Machine Learning

Homework 3

B02901143 楊筑雅

2016/11/18

I. Supervised Learning

1. Network Architecture

The network architecture mainly includes 6 convolution layers (64, 64, 128, 128, 256, 256) and 2 fully-connected layers (512, 10). Each convolution layer uses 3x3 kernels. Behind every convolutional or fully-connected layer are a batch normalization layer and an activation layer. In addition, behind every two convolutional layers are a max pooling layer and a dropout layer.

```
model = Sequential()

model.add(Convolution2D(64, 3, 3, input_shape=(3, 32, 32),
border_mode='same'))
model.add(BatchNormalization(axis=1))
model.add(Activation('relu'))

model.add(Convolution2D(64, 3, 3, border_mode='same'))
model.add(BatchNormalization(axis=1))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Convolution2D(128, 3, 3, border_mode='same'))
model.add(BatchNormalization(axis=1))
model.add(Activation('relu'))

model.add(Convolution2D(128, 3, 3, border_mode='same'))
model.add(BatchNormalization(axis=1))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Convolution2D(256, 3, 3, border_mode='same'))
model.add(BatchNormalization(axis=1))
model.add(Activation('relu'))

model.add(Convolution2D(256, 3, 3, border_mode='same'))
model.add(BatchNormalization(axis=1))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, init='normal'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(10, init='normal'))
model.add(BatchNormalization())
model.add(Activation('softmax'))
```

2. Batch Normalization
   Batch normalization layer normalizes the previous layer at each batch, and leads to faster learning and higher accuracy.

3. Dropout
   Dropout layer randomly drops partial units of the previous layer , and reducing overfitting. I use 0.25 for the fraction of the input units to drop in the network.

4. Data Augmentation
   In order to learn a better model, I use image data generator to augment data by rotating, shifting and horizontally flipping original data. I also set samples per epoch to 5 times the number of original data.

```
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.125,
    height_shift_range=0.125,
    horizontal_flip=True)

datagen.fit(train_x)

model.fit_generator(datagen.flow(train_x, train_y,
batch_size=100), samples_per_epoch=len(train_x)*5,
nb_epoch=50, validation_data=(val_x, val_y),
callbacks=[modelcheckpoint, earlystopping])
```

5. Results
   I get the accuracy of 67.34% and 69.22% for pubic testing data and private testing data respectively.

II. Semi-supervised Learning — Self-training

1. Method
   First, I use trained model by supervised learning to predict 45000 unlabeled data. Second, I labeled data that with class probability > 0.999 and added them into training data. Third, I trained the model again. Then I jumped back to the first to the third step again and again to make the model better and better.

