mnist_mlp-Copy1

August 2, 2018

0.1 Convolutional Neural Networks

In this notebook, we train an MLP to classify images from the MNIST database.

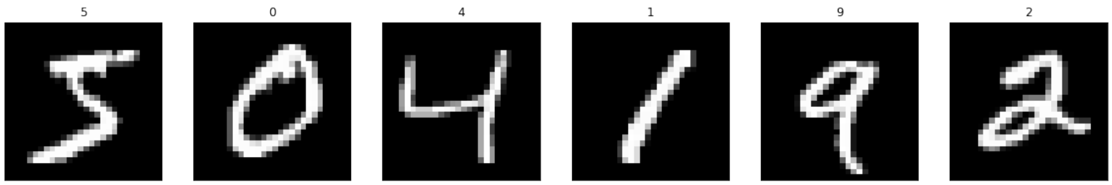
0.1.1 1. Load MNIST Database

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

        # use Keras to import pre-shuffled MNIST database
        (X_train, y_train), (X_test, y_test) = mnist.load_data()

        print("The MNIST database has a training set of %d examples." % len(X_train))
        print("The MNIST database has a test set of %d examples." % len(X_test))
```

```
Using TensorFlow backend.


The MNIST database has a training set of 60000 examples.
The MNIST database has a test set of 10000 examples.
```

0.1.2 2. Visualize the First Six Training Images

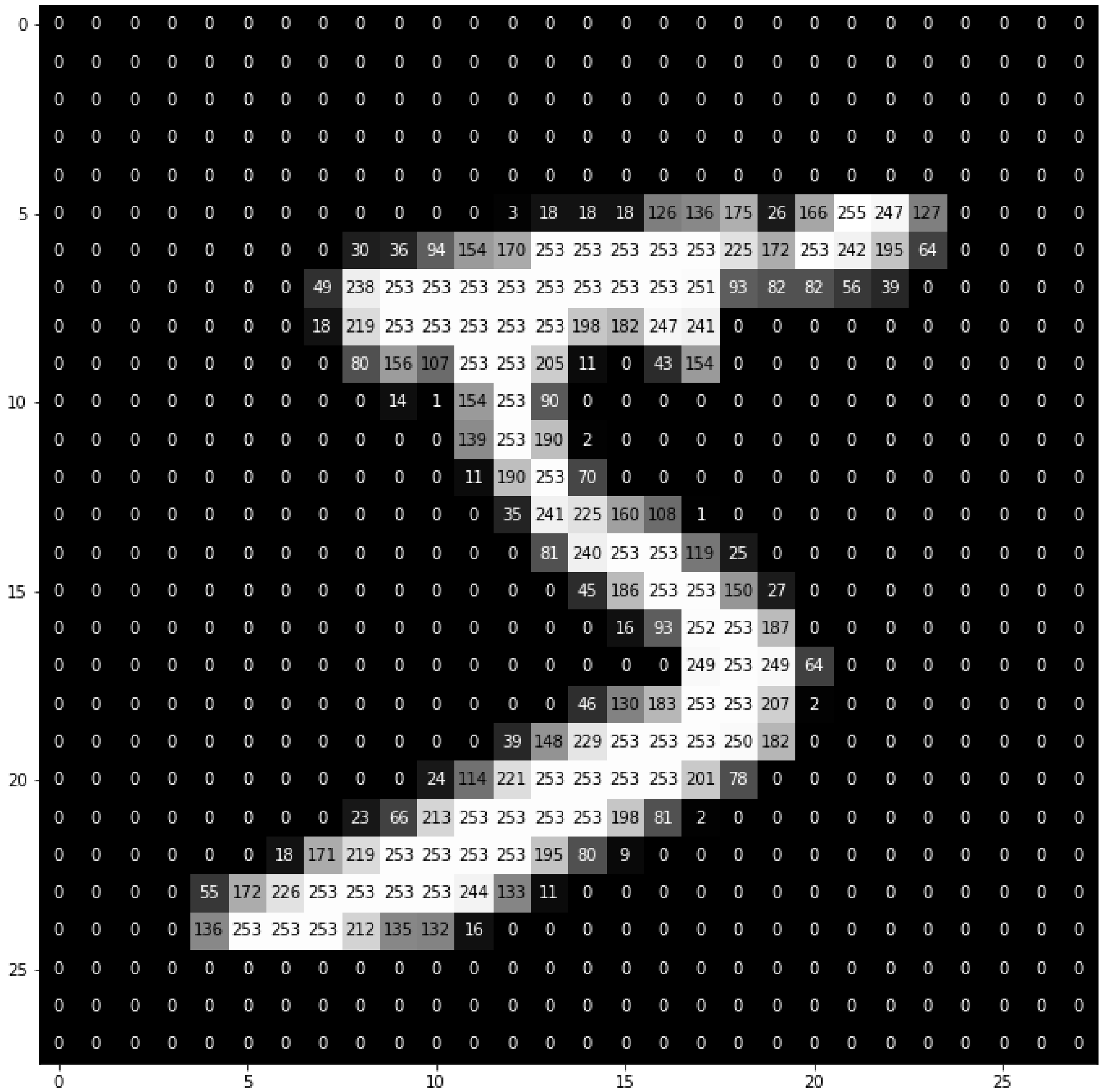
```
In [2]: import matplotlib.pyplot as plt
        %matplotlib inline
        import matplotlib.cm as cm
        import numpy as np

