Machine Learning HW3 Report

電機三 郭笛萱

B03901009

Discussed with b03901009 施順耀

1. Supervised Learning

In this method, I use three Convolution layers and three MaxPooling layers. In the last layer, I choose to use softmax as the Activation function. Adadelta is used as the optimizer and Cross_entropy as the loss function. Epoch is chosen to be 300, and batch number = 32.

I also use ImageDataGenerator to flip, rotate, and shift the images, so I can get more training data while training, and after adding ImageDataGerator, I found the validation score improves from 0.60 to 0.63, which shows a huge improvement.

CNN model

```
model = Sequential()

model.add(Convolution2D(32, 3, 3, border_mode='same', input_shape =
        (3,32,32)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Convolution2D(64, 3, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))
model.add(Convolution2D(64, 3, 3, border_mode='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
```

```
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))

keras.optimizers.Adadelta(lr = 1.0, rho = 0.95, epsilon = 1e-06)
model.compile(loss = 'categorical_crossentropy', optimizer = 'Adadelta', metrics
= ['accuracy'])
```

Data generator

```
datagen = ImageDataGenerator(
        rotation_range=20,    # randomly rotate images in (degrees, 0 to 20)
        width_shift_range=0.1,    # randomly shift images horizontally
        height_shift_range=0.1,    # randomly shift images vertically
        horizontal_flip=True)    # randomly flip images
```

2. Semi-supervised Learning 1 : Self-learning

In this method, I used the best model I got in Supervised Learning to predict unlabel data. If the probability is larger than 0.9995, then I will add this data to the training set, and train the whole data set with the model in supervised learning again. I choose to add 3000 unlabel data to the training set, and epoch is chosen to be 200, batch number to be 32.

```
p = model.predict_proba(x_unlabel, batch_size = 100, verbose = 0)
add = 0
add_x_train = [ ]
add_y_train = [ ]
delete_id = [ ]
#choose the unlabel with probability > 0.9995, and add it to the train data
for i in range(len(p)):
    ans = 0
    ans_id = 0
    for j in range(10):
        if p[i][j] > ans:
            ans = p[i][j]
            ans_id = j
```

```
        for j in range(10):
            if ans_id == j: p[i][j] = 1
            else : p[i][j] = 0
        if ans > 0.9995 and add < 3000:
            add += 1
            add_x_train.append(x_unlabel[i])
            add_y_train.append(p[i])
            delete_id.append(i)
    for i in range(len(delete_id)):
        x_unlabel = np.delete(x_unlabel, (i), axis = 0)
    add_x_train = np.array(add_x_train)
    add_y_train = np.array(add_y_train)
    x_train = np.concatenate(( x_train, add_x_train))
    y_train = np.concatenate(( y_train, add_y_train))
    datagen.fit(x_train)

    # fit the model on the batches generated by datagen.flow()
    model.fit_generator(datagen.flow(x_train, y_train,
                        batch_size=batch_size),
                        samples_per_epoch=x_train.shape[0],
                        nb_epoch=nb_epoch,
                        validation_data=(x_valid, y_valid))
```

3. Semi-supervised Learning 2 : Auto-encoder

I use a simple DNN as the encoder, and it involves 5 Dense layers, with activation = 'relu'. The model is optimized by SGD.

After encoding, I use K.function to get the output from the forth layer, and take this output as the features. Next, I calculate the mean of these features in each class, and calculate the absolute distances between these mean values and the feature of unlabel data, so I can label the data with the class having min distances. I pick 1000 data from the unlabel data and add them to the training data, and train the whole data set with the CNN model I used in the Supervised Learning method.

DNN model

```
model = Sequential()
model.add( Dense( length, activation = 'relu', input_shape = ( 3 * 32 * 32, ) ) )
```

```
model.add( Dense( length, activation = 'relu' ) )
model.add( Dense( length, activation = 'relu' ) )
model.add( Dense( length, activation = 'relu' ) )
model.add( Dense( 3 * 32 * 32, activation = 'linear' ) )
keras.optimizers.SGD(lr = 1, decay = 1e-6, momentum = 0.9, nesterov = True)
model.compile(loss = 'mse', optimizer = RMSprop(), metrics = ['accuracy'])
model.fit(x_train, x_train, nb_epoch = nb_epoch, batch_size = batch_size,
shuffle=True)
score = model.evaluate(x_valid, x_valid, batch_size = len(x_valid))
```

Clustering -- Calculate distances between unlabel data & each class

```
encode = K.function([model.layers[0].input], [model.layers[3].output])
encode_output = encode([x_train])[0]

class_mean = np.zeros((10, length))
for i in range(4500):     #calculate the mean of the features of each class
    for j in range(length):
        class_mean[i / 450][j] += encode_output[i][j]
        print encode_output[i][j]
        print class_mean[i/450][j]
for i in range (10):
    for j in range(length):
        class_mean[i][j] /= 450.0
encode_output = encode([x_unlabel])[0]
add = [ ]
y_ans = [0 for i in range(10)]
for i in range(1000):     #calculate the differences between unlabel data and class
    diff = [0 for j in range(10) ]
    min_diff = -1
    for j in range(10):
        for k in range(length):
            diff[j] += abs(encode_output[i][k] - class_mean[j][k])
    for j in range(10):
        if diff[j] < min_diff or min_diff == -1:
            min_diff = diff[j]
            ans = j
    add.append((i, ans, min_diff))
add.sort(key = lambda x : -x[2])
```

```
#add unlabel data to train data
add_x_train = [ ]
add_y_train = [ ]
for i in range( add_size ) :
    tmp_y = [ 0.0 for _ in range( 10 ) ]
    tmp_y[ add[ i ][ 1 ] ] = 1.0
    add_x_train.append( all_unlabel[ add[ i ][ 0 ] ] )
    add_y_train.append( tmp_y )
add_x_train = np.array(add_x_train)
add_y_train = np.array(add_y_train)

x_train = np.concatenate( (x_train, add_x_train) )
y_train = np.concatenate( (y_train, add_y_train) )
```

4. Compare and Analyze Your Results

In Supervised Learning, the results of validation score is 0.63 and the result on training score is 0.9638.

Putting this model into self-training method, the results show that after adding 3000 data with probability larger than 0.9995, the validation score improved from 0.63 to 0.67. I also test the model with adding 5000 data, however, there is no significant improvement on validation score, and the score on public data even get a little bit lower.

The result of auto-encoding is not very good, I only got 0.543 on training score, and 0.57 on validation score.