Machine Learning: HW3 Report

B03901034 吳泓霖

Supervised Learning

The following is the model I used as the main training model:

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
convolution2d_1 (Convolution2D)  (None, 32, 32L, 32L)  896         convolution2d_input_1[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)        (None, 32, 32L, 32L)  0           convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 32, 30L, 30L)  9248        activation_1[0][0]
____________________________________________________________________________________________________
activation_2 (Activation)        (None, 32, 30L, 30L)  0           convolution2d_2[0][0]
____________________________________________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 32, 15L, 15L)  0           activation_2[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 32, 15L, 15L)  0           maxpooling2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 64, 15L, 15L)  18496       dropout_1[0][0]
____________________________________________________________________________________________________
activation_3 (Activation)        (None, 64, 15L, 15L)  0           convolution2d_3[0][0]
____________________________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 64, 13L, 13L)  36928       activation_3[0][0]
____________________________________________________________________________________________________
activation_4 (Activation)        (None, 64, 13L, 13L)  0           convolution2d_4[0][0]
____________________________________________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 64, 6L, 6L)    0           activation_4[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 64, 6L, 6L)    0           maxpooling2d_2[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 2304)          0           dropout_2[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 512)           1180160     flatten_1[0][0]
____________________________________________________________________________________________________
activation_5 (Activation)        (None, 512)           0           dense_1[0][0]
____________________________________________________________________________________________________
dropout_3 (Dropout)              (None, 512)           0           activation_5[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 10)            5130        dropout_3[0][0]
____________________________________________________________________________________________________
activation_6 (Activation)        (None, 10)            0           dense_2[0][0]
====================================================================================================
Total params: 1250858
____________________________________________________________________________________________________
```

With the following parameters:

```
batch_size = 32
nb_classes = 10
nb_epoch = 30
optimizer = 'adam'
```

This model is referenced from the Keras example (cifar10_cnn.py), and the performance is pretty decent.

Semi-Supervised Learning (1)

(Only shows the different part)

```
# Same as supervised learning
...

# Predict the unlabeled data
predicts = model.predict(unlabeled, verbose=1)
# We store the results here: sampled[n] = [data, answer, weight]
sampled = []

for i in range(len(predicts)):
    # These steps are for finding the maximum confidence
    predict = predicts[i]
    max = predict[0]
    assumption = 0
    for j in range(len(predict)):
        if predict[j] > max:
            assumption = j
            max = predict[j]
    if max > threshold:
        index = assumption
        sampled
            .append([unlabeled[i], set_answer([0 for i in range(10)]), max])

# Add the data and the correspond answer into the original data
data = np.concatenate((labeled, map(lambda x: x[0], sampled)), axis=0)
ans = np.concatenate((ans, map(lambda x: x[1], sampled)), axis=0)

# Finally, train again
model.fit(data, ans, batch_size=batch_size, nb_epoch=nb_epoch,
            shuffle=True, sample_weight=weights)

...
```

What the code above will do is train the model with the labeled data, then predict the unlabeled data; finally, add those with high confidence back to the training data as if it is "labeled data", then train again.

Semi-Supervised Learning (2)

For method 2, I used autoencoder for clustering. Here are the differences:

```
...
# Read data
...

encode_layer = 5
encode_width = 512
# Only choose the best 3000 to add back to data
new_item_size = 3000

# Create encode NN
model = Sequential()
model.add(Dense(encode_width, activation='relu', input_shape=(3072, )))
for i in range(encode_layer) :
    model.add(Dense(encode_width, activation='relu'))
model.add(Dense(3072, activation='linear'))
model.compile(loss='mse', optimizer='rmsprop', metrics=[ 'accuracy' ])
# Train labeled-labeled ==> train both encoder and decoder
model.fit(labeled, labeled, batch_size=256, nb_epoch=200, verbose=1,
            validation_data=(labeled, labeled))

# The encoder is the first half
encoder = K.function([model.layers[0].input],
                         [model.layers[(encode_layer + 1) / 2].output])

# Get the clustered unlabled data
encoded = encoder([labeled])[0]

# "before" stores the average code for each labeled data
before = [[0.0 for x in range(encode_width)] for x in range(10)]
for feature in range(encode_width):
    for category in range(10):
        for i in range(500):
            before[category][feature]
                += encoded[category * 500 + i][feature]
        before[category][feature] /= 500

# Find the minimum mean square error to determine which category
after = encoder([unlabeled])[0]
cand_list = []
```

```python
for image_num in range(45000):
    best_category = 0
    min = float('inf')
    for category in range(10):
        mse = 0.0
        for feature in range(encode_width):
            mse += (after[image_num][feature]
                    - before[category][feature]) ** 2
        if mse < min:
            best_category = category
            min = mse
    cand_list.append([image_num, best_category, min])
# Sort it so we can get the best results
cand_list.sort(key = lambda x: x[2])

new_data = []
new_ans = []
for i in range(new_item_size):
    new_data.append(unlabeled[cand_list[i][0]])
    index = cand_list[i][1]
    new_ans.append(set_answer([0 for x in range(10)]))

# Add the new data and answers back to data
data = np.concatenate((labeled, new_data), axis=0)
         .reshape(5000 + new_item_size, 3, 32, 32)
ans = np.concatenate((ans, new_ans), axis=0)

...
# Same as supervised learning
...
```

It will select the best 3000 data from the encoder and add them along with the labeled data, and then train with the same model used in supervised learning.

Comparision

Running time:

`Supervised < Semi-supervised (1) <<< Semi-supervised (2)`

The main difference between method one and method two is that, in method two, we have to process data with CPU (line 29 - 35, 39 - 65), which cannot make good use of GPU acceleration, and the data is actually pretty large, which will slow down the whole process.

Performance (by accuracy) :

`Semi-supervised (1) > Supervised > Semi-supervised (2)`

I haven't spent much time investigating into method two, so I assume that the autoencoder can be improved if effort is put into it.

Things worth noting: the optimizer in semi-supervised learning will affect the result. In trial-and-error, `adam` is the best to use. Also, the threshold used to filter out bad results should be set to a very high value (e.g. 0.98), or the incorrect assumptions will pollute the labeled data, which will lead to bad results.