Classification of Grey-scale Images

Paula Kimmerling

April 26,2022

1. The Dataset

2. Training Choices

3. Fits

The Dataset

The Dataset

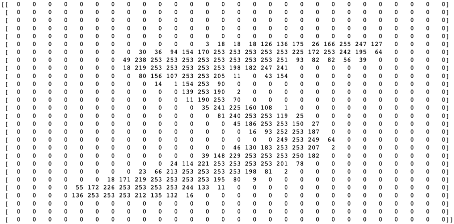
- Wanted to work with image classification, found the MNIST database of handwritten digits.

The Dataset

- Wanted to work with image classification, found the MNIST database of handwritten digits.
- There were 60,000 training samples and 10,000 testing samples already separated.

The Dataset

- Wanted to work with image classification, found the MNIST database of handwritten digits.
- There were 60,000 training samples and 10,000 testing samples already separated.
- Each sample was a pre-processed 28 x 28 matrix of integers 0 - 255, with 0 indicating no saturation vs 255 full saturation.



```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253 253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201  78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

My Desired Implementation

My Desired Implementation

1. Read in 60,000 matrices and reshape them to be 60,000 row vectors.

My Desired Implementation

1. Read in 60,000 matrices and reshape them to be 60,000 row vectors.
2. Encode output data 0-9 as binary (only need 4 outputs):

$$[0111] = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 4 + 2 + 1 = 7$$

My Desired Implementation

1. Read in 60,000 matrices and reshape them to be 60,000 row vectors.
2. Encode output data 0-9 as binary (only need 4 outputs):

$$[0111] = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 4 + 2 + 1 = 7$$

3. Use ensemble cost function.

My Desired Implementation

1. Read in 60,000 matrices and reshape them to be 60,000 row vectors.
2. Encode output data 0-9 as binary (only need 4 outputs):

$$[0111] = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 4 + 2 + 1 = 7$$

3. Use ensemble cost function.
4. Use LReLU as first activation and logistic as second activation.

My Desired Implementation

1. Read in 60,000 matrices and reshape them to be 60,000 row vectors.
2. Encode output data 0-9 as binary (only need 4 outputs):

$$[0111] = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 4 + 2 + 1 = 7$$

3. Use ensemble cost function.
4. Use LReLU as first activation and logistic as second activation.
5. Use bfgs as the non-linear solver.

My Desired Implementation

1. Read in 60,000 matrices and reshape them to be 60,000 row vectors.
2. Encode output data 0-9 as binary (only need 4 outputs):

$$[0111] = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 4 + 2 + 1 = 7$$

3. Use ensemble cost function.
4. Use LReLU as first activation and logistic as second activation.
5. Use bfgs as the non-linear solver.
6. Read back the binary numbers to integer to compare.

What Actually Happened

60,000 samples was too much for my local code. So I used the MLPCLassifier from sci-kit learn which runs faster because it has bits already compiled:

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver=‘lbfgs’,
    activation=‘logistic’,alpha=regParam,
hidden_layer_sizes=(k,), random_state=1)
start = time.time()
clf.fit(trainX,trainY)
end = time.time()
print("Time to train:", end-start)
```

`Time to train:` 581.859 $\approx$ 9.7 minutes

Fit to Train Data

It trained to fit the data perfectly in this case, because the ‘score’ attribute of the classifier gives an average percentage accuracy.

```
    pred = clf.predict(trainX)
    clf.score(trainX,trainY)
```

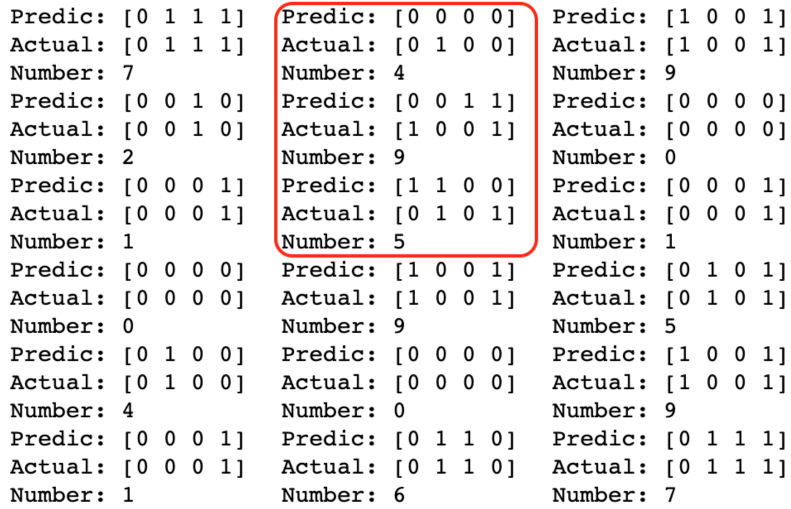
1.0 $= 100\%$

It also didn't do too badly on the testing data:

```
    pred = clf.predict(testX)
    clf.score(testX,testY)
```

0.8276 $= 82.76\%$

Examples of Test Data Outputs



```
Predic: [0 1 1 1]    Predic: [0 0 0 0]    Predic: [1 0 0 1]
Actual: [0 1 1 1]    Actual: [0 1 0 0]    Actual: [1 0 0 1]
Number: 7            Number: 4            Number: 9
Predic: [0 0 1 0]    Predic: [0 0 1 1]    Predic: [0 0 0 0]
Actual: [0 0 1 0]    Actual: [1 0 0 1]    Actual: [0 0 0 0]
Number: 2            Number: 9            Number: 0
Predic: [0 0 0 1]    Predic: [1 1 0 0]    Predic: [0 0 0 1]
Actual: [0 0 0 1]    Actual: [0 1 0 1]    Actual: [0 0 0 1]
Number: 1            Number: 5            Number: 1
Predic: [0 0 0 0]    Predic: [1 0 0 1]    Predic: [0 1 0 1]
Actual: [0 0 0 0]    Actual: [1 0 0 1]    Actual: [0 1 0 1]
Number: 0            Number: 9            Number: 5
Predic: [0 1 0 0]    Predic: [0 0 0 0]    Predic: [1 0 0 1]
Actual: [0 1 0 0]    Actual: [0 0 0 0]    Actual: [1 0 0 1]
Number: 4            Number: 0            Number: 9
Predic: [0 0 0 1]    Predic: [0 1 1 0]    Predic: [0 1 1 1]
Actual: [0 0 0 1]    Actual: [0 1 1 0]    Actual: [0 1 1 1]
Number: 1            Number: 6            Number: 7
```