

---

**Part1-Step1: Introduction For MNIST**


---

MNIST stands for Mixed National Institute of Standards and Technology, which has produced a handwritten digits dataset. This is one of the most researched datasets in machine learning, and is used to classify handwritten digits. This dataset is helpful for predictive analytics because of its sheer size, allowing deep learning to work its magic efficiently. This dataset contains 50000 examples and 784 dimensional feature vector, formatted as 28 x 28 pixel monochrome images.

---

**Part1-Step2: Training An SOM**


---

**In the process of training SOM, Gaussian distribution and Euclidean distance calculation are used.**

```
class EuclideanDistance {
public:
    float EvaluateDistance(Eigen::VectorXd v1, Eigen::VectorXd v2){
        float KernelValue = (v1 - v2).norm();
        return KernelValue;
    }
};
```

```
class GaussianDistribution {
public:
    float Gaussian(float x, float sigma){
        float value = exp(-((float)x/(float)sigma));
        return value;
    }
}
```

**And SOMLearningFunction, UpdateSOMLearningRate, UpdateSOMSigmaNeighbouring are constantly updated during the learning process.**

```
class UpdateSOMLearningRate {
public:
    float UpdateLearningRate(float L0, float LearningRateFinal, float Iter, float lambda){
        float value = (L0 - LearningRateFinal) * L0*exp(-((float)Iter/(float)lambda));
        return value;
    }
};
```

```
class UpdateSOMSigmaNeighbouring {
public:
    float UpdateSigmaNeighbouring(float SigmaNeighbouringZero, float SigmaNeighbouringFinal){
        float value = (SigmaNeighbouringZero - SigmaNeighbouringFinal) * exp(-((float)Iter/(float)lambda));
        return value;
    }
};
```

2

```

    }

    SOMTrain::LearningRate = UpdateLearningRate(SOMTrain::LearningRateInitial ,
    SOMTrain::SigmaNeighbouring = UpdateSigmaNeighbouring(SOMTrain::SigmaNeig
i , SOMTrain::lambda );

}
}
}

```

---

### Part1-Step3: Parameter Adjustment

---

Through continuous experiment adjustment, the initial value of 2D lattices is finally adjusted to 80 X 80, while continuously adjusting NeighbouringInitial, NeighbouringFinal, and learningrate, the experimental process is as follows:

