ex6_sol

June 5, 2019

1 Exercise 6 - Hyperparameter Optimization

This exercise is based on https://github.com/leriomaggio/deep-learning-keras-tensorflow, https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/ and https://github.com/fchollet/keras/blob/master/examples/mnist_sklearn_wrapper.py

We want to build simple CNN models to classify the MNIST dataset and uses sklearn's GridSearchCV to find the best hyperparameter model

1.1 Data Preparation

Let's first load and preprocess the data as we did in exercise 5:

In [1]:
```
#Import the required libraries
from keras import backend as K
import numpy as np
from keras.utils import np_utils
from keras.datasets import mnist
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

np.random.seed(1338)  # for reproducibilty!!

# input image dimensions
img_rows, img_cols = 28, 28
# number of classes
nb_classes = 10

#Data format
if K.image_data_format() == 'channels_first':
    shape_ord = (1, img_rows, img_cols)
else:  # channel_last
    shape_ord = (img_rows, img_cols, 1)

#Load the data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
        #Scale the data and convert to data format
        scaler = MinMaxScaler(feature_range=(0, 1))
        X_train = X_train.astype('float32')
        X_test = X_test.astype('float32')
        X_train = scaler.fit_transform(X_train.reshape(X_train.shape[0],img_rows*img_cols))
        X_test = scaler.transform(X_test.reshape(X_test.shape[0],img_rows*img_cols))
        X_train = X_train.reshape(X_train.shape[0],img_rows,img_cols)
        X_test = X_test.reshape(X_test.shape[0],img_rows,img_cols)
        X_train = X_train.reshape((X_train.shape[0],) + shape_ord)
        X_test = X_test.reshape((X_test.shape[0],) + shape_ord)

        #convert target vector
        Y_train = np_utils.to_categorical(y_train, nb_classes)
        Y_test = np_utils.to_categorical(y_test, nb_classes)

        #Take just 20k example for training for speed reasons
        X_train = X_train[:20000]
        Y_train = Y_train[:20000]
```

```
Using TensorFlow backend.
```

1.2 How to use Keras Models in scikit-learn

Keras models can be used in scikit-learn by wrapping them with the KerasClassifier or KerasRegressor class.

To use these wrappers you must define a function that creates and returns your Keras sequential model, then pass this function to the build_fn argument when constructing the KerasClassifier class.

You can learn more about the scikit-learn wrapper in Keras API documentation:
https://keras.io/scikit-learn-api/

1.3 Build Model

We will define a function which builds a similar model we used in exercise 5, but depends on all hyperparameter we would like to tune:

```
In [2]: from keras.models import Sequential
        from keras.layers import Dense, Dropout, Activation, Flatten
        from keras.layers import Conv2D, MaxPooling2D
        from keras.wrappers.scikit_learn import KerasClassifier
```

```
In [3]: def make_model(dense_layer_sizes, dense_activation, filters,
                       kernel_size, pool_size, padding_type, stride_size, dropout_rate, optimi
            '''Creates model comprised of 2 convolutional layers followed by dense layers

            dense_layer_sizes: List of layer sizes. This list has one number for each layer
            dense_activation: activation funciton in dense layer
            filters: Number of convolutional filters in each convolutional layer
```

```
    kernel_size: Convolutional kernel size
    pool_size: Size of pooling area for max pooling
    padding_type: type of padding: same or valid
    stride_size: symmetric stride size
    dropout_rate: dropout rate
    optimizer: optimizer used for mimizing
    '''

    model = Sequential()

    model.add(Conv2D(filters, (kernel_size, kernel_size), padding=padding_type,
                     strides=(stride_size, stride_size), activation='relu', input_shape
    model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))
    model.add(Dropout(dropout_rate))
    model.add(Flatten())
    for layer_size in dense_layer_sizes:
        model.add(Dense(layer_size, activation=dense_activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(nb_classes))
    model.add(Activation('softmax'))

    model.compile(loss='categorical_crossentropy', optimizer=optimizer,metrics=['accura

    return model

In [4]: my_cnn = KerasClassifier(make_model)
```

Note: Not even all hyperparameter are included here, we are not varying things like the neural network weight initialization

`model.add(Dense(layer_size, activation=dense_activation, kernel_initializer=init_mode))` with `init_mode = ['uniform', 'lecun_uniform', 'normal', 'zero', 'glorot_normal', 'glorot_uniform', 'he_normal', 'he_uniform']`

or we have not included kernel regularizer like l1/l2 for which we would need to change the strength. The parameter of the optimizer are also not yet included, but we will do that later.

1.4 How to use Grid Search in scikit-learn

Grid search is a model hyperparameter optimization technique. More information on hyperparameter optimization can be found here: http://scikit-learn.org/stable/modules/grid_search.html

In scikit-learn this technique is provided in the `GridSearchCV` class.

When constructing this class you must provide a dictionary of hyperparameters to evaluate in the param_grid argument. This is a map of the model parameter name and an array of values to try.

By default, accuracy is the score that is optimized, but other scores can be specified in the score argument of the GridSearchCV constructor.

By default, the grid search will only use one thread. By setting the n_jobs argument in the GridSearchCV constructor to -1, the process will use all cores on your machine. Depending on your Keras backend, this may interfere with the main neural network training process.

The GridSearchCV process will then construct and evaluate one model for each combination of parameters. Cross validation is used to evaluate each individual model and by default a 3-fold cross validation is used, although this can be overridden by specifying the cv argument to the GridSearchCV constructor. Below is an example of defining a simple grid search:

```
param_grid = dict(epochs=[10,20,30])       grid = GridSearchCV(estimator=model,
param_grid=param_grid, n_jobs=-1) grid_result = grid.fit(X, Y)
```

Once completed, you can access the outcome of the grid search in the result object returned from grid.fit(). The best_score_ member provides access to the best score observed during the optimization procedure and the best_params_ describes the combination of parameters that achieved the best results.

You can learn more about it here:

http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html#sklearn.grid_se

```
In [5]: from sklearn.model_selection import GridSearchCV
```

1.5 GridSearch HyperParameters

First we would like to optimize the convolutional part of the NN, we fix everything else and vary the filter, kernel size, pool size. We could change the padding type and the striding, but we know that padding won't have a large impact, since there is no information in the corners and we neglect striding for now.

```
In [6]: dense_size_candidates = [[32]]
        optimizer = ['Adam']
        activation = ['relu']

        param_grid={'dense_layer_sizes': dense_size_candidates,
                    'dense_activation' : activation,
                    'filters': [16, 32],
                    'kernel_size': [3, 5],
                    'pool_size': [2, 4],
                    'padding_type' : ['valid'],
                    'stride_size'  : [1],
                    'dropout_rate' : [0.5],
                    'optimizer' : optimizer,
                    # epochs and batch_size are avail for tuning even when not
                    # an argument to model building function
                    'epochs': [1],
                    'batch_size': [256]
                     }
```

