cnn

April 23, 2019

1 Home 3: Build a CNN for image recognition.

1.0.1 Name: Yao Xiao

1.1 1. Data preparation

1.1.1 1.1. Load data

```
In [2]: from keras.datasets import cifar10
        import numpy as np

        (x_train, y_train), (x_test, y_test) = cifar10.load_data()

        print('shape of x_train: ' + str(x_train.shape))
        print('shape of y_train: ' + str(y_train.shape))
        print('shape of x_test: ' + str(x_test.shape))
        print('shape of y_test: ' + str(y_test.shape))
        print('number of classes: ' + str(np.max(y_train) - np.min(y_train) + 1))
```

```
shape of x_train: (50000, 32, 32, 3)
shape of y_train: (50000, 1)
shape of x_test: (10000, 32, 32, 3)
shape of y_test: (10000, 1)
number of classes: 10
```

1.1.2 1.2. One-hot encode the labels

In the input, a label is a scalar in $\{0, 1, \cdots, 9\}$. One-hot encode transform such a scalar to a 10-dim vector. E.g., a scalar `y_train[j]=3` is transformed to the vector `y_train_vec[j]=[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]`.

1. Define a function `to_one_hot` that transforms an $n \times 1$ array to a $n \times 10$ matrix.

2. Apply the function to `y_train` and `y_test`.

```
In [3]: def to_one_hot(y, num_class=10):
            result = np.zeros((len(y), num_class))
            for i,label in enumerate(y):
```

```
            result[i,label]=1
        return result
    y_train_vec = to_one_hot(y_train)
    y_test_vec = to_one_hot(y_test)

    print('Shape of y_train_vec: ' + str(y_train_vec.shape))
    print('Shape of y_test_vec: ' + str(y_test_vec.shape))

    print(y_train[0])
    print(y_train_vec[0])
```

```
Shape of y_train_vec: (50000, 10)
Shape of y_test_vec: (10000, 10)
[6]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

Remark: the outputs should be

- Shape of y_train_vec: (50000, 10)
- Shape of y_test_vec: (10000, 10)
- [6]
- [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]

1.1.3 1.3. Randomly partition the training set to training and validation sets

Randomly partition the 50K training samples to 2 sets: * a training set containing 40K samples * a validation set containing 10K samples

```
In [4]: rand_indices = np.random.permutation(50000)
        train_indices = rand_indices[0:40000]
        valid_indices = rand_indices[40000:50000]

        x_val = x_train[valid_indices, :]
        y_val = y_train_vec[valid_indices, :]

        x_tr = x_train[train_indices, :]
        y_tr = y_train_vec[train_indices, :]

        print('Shape of x_tr: ' + str(x_tr.shape))
        print('Shape of y_tr: ' + str(y_tr.shape))
        print('Shape of x_val: ' + str(x_val.shape))
        print('Shape of y_val: ' + str(y_val.shape))
```

```
Shape of x_tr: (40000, 32, 32, 3)
Shape of y_tr: (40000, 10)
Shape of x_val: (10000, 32, 32, 3)
Shape of y_val: (10000, 10)
```

1.2 2. Build a CNN and tune its hyper-parameters

1. Build a convolutional neural network model
2. Use the validation data to tune the hyper-parameters (e.g., network structure, and optimization algorithm)
3. Try to achieve a validation accuracy as high as possible.

```
In [4]: from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
        from keras.models import Sequential

        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dense(10, activation='softmax'))

        model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               524416
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 545,098
Trainable params: 545,098
Non-trainable params: 0
_________________________________________________________________
```

```
In [5]: from keras import optimizers

        learning_rate = 1E-5 # to be tuned!

