MACHINE LEARNING HW3

Q1 CNN

```
# input image dimensions
img_rows, img_cols = 32, 32
# the CIFAR10 images are RGB
img_channels = 3

training_X = training_X.astype('float32') / 255
testing_X = testing_X.astype('float32') / 255
training_Y = np_utils.to_categorical(training_Y, nb_classes)
testing_Y = np_utils.to_categorical(testing_Y, nb_classes)

print("Training data shape: {}".format(training_X.shape))
print('# of train samples: {}'.format(len(training_Y)))
print('# of test samples: {}'.format(len(testing_Y)))

# -----initiate layers-----
model = Sequential()
model.add(Convolution2D(32, 3, 3, activation='relu', border_mode='same', input_shape=(img_channels, img_rows, img_cols),
                        dim_ordering='th', W_regularizer='l2', b_regularizer='l2'))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

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

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes, activation='softmax'))
# -----start-----
# print(model.summary())
# adam = Adam(lr=0.01, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])

checkpointer = ModelCheckpoint(filepath='cifar10_cnn_keras_model.hdf5', monitor='val_acc', verbose=1, save_best_only=True)
earlystopping = EarlyStopping(monitor='val_loss', patience=int(nb_epoch*0.05), verbose=0)

# this will do preprocessing and realtime 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,  # divide each input by its std
    zca_whitening=False,  # apply ZCA whitening
    rotation_range=0,  # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
    horizontal_flip=True  # randomly flip images
```

```
Epoch 00183: val_acc improved from 0.71800 to 0.72200, saving model to cifar10_cnn_keras_model.hdf5
1s - loss: 0.4083 - acc: 0.8676 - val_loss: 1.0661 - val_acc: 0.7220
```

Q2 CNN WITH SELF TRAINING

Same structure of CNN, but insert unlabeled data back into training dataset after first training, then retrain a new model

```
# ===== self training =====
for turns in range(1, 5):
    # -----(Re)train with CNN-----
    CNN(training_x, training_y, testing_x, testing_y, 5000*turns)
    # =====unlabel prediction after initial model trained=====
    from keras.models import load_model
    model = load_model('cifar10_cnn_keras_model.hdf5')
    predict_probability = model.predict_proba(unlabel_data, batch_size=32, verbose=1)
    predict_class = np.argmax(predict_probability, axis=1)
    predict_entropy = scipy.stats.entropy(predict_probability.T)
    # -----extract 25% most ensure prediction from unlabel data-----
    order = np.argsort(predict_entropy, axis=0)
    predict_probability = predict_probability[order]
    predict_entropy = predict_entropy[order]
    predict_class = predict_class[order]
    unlabel_data = unlabel_data[order]

    pseudo_label_data_list = []

    for idx, i in enumerate(predict_entropy):
        if i <= predict_entropy.mean() and (len(pseudo_label_data_list) < len(training_y)*0.5):
            pseudo_label_data_list.append(idx)
    pseudo_unlabel_x = unlabel_data[pseudo_label_data_list]
    pseudo_unlabel_y = predict_class[pseudo_label_data_list]
    unlabel_data = np.delete(unlabel_data, pseudo_label_data_list, axis=0)
    # -----add dataset into training data-----
    training_x = np.vstack((training_x, pseudo_unlabel_x))
    training_y = np.vstack((training_y, pseudo_unlabel_y.reshape(-1, 1)))
```

Although the dataset is larger, the validation accuracy does not improved. There may contain wrong concept in the unlabeled data from the first prediction, even only confidence prediction is selected into training dataset. The wrong concept may reinforce itself.

Ref: [Online] Semi-supervised Learning pp:6-7 - https://www.cs.utah.edu/~piyush/teaching/8-11-print.pdf

Q3 AUTOENCODER FOR CLUSTERING

```
def autoencode(training_X, nb_epoch=500, filepath='cifar10_autoencoder_model.hdf5'):
    from keras.layers import Input, Dense, Convolution2D, MaxPooling2D, UpSampling2D
    from keras.optimizers import Adam
    from keras.models import Model
    from keras.layers import PReLU
    from keras.regularizers import l1
    from keras.callbacks import ModelCheckpoint, EarlyStopping

    batch_size = int(len(training_X)/10)
    input_img = Input(shape=(3, 32, 32))

    x = Convolution2D(32, 3, 3, border_mode='same', subsample=(2, 2), W_regularizer=l1(0.01))(input_img)
    x = PReLU(init='zero', weights=None)(x)
    x = MaxPooling2D((2, 2), border_mode='same')(x)

    x = Convolution2D(16, 3, 3, border_mode='same')(x)
    x = PReLU()(x)
    encoded = MaxPooling2D((2, 2), border_mode='same')(x)

    x = UpSampling2D((2, 2))(encoded)
    x = Convolution2D(16, 3, 3, border_mode='same')(x)
    x = PReLU()(x)
    x = UpSampling2D((2, 2))(x)

    x = Convolution2D(32, 3, 3, border_mode='same')(x)
    x = PReLU()(x)
    x = UpSampling2D((2, 2))(x)

    decoded = Convolution2D(3, 3, 3, activation='sigmoid', border_mode='same')(x)

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

    autoencoder = Model(input_img, decoded)
    encoder_model = Model(input_img, encoded)
    autoencoder.compile(loss='mse', optimizer='Adam', metrics=['mean_squared_error'])

    checkpointer = ModelCheckpoint(filepath=filepath, verbose=1, save_best_only=True)
    earlystopping = EarlyStopping(monitor='val_loss', patience=int(nb_epoch*0.1), verbose=1)

    autoencoder.fit(training_X, training_X, batch_size=batch_size, nb_epoch=nb_epoch,
                    verbose=2, callbacks=[checkpointer, earlystopping], validation_split=0.05, validation_data=None,
                    shuffle=True, class_weight=None, sample_weight=None)

    return encoder_model
```

Use encoded training data for training the K-Mean clustering, however, it doesn’t work well.