 最初始模型(與範例一致)	約 0.47
# 增加 data generator 之後	約 0.53
# 改良 Model 之後(如上述)	約 0.64

2. Semi-supervised Learning ~ Self Training

```
for i in range(0, iterrange):
    print(repr(i)+'iter')
    if i > 9:
        epo2 = 10
    X_out = model.predict_proba(X_unlabel, batch_size=minibatch)
    toadd_index = np.where(X_out > 0.995)
    X_toadd = X_unlabel[toadd_index[0],:,:,:]
    X_unlabel = np.delete(X_unlabel, toadd_index[0], axis = 0)
    X_newlab = np.zeros((np.size(toadd_index[0]), 10))
    for k in range(0, np.size(toadd_index[0])):
        X_newlab[k, toadd_index[1][k]] = 1
    X_input = np.concatenate((X_input,X_toadd), axis=0)
    X_label = np.concatenate((X_label,X_newlab), axis=0)
    datagen.fit(X_input)
    model.fit_generator(datagen.flow(X_input, X_label,
                        batch_size=minibatch),
                        samples_per_epoch=X_input.shape[0],
                        nb_epoch=epo2,
                        validation_data=(X_validdata,X_validlabel))
```

*** Description on this method**

這題大架構和 supervised 部分相同，只是加上了 unlabeled 的使用。因此我前面的 code，便只附上後面加入 unlabeled data 的部分。這個手法我的 batch size=100, epoch=120, 20, 10(隨 iteration 往下降)

我加入 unlabeled data 的方式，使用的是的 self-training 的技巧，我會在每次針對上面的那個模型，train 完一定的次數後，將 unlabeled data 拿去做 prediction，若目前的模型顯示某筆 unlabeled data 有高於 0.995 的機率會屬於某一個 class，便將該筆 data 加入 labeled data—使用迴圈不斷的重覆這個動作，如此便可用 unlabeled data 來輔助完成最後的模型。

*** Performance (Validation accuracy and Kaggle public score)**

# 最初始模型(與範例一致)	約 0.63
# 增加 data generator 之後	約 0.73
# 改良 Model (改變 Activation layer)	約 0.75
# 改良 Model (使用 Batch Normalize)	約 0.77
# 改良 Model (加兩組 Convolution layer)	約 0.80
# 多組 train 出來結果的投票	約 0.82

3. Semi-supervised Learning (2) ~ Autoencoder

右為兩組 Autoencoder 以及最後 Softmax 進行 pretrain 的模型

```
#Convolutional layer
input_img = Input(shape=(32, 32, 3))

x = Convolution2D(32, 3, 3, activation='relu', border_mode='same')(input_img)
x = MaxPooling2D((2, 2), border_mode='same')(x)
x = Convolution2D(32, 3, 3, activation='relu', border_mode='same')(x)
x = MaxPooling2D((2, 2), border_mode='same')(x)
x = Convolution2D(64, 3, 3, activation='relu', border_mode='same')(x)
x = MaxPooling2D((2, 2), border_mode='same')(x)
x = Convolution2D(64, 3, 3, activation='relu', border_mode='same')(x)
encoded = MaxPooling2D((2, 2), border_mode='same')(x)

x = Convolution2D(64, 3, 3, activation='relu', border_mode='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Convolution2D(64, 3, 3, activation='relu', border_mode='same')(x)
x = UpSampling2D((2, 2))(x)
x = Convolution2D(32, 3, 3, activation='relu', border_mode='same')(x)
x = UpSampling2D((2, 2))(x)
x = Convolution2D(32, 3, 3, activation='relu', border_mode='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Convolution2D(3, 3, 3, activation='sigmoid', border_mode='same')(x)

encoder = Model(input=input_img, output=encoded)
autoencoder = Model(input=input_img, output=decoded)
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=["accuracy"])


#Dense Layer
input_img2 = Input(shape=(256,))
encoded2 = Dense(150, activation='relu')(input_img2)
decoded2 = Dense(256, activation='relu')(encoded2)
encoder2 = Model(input=input_img2, output=encoded2)
autoencoder2 = Model(input=input_img2, output=decoded2)
autoencoder2.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=["accuracy"])

#Softmax Layer
softlayer = Sequential()
softlayer.add(Dense(10, input_dim=150))
softlayer.add(Activation('softmax'))
softlayer.compile(loss='categorical_crossentropy', optimizer= 'adam', metrics=['accuracy'])
```

