ex4_sol

May 16, 2018

1 Exercise 4 - Fully Connected Networks and the MNIST dataset

This exercise is based on https://github.com/leriomaggio/deep-learning-keras-tensorflow

2 The MNIST database

The MNIST (Modified National Institute of Standards and Technology) database (link) has a database of handwritten digits. The dataset consists of 28x28 grayscale images of the 10 digits.

Since this dataset is **provided** with Keras, we just ask the `keras.dataset` model for training and test data.

`from keras.datasets import mnist` `(X_train, y_train), (X_test, y_test) = mnist.load_data()`

The training set has $60,000$ samples. The test set has $10,000$ samples. The digits are size-normalized and centered in a fixed-size image. The data page has description on how the data was collected. It also has reports the benchmark of various algorithms on the test dataset.

2.1 Task 1: Data preparation

- Download the data
- Inspect the data and plot a few of the images using `matplotlib.pyplot.imshow`
- Reshape the input data to be in vectorial form (original data are images)
- Convert the input data to do dtype `float32` using `astype` in order to scale it afterwards
- Normalize the design matrix to values between 0 and 1.
- How many classes do you have? How much data of each class?
- Convert the class vector to binary class matrices (**one-hot-vector**). Use the `to_categorical` function from `keras.utilis` to convert integer labels to **one-hot-vectors**.
- Split the training set into training and validation data (30%)

2.1.1 Download the datasets and display first entry

```
In [1]: from keras.datasets import mnist

        (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Using TensorFlow backend.
```

```
---------------------------------------------------------------------------

RuntimeError                              Traceback (most recent call last)

RuntimeError: module compiled against API version 0xc but this version of numpy is 0xb



---------------------------------------------------------------------------

RuntimeError                              Traceback (most recent call last)

RuntimeError: module compiled against API version 0xc but this version of numpy is 0xb
```

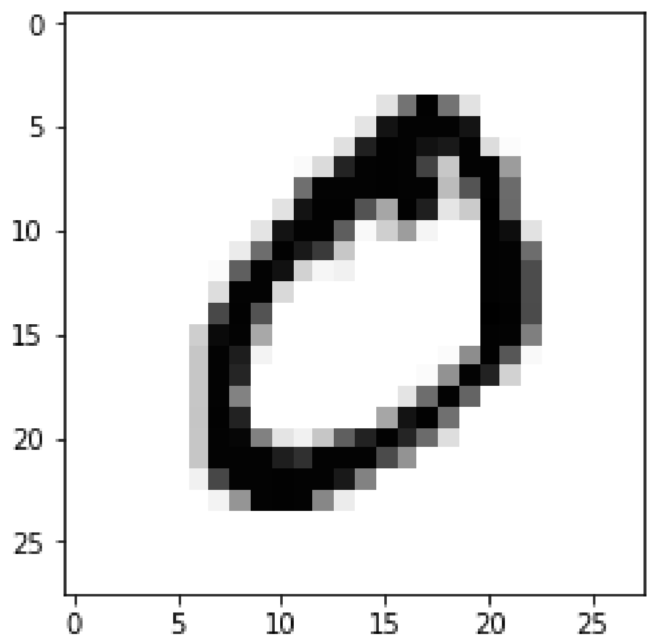
In [2]: X_train[1]

```
Out[2]: array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  51, 159, 253, 159,  50,   0,   0,   0,   0,   0,   0,
          0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  48, 238, 252, 252, 252, 237,   0,   0,   0,   0,   0,   0,
          0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         54, 227, 253, 252, 239, 233, 252,  57,   6,   0,   0,   0,   0,
          0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  10,  60,
        224, 252, 253, 252, 202,  84, 252, 253, 122,   0,   0,   0,   0,
          0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 163, 252,
        252, 252, 253, 252, 252,  96, 189, 253, 167,   0,   0,   0,   0,
          0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  51, 238, 253,
        253, 190, 114, 253, 228,  47,  79, 255, 168,   0,   0,   0,   0,
          0,   0],
```

```
[  0,   0,   0,   0,   0,   0,   0,   0,   0,  48, 238, 252, 252,
 179,  12,  75, 121,  21,   0,   0, 253, 243,  50,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  38, 165, 253, 233, 208,
  84,   0,   0,   0,   0,   0,   0, 253, 252, 165,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   7, 178, 252, 240,  71,  19,
  28,   0,   0,   0,   0,   0,   0, 253, 252, 195,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,  57, 252, 252,  63,   0,   0,
   0,   0,   0,   0,   0,   0,   0, 253, 252, 195,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0, 198, 253, 190,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0, 255, 253, 196,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  76, 246, 252, 112,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0, 253, 252, 148,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  85, 252, 230,  25,   0,   0,   0,
   0,   0,   0,   0,   0,   7, 135, 253, 186,  12,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  85, 252, 223,   0,   0,   0,   0,
   0,   0,   0,   0,   7, 131, 252, 225,  71,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  85, 252, 145,   0,   0,   0,   0,
   0,   0,   0,  48, 165, 252, 173,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  86, 253, 225,   0,   0,   0,   0,
   0,   0, 114, 238, 253, 162,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  85, 252, 249, 146,  48,  29,  85,
 178, 225, 253, 223, 167,  56,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  85, 252, 252, 252, 229, 215, 252,
 252, 252, 196, 130,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  28, 199, 252, 252, 253, 252, 252,
 233, 145,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,  25, 128, 252, 253, 252, 141,
  37,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
```

```
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0]], dtype=uint8)
```

In [3]:
```python
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.cm as cm

fig = plt.figure()
plt.imshow(X_train[1], cmap=cm.Greys)
plt.show()
```



In [4]:
```python
plt.figure(figsize=(12,10))
x, y = 10, 4
for i in range(40):
    plt.subplot(y, x, i+1)
    plt.imshow(X_train[i], interpolation='nearest', cmap=cm.Greys)
plt.show()
```

2.1.2 Reshape the datasets

```
In [5]: X_train.shape

Out[5]: (60000, 28, 28)

In [6]: X_train = X_train.reshape(60000, 784)
        X_test = X_test.reshape(10000, 784)

In [7]: X_train.shape

Out[7]: (60000, 784)
```

2.1.3 Convert to float32

```
In [8]: X_train.dtype

Out[8]: dtype('uint8')

In [9]: X_train = X_train.astype("float32")
        X_test = X_test.astype("float32")

In [10]: X_train.dtype

Out[10]: dtype('float32')
```

2.1.4 Scale the input data

```
In [11]: from sklearn.preprocessing import MinMaxScaler
         scaler = MinMaxScaler(feature_range=(0, 1))

         X_train = scaler.fit_transform(X_train)
         X_test = scaler.transform(X_test)
```

```
In [12]: X_train[1]
```

```
Out[12]: array([ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.20000002,  0.62352943,  0.99215692,
                 0.62352943,  0.19607845,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.18823531,
                 0.9333334 ,  0.98823535,  0.98823535,  0.98823535,  0.92941183,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
                 0.        ,  0.21176472,  0.89019614,  0.99215692,  0.98823535,
                 0.93725497,  0.91372555,  0.98823535,  0.22352943,  0.02352941,
                 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.03921569, 0.23529413, 0.87843144,
0.98823535, 0.99215692, 0.98823535, 0.79215693, 0.32941177,
0.98823535, 0.99215692, 0.4784314 , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.63921571, 0.98823535, 0.98823535, 0.98823535, 0.99215692,
0.98823535, 0.98823535, 0.37647063, 0.74117649, 0.99215692,
0.65490198, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.20000002, 0.9333334 , 0.99215692,
0.99215692, 0.74509805, 0.44705886, 0.99215692, 0.89411771,
0.18431373, 0.30980393, 1.        , 0.65882355, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.18823531,
0.9333334 , 0.98823535, 0.98823535, 0.7019608 , 0.04705883,
0.29411766, 0.47450984, 0.08235294, 0.        , 0.        ,
0.99215692, 0.95294124, 0.19607845, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.14901961, 0.64705884, 0.99215692, 0.91372555,
0.81568635, 0.32941177, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.99215692, 0.98823535,
0.64705884, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.02745098, 0.69803923,
0.98823535, 0.94117653, 0.27843139, 0.07450981, 0.10980393,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.99215692, 0.98823535, 0.76470596, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.22352943, 0.98823535, 0.98823535, 0.24705884,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.99215692,
0.98823535, 0.76470596, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.77647066,
0.99215692, 0.74509805, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 1.        , 0.99215692, 0.76862752,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.29803923, 0.96470594, 0.98823535, 0.43921572,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.99215692, 0.98823535, 0.58039218, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.33333334,
0.98823535, 0.90196085, 0.09803922, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.02745098, 0.52941179, 0.99215692, 0.72941178,
0.04705883, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.33333334, 0.98823535, 0.87450987,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.02745098, 0.51372552,
0.98823535, 0.88235301, 0.27843139, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.33333334, 0.98823535, 0.56862748, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.18823531, 0.64705884, 0.98823535, 0.67843139, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.33725491, 0.99215692,
0.88235301, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.44705886, 0.9333334 , 0.99215692,
0.63529414, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.33333334, 0.98823535, 0.97647065, 0.57254905,
0.18823531, 0.1137255 , 0.33333334, 0.69803923, 0.88235301,
0.99215692, 0.87450987, 0.65490198, 0.21960786, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.33333334,
0.98823535, 0.98823535, 0.98823535, 0.89803928, 0.84313732,
0.98823535, 0.98823535, 0.98823535, 0.76862752, 0.50980395,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.10980393, 0.78039223, 0.98823535,
0.98823535, 0.99215692, 0.98823535, 0.98823535, 0.91372555,
0.56862748, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.09803922, 0.50196081, 0.98823535, 0.99215692,
0.98823535, 0.5529412 , 0.14509805, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        ], dtype=float32)
```