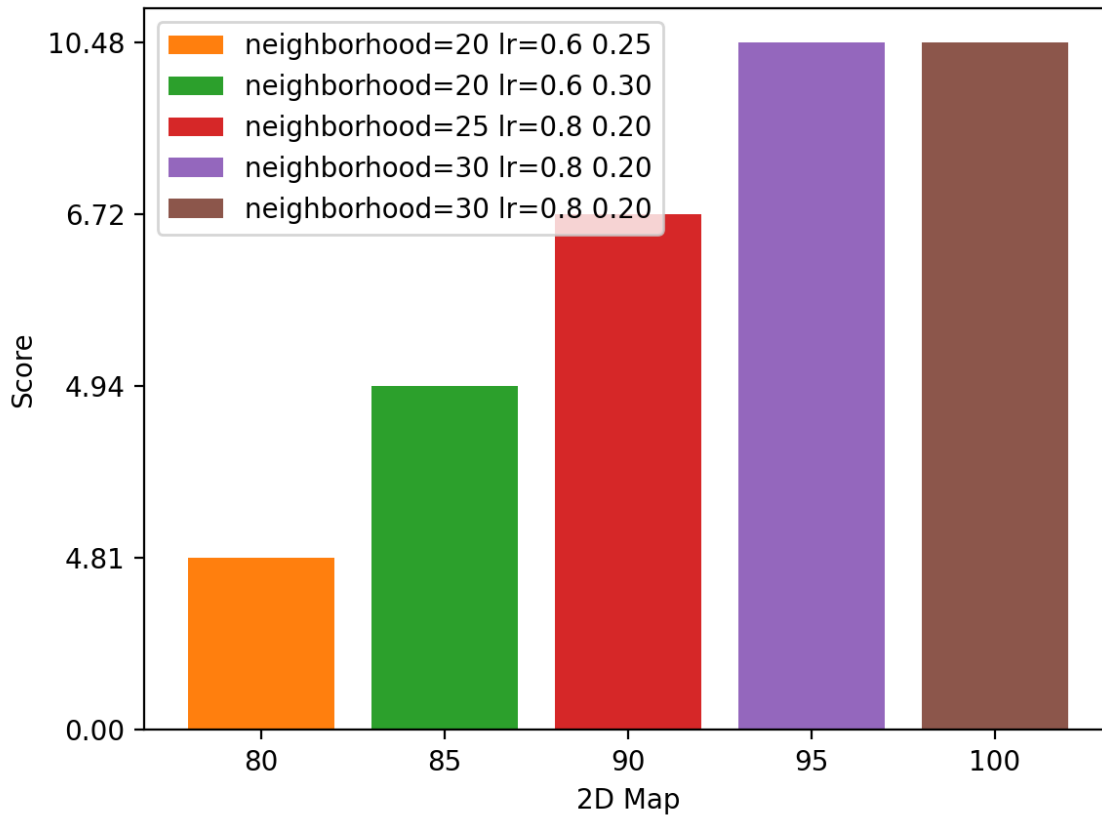


Figure 1: Parameter1

```

19 using namespace std;
20
21 int main()
22 {
23     SOM test;
24     test.ReadData( filename: "../SOM/MNIST_data.txt", NumberVectors: 130, InputVectorSize: 2000);
25     test.InitializeMap( xsize: 80, ysize: 80);
26     std::cout << "2D Map : " << test.xsize << "x" << test.ysize << "\n";
27     test.PrintToCSVFileRowWise( filename: "../SOM/BeforeTraining.txt", test.SOMMap, test.xsize, test.ysize, test.InputVectorSize);
28
29     test.SigmaNeighbouringInitial = 30;
30     test.SigmaNeighbourhoodFinal = 1;
31     test.LearningRateInitial = 0.8;
32     test.LearningRateFinal = 0.2;
33
34     test.Train( Iter: 100);
35     test.PrintToCSVFileRowWise( filename: "../SOM/Results.txt", test.SOMMap, test.xsize, test.ysize, test.InputVectorSize);
36
37     std::cout << "Lambda : " << test.lambda << "\n";
38     std::cout << "L0 : " << test.LearningRateInitial << "\n";
39     std::cout << "Sigma0 Neighbours : " << test.SigmaNeighbouringInitial << "\n";
40     std::cout << "L : " << test.LearningRate << "\n";
41     std::cout << "Sigma Neighbours : " << test.SigmaNeighbouring << "\n";
42     std::cout << "Score : " << test.Score << "\n";
43     return 0;
44 }
45
46 Declared In: ClassesSOM.h
47 public:
48 float SOMTrain::SigmaNeighbouring ;
49
50 Run: SOMcpp
51 /Users/deceber/Desktop/SOMcpp/cmake-build-debug/SOMcpp
52 Closing input file..
53 2D Map : 80x80
54 30
55 Lambda : 2.5
56 L0 : 0.8
57 Sigma0 Neighbours: 30
58 L : 0.2
59 Sigma Neighbours: 1
60 Score : 4.81626
61 Process finished with exit code 0