1.6 ModelCheckpoint

We want to save the best model for each grid search step (we could also save all of them, of course). For that we will use the ModelCheckpoint call back function, which is also useful to save a NN model after each epoch.

```
In [7]: from keras.callbacks import ModelCheckpoint
```

```
filepath = "best_cnn.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='acc', verbose=1, save_best_only=True, m
```

1.7 Run the grid search

We will use cross-validation with k=2 (speed) and `average precision` as a score value to find the best parameter set. The best metric for optimization depends on your problem, you can find a built-in list here: http://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter But you can also define your own scoring function: http://scikit-learn.org/stable/modules/model_evaluation.html#defining-your-scoring-strategy-from-metric-functions

```
In [8]: grid = GridSearchCV(my_cnn, param_grid, cv=2, scoring='average_precision', n_jobs=1)
        grid_result = grid.fit(X_train, Y_train, callbacks=[checkpoint])
```

```
WARNING:tensorflow:From /home/nackenho/miniconda/envs/TUDortmundMLSeminar/lib/python2.7/site-pa
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /home/nackenho/miniconda/envs/TUDortmundMLSeminar/lib/python2.7/site-pa
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
WARNING:tensorflow:From /home/nackenho/miniconda/envs/TUDortmundMLSeminar/lib/python2.7/site-pa
Instructions for updating:
Use tf.cast instead.
Epoch 1/1
10000/10000 [==============================] - 3s 330us/step - loss: 1.6647 - acc: 0.4329

Epoch 00001: acc improved from -inf to 0.43290, saving model to best_cnn.hdf5
Epoch 1/1
10000/10000 [==============================] - 2s 185us/step - loss: 1.8856 - acc: 0.3366

Epoch 00001: acc did not improve from 0.43290
Epoch 1/1
10000/10000 [==============================] - 1s 124us/step - loss: 2.1849 - acc: 0.2047

Epoch 00001: acc did not improve from 0.43290
Epoch 1/1
10000/10000 [==============================] - 1s 132us/step - loss: 2.1696 - acc: 0.2219

Epoch 00001: acc did not improve from 0.43290
Epoch 1/1
10000/10000 [==============================] - 2s 173us/step - loss: 1.7508 - acc: 0.4096

Epoch 00001: acc did not improve from 0.43290
Epoch 1/1
10000/10000 [==============================] - 2s 183us/step - loss: 1.7248 - acc: 0.4021

Epoch 00001: acc did not improve from 0.43290
```

```
Epoch 1/1
10000/10000 [==============================] - 1s 147us/step - loss: 2.1699 - acc: 0.2020

Epoch 00001: acc did not improve from 0.43290
Epoch 1/1
10000/10000 [==============================] - 2s 162us/step - loss: 2.2159 - acc: 0.1761

Epoch 00001: acc did not improve from 0.43290
Epoch 1/1
10000/10000 [==============================] - 3s 276us/step - loss: 1.4971 - acc: 0.5063

Epoch 00001: acc improved from 0.43290 to 0.50630, saving model to best_cnn.hdf5
Epoch 1/1
10000/10000 [==============================] - 3s 315us/step - loss: 1.5551 - acc: 0.4921

Epoch 00001: acc did not improve from 0.50630
Epoch 1/1
10000/10000 [==============================] - 2s 203us/step - loss: 2.0765 - acc: 0.2819

Epoch 00001: acc did not improve from 0.50630
Epoch 1/1
10000/10000 [==============================] - 2s 201us/step - loss: 2.0150 - acc: 0.2777

Epoch 00001: acc did not improve from 0.50630
Epoch 1/1
10000/10000 [==============================] - 3s 280us/step - loss: 1.6433 - acc: 0.4345

Epoch 00001: acc did not improve from 0.50630
Epoch 1/1
10000/10000 [==============================] - 3s 295us/step - loss: 1.5029 - acc: 0.4939

Epoch 00001: acc did not improve from 0.50630
Epoch 1/1
10000/10000 [==============================] - 2s 237us/step - loss: 1.9949 - acc: 0.3075

Epoch 00001: acc did not improve from 0.50630
Epoch 1/1
10000/10000 [==============================] - 2s 234us/step - loss: 2.0040 - acc: 0.2944

Epoch 00001: acc did not improve from 0.50630
Epoch 1/1
20000/20000 [==============================] - 5s 260us/step - loss: 1.1155 - acc: 0.6441

Epoch 00001: acc improved from 0.50630 to 0.64410, saving model to best_cnn.hdf5
```

```
In [9]: # summarize results
        print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
        means = grid_result.cv_results_['mean_test_score']
        stds = grid_result.cv_results_['std_test_score']
        params = grid_result.cv_results_['params']
        for mean, stdev, param in zip(means, stds, params):
            print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.916689 using {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'valid'
0.896588 (0.014013) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.699419 (0.003745) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.894765 (0.003553) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.700443 (0.003078) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.910830 (0.008910) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.760873 (0.007406) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.916689 (0.002882) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.819488 (0.024454) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
```

1.8 Task 1: Perform a sequentiell grid search to optimze the following hyperparameter. Save the best model for each of the sequentiell steps into a hdf5 file.

- Find the best optimizer: `'SGD'`, `'RMSprop'`, `'Adagrad'`, `'Adadelta'`, `'Adam'`, `'Adamax'`, `'Nadam'`
- Find the best dense structure: Change the width and the depth and try at least 4 different structures
- Find the best activation function for the dense network `'softmax'`, `'softplus'`, `'softsign'`, `'relu'`, `'tanh'`, `'sigmoid'`, `'hard_sigmoid'`, `'linear'`
- Find the best dropout rate: Change dropout between 0 and 0.5 in 0.1 steps
- Find the best optimizer parameter: In order to do that, you need to change the make_model function to adapt for that. Vary the parameter in a meaningful range.