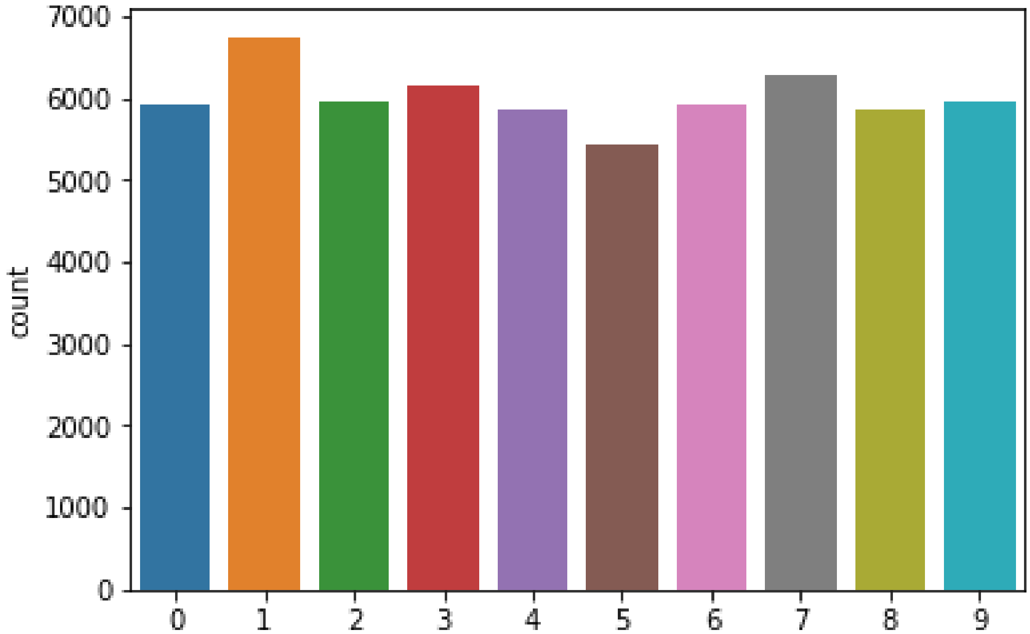
2.1.5 Convert class vectors to binary class matrices

```
In [13]: y_train
```

```
Out[13]: array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
In [14]: import seaborn as sns
         sns.countplot(y_train)
```

```
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9267626b90>
```




```
In [15]: from keras.utils import np_utils

         Y_train = np_utils.to_categorical(y_train, 10)
         Y_test = np_utils.to_categorical(y_test, 10)
```

```
In [16]: Y_train[1]
```

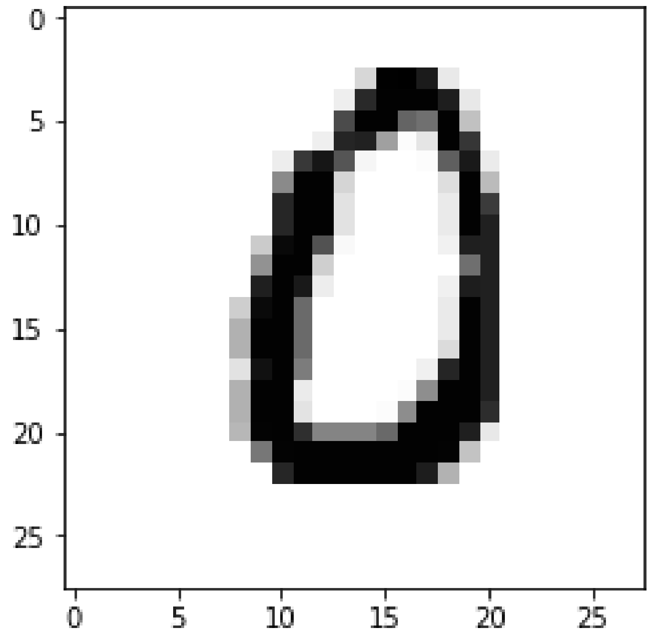
```
Out[16]: array([ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.], dtype=float32)
```

Split Training and Validation Data

```
In [17]: from sklearn.model_selection import train_test_split

         X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.3, rand
```

```
In [18]: plt.imshow(X_train[1].reshape(28, 28), cmap=cm.Greys)
```

```
Out[18]: <matplotlib.image.AxesImage at 0x7f9267683190>
```

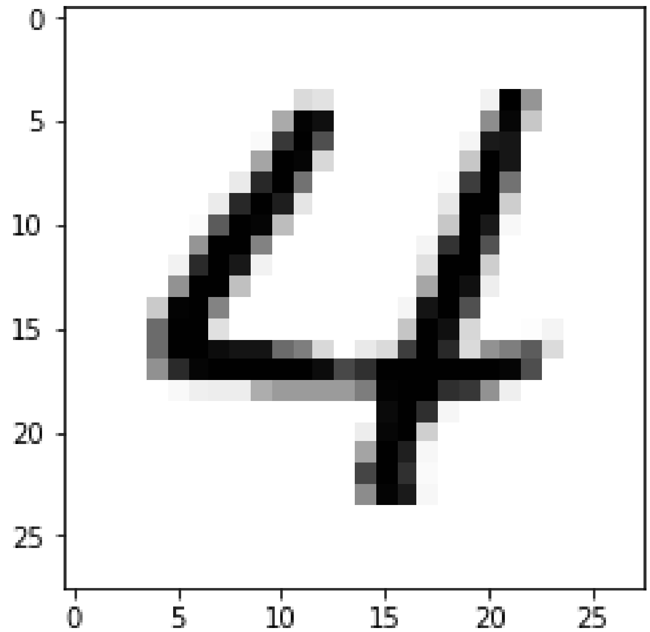



```
In [19]: import numpy as np
         print(np.asarray(range(10)))
         print(Y_train[1].astype('int'))
```

```
[0 1 2 3 4 5 6 7 8 9]
[1 0 0 0 0 0 0 0 0 0]
```

```
In [20]: plt.imshow(X_val[0].reshape(28, 28), cmap=cm.Greys)
```

```
Out[20]: <matplotlib.image.AxesImage at 0x7f9268001490>
```




```
In [21]: print(np.asarray(range(10)))
         print(Y_val[0].astype('int'))
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 0 0 0 1 0 0 0 0 0]
```

2.2 Task 2: Build and train a neural network

- Design a dense neural network structure.
- Choose `softmax` as activation for the output node (normalized multi-class probability)
- Use `categorical_crossentropy` as loss function (multi-class version of crossentropy)
- Use `adam` as optimizer and a batch size of 512 (speed things up)
- Train the NN over 50 epochs and plot the evolution of the training and validation loss as well as of one meaningful metric. What do you observe?
- Evaluate the performance on the test set using `sklearn.metrics`
- Plot the probability of being a *Zero* for true zeros and for all other numbers

2.3 Training

```
In [22]: from keras.models import Sequential
         from keras.layers.core import Dense

         model = Sequential()
         model.add(Dense(512, activation='relu', input_dim=784))
         model.add(Dense(256, activation='relu'))
```

```
model.add(Dense(10, activation='softmax'))

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

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 10)                2570
=================================================================
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
_________________________________________________________________
```

```
In [23]: history = model.fit(X_train, Y_train, batch_size=512, epochs=50, verbose=1, validation_
```

```
Train on 42000 samples, validate on 18000 samples
Epoch 1/50
42000/42000 [==============================] - 2s 51us/step - loss: 0.4453 - acc: 0.8770 - val_l
Epoch 2/50
42000/42000 [==============================] - 2s 46us/step - loss: 0.1658 - acc: 0.9520 - val_l
Epoch 3/50
42000/42000 [==============================] - 2s 45us/step - loss: 0.1154 - acc: 0.9663 - val_l
Epoch 4/50
42000/42000 [==============================] - 2s 46us/step - loss: 0.0904 - acc: 0.9739 - val_l
Epoch 5/50
42000/42000 [==============================] - 2s 51us/step - loss: 0.0624 - acc: 0.9822 - val_l
Epoch 6/50
42000/42000 [==============================] - 2s 53us/step - loss: 0.0526 - acc: 0.9845 - val_l
Epoch 7/50
42000/42000 [==============================] - 2s 47us/step - loss: 0.0333 - acc: 0.9914 - val_l
Epoch 8/50
42000/42000 [==============================] - 2s 51us/step - loss: 0.0315 - acc: 0.9909 - val_l
Epoch 9/50
42000/42000 [==============================] - 2s 55us/step - loss: 0.0193 - acc: 0.9956 - val_l
Epoch 10/50
42000/42000 [==============================] - 2s 50us/step - loss: 0.0311 - acc: 0.9910 - val_l
Epoch 11/50
42000/42000 [==============================] - 2s 50us/step - loss: 0.0112 - acc: 0.9977 - val_l
Epoch 12/50
42000/42000 [==============================] - 2s 54us/step - loss: 0.0087 - acc: 0.9987 - val_l
```

```
Epoch 13/50
42000/42000 [==============================] - 2s 51us/step - loss: 0.0067 - acc: 0.9991 - val_l
Epoch 14/50
42000/42000 [==============================] - 2s 47us/step - loss: 0.0063 - acc: 0.9989 - val_l
Epoch 15/50
42000/42000 [==============================] - 2s 54us/step - loss: 0.0036 - acc: 0.9997 - val_l
Epoch 16/50
42000/42000 [==============================] - 2s 56us/step - loss: 0.0025 - acc: 0.9999 - val_l
Epoch 17/50
42000/42000 [==============================] - 3s 69us/step - loss: 0.0019 - acc: 1.0000 - val_l
Epoch 18/50
42000/42000 [==============================] - 3s 71us/step - loss: 0.0035 - acc: 0.9995 - val_l
Epoch 19/50
42000/42000 [==============================] - 3s 65us/step - loss: 0.0014 - acc: 1.0000 - val_l
Epoch 20/50
42000/42000 [==============================] - 3s 62us/step - loss: 0.0011 - acc: 1.0000 - val_l
Epoch 21/50
42000/42000 [==============================] - 3s 63us/step - loss: 8.6685e-04 - acc: 1.0000 - v
Epoch 22/50
42000/42000 [==============================] - 3s 70us/step - loss: 7.5952e-04 - acc: 1.0000 - v
Epoch 23/50
42000/42000 [==============================] - 3s 68us/step - loss: 6.5437e-04 - acc: 1.0000 - v
Epoch 24/50
42000/42000 [==============================] - 3s 61us/step - loss: 5.7938e-04 - acc: 1.0000 - v
Epoch 25/50
42000/42000 [==============================] - 3s 61us/step - loss: 5.1534e-04 - acc: 1.0000 - v
Epoch 26/50
42000/42000 [==============================] - 3s 62us/step - loss: 4.5829e-04 - acc: 1.0000 - v
Epoch 27/50
42000/42000 [==============================] - 3s 69us/step - loss: 4.2828e-04 - acc: 1.0000 - v
Epoch 28/50
42000/42000 [==============================] - 3s 69us/step - loss: 3.7491e-04 - acc: 1.0000 - v
Epoch 29/50
42000/42000 [==============================] - 3s 63us/step - loss: 3.3716e-04 - acc: 1.0000 - v
Epoch 30/50
42000/42000 [==============================] - 3s 61us/step - loss: 3.2185e-04 - acc: 1.0000 - v
Epoch 31/50
42000/42000 [==============================] - 3s 63us/step - loss: 2.8666e-04 - acc: 1.0000 - v
Epoch 32/50
42000/42000 [==============================] - 3s 69us/step - loss: 2.5433e-04 - acc: 1.0000 - v
Epoch 33/50
42000/42000 [==============================] - 3s 70us/step - loss: 2.3013e-04 - acc: 1.0000 - v
Epoch 34/50
42000/42000 [==============================] - 3s 66us/step - loss: 2.1619e-04 - acc: 1.0000 - v
Epoch 35/50
42000/42000 [==============================] - 3s 70us/step - loss: 1.9521e-04 - acc: 1.0000 - v
Epoch 36/50
42000/42000 [==============================] - 3s 69us/step - loss: 1.9155e-04 - acc: 1.0000 - v
```

```
Epoch 37/50
42000/42000 [==============================] - 3s 63us/step - loss: 1.6839e-04 - acc: 1.0000 - v
Epoch 38/50
42000/42000 [==============================] - 3s 70us/step - loss: 1.5757e-04 - acc: 1.0000 - v
Epoch 39/50
42000/42000 [==============================] - 3s 64us/step - loss: 1.4156e-04 - acc: 1.0000 - v
Epoch 40/50
42000/42000 [==============================] - 3s 62us/step - loss: 1.3266e-04 - acc: 1.0000 - v
Epoch 41/50
42000/42000 [==============================] - 3s 66us/step - loss: 1.2259e-04 - acc: 1.0000 - v
Epoch 42/50
42000/42000 [==============================] - 4s 106us/step - loss: 1.1526e-04 - acc: 1.0000 -
Epoch 43/50
42000/42000 [==============================] - 3s 81us/step - loss: 1.0959e-04 - acc: 1.0000 - v
Epoch 44/50
42000/42000 [==============================] - 3s 70us/step - loss: 1.0430e-04 - acc: 1.0000 - v
Epoch 45/50
42000/42000 [==============================] - 3s 71us/step - loss: 9.2346e-05 - acc: 1.0000 - v
Epoch 46/50
42000/42000 [==============================] - 3s 63us/step - loss: 8.4654e-05 - acc: 1.0000 - v
Epoch 47/50
42000/42000 [==============================] - 3s 66us/step - loss: 8.2624e-05 - acc: 1.0000 - v
Epoch 48/50
42000/42000 [==============================] - 3s 67us/step - loss: 7.4904e-05 - acc: 1.0000 - v
Epoch 49/50
42000/42000 [==============================] - 3s 61us/step - loss: 7.1072e-05 - acc: 1.0000 - v
Epoch 50/50
42000/42000 [==============================] - 3s 70us/step - loss: 6.5492e-05 - acc: 1.0000 - v
```