        # plot first six training images

        fig = plt.figure(figsize=(20,20))
        for i in range(6):
            ax = fig.add_subplot(1, 6, i+1, xticks=[], yticks=[])
            ax.imshow(X_train[i], cmap='gray')
            ax.set_title(str(y_train[i]))
```



0.1.3 3. View an Image in More Detail

```
In [3]: def visualize_input(img, ax):
            ax.imshow(img, cmap='gray')
            width, height = img.shape
            thresh = img.max()/2.5
            print(thresh)
            for x in range(width):
                for y in range(height):

                    ax.annotate(str(round(img[x][y],2)), xy=(y,x),
                                horizontalalignment='center',
                                verticalalignment='center',
                                color='white' if img[x][y]<thresh else 'black')

        fig = plt.figure(figsize = (12,12))
        ax = fig.add_subplot(111)
        visualize_input(X_train[0], ax)
```

```
102.0
```



0.1.4 4. Rescale the Images by Dividing Every Pixel in Every Image by 255

In [4]:
```
# rescale [0,255] --> [0,1]
X_train = X_train.astype('float32')/255
X_test = X_test.astype('float32')/255
```

0.1.5 5. Encode Categorical Integer Labels Using a One-Hot Scheme

In [5]:
```
from keras.utils import np_utils

# print first ten (integer-valued) training labels
print('Integer-valued labels:')
print(y_train[:10])
```

```
# one-hot encode the labels
y_train = np_utils.to_categorical(y_train, 10)
y_test = np_utils.to_categorical(y_test, 10)

# print first ten (one-hot) training labels
print('One-hot labels:')
print(y_train[:10])
```

```
Integer-valued labels:
[5 0 4 1 9 2 1 3 1 4]
One-hot labels:
[[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
```

0.1.6 6. Define the Model Architecture

```
In [6]: from keras.models import Sequential
        from keras.layers import Dense, Dropout, Flatten

        # define the model
        model = Sequential()
        model.add(Flatten(input_shape=X_train.shape[1:]))
        model.add(Dense(512, activation='relu')) #512
        model.add(Dropout(0.2))
        model.add(Dense(512, activation='relu'))
        model.add(Dropout(0.2))

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

        # summarize the model
        model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               401920
```

```
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
```

0.1.7 7. Compile the Model

```
In [7]: # compile the model
        model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
                      metrics=['accuracy'])
```

0.1.8 8. Calculate the Classification Accuracy on the Test Set (Before Training)

```
In [8]: # evaluate test accuracy
        score = model.evaluate(X_test, y_test, verbose=0)
        accuracy = 100*score[1]