1.9 Change the optimizer

```
In [10]: optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
         param_grid={'dense_layer_sizes': dense_size_candidates,
                     'dense_activation' : activation,
                      'filters': [32],
                      'kernel_size': [5],
                      'pool_size': [2],
                      'padding_type' : ['valid'],
                      'stride_size'  : [1],
                      'dropout_rate' : [0.5],
                      'optimizer' : optimizer,
                      # epochs and batch_size are avail for tuning even when not
                      # an argument to model building function
                      'epochs': [1],
                      'batch_size': [256]
                       }
```

```
In [11]: filepath = "best_optimizer.hdf5"
         checkpoint = ModelCheckpoint(filepath, monitor='acc', verbose=1, save_best_only=True,

         grid = GridSearchCV(my_cnn, param_grid, cv=2, scoring='average_precision', n_jobs=1)
         grid_result = grid.fit(X_train, Y_train, callbacks=[checkpoint])

Epoch 1/1
10000/10000 [==============================] - 4s 409us/step - loss: 2.2812 - acc: 0.1368

Epoch 00001: acc improved from -inf to 0.13680, saving model to best_optimizer.hdf5
Epoch 1/1
10000/10000 [==============================] - 3s 282us/step - loss: 2.2716 - acc: 0.1514

Epoch 00001: acc improved from 0.13680 to 0.15140, saving model to best_optimizer.hdf5
Epoch 1/1
10000/10000 [==============================] - 3s 292us/step - loss: 1.4237 - acc: 0.5320

Epoch 00001: acc improved from 0.15140 to 0.53200, saving model to best_optimizer.hdf5
Epoch 1/1
10000/10000 [==============================] - 4s 378us/step - loss: 1.4087 - acc: 0.5399

Epoch 00001: acc improved from 0.53200 to 0.53990, saving model to best_optimizer.hdf5
Epoch 1/1
10000/10000 [==============================] - 3s 338us/step - loss: 1.3975 - acc: 0.5133

Epoch 00001: acc did not improve from 0.53990
Epoch 1/1
10000/10000 [==============================] - 4s 433us/step - loss: 1.2464 - acc: 0.5717

Epoch 00001: acc improved from 0.53990 to 0.57170, saving model to best_optimizer.hdf5
Epoch 1/1
10000/10000 [==============================] - 6s 575us/step - loss: 1.4169 - acc: 0.5259

Epoch 00001: acc did not improve from 0.57170
Epoch 1/1
10000/10000 [==============================] - 4s 421us/step - loss: 1.5349 - acc: 0.4924

Epoch 00001: acc did not improve from 0.57170
Epoch 1/1
10000/10000 [==============================] - 3s 342us/step - loss: 1.4979 - acc: 0.4939

Epoch 00001: acc did not improve from 0.57170
Epoch 1/1
10000/10000 [==============================] - 4s 379us/step - loss: 1.4930 - acc: 0.4870

Epoch 00001: acc did not improve from 0.57170
Epoch 1/1
10000/10000 [==============================] - 4s 380us/step - loss: 1.5942 - acc: 0.4642
```

```
Epoch 00001: acc did not improve from 0.57170
Epoch 1/1
10000/10000 [==============================] - 3s 345us/step - loss: 1.4891 - acc: 0.4954

Epoch 00001: acc did not improve from 0.57170
Epoch 1/1
10000/10000 [==============================] - 4s 390us/step - loss: 1.3234 - acc: 0.5472

Epoch 00001: acc did not improve from 0.57170
Epoch 1/1
10000/10000 [==============================] - 4s 383us/step - loss: 1.2547 - acc: 0.5856

Epoch 00001: acc improved from 0.57170 to 0.58560, saving model to best_optimizer.hdf5
Epoch 1/1
20000/20000 [==============================] - 5s 257us/step - loss: 0.7949 - acc: 0.7379

Epoch 00001: acc improved from 0.58560 to 0.73795, saving model to best_optimizer.hdf5
```

```
In [12]: # summarize results
         print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
         means = grid_result.cv_results_['mean_test_score']
         stds = grid_result.cv_results_['std_test_score']
         params = grid_result.cv_results_['params']
         for mean, stdev, param in zip(means, stds, params):
             print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.945193 using {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'valid'
0.432661 (0.031049) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.916992 (0.007793) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.945193 (0.001320) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.919770 (0.005383) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.920731 (0.017727) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.920759 (0.002875) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.941536 (0.001952) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
```

1.10 Change the dense structure

```
In [13]: optimizer = ['Adagrad']
         dense_size_candidates = [[32], [64], [32, 32], [64, 64]]

         param_grid={'dense_layer_sizes': dense_size_candidates,
                     'dense_activation' : activation,
                      'filters': [32],
                      'kernel_size': [5],
                      'pool_size': [2],
```

```
                    'padding_type' : ['valid'],
                    'stride_size'  : [1],
                    'dropout_rate' : [0.5],
                    'optimizer' : optimizer,
                    # epochs and batch_size are avail for tuning even when not
                    # an argument to model building function
                    'epochs': [1],
                    'batch_size': [256]
                   }

In [14]: filepath = "best_dnnstruc.hdf5"
         checkpoint = ModelCheckpoint(filepath, monitor='acc', verbose=1, save_best_only=True,

         grid = GridSearchCV(my_cnn, param_grid, cv=2, scoring='average_precision', n_jobs=1)
         grid_result = grid.fit(X_train, Y_train, callbacks=[checkpoint])

Epoch 1/1
10000/10000 [==============================] - 4s 351us/step - loss: 1.1194 - acc: 0.6244

Epoch 00001: acc improved from -inf to 0.62440, saving model to best_dnnstruc.hdf5
Epoch 1/1
10000/10000 [==============================] - 4s 353us/step - loss: 1.2089 - acc: 0.5946

Epoch 00001: acc did not improve from 0.62440
Epoch 1/1
10000/10000 [==============================] - 4s 378us/step - loss: 0.8700 - acc: 0.7224

Epoch 00001: acc improved from 0.62440 to 0.72240, saving model to best_dnnstruc.hdf5
Epoch 1/1
10000/10000 [==============================] - 4s 376us/step - loss: 1.0742 - acc: 0.6539

Epoch 00001: acc did not improve from 0.72240
Epoch 1/1
10000/10000 [==============================] - 4s 419us/step - loss: 1.4062 - acc: 0.5087

Epoch 00001: acc did not improve from 0.72240
Epoch 1/1
10000/10000 [==============================] - 5s 468us/step - loss: 1.3188 - acc: 0.5492

Epoch 00001: acc did not improve from 0.72240
Epoch 1/1
10000/10000 [==============================] - 6s 626us/step - loss: 1.0903 - acc: 0.6353

Epoch 00001: acc did not improve from 0.72240
Epoch 1/1
10000/10000 [==============================] - 9s 945us/step - loss: 0.9624 - acc: 0.6891

Epoch 00001: acc did not improve from 0.72240
```

```
Epoch 1/1
20000/20000 [==============================] - 11s 567us/step - loss: 0.6279 - acc: 0.8062

Epoch 00001: acc improved from 0.72240 to 0.80620, saving model to best_dnnstruc.hdf5
```

```
In [15]: # summarize results
         print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
         means = grid_result.cv_results_['mean_test_score']
         stds = grid_result.cv_results_['std_test_score']
         params = grid_result.cv_results_['params']
         for mean, stdev, param in zip(means, stds, params):
             print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.971635 using {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'valid'
0.955204 (0.003030) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.971635 (0.004167) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.931201 (0.010672) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.965938 (0.000290) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
```