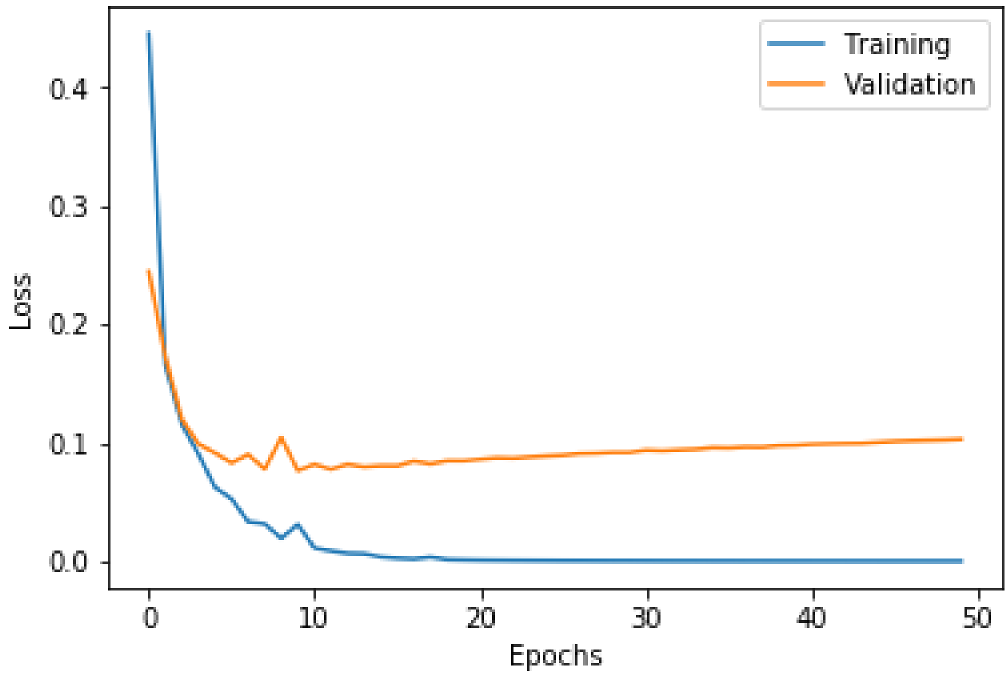
2.3.1 Plotting the network history

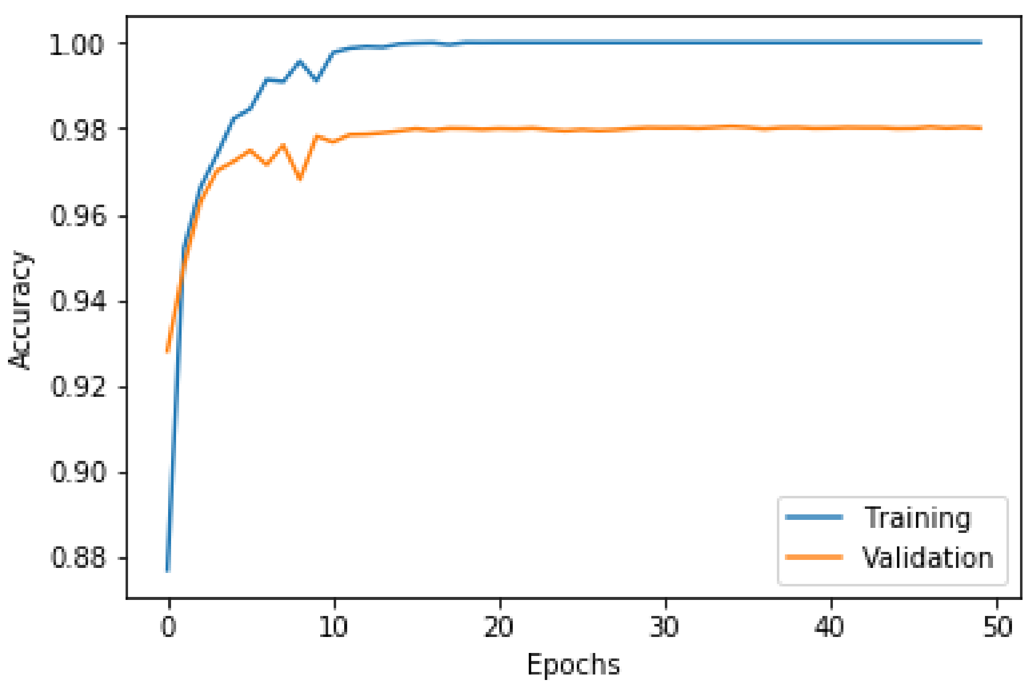
As seen before, the return value of the `fit` function is a `keras.callbacks.History` object which contains the entire history of training/validation loss and defined metric (accuracy) for each epoch. Let's define a function to plot the history:

```
In [24]: 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()
```

In [25]:
```
plot_history(history)
```





2.4 Evaluation

```
In [26]: loss_and_metrics = model.evaluate(X_test, Y_test, batch_size=256)
         print loss_and_metrics
```

```
10000/10000 [==============================] - 0s 28us/step
[0.09768963857802318, 0.98050000000000004]
```

```
In [27]: # 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)
```

```
In [28]: from sklearn.metrics import classification_report
         print 'Classification Report:\n', classification_report(Y_true,Y_cls)
```

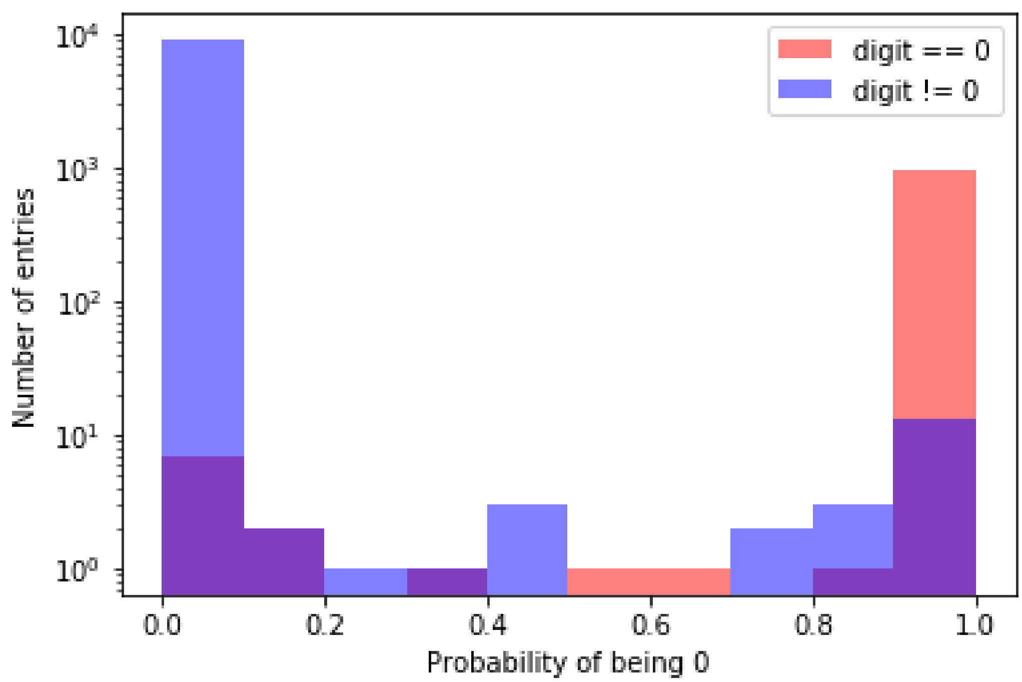
```
Classification Report:
             precision    recall  f1-score   support

          0       0.98      0.99      0.99       980
          1       0.99      0.99      0.99      1135
          2       0.97      0.98      0.97      1032
          3       0.98      0.98      0.98      1010
          4       0.98      0.98      0.98       982
          5       0.98      0.98      0.98       892
          6       0.99      0.98      0.98       958
          7       0.98      0.98      0.98      1028
          8       0.98      0.97      0.97       974
          9       0.98      0.98      0.98      1009

avg / total       0.98      0.98      0.98     10000
```