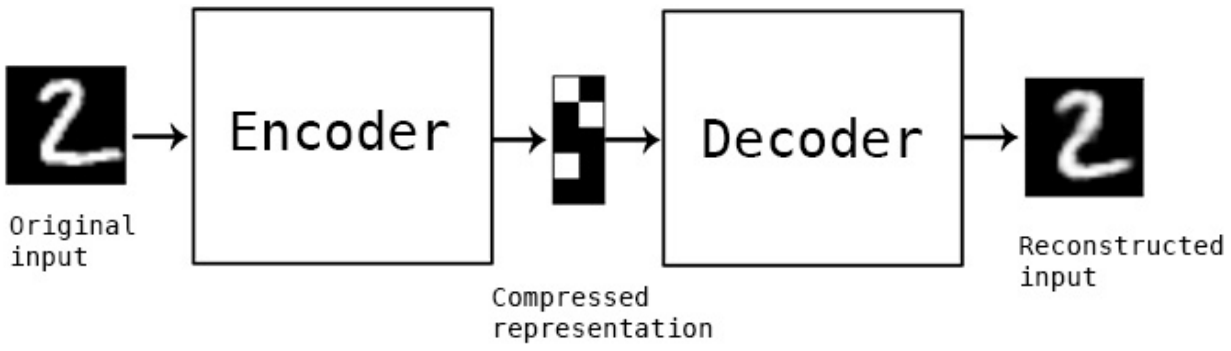
```
for it in range(5):
    model.load_weights(model_file)
    unlabel_proba = model.predict_proba(unlabel_x,
                    batch_size=100)
    new_label_dict = {}
    for i in range(10):
        new_label_dict[i] = []
    for index, label in enumerate(unlabel_proba):
        proba_max = label[0]
        for cla, proba in enumerate(label):
            if proba > proba_max:
                proba_max = proba
        if proba_max > 0.999:
            new_label_dict[cla].append(index)
    new_label_indexes = []
    for cla in range(10):
        new_label_indexes += new_label_dict[cla]
    index_dict = dict(zip(sorted(new_label_indexes),
                 [i for i in range(len(new_label_indexes))]))
    new_label_y = np.zeros((len(new_label_indexes), 10))
    for cla in range(10):
        for index in new_label_dict[cla]:
            new_label_y[index_dict[index], cla] = 1
    new_label_x = unlabel_x[new_label_indexes]
    print "Add", len(new_label_indexes), "data into",
           len(train_x), "data"
    train_x = np.concatenate((train_x, new_label_x))
    train_y = np.concatenate((train_y, new_label_y))
    p = np.random.permutation(len(train_x))
    train_x = train_x[p]
    train_y = train_y[p]
    print "Training", len(train_x), "data..."
    model.fit_generator(datagen.flow(train_x, train_y,
          batch_size=100), samples_per_epoch=len(train_x)*5,
          nb_epoch=25, validation_data=(val_x, val_y),
          callbacks=[modelcheckpoint])
```

2. Results

   I get the accuracy of 79.28% for validation data, which is about 10% higher than the one using supervised learning only . (I finished training after kaggle deadline, so I did not have this number on kaggle.)

III. Semi-supervised learning — Variational Autoencoder and Self-training

1. Variational Autoencoder (VAE)



Autoencoding does data compression, and variational autoencoder is the kind that learns a latent variable model for its input data. That is, the encoder network turns the input into two parameters (z_mean and z_log_sigma) in a latent space, and the decoder network turns the latent points back into the reconstructed input.

```
h = Dense(intermediate_dim, activation='relu')(x)
z_mean = Dense(latent_dim)(h)
z_log_var = Dense(latent_dim)(h)

def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(batch_size, latent_dim),
                              mean=0., std=epsilon_std)
    return z_mean + K.exp(z_log_var / 2) * epsilon

z = Lambda(sampling, output_shape=(latent_dim,))([z_mean,
                                                    z_log_var])

decoder_h = Dense(intermediate_dim, activation='relu')
decoder_mean = Dense(original_dim, activation='sigmoid')
h_decoded = decoder_h(z)
x_decoded_mean = decoder_mean(h_decoded)

def vae_loss(x, x_decoded_mean):
    xent_loss = original_dim *
            objectives.binary_crossentropy(x, x_decoded_mean)
    kl_loss = - 0.5 * K.sum(1 + z_log_var - K.square(z_mean)
            - K.exp(z_log_var), axis=-1)
    return xent_loss + kl_loss

vae = Model(x, x_decoded_mean)
vae.compile(optimizer='rmsprop', loss=vae_loss)
```

2. Method

   To begin with, I train VAE to learn robust parameters. Second, I build an encoder with those parameters to encode data and compress data.

```
# build a model to project inputs on the latent space
encoder = Model(x, z_mean)
# encode data
train_x = encoder.predict(train_x, batch_size=batch_size)
val_x = encoder.predict(val_x, batch_size=batch_size)
unlabel_x = encoder.predict(all_unlabel_x,
                            batch_size=batch_size)
```

   Then I use the encoded data to train a neural network with 3 fully-connected layers. Besides, I also use self-training to make use of unlabeled data. (I save two models. One for the neural network and one for the encoder.)

3. Results

   I get the accuracy of 20% for both supervised learning and semi-supervised learning, which is only a bit better than guessing (10%). In other words, self-training nearly does not help.

IV. Discussion

From I and II, the accuracy of CNN improves a lot by using self-training. As for III, the accuracy is quite low compared with I and II. However, the speed is far faster than I and II since it has simpler architecture and less parameters.