1.11 Change the dense activation

```
In [16]: optimizer = ['Adagrad']
         dense_size_candidates = [[64]]
         activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hard_sign

         param_grid={'dense_layer_sizes': dense_size_candidates,
                     'dense_activation' : activation,
                      'filters': [32],
                      'kernel_size': [5],
                      'pool_size': [2],
                      'padding_type' : ['valid'],
                      'stride_size'  : [1],
                      'dropout_rate' : [0.5],
                      'optimizer' : optimizer,
                      # epochs and batch_size are avail for tuning even when not
                      # an argument to model building function
                      'epochs': [1],
                      'batch_size': [256]
                       }
```

```
In [17]: filepath = "best_dnnact.hdf5"
         checkpoint = ModelCheckpoint(filepath, monitor='acc', verbose=1, save_best_only=True,

         grid = GridSearchCV(my_cnn, param_grid, cv=2, scoring='average_precision', n_jobs=1)
         grid_result = grid.fit(X_train, Y_train, callbacks=[checkpoint])
```

```
Epoch 1/1
10000/10000 [==============================] - 6s 557us/step - loss: 2.0600 - acc: 0.3987
```

```
Epoch 00001: acc improved from -inf to 0.39870, saving model to best_dnnact.hdf5
Epoch 1/1
10000/10000 [==============================] - 6s 645us/step - loss: 2.0553 - acc: 0.4182

Epoch 00001: acc improved from 0.39870 to 0.41820, saving model to best_dnnact.hdf5
Epoch 1/1
10000/10000 [==============================] - 6s 570us/step - loss: 0.9643 - acc: 0.6856

Epoch 00001: acc improved from 0.41820 to 0.68560, saving model to best_dnnact.hdf5
Epoch 1/1
10000/10000 [==============================] - 6s 588us/step - loss: 1.0423 - acc: 0.6627

Epoch 00001: acc did not improve from 0.68560
Epoch 1/1
10000/10000 [==============================] - 6s 617us/step - loss: 0.7268 - acc: 0.7927

Epoch 00001: acc improved from 0.68560 to 0.79270, saving model to best_dnnact.hdf5
Epoch 1/1
10000/10000 [==============================] - 6s 630us/step - loss: 0.7798 - acc: 0.7762

Epoch 00001: acc did not improve from 0.79270
Epoch 1/1
10000/10000 [==============================] - 8s 791us/step - loss: 0.8793 - acc: 0.7179

Epoch 00001: acc did not improve from 0.79270
Epoch 1/1
10000/10000 [==============================] - 5s 522us/step - loss: 0.9177 - acc: 0.7164

Epoch 00001: acc did not improve from 0.79270
Epoch 1/1
10000/10000 [==============================] - 5s 538us/step - loss: 0.6787 - acc: 0.7942

Epoch 00001: acc improved from 0.79270 to 0.79420, saving model to best_dnnact.hdf5
Epoch 1/1
10000/10000 [==============================] - 5s 488us/step - loss: 0.6925 - acc: 0.7968

Epoch 00001: acc improved from 0.79420 to 0.79680, saving model to best_dnnact.hdf5
Epoch 1/1
10000/10000 [==============================] - 5s 499us/step - loss: 1.0631 - acc: 0.6921

Epoch 00001: acc did not improve from 0.79680
Epoch 1/1
10000/10000 [==============================] - 6s 568us/step - loss: 1.0696 - acc: 0.6952

Epoch 00001: acc did not improve from 0.79680
Epoch 1/1
10000/10000 [==============================] - 6s 611us/step - loss: 1.0393 - acc: 0.7049
```

```
Epoch 00001: acc did not improve from 0.79680
Epoch 1/1
10000/10000 [==============================] - 6s 553us/step - loss: 1.0008 - acc: 0.7054

Epoch 00001: acc did not improve from 0.79680
Epoch 1/1
10000/10000 [==============================] - 5s 524us/step - loss: 0.7252 - acc: 0.7781

Epoch 00001: acc did not improve from 0.79680
Epoch 1/1
10000/10000 [==============================] - 6s 572us/step - loss: 0.8230 - acc: 0.7699

Epoch 00001: acc did not improve from 0.79680
Epoch 1/1
20000/20000 [==============================] - 8s 397us/step - loss: 0.6913 - acc: 0.7843

Epoch 00001: acc did not improve from 0.79680
```

```
In [18]: # summarize results
         print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
         means = grid_result.cv_results_['mean_test_score']
         stds = grid_result.cv_results_['std_test_score']
         params = grid_result.cv_results_['params']
         for mean, stdev, param in zip(means, stds, params):
             print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.975182 using {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'valid'
0.797115 (0.023483) with: {'dropout_rate': 0.5, 'dense_activation': 'softmax', 'padding_type':
0.960210 (0.003852) with: {'dropout_rate': 0.5, 'dense_activation': 'softplus', 'padding_type'
0.972582 (0.002216) with: {'dropout_rate': 0.5, 'dense_activation': 'softsign', 'padding_type'
0.975182 (0.000176) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
0.973485 (0.004768) with: {'dropout_rate': 0.5, 'dense_activation': 'tanh', 'padding_type': 'va
0.953993 (0.003981) with: {'dropout_rate': 0.5, 'dense_activation': 'sigmoid', 'padding_type':
0.955436 (0.008407) with: {'dropout_rate': 0.5, 'dense_activation': 'hard_sigmoid', 'padding_ty
0.972310 (0.004212) with: {'dropout_rate': 0.5, 'dense_activation': 'linear', 'padding_type':
```