2.4.1 Plotting the normalized probability prob_i = p_i / sum(p_i)

In multi-class problems the interpretation of the n-dimensional output is not always trivial, in particular if an output activation function is used which can not be interpreted as a probability. If one is only interested in distinguishing two of the classes one could build the ratio of these two class responses in order to get the best discrimination. Similarly, one could weight different classes according their importance for the specific problem. Because we have used a softmax activation together with the categorical cross-entropy we can directly interpret our output as probabilities and don't need to normalize it. If that is not the case you can define a multi-class probability for instance in the following way:

In [29]:
```
def prob_multiclass(Y_pred, index):
    n_cls = len(Y_pred[0])

    Y_prob=[]
    for i in range(len(Y_pred)):
        numerator=Y_pred[i,index]
        denominator=0.0
        for idx in range(n_cls):
            denominator+=Y_pred[i,idx]

        Y_prob.append(numerator/denominator)

    return np.asarray(Y_prob)
```

In [30]:
```
label=0
Y_pred_prob = prob_multiclass(Y_pred, label)
plt.hist(Y_pred_prob[Y_true == label], alpha=0.5, color='red', bins=10, log = True)
plt.hist(Y_pred_prob[Y_true != label], alpha=0.5, color='blue', bins=10, log = True)
plt.legend(['digit == 0', 'digit != 0'], loc='upper right')
plt.xlabel('Probability of being 0')
plt.ylabel('Number of entries')
plt.show()
```

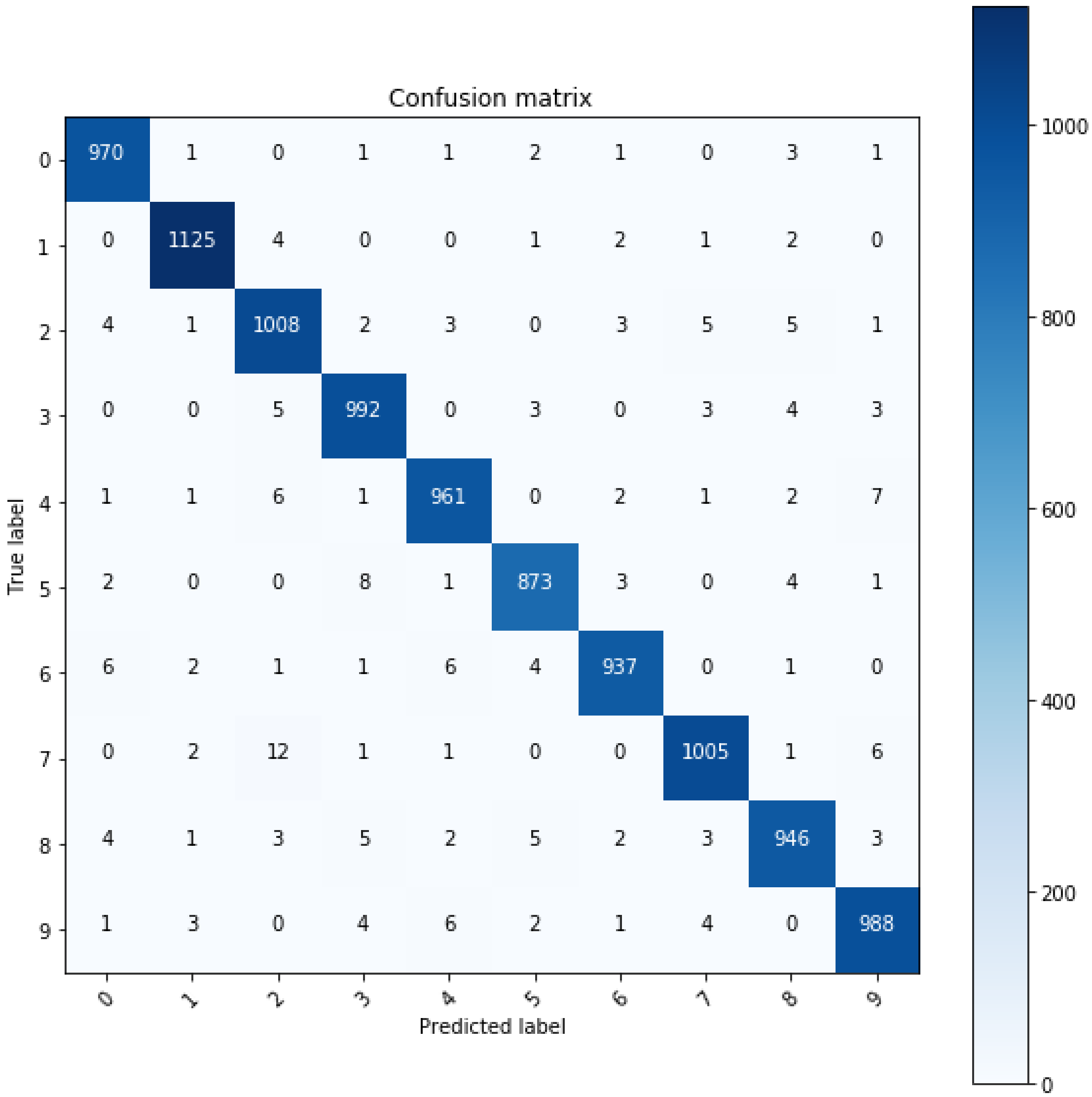


2.4.2 Plot the confusion matrix

A good way to show the performance of a multi-class output is the confusion matrix: http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

```
In [31]: #Note, this code is taken straight from the SKLEARN website, an nice way of viewing con
         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')
```

```
In [32]: from sklearn.metrics import confusion_matrix
         # 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))
```



2.4.3 Plot wrong associations

Errors are difference between predicted labels and true labels

```
In [33]: 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]
```

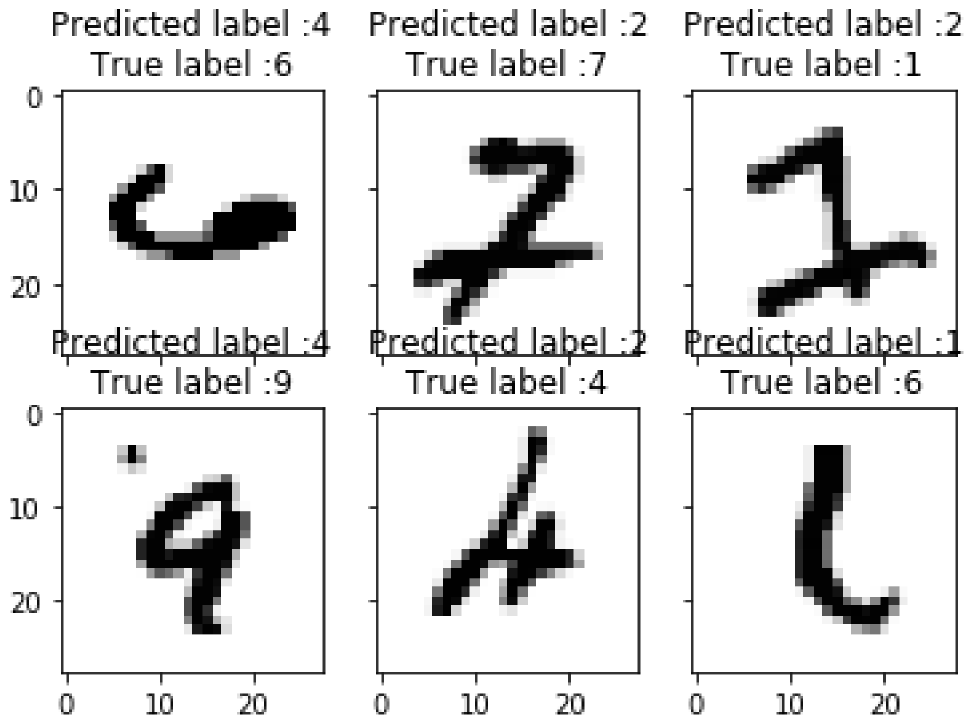
Define plotting function

In [34]:
```
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, int
            ax[row,col].set_title("Predicted label :{}\nTrue label :{}".format(pred_err
            n += 1
```

Rank errors by difference in probability

In [35]:
```
# 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:]
```

In [36]:
```
# Show the top 6 errors
display_errors(most_important_errors, X_test_errors, Y_cls_errors, Y_true_errors)
```



2.5 Using Dropout Layers

As we have learned last time, the trainings and validation loss of the fit history is not comparable when using dropout. We can define our own callback function which calculates the loss and metric after each epoch for any dataset

```
In [37]: from keras.callbacks import Callback

         class HistoryEpoch(Callback):
             def __init__(self, data):
                 self.data = data

             def on_train_begin(self, logs={}):
                 self.loss = []
                 self.acc = []

             def on_epoch_end(self, epoch, logs={}):
                 x, y = self.data
                 l, a = self.model.evaluate(x, y, verbose=0)
                 self.loss.append(l)
                 self.acc.append(a)
```

2.6 Task 3: Using regularizer

- Modify your previous example network by adding a Dropout layer after each hidden layer
- Add l2 regularization to the hidden layers
- Use the new defined `HistoryEpoch` for training, validation and test data set in order to save a comparable loss function and metric. This is done by e.g.: `train_hist=HistoryEpoch((X_train, Y_train))`. In the `fit` function you can call the callback then by specifying `callbacks=[train_hist]`.
- Plot the loss and metric evolution and compare the calculated loss with the default loss from the history
- Evaluate the performance of the NN as for the unregularized NN and compare the performance

In [38]:
```
from keras.layers.core import Dropout
from keras.regularizers import l2

dropout=0.5
l2_lambda = 0.0001

model_dropout = Sequential()
model_dropout.add(Dense(512, activation='relu', kernel_regularizer=l2(l2_lambda), input
model_dropout.add(Dropout(dropout))
model_dropout.add(Dense(256, activation='relu', kernel_regularizer=l2(l2_lambda)))
model_dropout.add(Dropout(dropout))
model_dropout.add(Dense(10, activation='softmax'))

train_hist=HistoryEpoch((X_train, Y_train))
val_hist=HistoryEpoch((X_val, Y_val))
test_hist=HistoryEpoch((X_test, Y_test))


model_dropout.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accu
model_dropout.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2570
=================================================================
Total params: 535,818
Trainable params: 535,818
```

```
Non-trainable params: 0
_________________________________________________________________
```