```

Figure 2: Parameter1

```

21 int main()
22 {
23     SOM test;
24     test.ReadData( filename: "../SOM/MNIST_data.txt", NumberVectors: 130, InputVectorSize: 2000);
25     test.InitializeMap( xsize: 80, ysize: 80);
26     std::cout << "2D Map : " << test.xsize << "x" << test.ysize << "\n";
27     test.PrintToCSVFileRowWise( filename: "../SOM/BeforeTraining.txt", test.SOMMap, test.xsize, test.ysize, test.InputVectorSize);
28
29     test.SigmaNeighbouringInitial = 30;
30     test.SigmaNeighbourhoodFinal = 1;
31     test.LearningRateInitial = 0.8;
32     test.LearningRateFinal = 0.2;
33
34     test.Train( Iter: 100);
35     test.PrintToCSVFileRowWise( filename: "../SOM/Results.txt", test.SOMMap, test.xsize, test.ysize, test.InputVectorSize);
36
37     std::cout << "Lambda : " << test.lambda << "\n";
38     std::cout << "L0 : " << test.LearningRateInitial << "\n";
39     std::cout << "Sigma0 Neighbours : " << test.SigmaNeighbouringInitial << "\n";
40     std::cout << "L : " << test.LearningRate << "\n";
41     std::cout << "Sigma Neighbours : " << test.SigmaNeighbouring << "\n";
42     std::cout << "Score : " << test.Score << "\n";
43     return 0;
44 }
45
46 Declared In: ClassesSOM.h
47 public:
48 float SOMTrain::SigmaNeighbouring ;
49
50 Run: SOMcpp
51 Closing input file..
52 2D Map : 80x80
53 30
54 Lambda : 2.5
55 L0 : 0.8
56 Sigma0 Neighbours: 30
57 L : 0.2
58 Sigma Neighbours: 1
59 Score : 6.72422
60 Process finished with exit code 0

```

Figure 3: Parameter2

The screenshot shows a C++ IDE with the following components:

- Project Explorer (Left):** Shows the project structure for 'SOMcpp'. The 'SOM' folder is expanded, showing files like 'BeforeTraining.txt', 'ClassesSOM.h', 'GenerateColorsExample.R', 'main.cpp', 'PlotColorMatrix.R', 'Results.txt', 'SOM1', 'SOM.cpp', 'SOM.h', 'SOM\_MNIST\_data.txt', and 'SOM.xcodeproj'.
- Code Editor (Center):** Displays the 'main.cpp' file. The code includes headers for `<iostream>`, `<fstream>`, and `<string>`, and includes 'SOM.h' and 'ClassesSOM.h'. It uses the `std` namespace and defines a `main` function. The code performs an SOM test by reading data from 'SOM\_MNIST\_data.txt', initializing a SOM map (20x20), and training it with 100 iterations. It prints the results to 'Results.txt'.
- Run Console (Bottom):** Shows the output of the program. It indicates that the input file is closed, the map size is 20x20, and the final score is 18.4885. The process finished with exit code 0.

Figure 4: Parameter3

## Part1-Step4: Result

The image shows a CMakeLists.txt file in a code editor. The file is titled "The file is too large 466.56 MB. Showing read-only mode." and contains a large list of numerical values, likely coordinates or weights, arranged in a grid-like structure. The values are organized into sections, with some sections having headers like "SOM1", "SOM2", "SOM3", "SOM4", "SOM5", "SOM6", "SOM7", "SOM8", "SOM9", "SOM10", "SOM11", "SOM12", "SOM13", "SOM14", "SOM15", "SOM16", "SOM17", "SOM18", "SOM19", "SOM20", "SOM21", "SOM22", "SOM23", "SOM24", "SOM25", "SOM26", "SOM27", "SOM28", "SOM29", "SOM30", "SOM31", "SOM32", "SOM33", "SOM34", "SOM35", "SOM36", "SOM37", "SOM38", "SOM39", "SOM40", "SOM41", "SOM42", "SOM43", "SOM44", "SOM45", "SOM46", "SOM47", "SOM48", "SOM49", "SOM50", "SOM51", "SOM52", "SOM53", "SOM54", "SOM55", "SOM56", "SOM57", "SOM58", "SOM59", "SOM60", "SOM61", "SOM62", "SOM63", "SOM64", "SOM65", "SOM66", "SOM67", "SOM68", "SOM69", "SOM70", "SOM71", "SOM72", "SOM73", "SOM74", "SOM75", "SOM76", "SOM77", "SOM78", "SOM79", "SOM80", "SOM81", "SOM82", "SOM83", "SOM84", "SOM85", "SOM86", "SOM87", "SOM88", "SOM89", "SOM90", "SOM91", "SOM92", "SOM93", "SOM94", "SOM95", "SOM96", "SOM97", "SOM98", "SOM99", "SOM100".