1.12 Change the dropout rate

```
In [19]: optimizer = ['Adagrad']
         dense_size_candidates = [[64]]
         activation = ['relu']

         param_grid={'dense_layer_sizes': dense_size_candidates,
                     'dense_activation' : activation,
                      'filters': [32],
```

```
                        'kernel_size': [5],
                        'pool_size': [2],
                        'padding_type' : ['valid'],
                        'stride_size'  : [1],
                        'dropout_rate' : [0.1, 0.2, 0.3, 0.4, 0.5],
                        'optimizer' : optimizer,
                        # epochs and batch_size are avail for tuning even when not
                        # an argument to model building function
                        'epochs': [1],
                        'batch_size': [256]
                         }

In [20]: filepath = "best_dropout.hdf5"
         checkpoint = ModelCheckpoint(filepath, monitor='acc', verbose=1, save_best_only=True,

         grid = GridSearchCV(my_cnn, param_grid, cv=2, scoring='average_precision', n_jobs=1)
         grid_result = grid.fit(X_train, Y_train, callbacks=[checkpoint])

Epoch 1/1
10000/10000 [==============================] - 5s 541us/step - loss: 0.7639 - acc: 0.7710

Epoch 00001: acc improved from -inf to 0.77100, saving model to best_dropout.hdf5
Epoch 1/1
10000/10000 [==============================] - 5s 511us/step - loss: 0.7511 - acc: 0.7624

Epoch 00001: acc did not improve from 0.77100
Epoch 1/1
10000/10000 [==============================] - 6s 566us/step - loss: 0.6678 - acc: 0.7901

Epoch 00001: acc improved from 0.77100 to 0.79010, saving model to best_dropout.hdf5
Epoch 1/1
10000/10000 [==============================] - 5s 535us/step - loss: 0.7720 - acc: 0.7690

Epoch 00001: acc did not improve from 0.79010
Epoch 1/1
10000/10000 [==============================] - 6s 570us/step - loss: 0.7833 - acc: 0.7565

Epoch 00001: acc did not improve from 0.79010
Epoch 1/1
10000/10000 [==============================] - 8s 789us/step - loss: 0.7472 - acc: 0.7639

Epoch 00001: acc did not improve from 0.79010
Epoch 1/1
10000/10000 [==============================] - 6s 563us/step - loss: 0.8522 - acc: 0.7258

Epoch 00001: acc did not improve from 0.79010
Epoch 1/1
10000/10000 [==============================] - 6s 563us/step - loss: 0.8956 - acc: 0.7245
```

```
Epoch 00001: acc did not improve from 0.79010
Epoch 1/1
10000/10000 [==============================] - 6s 592us/step - loss: 0.9507 - acc: 0.6973

Epoch 00001: acc did not improve from 0.79010
Epoch 1/1
10000/10000 [==============================] - 6s 569us/step - loss: 0.8580 - acc: 0.7336

Epoch 00001: acc did not improve from 0.79010
Epoch 1/1
20000/20000 [==============================] - 7s 372us/step - loss: 0.4753 - acc: 0.8578

Epoch 00001: acc improved from 0.79010 to 0.85775, saving model to best_dropout.hdf5
```

```
In [21]: # summarize results
         print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
         means = grid_result.cv_results_['mean_test_score']
         stds = grid_result.cv_results_['std_test_score']
         params = grid_result.cv_results_['params']
         for mean, stdev, param in zip(means, stds, params):
             print("%f (%f) with: %r" % (mean, stdev, param))

Best: 0.975177 using {'dropout_rate': 0.2, 'dense_activation': 'relu', 'padding_type': 'valid'
0.961766 (0.001120) with: {'dropout_rate': 0.1, 'dense_activation': 'relu', 'padding_type': 'va
0.975177 (0.004861) with: {'dropout_rate': 0.2, 'dense_activation': 'relu', 'padding_type': 'va
0.971155 (0.000695) with: {'dropout_rate': 0.3, 'dense_activation': 'relu', 'padding_type': 'va
0.942995 (0.025002) with: {'dropout_rate': 0.4, 'dense_activation': 'relu', 'padding_type': 'va
0.974508 (0.000464) with: {'dropout_rate': 0.5, 'dense_activation': 'relu', 'padding_type': 'va
```

1.13 Change the optimizer parameter

```
In [22]: from keras.optimizers import Adam

         def make_model(dense_layer_sizes, dense_activation, filters,
                        kernel_size, pool_size, padding_type, stride_size, dropout_rate, learn
             '''Creates model comprised of 2 convolutional layers followed by dense layers

             dense_layer_sizes: List of layer sizes. This list has one number for each layer
             dense_activation: activation funciton in dense layer
             filters: Number of convolutional filters in each convolutional layer
             kernel_size: Convolutional kernel size
             pool_size: Size of pooling area for max pooling
             padding_type: type of padding: same or valid
             stride_size: symmetric stride size
             dropout_rate: dropout rate
```

```
        optimizer: optimizer used for mimizing
        '''

        optimizer = Adam(lr=learn_rate, beta_1=beta_1, beta_2=beta_2, decay=decay)

        model = Sequential()

        model.add(Conv2D(filters, (kernel_size, kernel_size), padding=padding_type,
                         strides=(stride_size, stride_size), activation='relu', input_shap
        model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))
        model.add(Dropout(dropout_rate))
        model.add(Flatten())
        for layer_size in dense_layer_sizes:
            model.add(Dense(layer_size, activation=dense_activation))
        model.add(Dropout(dropout_rate))
        model.add(Dense(nb_classes))
        model.add(Activation('softmax'))

        model.compile(loss='categorical_crossentropy', optimizer=optimizer,metrics=['accu

        return model

    my_cnn = KerasClassifier(make_model)
```

In [23]:
```
dense_size_candidates = [[64]]
activation = ['relu']

param_grid={'dense_layer_sizes': dense_size_candidates,
            'dense_activation' : activation,
             'filters': [32],
             'kernel_size': [5],
             'pool_size': [2],
             'padding_type' : ['valid'],
             'stride_size'  : [1],
             'dropout_rate' : [0.2],
             # epochs and batch_size are avail for tuning even when not
             # an argument to model building function
             'epochs': [1],
             'batch_size': [256],
             'learn_rate' : [0.001, 0.0001],
             'beta_1' : [0.9, 0.8],
             'beta_2' : [0.999],
             'decay'  : [0.0, 0.3]

              }
```