下方為 Autoencoder 進行 pretrain 和 fine-tune 的過程：

```
#Apply Noise Factor
noise_factor = 0.5
X_withnoise = X_inputall + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_inputall.shape)
X_withnoise = np.clip(X_withnoise, 0., 1.)


#Autoencoder fitting (Pretrain)
datagen.fit(X_withnoise)
autoencoder.fit_generator(datagen.flow(X_withnoise, X_inputall, batch_size=minibatch),
                                       samples_per_epoch=X_withnoise.shape[0],
                                       nb_epoch=epo)
X_inputall = encoder.predict(X_inputall)
X_inputenc = encoder.predict(X_input)


X_inputall = np.reshape(X_inputall, (X_inputall.shape[0],256))
X_withnoise = X_inputall + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_inputall.shape)
X_withnoise = np.clip(X_withnoise, 0., 1.)
autoencoder2.fit(X_withnoise, X_inputall, nb_epoch=epo2, batch_size=minibatch)

X_inputall = encoder2.predict(np.reshape(X_inputenc, (X_inputenc.shape[0],256)))
softlayer.fit(X_inputall,X_label,batch_size=minibatch,nb_epoch=epo)

encoders = []
encoders.append(encoder)
encoders.append(encoder2)
encoders.append(softlayer)

# Fine-tune
model = Sequential()
model.add(encoders[0])
model.add(Flatten())
model.add(encoders[1])
model.add(encoders[2])
model.compile(loss='categorical_crossentropy',
              optimizer= 'adam',
              metrics=['accuracy'])

datagen.fit(X_input)

model.fit_generator(datagen.flow(X_input, X_label,
                    batch_size=minibatch),
                    samples_per_epoch=X_input.shape[0],
                    nb_epoch=epo,validation_data=(X_validdata, X_validlabel))
```

*** Description on this method**

這題我的手法是利用 Autoencoder 進行 pretrain。

基本架構我也是參考 Keras 上 autoencoder 的範例程式。這邊的 Model 跟前面不一樣在於，除了寫 encoder 部分之外，還要寫一個相對應能夠 decode 回去的 layers，然後 encoder 加上 decoder 一起 train，使用的 loss function 是 binary_crossentropy，代表 decode 後的還原度。

原則上我也是使用 Convolution + Dense + Softmax，然後我把三個部分分開進行 pretrain，一層一層 train 完之後，便可以加在一起，形成一個新的 model。而這個 model 在使用原先的 labeled data 進行一些最終調整，在 Machine Learning 上稱之為 fine-tune，調整完的 model 經過 predict 便可以得到最後的結果了！

*** Performance (Validation accuracy and Kaggle public score)**

# 僅進行 Pretrain	約 0.37
# Pretrain 後加上 Fine-tune	約 0.55

4. Compare, Analyze, and Discuss

(1). Supervised vs. Semi-supervised

結果上來說，Semi-supervised 的成效比較好！

這是因為，Semi-supervised 加上使用了九倍的 unlabeled data，在這種狀況下，unlabeled data 可能會提供程式一些本來 labeled data 不包含的訊息，而改變最後進行 Classification 的依據。每次在新增資料的時候，都要擁有超過 0.995 的信心，因此還算準確。

當然，這跟 training data 和 test data 的設計還是有些關係的，某些 data set 可能使用 supervised 反而保證不出錯，效果更好—但是這題 unlabeled data 顯然含有一些能夠輔助判斷 test data 的訊息就是。

(2). Usage of validation set

我這次的作業還有使用 validation set。我從 labeled data 中切出了 500 筆的 data 來做為 validation 使用。(有先將 labeled data 進行 shuffle，否則的話會出現 500 個 label 10…)

Validation Set 使得我可以看到其 accuracy，隨著 training 增加，會發現 training data 的 accuracy 接近 0.97，實為 overfitting。參考 validation set 的正確率，這次因為資料量大，幾乎可以準確估計 Kaggle 上的分數，通常不會差超過一個百分點。這使我能夠進行有效篩選，尤其是在我不斷嘗試調整我的 model 的時候，可以知道哪些方向較為正確。

(3). Comment on Autoencoder

Autoencoder 是這次我三種方法裡面效果最差的。

Autoencoder 進行 pretrain 的時候，時常是完全不使用 label 的，也就是處在一個 unsupervised 的狀況下—unsupervised 相對費時，因此我可能提供不夠久的時間；此外，這次的 data set 有足夠多的 label 來對程式進行 supervise 的動作，明明有資訊卻不使用，效果差一些也是可想而知。

但是 pretraining 仍是有效果的，在進行 fine-tune 之前，我的 accuracy 便能夠達到 0.37 左右，已經比 supervised 最一開始接近 random 的分布要(0.1)來的好，但是隨著 fine-tune 下去，因為 supervised 我使用的 model 比較強，因此 performance 便又被超越了。

Appendix

(1)我有參考 github 上的 cifar 10 範例程式的 model 寫法 (supervised, self-training part)以及 blog.keras.io 上 autoencoder 的寫法

(2)我 Kaggle 上最佳成績是 vote 出來的，而上傳的程式碼以及 model 則是如報告內容所述，public score 在 0.80 附近

(3) Collaborators: B02901092 江貫榮、B02901056 吳雨葳、B02901019 王志新、B02901051 呂弈臻