The terminal output shows the build process, including the command "cmake -S . -B build" and the output "Build finished in 5 s 54 ms (today 01:35)".

Figure 5: Output

---

**Part2: Introduction On MNIST**

---

The MNIST data is split into three parts: 55,000 data points of training data (mnist.train), 10,000 points of test data (mnist.test), and 5,000 points of validation data (mnist.validation). This split is very important: it's essential in machine learning that we have separate data which we don't learn from so that we can make sure that what we've learned actually generalizes!

the training images are mnist.train.images and the training labels are mnist.train.labels.

Each image is 28 pixels by 28 pixels. Each entry in the tensor is a pixel intensity between 0 and 1, for a particular pixel in a particular image.

Each image in MNIST has a corresponding label, a number between 0 and 9 representing the digit drawn in the image.

---

**Part2: Multinomial Logistic Regression**

---

```
learning_rate = 0.09
training_epochs = 20
batch_size = 80
display_step = 1

# mnist data image of shape 28*28=784
# 0 - 9 recognition -> 10 classes
x = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])

W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# softmax model
pred = tf.nn.softmax(tf.matmul(x, W) + b)

# using cross entropy
# use gradient descent
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred), reduction_indices=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

```

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)

    for epoch in range(training_epochs):
        avg_cost = 0
        total_batch = int(mnist.train.num_examples/batch_size)
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            _, c = sess.run([optimizer, cost], feed_dict={x: batch_xs,
                                                            y: batch_ys})
            avg_cost += c / total_batch
        if (epoch+1) % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))

    print("Finished!")

    correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    print("Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels}))

Epoch: 0001 cost= 0.532864098
Epoch: 0002 cost= 0.361025149
Epoch: 0003 cost= 0.330213694
Epoch: 0004 cost= 0.316047758
Epoch: 0005 cost= 0.308028520
Epoch: 0006 cost= 0.301304304
Epoch: 0007 cost= 0.294422028
Epoch: 0008 cost= 0.291574168
Epoch: 0009 cost= 0.286512886
Epoch: 0010 cost= 0.288759521
Epoch: 0011 cost= 0.278943424
Epoch: 0012 cost= 0.281038634
Epoch: 0013 cost= 0.278534079
Epoch: 0014 cost= 0.275540369
Epoch: 0015 cost= 0.276986441
Epoch: 0016 cost= 0.272276273
Epoch: 0017 cost= 0.271651880
Epoch: 0018 cost= 0.270200381
Epoch: 0019 cost= 0.270604570
Epoch: 0020 cost= 0.268859776
Finished!
Accuracy: 0.925

```

Figure 6: Multinomial Logistic Regression Output

## Part2: CNN

```

learning_rate = 0.009
num_steps = 2000
batch_size = 128

```

```

num_input = 784
num_classes = 10
dropout = 0.25

```

```

# Create the neural network
def conv_net(x_dict, n_classes, dropout, reuse, is_training):
    with tf.variable_scope('ConvNet', reuse=reuse):
        x = x_dict['images']

        x = tf.reshape(x, shape=[-1, 28, 28, 1])

        # 32 filters and 5 kernels in Convolution Layer
        conv1 = tf.layers.conv2d(x, 32, 5, activation=tf.nn.relu)
        conv1 = tf.layers.max_pooling2d(conv1, 2, 2)

        # 64 filters and 3 kernels in Convolution Layer2
        conv2 = tf.layers.conv2d(conv1, 64, 3, activation=tf.nn.relu)

```