In [24]:
```
filepath = "best_adampar.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='acc', verbose=1, save_best_only=True,
```

```
        grid = GridSearchCV(my_cnn, param_grid, cv=2, scoring='average_precision', n_jobs=1)
        grid_result = grid.fit(X_train, Y_train, callbacks=[checkpoint])
```

```
Epoch 1/1
10000/10000 [==============================] - 6s 587us/step - loss: 1.0065 - acc: 0.7011

Epoch 00001: acc improved from -inf to 0.70110, saving model to best_adampar.hdf5
Epoch 1/1
10000/10000 [==============================] - 6s 601us/step - loss: 0.9452 - acc: 0.7192

Epoch 00001: acc improved from 0.70110 to 0.71920, saving model to best_adampar.hdf5
Epoch 1/1
10000/10000 [==============================] - 6s 611us/step - loss: 2.0914 - acc: 0.3970

Epoch 00001: acc did not improve from 0.71920
Epoch 1/1
10000/10000 [==============================] - 6s 588us/step - loss: 2.1180 - acc: 0.3989

Epoch 00001: acc did not improve from 0.71920
Epoch 1/1
10000/10000 [==============================] - 6s 600us/step - loss: 1.6890 - acc: 0.5597

Epoch 00001: acc did not improve from 0.71920
Epoch 1/1
10000/10000 [==============================] - 6s 605us/step - loss: 1.7077 - acc: 0.6024

Epoch 00001: acc did not improve from 0.71920
Epoch 1/1
10000/10000 [==============================] - 6s 627us/step - loss: 2.2652 - acc: 0.1989

Epoch 00001: acc did not improve from 0.71920
Epoch 1/1
10000/10000 [==============================] - 6s 626us/step - loss: 2.2447 - acc: 0.1955

Epoch 00001: acc did not improve from 0.71920
Epoch 1/1
10000/10000 [==============================] - 6s 626us/step - loss: 0.9833 - acc: 0.7222

Epoch 00001: acc improved from 0.71920 to 0.72220, saving model to best_adampar.hdf5
Epoch 1/1
10000/10000 [==============================] - 6s 634us/step - loss: 0.9229 - acc: 0.7319

Epoch 00001: acc improved from 0.72220 to 0.73190, saving model to best_adampar.hdf5
Epoch 1/1
10000/10000 [==============================] - 7s 657us/step - loss: 2.0372 - acc: 0.3787

Epoch 00001: acc did not improve from 0.73190
```

```
Epoch 1/1
10000/10000 [==============================] - 7s 738us/step - loss: 2.0335 - acc: 0.3773

Epoch 00001: acc did not improve from 0.73190
Epoch 1/1
10000/10000 [==============================] - 7s 712us/step - loss: 1.6256 - acc: 0.6032

Epoch 00001: acc did not improve from 0.73190
Epoch 1/1
10000/10000 [==============================] - 7s 688us/step - loss: 1.6205 - acc: 0.6060

Epoch 00001: acc did not improve from 0.73190
Epoch 1/1
10000/10000 [==============================] - 7s 676us/step - loss: 2.1862 - acc: 0.2531

Epoch 00001: acc did not improve from 0.73190
Epoch 1/1
10000/10000 [==============================] - 7s 691us/step - loss: 2.2455 - acc: 0.1987

Epoch 00001: acc did not improve from 0.73190
Epoch 1/1
20000/20000 [==============================] - 9s 456us/step - loss: 0.6990 - acc: 0.7959

Epoch 00001: acc improved from 0.73190 to 0.79590, saving model to best_adampar.hdf5
```

```
In [25]: # summarize results
         print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
         means = grid_result.cv_results_['mean_test_score']
         stds = grid_result.cv_results_['std_test_score']
         params = grid_result.cv_results_['params']
         for mean, stdev, param in zip(means, stds, params):
             print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.957188 using {'beta_1': 0.8, 'learn_rate': 0.001, 'dropout_rate': 0.2, 'beta_2': 0.999
0.952343 (0.002845) with: {'beta_1': 0.9, 'learn_rate': 0.001, 'dropout_rate': 0.2, 'beta_2': (
0.812165 (0.012700) with: {'beta_1': 0.9, 'learn_rate': 0.0001, 'dropout_rate': 0.2, 'beta_2':
0.857559 (0.000297) with: {'beta_1': 0.9, 'learn_rate': 0.001, 'dropout_rate': 0.2, 'beta_2': (
0.429046 (0.016940) with: {'beta_1': 0.9, 'learn_rate': 0.0001, 'dropout_rate': 0.2, 'beta_2':
0.957188 (0.002630) with: {'beta_1': 0.8, 'learn_rate': 0.001, 'dropout_rate': 0.2, 'beta_2': (
0.809175 (0.022177) with: {'beta_1': 0.8, 'learn_rate': 0.0001, 'dropout_rate': 0.2, 'beta_2':
0.875474 (0.006918) with: {'beta_1': 0.8, 'learn_rate': 0.001, 'dropout_rate': 0.2, 'beta_2': (
0.483471 (0.018391) with: {'beta_1': 0.8, 'learn_rate': 0.0001, 'dropout_rate': 0.2, 'beta_2':
```

1.14 Tips for Hyperparameter Optimization

This section lists some handy tips to consider when tuning hyperparameters of your neural network.

- **k-fold Cross Validation.** You can see that the results from the examples in this post show some variance. For speed reasons, we used a cross-validation of 2, but perhaps k=5 or k=10 would be more stable. Carefully choose your cross validation configuration to ensure your results are stable.
- **Review the Whole Grid.** Do not just focus on the best result, review the whole grid of results and look for trends to support configuration decisions.
- **Parallelize.** Use all your cores if you can, neural networks are slow to train and we often want to try a lot of different parameters. Consider using cluster instances if available.
- **Use a Subsample of Your Dataset.** Because networks are slow to train, try training them on a smaller sample of your training dataset, just to get an idea of general directions of parameters rather than optimal configurations.
- **Start with Coarse Grids.** Start with coarse-grained grids and zoom into finer grained grids once you can narrow the scope.
- **Do not Transfer Results.** Results are generally problem specific. Try to avoid favorite configurations on each new problem that you see. It is unlikely that optimal results you discover on one problem will transfer to your next project. Instead look for broader trends like number of layers or relationships between parameters.
- **Reproducibility is a Problem.** Although we set the seed for the random number generator in NumPy, the results are not 100% reproducible. There is more to reproducibility when grid searching wrapped Keras models than is presented in this post.

1.15 Task 2: Load the best model and evaluate it using the function below

You can load the model using `from keras.models import load_model model= load_model('filename')`

In [26]:
```
##############################################################################

import matplotlib.pyplot as plt
%matplotlib inline

def plot_history(network_history):
    plt.figure()
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.plot(network_history.history['loss'])
    plt.plot(network_history.history['val_loss'])
    plt.legend(['Training', 'Validation'])

    plt.figure()
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.plot(network_history.history['acc'])
    plt.plot(network_history.history['val_acc'])
    plt.legend(['Training', 'Validation'], loc='lower right')
    plt.show()

##############################################################################
```

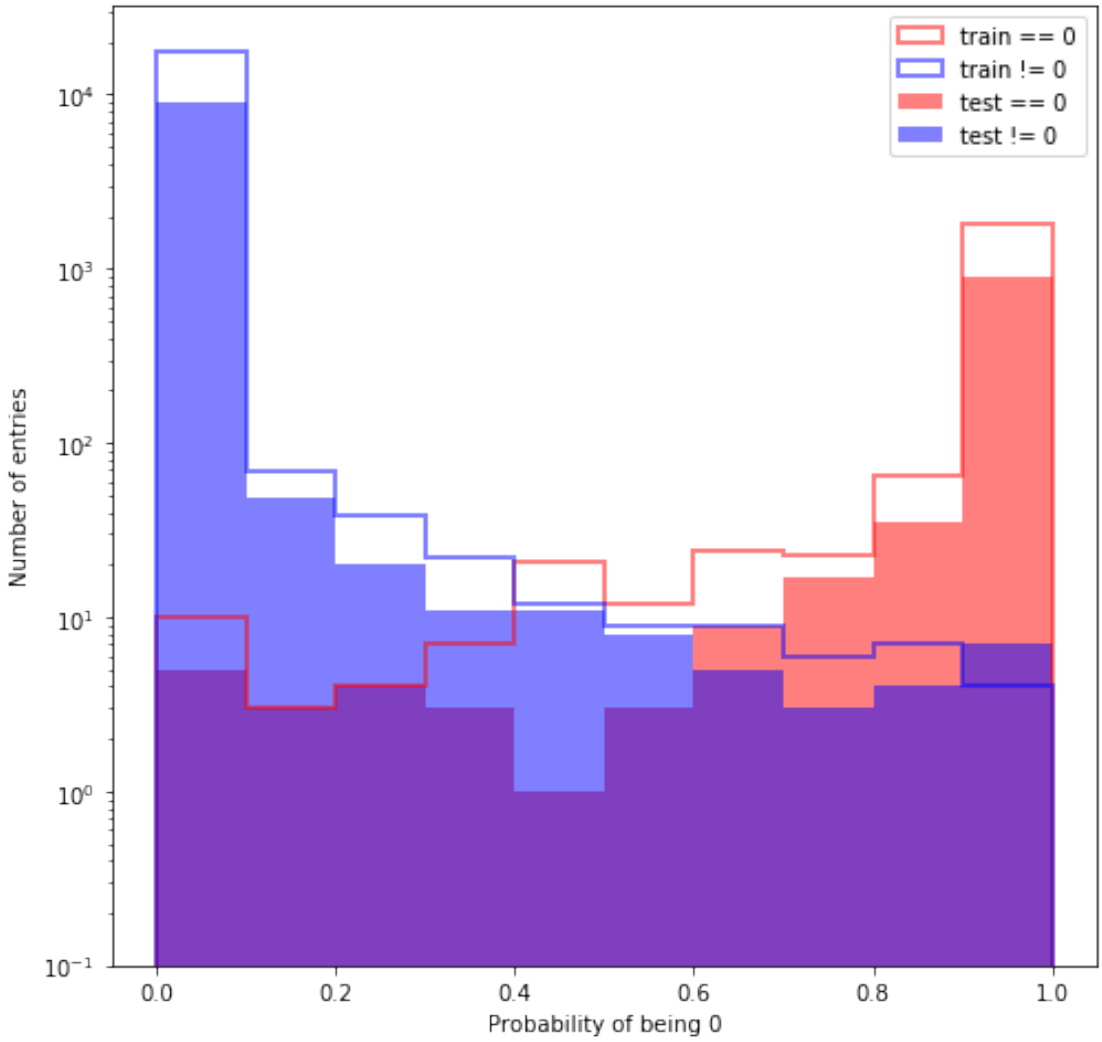
```
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

################################################################################
import matplotlib.cm as cm
def display_errors(errors_index,img_errors,pred_errors, obs_errors):
    """ This function shows 6 images with their predicted and real labels"""
    n = 0
    nrows = 2
    ncols = 3
    fig, ax = plt.subplots(nrows,ncols,sharex=True,sharey=True)
    for row in range(nrows):
        for col in range(ncols):
            error = errors_index[n]
            ax[row,col].imshow((img_errors[error]).reshape((28,28)), cmap=cm.Greys, i
            ax[row,col].set_title("Predicted label :{}\nTrue label :{}".format(pred_e
            n += 1

################################################################################
from sklearn.metrics import confusion_matrix,classification_report
```