        model.compile(loss='categorical_crossentropy',
```

```
                     optimizer=optimizers.RMSprop(lr=learning_rate),
                     metrics=['acc'])
```

In [6]:
```
history = model.fit(x_tr, y_tr, batch_size=32, epochs=10, validation_data=(x_val, y_val
```

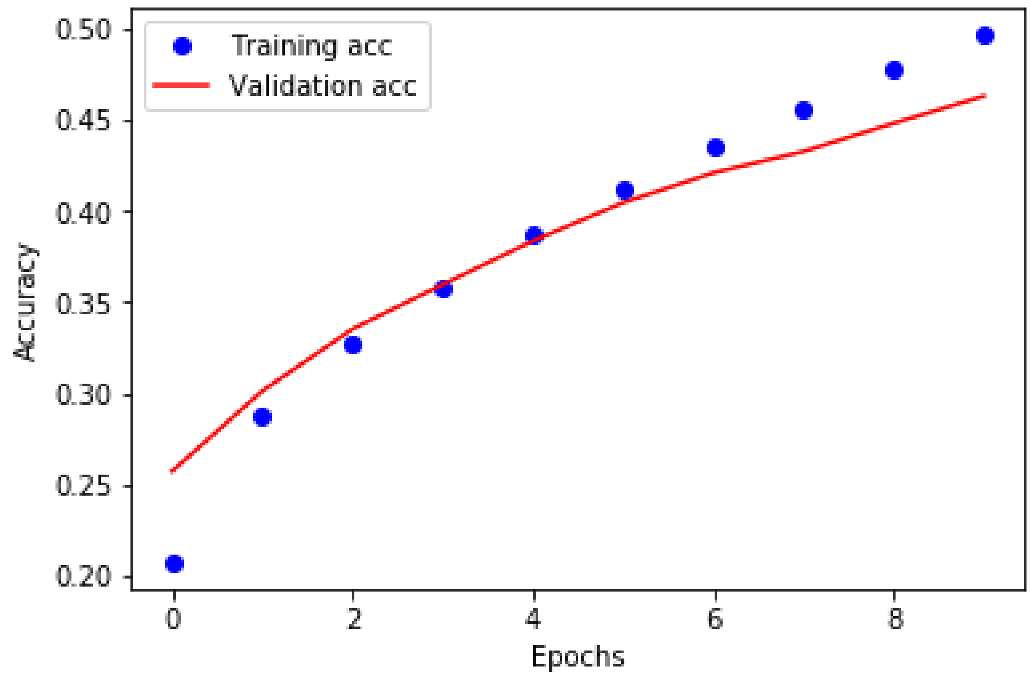
```
Train on 40000 samples, validate on 10000 samples
Epoch 1/10
40000/40000 [==============================] - 36s 905us/step - loss: 8.2007 - acc: 0.2073 - va
Epoch 2/10
40000/40000 [==============================] - 39s 964us/step - loss: 4.9410 - acc: 0.2873 - va
Epoch 3/10
40000/40000 [==============================] - 39s 970us/step - loss: 3.2000 - acc: 0.3265 - va
Epoch 4/10
40000/40000 [==============================] - 39s 972us/step - loss: 2.5738 - acc: 0.3583 - va
Epoch 5/10
40000/40000 [==============================] - 39s 975us/step - loss: 2.2260 - acc: 0.3864 - va
Epoch 6/10
40000/40000 [==============================] - 39s 982us/step - loss: 1.9925 - acc: 0.4120 - va
Epoch 7/10
40000/40000 [==============================] - 39s 986us/step - loss: 1.8287 - acc: 0.4350 - va
Epoch 8/10
40000/40000 [==============================] - 40s 988us/step - loss: 1.7027 - acc: 0.4558 - va
Epoch 9/10
40000/40000 [==============================] - 40s 990us/step - loss: 1.6063 - acc: 0.4773 - va
Epoch 10/10
40000/40000 [==============================] - 40s 993us/step - loss: 1.5326 - acc: 0.4964 - va
```

In [7]:
```
import matplotlib.pyplot as plt
%matplotlib inline

acc = history.history['acc']
val_acc = history.history['val_acc']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



1.3 3. Train (again) and evaluate the model

- To this end, we have found the "best" hyper-parameters.
- Now, fix the hyper-parameters and train the network on the entire training set (all the 50K training samples)
- Evaluate the model on the test set.

