ML HW3 Report

R04922108 林俊佑

1. Supervised

第一題我使用兩層 Convolution layer, 一層全連層, 第一層 conv layer 為 128 個 3 *3 的 filter, 接著接上 batch normalization , 然後再將 feature map 整個丟進 relu activation function, 第二層 convolution layer 只與第一層差在使用的是 256 個 filter, 做完 convolution layer 後面把 feature map 拉直進入全連層, 我使用的是 1024 個 node, 在前面兩層 convolution layer 後面我都有加上一個 kernel size 2*2 strides 2*2 的 max pooling 層, 另外, 再除了 input layer 以外的每一層後面, 我都有接上 dropout rate 0.6 的 dropout layer, 在 data augmentation 方面, 我只使用了將影像左右翻轉而已, 這次作業我使用 Tensorflow 實作, performance 方面, 在只使用這 10000 筆 labeled data 的情況下, 在 kaggle public set 上面的 accuracy 是 62%左右

2. Semi Supervised Method1: Self training

第二題的架構跟第一題長得一模一樣, 差別在於在做完 training 後, 直接對剩下 45000 筆 data 去做 labeling, 並將 confidence 大於 0.9 的 data 加進 labeled data set 裡面, 在 train 一個 model 出來, 這樣子連續做了兩次循環,

```
    for i in range(100):
        batch_softmax_result, batch_argmax_result = sess.run([y_conv_softmax, unlabeled_batch_result], feed_dict =
ob_in: 1.0, keep_prob: 1.0, phase_train: False})
        unlabeled_all_softmax_result.append(batch_softmax_result)
        unlabeled_all_argmax_result.append(batch_argmax_result)
    unlabeled_all_softmax_result = np.asarray(unlabeled_all_softmax_result).reshape(-1, 10)
    index = np.amax(unlabeled_all_softmax_result, axis = 1) > 0.9
    unlabeled_all_argmax_result = np.asarray(unlabeled_all_argmax_result).reshape(-1)
    hard_label = np.zeros(shape = unlabeled_all_softmax_result.shape)
    for i in range(hard_label.shape[0]):
        hard_label[i, unlabeled_all_argmax_result[i]] = 1.
    self_labeled_image = np.concatenate((train_image, unlabeled_image[index]), axis = 0)
    self_label = np.concatenate((train_label, hard_label[index]), axis = 0)
```

最終在使用 unlabeled data 一起 train 的情況下, 在 kaggle public set 上面的 accuracy 是 64.8%

3. Semi Supervised Method2: Self training with initial by fine-tune auto-encoder

第三題我使用 autoencoder 去對前面兩題的那個架構做 fine-tune weight initial, 首先先用第一層 train 一個 autoencoder, 再把 decode part 拿掉, 在固定 weight 加入第二層在 train 一個 autoencoder, 一樣, 再把 decode part 拿掉, 最後將這兩層的 weight 當作 initial, 後面接上全連層, 先使用 labeled data, 在使用 unlabeled data 做 self training, 這題的結果也差不多事 64.8% 左右, 不過在 training 上面得到很大的幫助, 有做 initial weight 的方法在 epoch 50 左右就已經 fit training data 了, 但沒有做 initial weight 的方法 50

epoch 的時候在 training set 上只有 88%的準確率

```
for l in range(1, L + 1, 1):
#---construct model---#
    encoder_op = encoder(l, X)
    print encoder_op.get_shape()
    #  autoencoder model
    decoder_op = tf.reshape(decoder(l, encoder_op), [-1, n_input])
    y_pred = decoder_op
    y_true = X
    cost = tf.reduce_mean(tf.pow(y_true - y_pred, 2))
    optimizer = tf.train.AdamOptimizer(0.0005).minimize(cost)
    uninitialized_vars = []
    for var in tf.all_variables():
        try:
            sess.run(var)
        except tf.errors.FailedPreconditionError:
            uninitialized_vars.append(var)
    init_new_vars_op = tf.initialize_variables(uninitialized_vars)
    sess.run(init_new_vars_op)
    #---train the autoencoder---#
    e = 5 * l
    for epoch in range(e):
        for i in range(batch.shape[0]):
            , c, y_p, y_t = sess.run([optimizer, cost, y_pred, y_true], feed_dict = {X: all_image[batch[i]], phase_train: True})
            print 'fine tune, layer ' + str(l) + '/' + str(L)
            print 'epoch ' + str(epoch + 1) + '/'+ str(e) + ' batch '  + str(i + 1) + '/' + str(batch.shape[0]) +' cost:' + str(c)
```

4. Result Analysis and Comparison

- 這次作業，在 unlabeled data labeling 的部分，也就是第三題，其實我一開始嘗試過使用 raw data 算 image 距離, autoencoder code 算 code 距離, autoencoder + K nearest neighbor 做 clustering，等等方法，在 raw data 上面，使用 validation set, labeling 大概只有 23%的 accuracy，使用 autoencoder 的 code 的話，可以提升到 28%左右，如果使用 autoencoder + KNN 的話，在 K = 20, vote 方法設為 exp^(-distance) 可以達到 38%準確率，但這樣子還是太低了，我擔心使用他 label 出來的 model 來 training，可能會造成更差的效果，因此我試過使用 autoencoder + KNN + labeled data trained model，來做 vote, vote 的分數設為 exp^(-distance) * (trained model 給這筆 data 這個 label 的 confidence)，但使用這種 vote 後，結果幾乎跟直接使用 self training 差不多，因此我捨棄這些方法，只使用 autoencoder 來對原本的 model 做 weight initial，我原本有參考一篇 paper 實作 7 層的 ALL CNN model，這個 model train 起來非常的耗時，但在使用 autoencoder 後再 training 的一開始就有 30%的準確率了，減少了很多 training 的時間.
- 最後在 7 層 ALL-CNN 以及兩層 CNN+一層 DENSE 下做選擇，兩者的 performance 差不多，但 model 大小差了非常多，因此我使用了兩層 CNN 的結構
- 另外這次使用 tensorflow，有關 mini batch, batch normalization，皆為我自己實作出來的
- Comparison:
  Autoencoder init + self training: 64.8% 更快找到 optimal function
  Self training: 64.8%
  Supervised: 62%