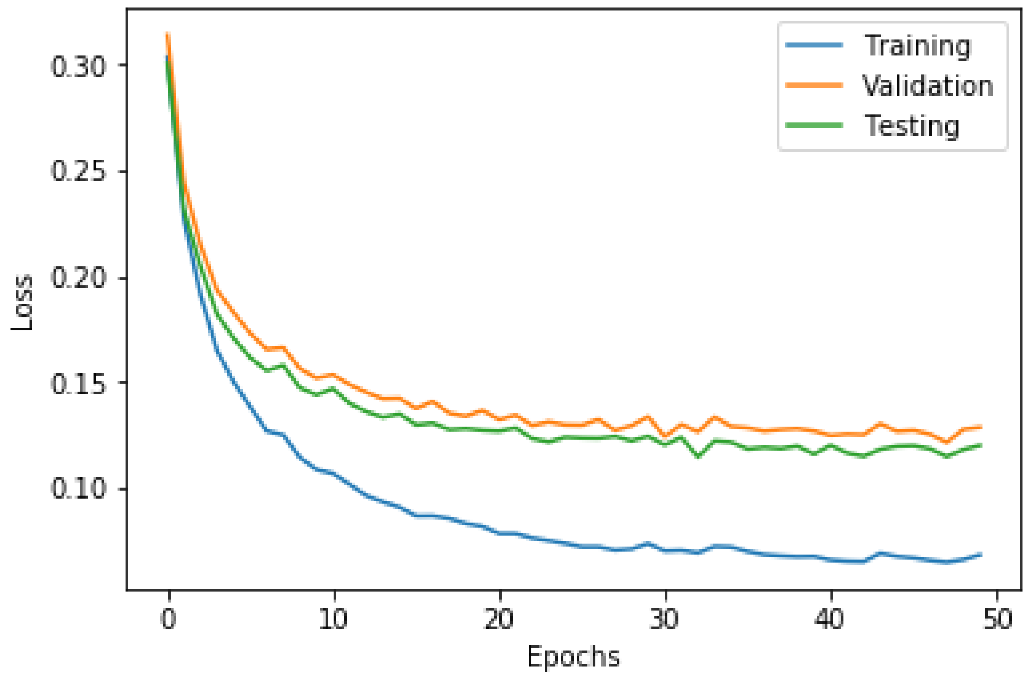
```
In [39]: history_dropout = model_dropout.fit(X_train, Y_train, batch_size=512, epochs=50, verbos
```

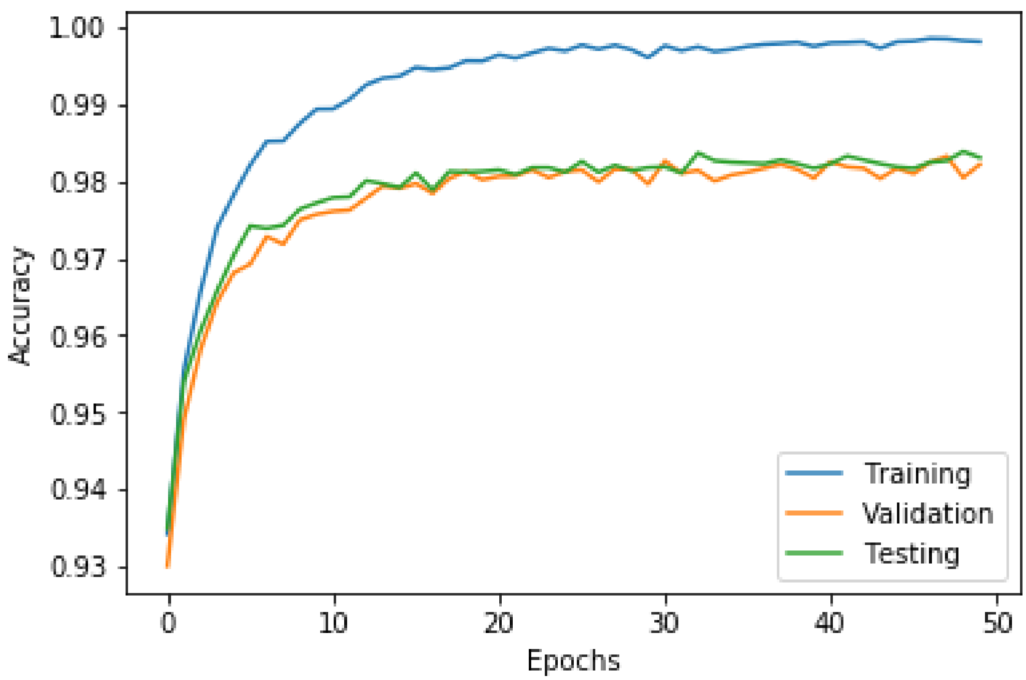
```
Train on 42000 samples, validate on 18000 samples
Epoch 1/50
42000/42000 [==============================] - 3s 80us/step - loss: 0.7435 - acc: 0.7940 - val_l
Epoch 2/50
42000/42000 [==============================] - 3s 64us/step - loss: 0.3466 - acc: 0.9219 - val_l
Epoch 3/50
42000/42000 [==============================] - 4s 86us/step - loss: 0.2801 - acc: 0.9408 - val_l
Epoch 4/50
42000/42000 [==============================] - 3s 77us/step - loss: 0.2390 - acc: 0.9515 - val_l
Epoch 5/50
42000/42000 [==============================] - 3s 70us/step - loss: 0.2223 - acc: 0.9557 - val_l
Epoch 6/50
42000/42000 [==============================] - 3s 72us/step - loss: 0.1979 - acc: 0.9633 - val_l
Epoch 7/50
42000/42000 [==============================] - 3s 73us/step - loss: 0.1853 - acc: 0.9670 - val_l
Epoch 8/50
42000/42000 [==============================] - 4s 96us/step - loss: 0.1810 - acc: 0.9677 - val_l
Epoch 9/50
42000/42000 [==============================] - 4s 84us/step - loss: 0.1687 - acc: 0.9719 - val_l
Epoch 10/50
42000/42000 [==============================] - 3s 83us/step - loss: 0.1620 - acc: 0.9736 - val_l
Epoch 11/50
42000/42000 [==============================] - 3s 74us/step - loss: 0.1541 - acc: 0.9743 - val_l
Epoch 12/50
42000/42000 [==============================] - 4s 100us/step - loss: 0.1452 - acc: 0.9768 - val_
Epoch 13/50
42000/42000 [==============================] - 4s 87us/step - loss: 0.1449 - acc: 0.9765 - val_l
Epoch 14/50
42000/42000 [==============================] - 3s 74us/step - loss: 0.1392 - acc: 0.9787 - val_l
Epoch 15/50
42000/42000 [==============================] - 3s 69us/step - loss: 0.1348 - acc: 0.9794 - val_l
Epoch 16/50
42000/42000 [==============================] - 4s 103us/step - loss: 0.1268 - acc: 0.9816 - val_
Epoch 17/50
42000/42000 [==============================] - 3s 71us/step - loss: 0.1274 - acc: 0.9811 - val_l
Epoch 18/50
42000/42000 [==============================] - 3s 68us/step - loss: 0.1232 - acc: 0.9820 - val_l
Epoch 19/50
42000/42000 [==============================] - 2s 59us/step - loss: 0.1287 - acc: 0.9805 - val_l
Epoch 20/50
42000/42000 [==============================] - 2s 59us/step - loss: 0.1195 - acc: 0.9827 - val_l
Epoch 21/50
```

```
42000/42000 [==============================] - 3s 61us/step - loss: 0.1162 - acc: 0.9843 - val_l
Epoch 22/50
42000/42000 [==============================] - 3s 64us/step - loss: 0.1142 - acc: 0.9847 - val_l
Epoch 23/50
42000/42000 [==============================] - 3s 71us/step - loss: 0.1108 - acc: 0.9848 - val_l
Epoch 24/50
42000/42000 [==============================] - 3s 76us/step - loss: 0.1098 - acc: 0.9854 - val_l
Epoch 25/50
42000/42000 [==============================] - 3s 76us/step - loss: 0.1080 - acc: 0.9857 - val_l
Epoch 26/50
42000/42000 [==============================] - 4s 95us/step - loss: 0.1090 - acc: 0.9850 - val_l
Epoch 27/50
42000/42000 [==============================] - 4s 88us/step - loss: 0.1042 - acc: 0.9858 - val_l
Epoch 28/50
42000/42000 [==============================] - 4s 85us/step - loss: 0.1050 - acc: 0.9854 - val_l
Epoch 29/50
42000/42000 [==============================] - 3s 81us/step - loss: 0.1031 - acc: 0.9866 - val_l
Epoch 30/50
42000/42000 [==============================] - 4s 84us/step - loss: 0.1054 - acc: 0.9854 - val_l
Epoch 31/50
42000/42000 [==============================] - 3s 71us/step - loss: 0.1032 - acc: 0.9861 - val_l
Epoch 32/50
42000/42000 [==============================] - 4s 107us/step - loss: 0.1053 - acc: 0.9853 - val_
Epoch 33/50
42000/42000 [==============================] - 3s 80us/step - loss: 0.0990 - acc: 0.9871 - val_l
Epoch 34/50
42000/42000 [==============================] - 3s 78us/step - loss: 0.1116 - acc: 0.9825 - val_l
Epoch 35/50
42000/42000 [==============================] - 4s 95us/step - loss: 0.1082 - acc: 0.9854 - val_l
Epoch 36/50
42000/42000 [==============================] - 4s 91us/step - loss: 0.1035 - acc: 0.9857 - val_l
Epoch 37/50
42000/42000 [==============================] - 4s 87us/step - loss: 0.1014 - acc: 0.9867 - val_l
Epoch 38/50
42000/42000 [==============================] - 3s 81us/step - loss: 0.1010 - acc: 0.9862 - val_l
Epoch 39/50
42000/42000 [==============================] - 3s 83us/step - loss: 0.0982 - acc: 0.9871 - val_l
Epoch 40/50
42000/42000 [==============================] - 5s 107us/step - loss: 0.0967 - acc: 0.9882 - val_
Epoch 41/50
42000/42000 [==============================] - 5s 108us/step - loss: 0.0955 - acc: 0.9882 - val_
Epoch 42/50
42000/42000 [==============================] - 4s 87us/step - loss: 0.0947 - acc: 0.9883 - val_l
Epoch 43/50
42000/42000 [==============================] - 4s 86us/step - loss: 0.0927 - acc: 0.9886 - val_l
Epoch 44/50
42000/42000 [==============================] - 4s 90us/step - loss: 0.1105 - acc: 0.9831 - val_l
Epoch 45/50
```