1.3.1 3.1. Train the model on the entire training set

Why? Previously, we used 40K samples for training; we wasted 10K samples for the sake of hyper-parameter tuning. Now we already know the hyper-parameters, so why not using all the 50K samples for training?

```
In [28]: from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation
         from keras.models import Sequential
         from keras import layers
         model1 = Sequential()
         model1.add(layers.Conv2D(32, (3, 3), padding='same', input_shape=(32, 32, 3)))
         model1.add(layers.BatchNormalization())
         model1.add(Activation('relu'))
         model1.add(layers.MaxPooling2D((2, 2)))
         model1.add(layers.Conv2D(64, (3, 3), padding='same'))
         model1.add(layers.BatchNormalization())
         model1.add(Activation('relu'))
         model1.add(layers.MaxPooling2D((2, 2)))
```

```python
model1.add(layers.Conv2D(128, (3, 3), padding='same'))
model1.add(layers.BatchNormalization())
model1.add(Activation('relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Flatten())
model1.add(layers.Dropout(0.5))
model1.add(layers.Dense(512))
model1.add(layers.BatchNormalization())
model1.add(Activation('relu'))
model1.add(layers.Dense(10, activation='softmax'))

model1.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_32 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization_29 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
activation_29 (Activation)   (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_30 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
activation_30 (Activation)   (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_31 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
activation_31 (Activation)   (None, 8, 8, 128)         0
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
flatten_10 (Flatten)         (None, 2048)              0
_________________________________________________________________
dropout_9 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_16 (Dense)             (None, 512)               1049088
_________________________________________________________________
batch_normalization_32 (Batc (None, 512)               2048
_________________________________________________________________
```

```
activation_32 (Activation)   (None, 512)              0
_________________________________________________________________
dense_17 (Dense)             (None, 10)               5130
=================================================================
Total params: 1,150,410
Trainable params: 1,148,938
Non-trainable params: 1,472
_________________________________________________________________
```

```
In [29]: from keras import optimizers

         learning_rate = 3E-4 # to be tuned!
         model1.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=learni
```

```
In [30]: history = model1.fit(x_train, y_train_vec, batch_size=128, epochs=50)
```

```
Epoch 1/50
50000/50000 [==============================] - 132s 3ms/step - loss: 1.4575 - acc: 0.4771
Epoch 2/50
50000/50000 [==============================] - 129s 3ms/step - loss: 1.1073 - acc: 0.6070
Epoch 3/50
50000/50000 [==============================] - 129s 3ms/step - loss: 0.9777 - acc: 0.6540
Epoch 4/50
50000/50000 [==============================] - 127s 3ms/step - loss: 0.8880 - acc: 0.6880
Epoch 5/50
50000/50000 [==============================] - 118s 2ms/step - loss: 0.8225 - acc: 0.7113
Epoch 6/50
50000/50000 [==============================] - 119s 2ms/step - loss: 0.7734 - acc: 0.7268
Epoch 7/50
50000/50000 [==============================] - 129s 3ms/step - loss: 0.7264 - acc: 0.7449
Epoch 8/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.6924 - acc: 0.7574
Epoch 9/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.6564 - acc: 0.7680
Epoch 10/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.6276 - acc: 0.7795
Epoch 11/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.5939 - acc: 0.7916
Epoch 12/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.5707 - acc: 0.7979
Epoch 13/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.5450 - acc: 0.8064
Epoch 14/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.5189 - acc: 0.8189
Epoch 15/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.5037 - acc: 0.8240
Epoch 16/50
```

```
50000/50000 [==============================] - 131s 3ms/step - loss: 0.4829 - acc: 0.8293
Epoch 17/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.4656 - acc: 0.8364
Epoch 18/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.4479 - acc: 0.8422
Epoch 19/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.4309 - acc: 0.8477
Epoch 20/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.4112 - acc: 0.8557
Epoch 21/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.3950 - acc: 0.8595
Epoch 22/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.3857 - acc: 0.8655
Epoch 23/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.3663 - acc: 0.8716
Epoch 24/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.3550 - acc: 0.8746
Epoch 25/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.3447 - acc: 0.8797
Epoch 26/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.3318 - acc: 0.8845
Epoch 27/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.3163 - acc: 0.8890
Epoch 28/50
50000/50000 [==============================] - 131s 3ms/step - loss: 0.3040 - acc: 0.8943
Epoch 29/50
50000/50000 [==============================] - 132s 3ms/step - loss: 0.2946 - acc: 0.8971
Epoch 30/50
50000/50000 [==============================] - 132s 3ms/step - loss: 0.2850 - acc: 0.9015
Epoch 31/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.2712 - acc: 0.9059
Epoch 32/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.2618 - acc: 0.9080
Epoch 33/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.2536 - acc: 0.9108
Epoch 34/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.2502 - acc: 0.9133
Epoch 35/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.2386 - acc: 0.9182
Epoch 36/50
50000/50000 [==============================] - 119s 2ms/step - loss: 0.2310 - acc: 0.9193
Epoch 37/50
50000/50000 [==============================] - 123s 2ms/step - loss: 0.2251 - acc: 0.9211
Epoch 38/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.2164 - acc: 0.9245
Epoch 39/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.2133 - acc: 0.9263
Epoch 40/50
```

```
50000/50000 [==============================] - 130s 3ms/step - loss: 0.2024 - acc: 0.9292
Epoch 41/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.1999 - acc: 0.9303
Epoch 42/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.1955 - acc: 0.9320
Epoch 43/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.1895 - acc: 0.9334
Epoch 44/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.1812 - acc: 0.9374
Epoch 45/50
50000/50000 [==============================] - 132s 3ms/step - loss: 0.1753 - acc: 0.9394
Epoch 46/50
50000/50000 [==============================] - 132s 3ms/step - loss: 0.1729 - acc: 0.9402
Epoch 47/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.1651 - acc: 0.9431
Epoch 48/50
50000/50000 [==============================] - 130s 3ms/step - loss: 0.1638 - acc: 0.9433
Epoch 49/50
50000/50000 [==============================] - 190s 4ms/step - loss: 0.1593 - acc: 0.9452
Epoch 50/50
50000/50000 [==============================] - 132s 3ms/step - loss: 0.1594 - acc: 0.9447
```