```

conv2 = tf.layers.max_pooling2d(conv2, 2, 2)

fc1 = tf.contrib.layers.flatten(conv2)

fc1 = tf.layers.dense(fc1, 1024)
fc1 = tf.layers.dropout(fc1, rate=dropout, training=is_training)

out = tf.layers.dense(fc1, n_classes)

return out

```

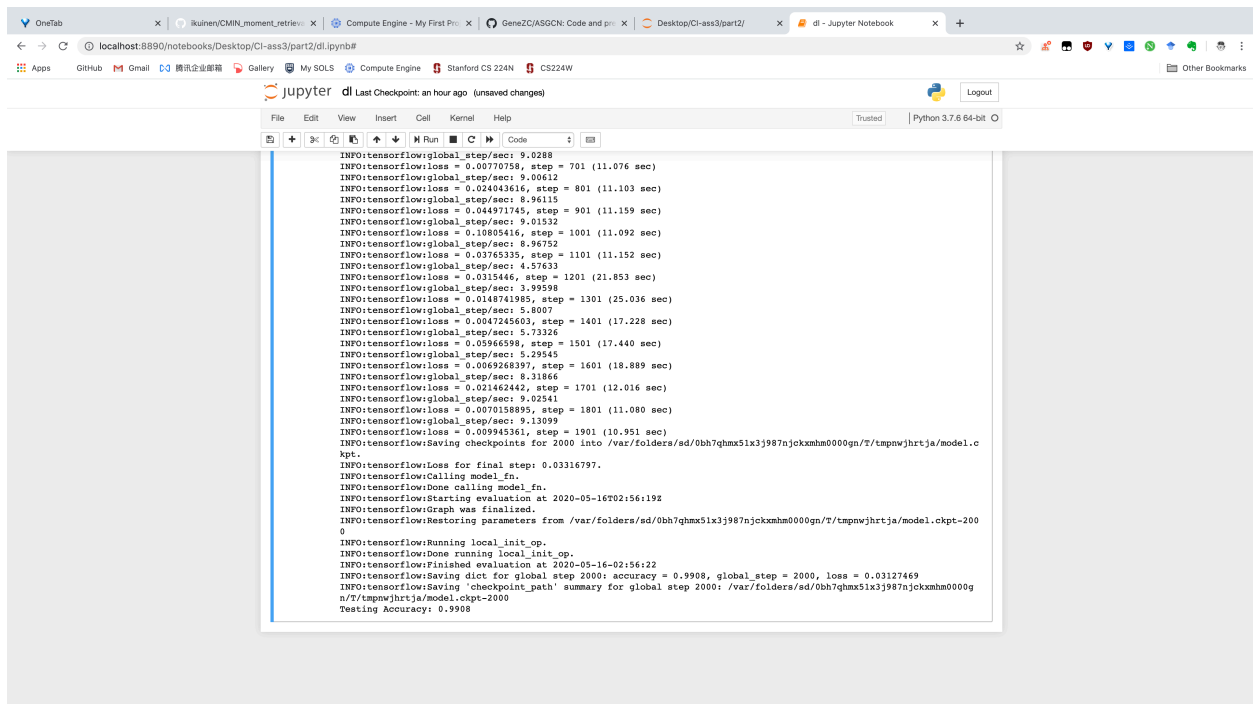


Figure 7: CNN Output

## Part2: Cell Image Classification

```
training_set_size = 25998
```

```
test_set_size = 1560
```

```

model = keras.Sequential([
    keras.layers.Convolution2D(5,(3,3),input_shape=(img_size,img_size,training_set_size),
    keras.layers.MaxPooling2D((2, 2), padding='same'),
    keras.layers.Convolution2D(10,(3,3),padding='same',data_format='channels_last'),
    keras.layers.MaxPooling2D((2, 2), padding='same'),
    keras.layers.Flatten(),
    #keras.layers.Dense(10, activation = 'relu'), # if you have a good CPU/GPU :S
    keras.layers.Dense(1, activation = 'sigmoid')
])

```

```
])

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()

epochs = 10

model.fit_generator(training_set,
                    steps_per_epoch = (training_set_size/batch_size),
                    epochs = epochs,
                    validation_data = test_set,
                    validation_steps = (test_set_size/batch_size))
```