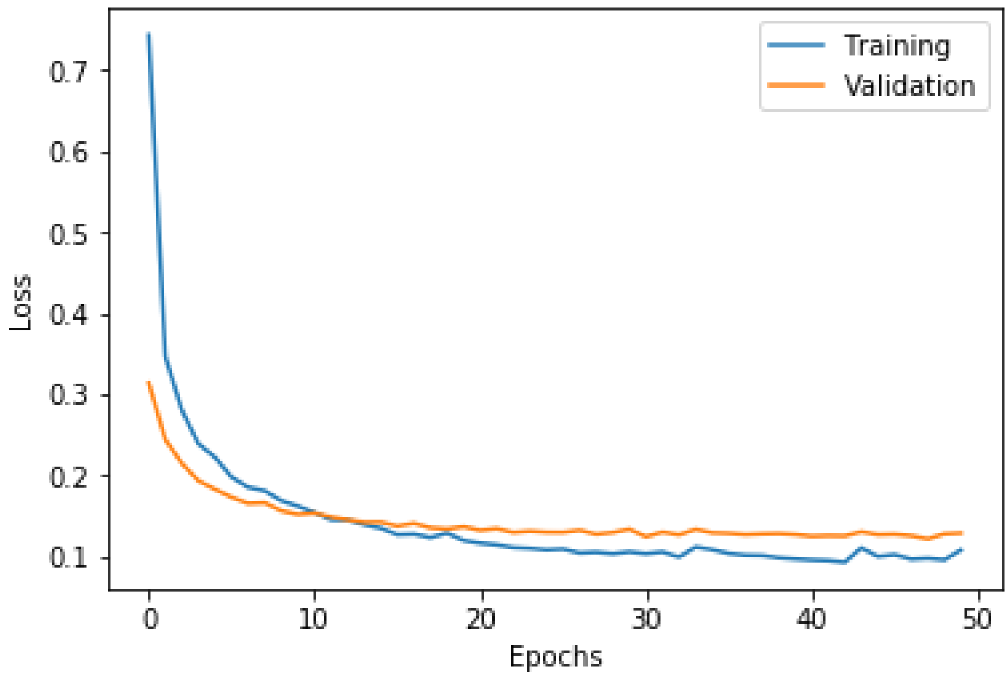
```
42000/42000 [==============================] - 4s 86us/step - loss: 0.0997 - acc: 0.9867 - val_l
Epoch 46/50
42000/42000 [==============================] - 4s 90us/step - loss: 0.1021 - acc: 0.9869 - val_l
Epoch 47/50
42000/42000 [==============================] - 5s 131us/step - loss: 0.0965 - acc: 0.9877 - val_
Epoch 48/50
42000/42000 [==============================] - 4s 96us/step - loss: 0.0976 - acc: 0.9874 - val_l
Epoch 49/50
42000/42000 [==============================] - 4s 90us/step - loss: 0.0956 - acc: 0.9875 - val_l
Epoch 50/50
42000/42000 [==============================] - 4s 94us/step - loss: 0.1077 - acc: 0.9842 - val_l
```

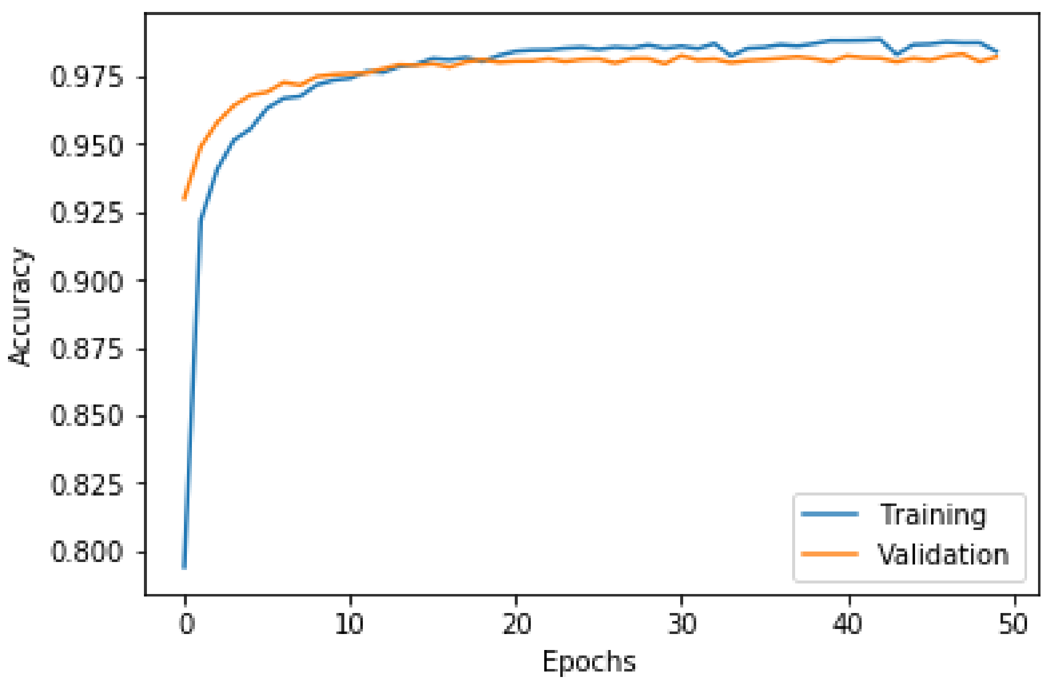
In [40]:
```
plt.figure()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.plot(train_hist.loss)
plt.plot(val_hist.loss)
plt.plot(test_hist.loss)
plt.legend(['Training', 'Validation', 'Testing'])

plt.figure()
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.plot(train_hist.acc)
plt.plot(val_hist.acc)
plt.plot(test_hist.acc)
plt.legend(['Training', 'Validation', 'Testing'], loc='lower right')
plt.show()
```






```
In [41]: plot_history(history_dropout)
```


Training
Validation
Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
Epochs


Training
Validation
Accuracy
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0
10
20
30
40
50
Epochs

2.7 Evaluation

In [42]:
```
loss_and_metrics = model_dropout.evaluate(X_test, Y_test, batch_size=256)
print loss_and_metrics
```

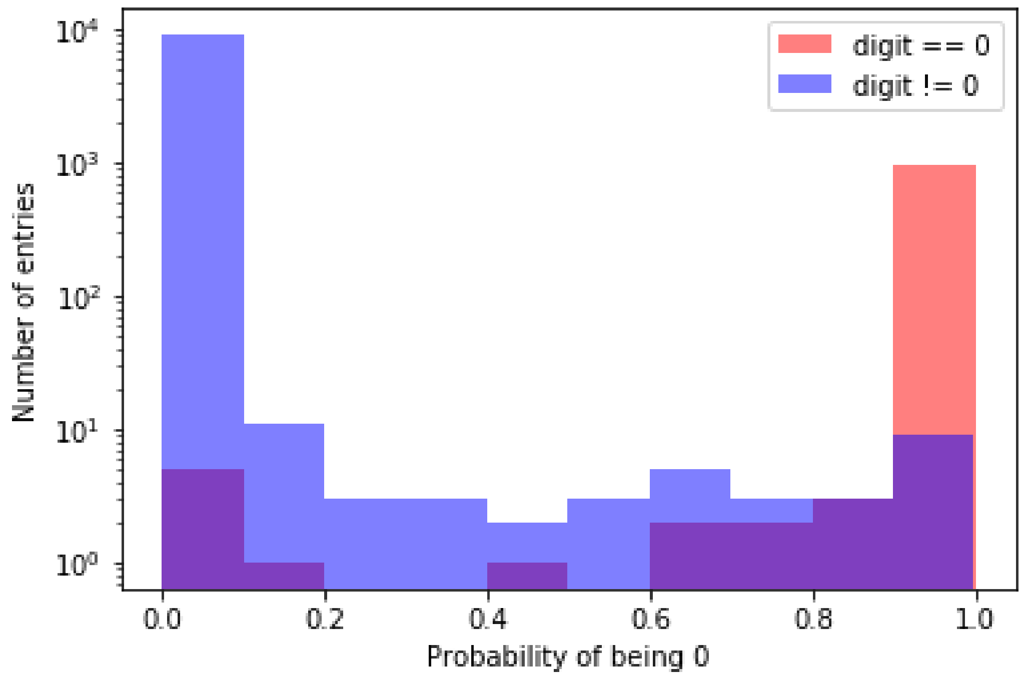
```
10000/10000 [==============================] - 0s 39us/step
[0.12014759578704834, 0.98309999999999997]
```

In [43]:
```
# Predict the values from the test dataset
Y_pred = model_dropout.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)
```

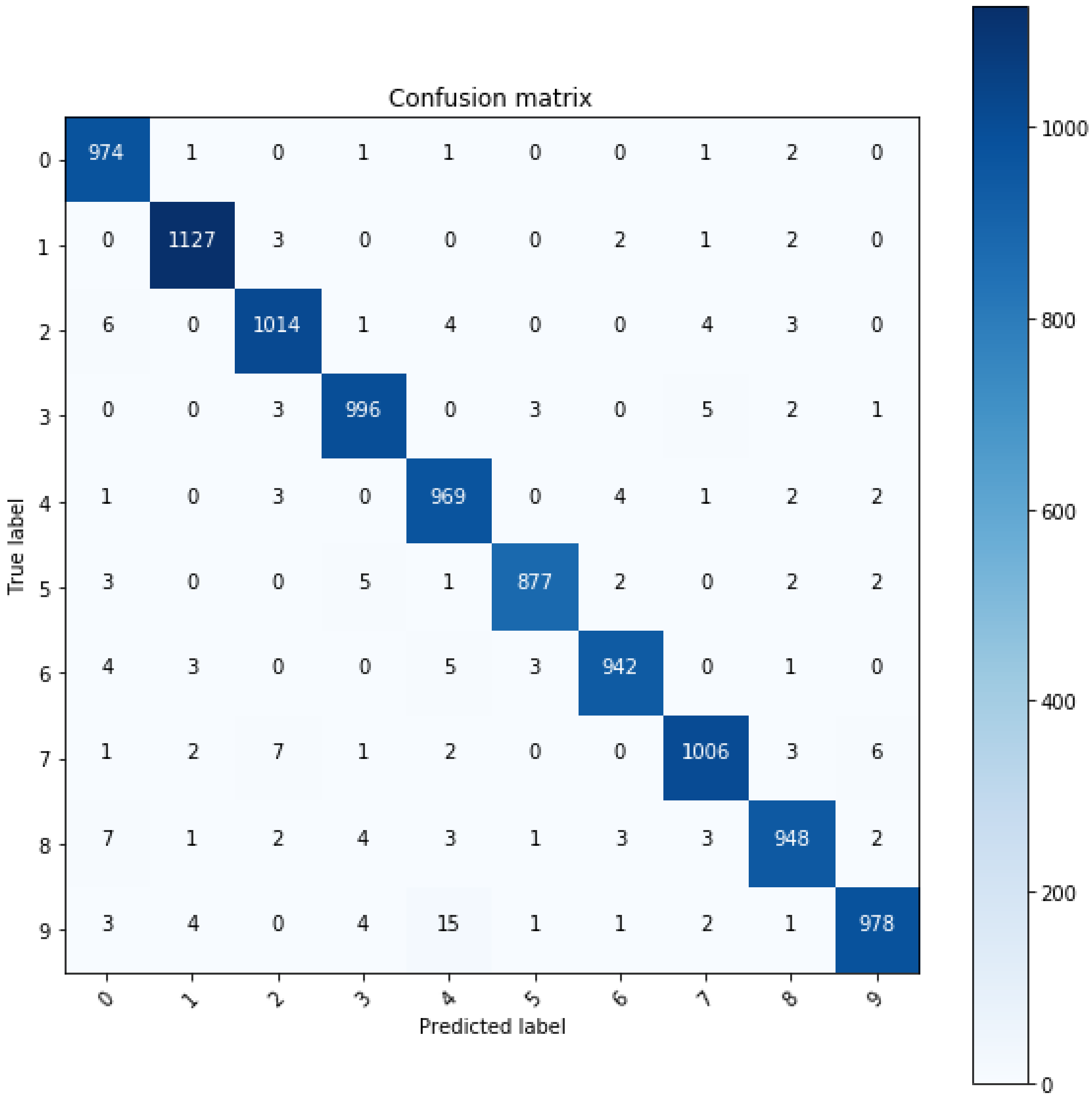
In [44]:
```
from sklearn.metrics import classification_report
print 'Classification Report:\n', classification_report(Y_true,Y_cls)
```

```
Classification Report:
             precision    recall  f1-score   support

          0       0.97      0.99      0.98       980
          1       0.99      0.99      0.99      1135
          2       0.98      0.98      0.98      1032
          3       0.98      0.99      0.99      1010
          4       0.97      0.99      0.98       982
          5       0.99      0.98      0.99       892
          6       0.99      0.98      0.99       958
          7       0.98      0.98      0.98      1028
          8       0.98      0.97      0.98       974
          9       0.99      0.97      0.98      1009

avg / total       0.98      0.98      0.98     10000
```