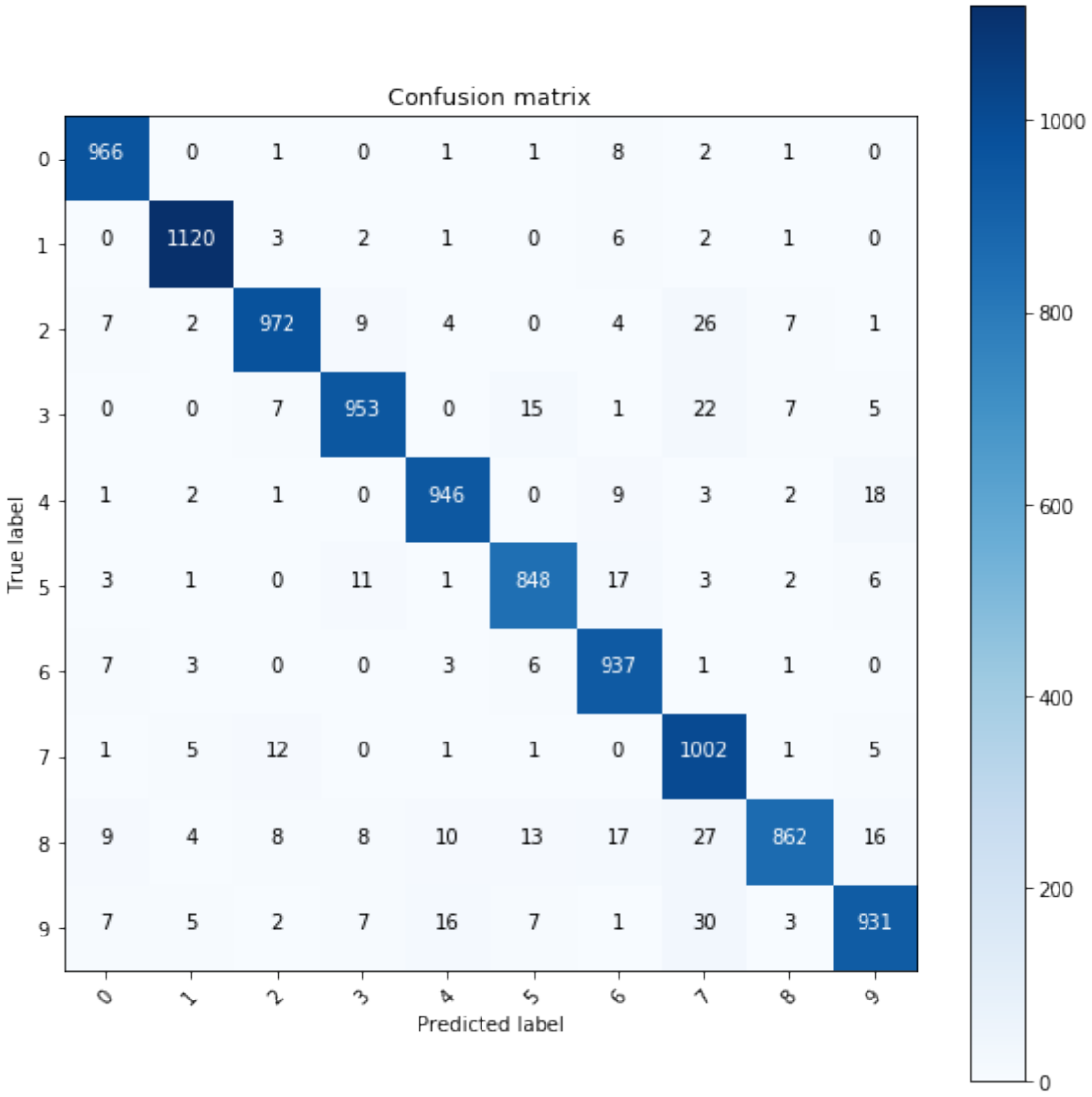
```
def evaluate(X_test, Y_test, model):

    ##Evaluate loss and metrics
    loss, accuracy = model.evaluate(X_test, Y_test, verbose=0)
    print('Test Loss:', loss)
    print('Test Accuracy:', accuracy)
    # Predict the values from the test dataset
    Y_pred = model.predict(X_test)
    # Convert predictions classes to one hot vectors
    Y_cls = np.argmax(Y_pred, axis = 1)
    # Convert validation observations to one hot vectors
    Y_true = np.argmax(Y_test, axis = 1)
    print 'Classification Report:\n', classification_report(Y_true,Y_cls)

    ## Plot 0 probability including overtraining test
    plt.figure(figsize=(8,8))

    label=0
    #Test prediction
    Y_pred_prob = Y_pred[:,label]
    plt.hist(Y_pred_prob[Y_true == label], alpha=0.5, color='red', range=[0, 1], bins=
    plt.hist(Y_pred_prob[Y_true != label], alpha=0.5, color='blue', range=[0, 1], bins
    #Train prediction
    Y_train_pred = model.predict(X_train)
    Y_train_pred_prob = Y_train_pred[:,label]
    Y_train_true = np.argmax(Y_train, axis = 1)
    plt.hist(Y_train_pred_prob[Y_train_true == label], alpha=0.5, color='red', range=
    plt.hist(Y_train_pred_prob[Y_train_true != label], alpha=0.5, color='blue', range=

    plt.legend(['train == 0', 'train != 0', 'test == 0', 'test != 0'], loc='upper rig
    plt.xlabel('Probability of being 0')
    plt.ylabel('Number of entries')
    plt.show()

    # compute the confusion matrix
    confusion_mtx = confusion_matrix(Y_true, Y_cls)
    # plot the confusion matrix
    plt.figure(figsize=(8,8))
    plot_confusion_matrix(confusion_mtx, classes = range(10))

    #Plot largest errors
    errors = (Y_cls - Y_true != 0)
    Y_cls_errors = Y_cls[errors]
    Y_pred_errors = Y_pred[errors]
    Y_true_errors = Y_true[errors]
    X_test_errors = X_test[errors]
    # Probabilities of the wrong predicted numbers
    Y_pred_errors_prob = np.max(Y_pred_errors,axis = 1)
```

