National Taiwan University

Machine Learning: HW3

EE3 b03901016
陳昊

November 17, 2016

---

1 Supervised Learning

The first thing after read data from pickle file is to reshape it, to match the original (3072,) shape to (3, 32, 32) or (32, 32, 3) depends on different backend, so that the image can be used in a CNN network. Besides, I divide all the value in 255 so that the value in the data is normalize to [0,1].

The model that I got the best public score on Kaggle is a CNN network, which contains 3 convolution layers and 2 maxpooling layers, then connect to 3 fully connected hidden layers, shown below.

In addition to the CNN network, I also use `ImageDataGenerator` to simply generate more training data by randomly shift and rotate and flip the original images, this method increase the accuracy on testing data a lot. Without using ImageDataGenerator the best score I got on Kaggle public set is 0.60880, while I got an accuracy of 0.68080 with this method. However, it needs more epochs when training if I use `ImageDataGenerator`, usually I got about 0.96 training acc in 70 epochs without using it, but need to run about 250 epochs to reach the same training acc when using it. (batch_size = 32)

```
if K.image_dim_ordering() == 'th':
   x_train = x_train.reshape(x_train.shape[0], img_channels, img_rows, img_cols)
   x_test = x_test.reshape(x_test.shape[0], img_channels, img_rows, img_cols)
   input_shape = (img_channels, img_rows, img_cols)
else:
   x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, img_channels)
   x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, img_channels)
   input_shape = (img_rows, img_cols, img_channels)

x_train = x_train.astype('float32') / 255
y_train = y_train.astype('float32')
x_test = x_test.astype('float32') / 255

# define model
model = Sequential()
```

```
model.add(Convolution2D(32, 3, 3, input_shape = input_shape))
model.add(Activation('relu'))

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

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



model.add(Flatten())
model.add(Dense(output_dim = 512))
model.add(Activation('relu'))
model.add(Dense(output_dim = 256))
model.add(Activation('relu'))
model.add(Dense(output_dim = 128))
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(output_dim = nb_classes))
model.add(Activation('softmax'))

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

if data_augmentation == True:
   print "Data augmentation..."
   datagen = ImageDataGenerator(
          featurewise_center = False, # set input mean to 0 over the dataset
          samplewise_center = False, # set each sample mean to 0
          featurewise_std_normalization = False, # divide inputs by std of the dataset
          samplewise_std_normalization = False, # devide each input by its std
          zca_whitening = False, # apply ZCA whitening
          rotation_range = 12, # randomly rotate images in range
          width_shift_range = 0.1, # randomly shift images horizontally
          height_shift_range = 0.1, # randomly shift images vertically
          horizontal_flip = True, # randomly flip images
          vertical_flip = False) # randomly flip images
   datagen.fit(x_train)
   model.fit_generator(datagen.flow(x_train, y_train,
                     batch_size = batch_size),
                     samples_per_epoch = x_train.shape[0],
```

```
                    nb_epoch = nb_epoch)
```

2 Self-Learning

The method I use to implement self-learning is very similar to supervised learning. After training the model for the first round (only used labelled data), I predict all the unlabelled data, and pick the data whose max value in y_train exceed threshold value then put it into x_train, and iterate several times (iteration = 8). Because the last activation layer of the model is `softmax`, so the threshold should be close to 1 (In my implementation, it's setted to 0.98).

Another technique I used in this implementation is reset `nb_epoch` after the first round, so that it won't train too many times.