        # print test accuracy
        print('Test accuracy: %.4f%%' % accuracy)
```

```
Test accuracy: 11.9600%
```

0.1.9 9. Train the Model

```
In [9]: from keras.callbacks import ModelCheckpoint

        # train the model
        checkpointer = ModelCheckpoint(filepath='mnist.model.best.hdf5',
                                       verbose=1, save_best_only=True)
        hist = model.fit(X_train, y_train, batch_size=250, epochs=10,
                  validation_split=0.2, callbacks=[checkpointer],
                  verbose=1, shuffle=True)
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 6s 116us/step - loss: 0.3266 - acc: 0.8993 - va

Epoch 00001: val_loss improved from inf to 0.12071, saving model to mnist.model.best.hdf5
```

```
Epoch 2/10
48000/48000 [==============================] - 6s 116us/step - loss: 0.1266 - acc: 0.9608 - va

Epoch 00002: val_loss improved from 0.12071 to 0.09419, saving model to mnist.model.best.hdf5
Epoch 3/10
48000/48000 [==============================] - 6s 121us/step - loss: 0.0846 - acc: 0.9732 - va

Epoch 00003: val_loss improved from 0.09419 to 0.08274, saving model to mnist.model.best.hdf5
Epoch 4/10
48000/48000 [==============================] - 8s 158us/step - loss: 0.0649 - acc: 0.9797 - va

Epoch 00004: val_loss improved from 0.08274 to 0.07943, saving model to mnist.model.best.hdf5
Epoch 5/10
48000/48000 [==============================] - 8s 174us/step - loss: 0.0519 - acc: 0.9828 - va

Epoch 00005: val_loss did not improve from 0.07943
Epoch 6/10
48000/48000 [==============================] - 8s 159us/step - loss: 0.0419 - acc: 0.9865 - va

Epoch 00006: val_loss did not improve from 0.07943
Epoch 7/10
48000/48000 [==============================] - 5s 107us/step - loss: 0.0329 - acc: 0.9888 - va

Epoch 00007: val_loss did not improve from 0.07943
Epoch 8/10
48000/48000 [==============================] - 5s 104us/step - loss: 0.0293 - acc: 0.9901 - va

Epoch 00008: val_loss did not improve from 0.07943
Epoch 9/10
48000/48000 [==============================] - 6s 116us/step - loss: 0.0277 - acc: 0.9907 - va

Epoch 00009: val_loss did not improve from 0.07943
Epoch 10/10
48000/48000 [==============================] - 7s 136us/step - loss: 0.0232 - acc: 0.9922 - va

Epoch 00010: val_loss did not improve from 0.07943
```

0.1.10 10. Load the Model with the Best Classification Accuracy on the Validation Set

In [10]:
```
# load the weights that yielded the best validation accuracy
model.load_weights('mnist.model.best.hdf5')
```

0.1.11 11. Calculate the Classification Accuracy on the Test Set

In [11]:
```
# evaluate test accuracy
score = model.evaluate(X_test, y_test, verbose=0)
accuracy = 100*score[1]
```

```
        # print test accuracy
        print('Test accuracy: %.4f%%' % accuracy)
```

```
Test accuracy: 98.0300%
```

0.1.12 Grid Search

```
In [ ]: import numpy
        from sklearn.model_selection import GridSearchCV
        from keras.models import Sequential
        from keras.layers import Dense
        from keras.wrappers.scikit_learn import KerasClassifier
        # Function to create model, required for KerasClassifier
        def create_model():
                # create model
                model = Sequential()
                model.add(Dense(12, input_dim=8, activation='relu'))
                model.add(Dense(1, activation='sigmoid'))
                # Compile model
                model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy
                return model
        # fix random seed for reproducibility
        seed = 7
        numpy.random.seed(seed)
        # load dataset
        dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
        # split into input (X) and output (Y) variables
        X = dataset[:,0:8]
        Y = dataset[:,8]
        # create model
        model = KerasClassifier(build_fn=create_model, verbose=0)
        # define the grid search parameters
        batch_size = [10, 20, 40, 60, 80, 100]
        epochs = [10, 50, 100]
        param_grid = dict(batch_size=batch_size, epochs=epochs)
        grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
        grid_result = grid.fit(X, Y)
        # 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))
```