In [45]:
```
label=0
Y_pred_prob = prob_multiclass(Y_pred, label)
plt.hist(Y_pred_prob[Y_true == label], alpha=0.5, color='red', bins=10, log = True)
plt.hist(Y_pred_prob[Y_true != label], alpha=0.5, color='blue', bins=10, log = True)
plt.legend(['digit == 0', 'digit != 0'], loc='upper right')
plt.xlabel('Probability of being 0')
plt.ylabel('Number of entries')
plt.show()
```

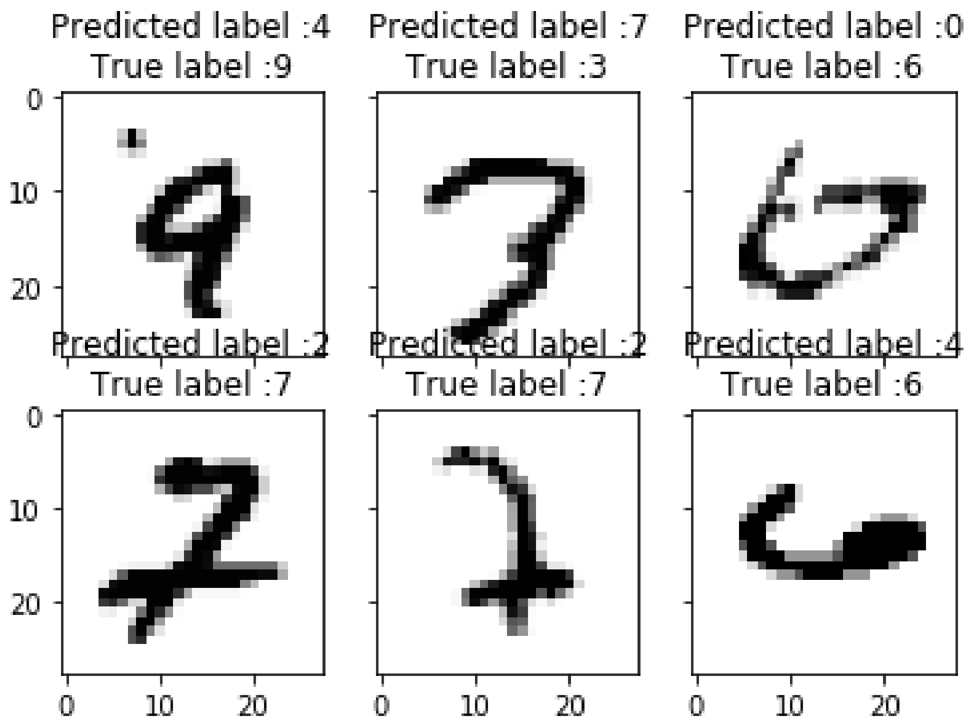


In [46]:
```
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_cls)
# plot the confusion matrix
plt.figure(figsize=(8,8))
plot_confusion_matrix(confusion_mtx, classes = range(10))
```




```
In [47]: #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)
```



2.8 Early Stopping as a regularizer

- If you continue training, at some point the validation loss will start to increase: that is when the model starts to **overfit**. We can use EarlyStopping as a regularizer:

```
In [48]: from keras.callbacks import EarlyStopping

         early_stop = EarlyStopping(monitor='val_loss', patience=5, verbose=1)
         #Also possible choice:
         #early_stop = EarlyStopping(monitor='val_acc', patience=5, verbose=1)

         dropout=0.5
```

```
model_ES = Sequential()
model_ES.add(Dense(512, activation='relu', kernel_regularizer=l2(l2_lambda), input_dim=
model_ES.add(Dropout(dropout))
model_ES.add(Dense(256, activation='relu', kernel_regularizer=l2(l2_lambda)))
model_ES.add(Dropout(dropout))
model_ES.add(Dense(10, activation='softmax'))

model_ES.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'

model_dropout.summary()
```

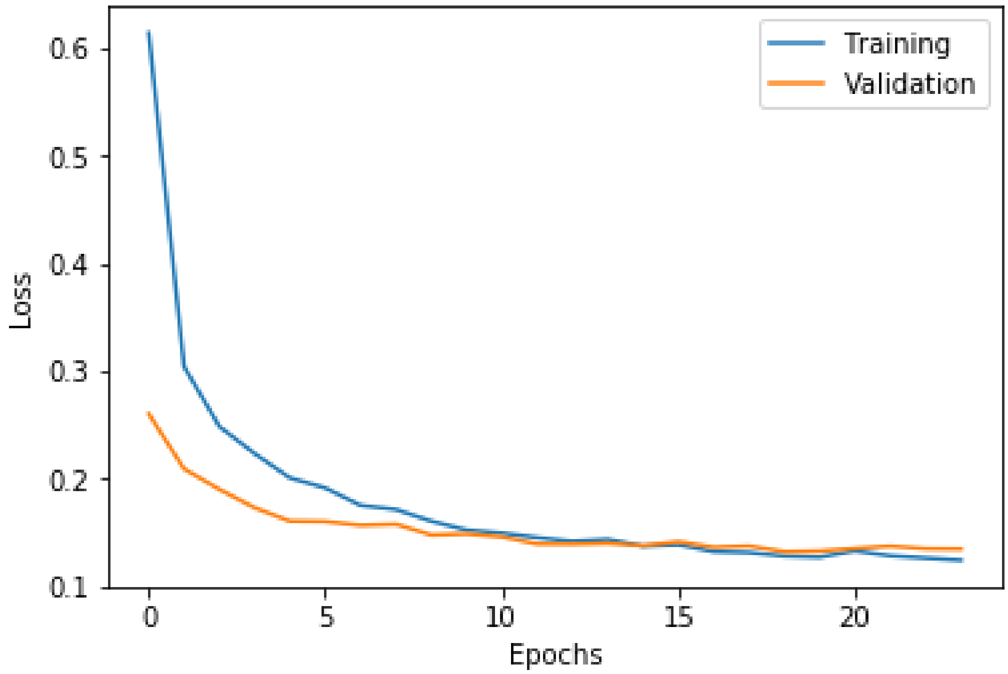
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2570
=================================================================
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
_________________________________________________________________
```

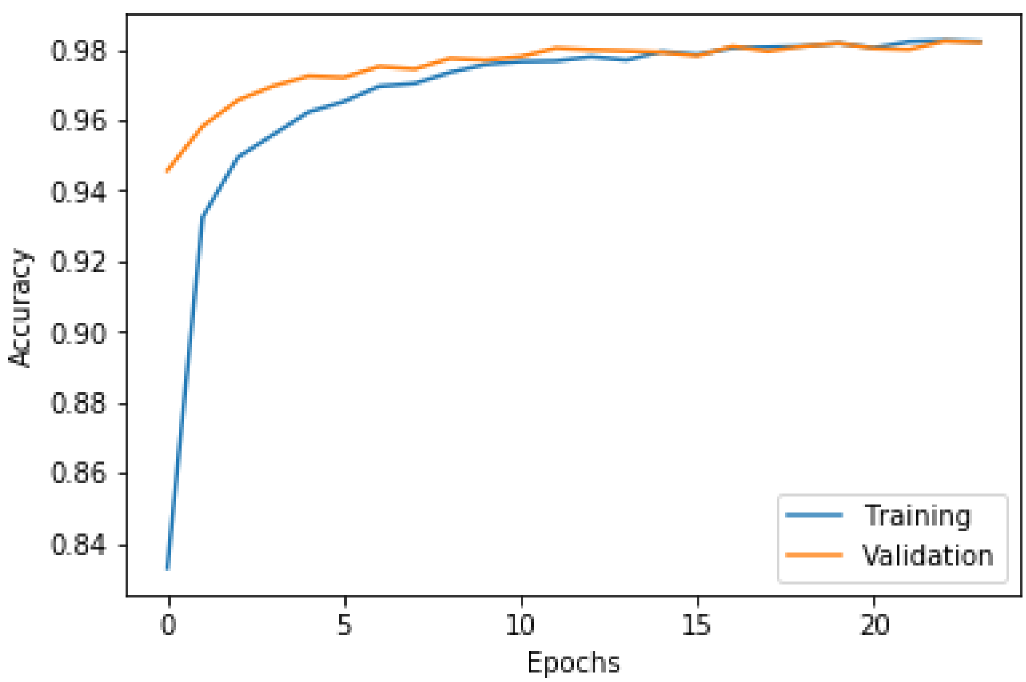
```
In [49]: history_ES = model_ES.fit(X_train, Y_train, validation_data = (X_test, Y_test), epochs=
                      callbacks=[early_stop])
```

```
Train on 42000 samples, validate on 10000 samples
Epoch 1/100
42000/42000 [==============================] - 6s 137us/step - loss: 0.6138 - acc: 0.8329 - val_
Epoch 2/100
42000/42000 [==============================] - 5s 123us/step - loss: 0.3040 - acc: 0.9324 - val_
Epoch 3/100
42000/42000 [==============================] - 8s 186us/step - loss: 0.2485 - acc: 0.9494 - val_
Epoch 4/100
42000/42000 [==============================] - 6s 133us/step - loss: 0.2234 - acc: 0.9558 - val_
Epoch 5/100
42000/42000 [==============================] - 5s 117us/step - loss: 0.2007 - acc: 0.9622 - val_
Epoch 6/100
42000/42000 [==============================] - 4s 107us/step - loss: 0.1915 - acc: 0.9652 - val_
Epoch 7/100
```