By using self-learning, I got better score on Kaggle public set, the accuracy increase to 0.69540.

```
nb_classes = 10
batch_size = 128
nb_epoch = 250
encoding_dim = 256
add_size = 5000

# encoder
model = Sequential()
model.add(Dense(encoding_dim, activation = 'relu', input_shape = (3072,)))
model.add(Dense(encoding_dim, activation = 'relu'))
model.add(Dense(encoding_dim, activation = 'relu'))
model.add(Dense(encoding_dim, activation = 'relu'))
model.add(Dense(encoding_dim, activation = 'relu'))
model.add(Dense(encoding_dim, activation = 'relu'))
model.add(Dense(3072, activation = 'linear'))

model.compile(loss = 'mse', optimizer = 'rmsprop', metrics = ['accuracy'])

model.fit(x_train, x_train,
              batch_size = batch_size,
              nb_epoch = nb_epoch,
              verbose = 1,
              validation_data = (x_test, x_test))

encoder = K.function([model.layers[0].input], [model.layers[2].output])
encoded_x_train = encoder([x_train])[0]

# calculate k-means
ave = []
for i in range(nb_classes):
    ave.append([0.0 for m in range(encoding_dim)])

for i in range(nb_classes):
    for idx in range(500):
```

```
        pos = i * 500 + idx
        for j in range(encoding_dim):
            ave[i][j] += encoded_x_train[pos][j]
    for k in range(encoding_dim):
        ave[i][k] /= 500
print 'phase 1'
encoded_ul = encoder([x_ul])[0]
c = []
for i in range(len(x_ul)):
    lb = -1
    m = 1e10
    for j in range(nb_classes):
        mse = 0.0
        for k in range(encoding_dim):
            mse += (encoded_ul[i][k] - ave[j][k]) ** 2
        if mse < m:
            lb = j
            m = mse
    c.append((i, lb, m))
c.sort(key = lambda x: x[2])
print 'phase 2'
new_x = []
new_y = []
for i in range(add_size):
    tmp_y = [0.] * nb_classes
    tmp_y[c[i][1]] = 1.
    new_x.append(x_ul[c[i][0]])
    new_y.append(tmp_y)

new_x = np.array(new_x)
new_y = np.array(new_y)

new_x = new_x.astype('float32') / 255

print 'y_train shape', y_train.shape
print 'new_y shape', new_y.shape

x_train = np.concatenate((x_train, new_x), axis = 0)
y_train = np.concatenate((y_train, new_y), axis = 0)
x_train = x_train.reshape(len(x_train), 3, 32, 32)
```

3 Autoencoder Clustering

First, I design an deep autoencoder to reconstruct the feature of input images, the method I used to construct the autoencoder is just simply add some layers and the output of the model have the same dimension with the input images. Then the output of the middle layer is the encoded data, with new features.

Second, calculate the K-means boundary, and label those unlabelled data. The method I label those unlabelled data is to calculate the distance with the mean of each encoded feature, and pick the closest. then add the unlabelled data to x_train.

Last, simply use the CNN network constructed in supervised learning to train all the data. The best score I got using this method is 0.68140. However, I think this is a good way with high potential, I think this will be better if I had enough time to try more models.

```
iteration = 8
threshold = 0.98

for i in range(iteration):
    if i > 0:
        nb_epoch = 100
    if data_augmentation is True:
        model.fit_generator(datagen.flow(x_train, y_train,
                                 batch_size = batch_size),
                                 samples_per_epoch = x_train.shape[0],
                                 nb_epoch = nb_epoch)
    else:
        model.fit(x_train, y_train,
                  batch_size = batch_size,
                  nb_epoch = nb_epoch,
                  shuffle = True)
    r = model.predict(x_ul)
    tmp_x = []
    tmp_y = []
    t = []
    for j in range(len(r)):
        m, idx = 0, 0
        for k in range(len(r[j])):
            if r[j][k] > m:
                m = r[j][k]
                idx = k
        if m > threshold:
            temp = [0] * nb_classes
            temp[idx] = 1
            tmp_x.append(x_ul[j])
            tmp_y.append(temp)
            t.append(j)
    if len(tmp_x) > 0 and len(tmp_y) > 0:
        tmp_x = np.array(tmp_x)
        tmp_y = np.array(tmp_y)
        x_train = np.concatenate((x_train, tmp_x), axis = 0)
        y_train = np.concatenate((y_train, tmp_y), axis = 0)
    x_ul = np.delete(x_ul, t, axis = 0)
```

4 Result and Analysis

Here I list the best score (Acc) I got on Kaggle public set.

- Supervised (No datagen): 0.60820 (epoch =70, batch_size = 32)
- Supervised (With datagen): 0.68080 (epoch = 250, batch_size = 32)
- Self-learning (No datagen): 0.63960 (epoch = 70, batch_size = 32, iteration = 8)
- Self-learning (With datagen): 0.69540 (epoch = 250, batch_size = 32, iteration = 8)
- Autoencoder Clustering (With datagen): 0.68140 (epoch = 250, batch_size = 128, add 5000 unlabelled)

I found that self-learning is useful, mostly have enhancement of 3% to 5%, and I think autoencoder is also a useful method if I add all the unlabelled data to x_train and train more epochs, it may get even better result.