1.3.2 3.2. Evaluate the model on the test set

In [32]:
```
loss_and_acc = model1.evaluate(x_test, y_test_vec)
print('loss = ' + str(loss_and_acc[0]))
print('accuracy = ' + str(loss_and_acc[1]))
```

```
10000/10000 [==============================] - 5s 471us/step
loss = 0.6779227051258088
accuracy = 0.8011
```

1.4 Data augmentation

In [9]:
```
from keras.preprocessing.image import ImageDataGenerator

datagen_train = ImageDataGenerator(
width_shift_range = 0.1,
height_shift_range = 0.1,
horizontal_flip = True)

datagen_train.fit(x_train)
```

In [21]:
```
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation
from keras.models import Sequential
from keras import layers
model1 = Sequential()
```

```python
model1.add(layers.Conv2D(32, (3, 3), padding='same', input_shape=(32, 32, 3)))
model1.add(layers.BatchNormalization())
model1.add(Activation('relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Conv2D(64, (3, 3), padding='same'))
model1.add(layers.BatchNormalization())
model1.add(Activation('relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Conv2D(128, (3, 3), padding='same'))
model1.add(layers.BatchNormalization())
model1.add(Activation('relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Flatten())
model1.add(layers.Dropout(0.5))
model1.add(layers.Dense(512))
model1.add(layers.BatchNormalization())
model1.add(Activation('relu'))
model1.add(layers.Dense(10, activation='softmax'))

model1.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization_9 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
activation_9 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_10 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
activation_10 (Activation)   (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_11 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
activation_11 (Activation)   (None, 8, 8, 128)         0
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
```

```
flatten_5 (Flatten)          (None, 2048)              0
_________________________________________________________________
dropout_7 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_8 (Dense)              (None, 512)               1049088
_________________________________________________________________
batch_normalization_12 (Batc (None, 512)               2048
_________________________________________________________________
activation_12 (Activation)   (None, 512)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                5130
=================================================================
Total params: 1,150,410
Trainable params: 1,148,938
Non-trainable params: 1,472
_________________________________________________________________
```

In [22]:
```python
model1.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accura
```

In [17]:
```python
from keras.callbacks import ModelCheckpoint

batch_size = 32
checkpoint = ModelCheckpoint(filepath='MLP.weights.best.hdf5', verbose=1, save_best_o
#history = model2.fit(x_train, y_train_vec, batch_size=128, epochs=10)
```