```
42000/42000 [==============================] - 4s 102us/step - loss: 0.1752 - acc: 0.9695 - val_
Epoch 8/100
42000/42000 [==============================] - 6s 149us/step - loss: 0.1714 - acc: 0.9703 - val_
Epoch 9/100
42000/42000 [==============================] - 6s 146us/step - loss: 0.1605 - acc: 0.9734 - val_
Epoch 10/100
42000/42000 [==============================] - 5s 123us/step - loss: 0.1523 - acc: 0.9757 - val_
Epoch 11/100
42000/42000 [==============================] - 5s 130us/step - loss: 0.1494 - acc: 0.9765 - val_
Epoch 12/100
42000/42000 [==============================] - 5s 124us/step - loss: 0.1453 - acc: 0.9767 - val_
Epoch 13/100
42000/42000 [==============================] - 5s 122us/step - loss: 0.1418 - acc: 0.9779 - val_
Epoch 14/100
42000/42000 [==============================] - 5s 127us/step - loss: 0.1436 - acc: 0.9769 - val_
Epoch 15/100
42000/42000 [==============================] - 5s 130us/step - loss: 0.1373 - acc: 0.9793 - val_
Epoch 16/100
42000/42000 [==============================] - 5s 116us/step - loss: 0.1386 - acc: 0.9785 - val_
Epoch 17/100
42000/42000 [==============================] - 5s 125us/step - loss: 0.1323 - acc: 0.9803 - val_
Epoch 18/100
42000/42000 [==============================] - 5s 122us/step - loss: 0.1314 - acc: 0.9806 - val_
Epoch 19/100
42000/42000 [==============================] - 5s 127us/step - loss: 0.1281 - acc: 0.9809 - val_
Epoch 20/100
42000/42000 [==============================] - 5s 124us/step - loss: 0.1271 - acc: 0.9817 - val_
Epoch 21/100
42000/42000 [==============================] - 5s 121us/step - loss: 0.1329 - acc: 0.9804 - val_
Epoch 22/100
42000/42000 [==============================] - 5s 119us/step - loss: 0.1284 - acc: 0.9821 - val_
Epoch 23/100
42000/42000 [==============================] - 6s 134us/step - loss: 0.1263 - acc: 0.9825 - val_
Epoch 24/100
42000/42000 [==============================] - 5s 121us/step - loss: 0.1242 - acc: 0.9822 - val_
Epoch 00024: early stopping
```

```
In [50]: plot_history(history_ES)
```


Training
Validation
0.6
0.5
0.4
0.3
0.2
0.1
Loss
0
5
10
15
20
Epochs


0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
Accuracy
Training
Validation
0
5
10
15
20
Epochs

3 Bonus: Inspecting Layers

```
In [51]: # We already used `summary`
         model_dropout.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2570
=================================================================
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
_________________________________________________________________
```

3.0.1 `model.layers` is iterable

```
In [52]: print('Model Input Tensors: ', model.input)
         print('Layers - Network Configuration:')
         for layer in model.layers:
             print(layer.name, layer.trainable)
             print('Layer Configuration:')
             print(layer.get_config(), )
         print('Model Output Tensors: ', model.output)
```

```
('Model Input Tensors: ', <tf.Tensor 'dense_1_input:0' shape=(?, 784) dtype=float32>)
Layers - Network Configuration:
('dense_1', True)
Layer Configuration:
({'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'distribution': 'uniform',
('dense_2', True)
Layer Configuration:
({'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'distribution': 'uniform',
('dense_3', True)
Layer Configuration:
({'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'distribution': 'uniform',
('Model Output Tensors: ', <tf.Tensor 'dense_3/Softmax:0' shape=(?, 10) dtype=float32>)
```

3.1 Extract hidden layer representation of the given data

One **simple** way to do it is to use the weights of your model to build a new model that's truncated at the layer you want to read.

Then you can run the `._predict(X_batch)` method to get the activations for a batch of inputs.

```
In [53]: model_truncated = Sequential()
         model_truncated.add(Dense(512, activation='relu', input_shape=(784,)))
         model_truncated.add(Dropout(dropout))
         model_truncated.add(Dense(256, activation='relu'))

         for i, layer in enumerate(model_truncated.layers):
             layer.set_weights(model_dropout.layers[i].get_weights())

         model_truncated.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['acc
```

```
In [54]: # Check
         np.all(model_truncated.layers[0].get_weights()[0] == model.layers[0].get_weights()[0])
```

```
Out[54]: False
```

```
In [55]: hidden_features = model_truncated.predict(X_train)
```

```
In [56]: hidden_features.shape
```

```
Out[56]: (42000, 256)
```

```
In [57]: X_train.shape
```

```
Out[57]: (42000, 784)
```

Hint: Alternative Method to get activations (Using `keras.backend` `function` on Tensors)

```
def get_activations(model, layer, X_batch):
    activations_f = K.function([model.layers[0].input, K.learning_phase()], [layer.output,])
    activations = activations_f((X_batch, False))
    return activations
```

3.1.1 Generate the Embedding of Hidden Features

Dimensionality reduction to dim=20 by using principal component analysis (PCA)

```
In [58]: from sklearn.decomposition import PCA
         pca = PCA(n_components=20)
         pca_result = pca.fit_transform(hidden_features)
         print('Variance PCA: {}'.format(np.sum(pca.explained_variance_ratio_)))
```

```
Variance PCA: 0.950339972973
```

Dimensionality reduction to dim=2 by using t-distributed stochastic neighbor embedding (TSNE)

```
In [59]: from sklearn.manifold import TSNE

         tsne = TSNE(n_components=2)
         X_tsne = tsne.fit_transform(pca_result[:1000]) ## Reduced for computational issues
```

```
In [60]: colors_map = np.argmax(Y_train, axis=1)
```

```
In [61]: X_tsne.shape
```

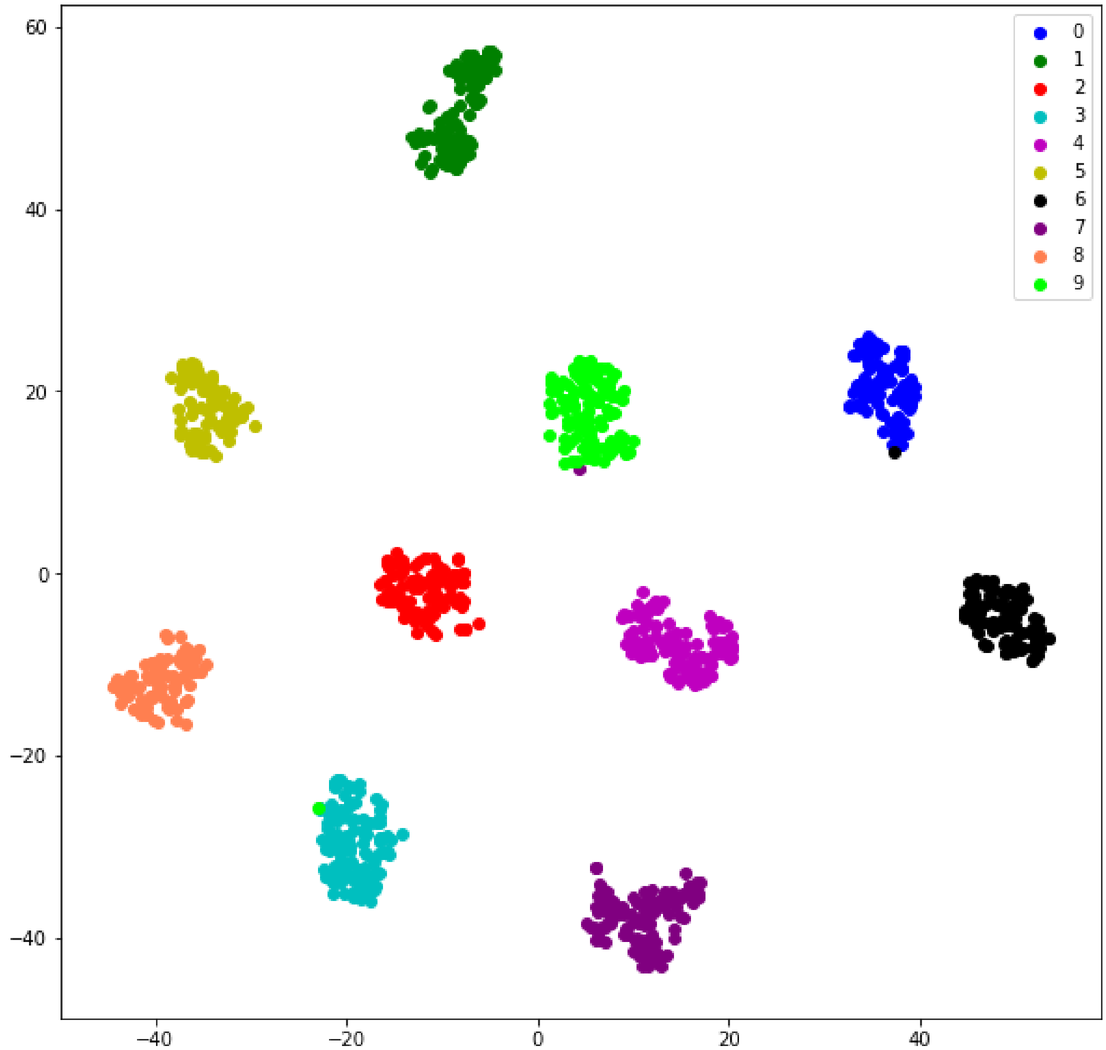
```
Out[61]: (1000, 2)
```

```
In [62]: nb_classes=10
```

```
In [63]: np.where(colors_map==6)
```

```
Out[63]: (array([    2,     8,    23, ..., 41927, 41983, 41988]),)
```

```
In [64]: colors = np.array([x for x in 'b-g-r-c-m-y-k-purple-coral-lime'.split('-')])
         colors_map = np.argmax(Y_train, axis=1)
         colors_map = colors_map[:1000]
         plt.figure(figsize=(10,10))
         for cl in range(nb_classes):
             indices = np.where(colors_map==cl)
             plt.scatter(X_tsne[indices,0], X_tsne[indices, 1], c=colors[cl], label=cl)
         plt.legend()
         plt.show()
```


60
40
20
0
−20
−40
−40
−20
0
20
40
0
1
2
3
4
5
6
7
8
9