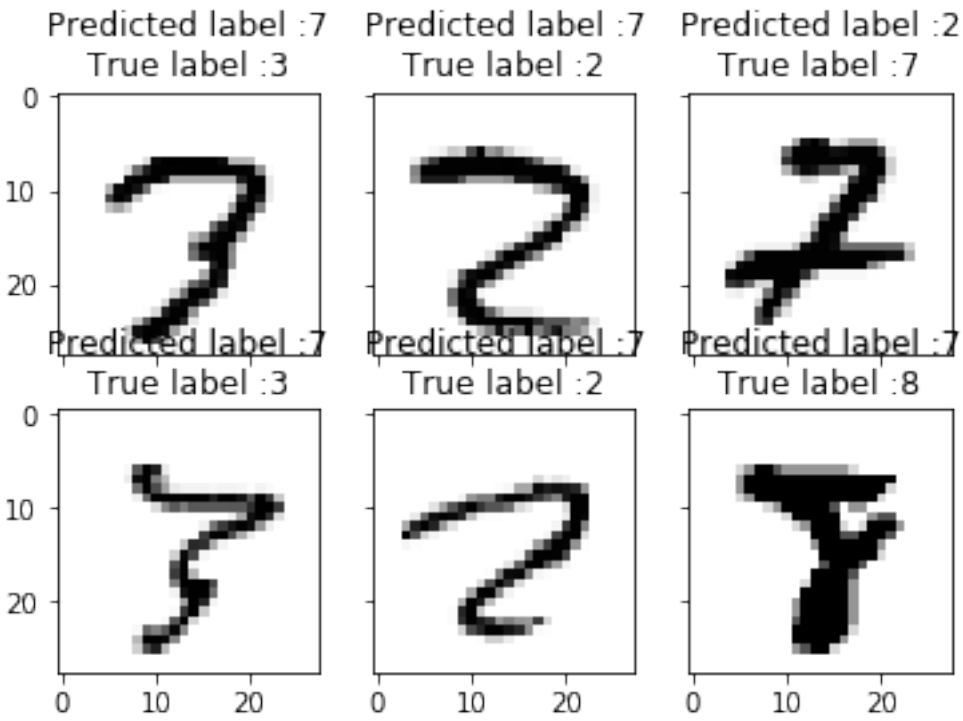
```
# Predicted probabilities of the true values in the error set
true_prob_errors = np.diagonal(np.take(Y_pred_errors, Y_true_errors, axis=1))
# Difference between the probability of the predicted label and the true label
delta_pred_true_errors = Y_pred_errors_prob - true_prob_errors
# Sorted list of the delta prob errors
sorted_dela_errors = np.argsort(delta_pred_true_errors)
# Top 6 errors
most_important_errors = sorted_dela_errors[-6:]
# Show the top 6 errors
display_errors(most_important_errors, X_test_errors, Y_cls_errors, Y_true_errors)

##Plot predictions
slice = 15
predicted = model.predict(X_test[:slice]).argmax(-1)
plt.figure(figsize=(16,8))
for i in range(slice):
    plt.subplot(1, slice, i+1)
    plt.imshow(X_test[i].reshape(28,28), interpolation='nearest')
    plt.text(0, 0, predicted[i], color='black',
             bbox=dict(facecolor='white', alpha=1))
    plt.axis('off')
```

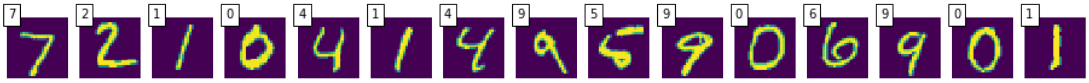
1.16 Load best model and evaluate

```
In [27]: from keras.models import load_model
         model= load_model('best_dropout.hdf5')

In [28]: evaluate(X_test, Y_test, model)

('Test Loss:', 0.1637829773247242)
('Test Accuracy:', 0.9537)
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.98      0.99      0.98      1135
           2       0.97      0.94      0.95      1032
           3       0.96      0.94      0.95      1010
           4       0.96      0.96      0.96       982
           5       0.95      0.95      0.95       892
           6       0.94      0.98      0.96       958
           7       0.90      0.97      0.93      1028
           8       0.97      0.89      0.93       974
           9       0.95      0.92      0.94      1009

   micro avg       0.95      0.95      0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```


train == 0
train != 0
test == 0
test != 0
Number of entries
10^4
10^3
10^2
10^1
10^0
10^{-1}
0.0
0.2
0.4
0.6
0.8
1.0
Probability of being 0


Confusion matrix
True label
Predicted label
0 1 2 3 4 5 6 7 8 9
966 0 1 0 1 1 8 2 1 0
0 1120 3 2 1 0 6 2 1 0
7 2 972 9 4 0 4 26 7 1
0 0 7 953 0 15 1 22 7 5
1 2 1 0 946 0 9 3 2 18
3 1 0 11 1 848 17 3 2 6
7 3 0 0 3 6 937 1 1 0
1 5 12 0 1 1 0 1002 1 5
9 4 8 8 10 13 17 27 862 16
7 5 2 7 16 7 1 30 3 931
1000
800
600
400
200
0





---

2 There's more:

The `GridSearchCV` model in scikit-learn performs a complete search, considering **all** the possible combinations of Hyper-parameters we want to optimise.

If we want to apply for an optmised and bounded search in the hyper-parameter space, I suggest to take a look at:

- `Keras + hyperopt == hyperas`: http://maxpumperla.github.io/hyperas/