In [28]:
```python
model1.fit_generator(datagen_train.flow(x_train, y_train_vec, batch_size=32),
                     steps_per_epoch=x_train.shape[0] // batch_size,
                     epochs = 25,
                     verbose=2,
                     callbacks=[checkpoint],
                     validation_data=(x_test, y_test_vec),
                     validation_steps=x_test.shape[0] // batch_size)
```

```
Epoch 1/25
 - 156s - loss: 0.7768 - acc: 0.7360 - val_loss: 0.7090 - val_acc: 0.7625

Epoch 00001: val_loss improved from 0.71476 to 0.70901, saving model to MLP.weights.best.hdf5
Epoch 2/25
 - 157s - loss: 0.7601 - acc: 0.7439 - val_loss: 0.7014 - val_acc: 0.7745

Epoch 00002: val_loss improved from 0.70901 to 0.70140, saving model to MLP.weights.best.hdf5
Epoch 3/25
 - 155s - loss: 0.7428 - acc: 0.7480 - val_loss: 1.3019 - val_acc: 0.6793

Epoch 00003: val_loss did not improve from 0.70140
Epoch 4/25
 - 153s - loss: 0.7327 - acc: 0.7529 - val_loss: 0.7865 - val_acc: 0.7423
```

```
Epoch 00004: val_loss did not improve from 0.70140
Epoch 5/25
 - 153s - loss: 0.7190 - acc: 0.7571 - val_loss: 0.8275 - val_acc: 0.7400

Epoch 00005: val_loss did not improve from 0.70140
Epoch 6/25
 - 154s - loss: 0.7077 - acc: 0.7611 - val_loss: 0.8883 - val_acc: 0.7056

Epoch 00006: val_loss did not improve from 0.70140
Epoch 7/25
 - 153s - loss: 0.7028 - acc: 0.7643 - val_loss: 0.6701 - val_acc: 0.7832

Epoch 00007: val_loss improved from 0.70140 to 0.67005, saving model to MLP.weights.best.hdf5
Epoch 8/25
 - 148s - loss: 0.6931 - acc: 0.7665 - val_loss: 0.6966 - val_acc: 0.7820

Epoch 00008: val_loss did not improve from 0.67005
Epoch 9/25
 - 154s - loss: 0.6881 - acc: 0.7696 - val_loss: 0.8197 - val_acc: 0.7315

Epoch 00009: val_loss did not improve from 0.67005
Epoch 10/25
 - 157s - loss: 0.6801 - acc: 0.7720 - val_loss: 0.7311 - val_acc: 0.7605

Epoch 00010: val_loss did not improve from 0.67005
Epoch 11/25
 - 154s - loss: 0.6689 - acc: 0.7749 - val_loss: 0.6025 - val_acc: 0.8047

Epoch 00011: val_loss improved from 0.67005 to 0.60250, saving model to MLP.weights.best.hdf5
Epoch 12/25
 - 162s - loss: 0.6627 - acc: 0.7766 - val_loss: 0.6843 - val_acc: 0.7736

Epoch 00012: val_loss did not improve from 0.60250
Epoch 13/25
 - 159s - loss: 0.6610 - acc: 0.7790 - val_loss: 0.8087 - val_acc: 0.7282

Epoch 00013: val_loss did not improve from 0.60250
Epoch 14/25
 - 149s - loss: 0.6517 - acc: 0.7812 - val_loss: 0.6933 - val_acc: 0.7773

Epoch 00014: val_loss did not improve from 0.60250
Epoch 15/25
 - 148s - loss: 0.6467 - acc: 0.7826 - val_loss: 0.5933 - val_acc: 0.8062

Epoch 00015: val_loss improved from 0.60250 to 0.59333, saving model to MLP.weights.best.hdf5
Epoch 16/25
 - 152s - loss: 0.6402 - acc: 0.7859 - val_loss: 0.6885 - val_acc: 0.7700
```

```
Epoch 00016: val_loss did not improve from 0.59333
Epoch 17/25
 - 153s - loss: 0.6360 - acc: 0.7876 - val_loss: 0.7582 - val_acc: 0.7878

Epoch 00017: val_loss did not improve from 0.59333
Epoch 18/25
 - 151s - loss: 0.6338 - acc: 0.7885 - val_loss: 0.6813 - val_acc: 0.7894

Epoch 00018: val_loss did not improve from 0.59333
Epoch 19/25
 - 152s - loss: 0.6293 - acc: 0.7872 - val_loss: 0.5339 - val_acc: 0.8263

Epoch 00019: val_loss improved from 0.59333 to 0.53386, saving model to MLP.weights.best.hdf5
Epoch 20/25
 - 149s - loss: 0.6225 - acc: 0.7912 - val_loss: 0.7426 - val_acc: 0.7630

Epoch 00020: val_loss did not improve from 0.53386
Epoch 21/25
 - 154s - loss: 0.6190 - acc: 0.7937 - val_loss: 0.6205 - val_acc: 0.8052

Epoch 00021: val_loss did not improve from 0.53386
Epoch 22/25
 - 152s - loss: 0.6120 - acc: 0.7960 - val_loss: 0.6579 - val_acc: 0.8069

Epoch 00022: val_loss did not improve from 0.53386
Epoch 23/25
 - 152s - loss: 0.6109 - acc: 0.7956 - val_loss: 0.6266 - val_acc: 0.8027

Epoch 00023: val_loss did not improve from 0.53386
Epoch 24/25
 - 155s - loss: 0.6071 - acc: 0.7969 - val_loss: 0.5392 - val_acc: 0.8273

Epoch 00024: val_loss did not improve from 0.53386
Epoch 25/25
 - 162s - loss: 0.6021 - acc: 0.7992 - val_loss: 0.6713 - val_acc: 0.7905

Epoch 00025: val_loss did not improve from 0.53386
```

```
Out[28]: <keras.callbacks.History at 0xb3c793f98>
```

```
In [30]: model1.load_weights('MLP.weights.best.hdf5')
         loss_and_acc = model1.evaluate(x_test, y_test_vec)
         print('loss = ' + str(loss_and_acc[0]))
         print('accuracy = ' + str(loss_and_acc[1]))
```

```
10000/10000 [==============================] - 5s 499us/step
loss = 0.5338634187221527
```

```
accuracy = 0.8263
```