```
for (int j = 0; j < 10; j++) {
    for (int x = 0; x < 28 * 28; ++x) {
        y1[j] += m_MnistTrainInput[i][x] * m_W[0][j][x];
    }
    y1[j] += m_B[0][j];
    y1[j] = 1. / (1 + exp(-y1[j]));

    for (int x = 0; x < 14 * 14; ++x) {
        y2[j] += m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
            * m_W[1][j][4 * x];
        y2[j] += m_MnistTrainInput[i][(x / 14 * 2 + 1) * 28 + x % 14 * 2 + 1]
            * m_W[1][j][4 * x + 1];
        y2[j] += m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
            * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
            * m_W[1][j][4 * x + 2];
        y2[j] += m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
            * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
            * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
            * m_W[1][j][4 * x + 3];

        y3[j] += m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
            * m_W[2][j][4 * x];
        y3[j] += (m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
            + m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2 + 1]
            + m_MnistTrainInput[i][(x / 14 * 2 + 1) * 28 + x % 14 * 2]
            + m_MnistTrainInput[i][(x / 14 * 2 + 1) * 28 + x % 14 * 2 + 1]) / 4
            * m_W[2][j][4 * x + 1];
        y3[j] += (m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
            - m_MnistTrainInput[i][(x / 14 * 2 + 1) * 28 + x % 14 * 2 + 1])
            * m_W[2][j][4 * x + 2];
        y3[j] += m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
            * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
            * m_W[2][j][4 * x + 3];
    }
    y2[j] += m_B[1][j];
    y2[j] = 1. / (1 + exp(-y2[j]));

    y3[j] += m_B[2][j];
    y3[j] = 1. / (1 + exp(-y3[j]));
}
```

1. 3개의 분류기 작성

맨 위에서부터 순서대로 28 * 28 모든 화소 / 14 * 14 (0, 0) 기준, 14 * 14 (1, 1) 기준, 14 * 14 (0, 0) 기준의 제곱, 14 * 14 (0, 0) 기준의 세제곱 / 14 * 14 (0, 0) 기준, 14 * 14 (0, 0) 기준의 평균, 14 * 14 (0, 0), (1, 1)의 차, 14 * 14 (0, 0) 기준의 제곱 분류기이다.

각각 입력 벡터에 가중치를 곱해준 뒤 바이어스를 더해주고 로지스틱 시그모이드 활성 함수를 통과시켜 값의 범위를 (0, 1)로 제한했다.

```
int nOutputCnt = 10;

double** OutputList = new double* [m_nBatch];
for (int i = 0; i < m_nBatch; ++i)
    OutputList[i] = new double[nOutputCnt];

for (int i = 0; i < m_nBatch; ++i)
{
    for (int j = 0; j < nOutputCnt; ++j)
    {
```

```
        OutputList[i][j] = 0;
        if (m_MnistTrainOutput[nIndex + i] == j) OutputList[i][j] = 1;
    }
}
```

다음으로 출력 노드가 10개가 된 모델의 형식에 맞게 m_MnistTrainOutput 데이터를 가공한다. 예를 들어 원래 출력값이 4였다면 이를 원핫 코딩하여 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]의 10차원 벡터로 변환해준다.

```
    bool predict1 = true;
    bool predict2 = true;
    bool predict3 = true;

    for (int j = 0; j < 10; j++) {
        double d = OutputList[i - nIndex][j];
        Loss1 += (d - y1[j]) * (d - y1[j]) * 0.5;
        Loss2 += (d - y2[j]) * (d - y2[j]) * 0.5;
        Loss3 += (d - y3[j]) * (d - y3[j]) * 0.5;

        if (y1[j] > 0.5 && d == 0) predict1 = false;
        if (y1[j] <= 0.5 && d == 1) predict1 = false;

        if (y2[j] > 0.5 && d == 0) predict2 = false;
        if (y2[j] <= 0.5 && d == 1) predict2 = false;

        if (y3[j] > 0.5 && d == 0) predict3 = false;
        if (y3[j] <= 0.5 && d == 1) predict3 = false;

        for (int x = 0; x < 28 * 28; ++x) {
            m_W[0][j][x] += lr * (d - y1[j]) * y1[j] * (1 - y1[j]) * m_MnistTrainInput[i][x];
        }
        m_B[0][j] += lr * (d - y1[j]) * y1[j] * (1 - y1[j]);

        for (int x = 0; x < 14 * 14; ++x) {
            m_W[1][j][4 * x] += lr * (d - y2[j]) * y2[j] * (1 - y2[j])
                * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2];
            m_W[1][j][4 * x + 1] += lr * (d - y2[j]) * y2[j] * (1 - y2[j])
                * m_MnistTrainInput[i][(x / 14 * 2 + 1) * 28 + x % 14 * 2 + 1];
            m_W[1][j][4 * x + 2] += lr * (d - y2[j]) * y2[j] * (1 - y2[j])
                * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2];
            m_W[1][j][4 * x + 3] += lr * (d - y2[j]) * y2[j] * (1 - y2[j])
                * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2];

            m_W[2][j][4 * x] += lr * (d - y3[j]) * y3[j] * (1 - y3[j])
                * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2];
            m_W[2][j][4 * x + 1] += lr * (d - y3[j]) * y3[j] * (1 - y3[j])
                * (m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                    + m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2 + 1]
                    + m_MnistTrainInput[i][(x / 14 * 2 + 1) * 28 + x % 14 * 2]
                    + m_MnistTrainInput[i][(x / 14 * 2 + 1) * 28 + x % 14 * 2 + 1]) / 4;
            m_W[2][j][4 * x + 2] += lr * (d - y3[j]) * y3[j] * (1 - y3[j])
                * (m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                    - m_MnistTrainInput[i][(x / 14 * 2 + 1) * 28 + x % 14 * 2 + 1]);
            m_W[2][j][4 * x + 3] += lr * (d - y3[j]) * y3[j] * (1 - y3[j])
                * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_MnistTrainInput[i][(x / 14 * 2) * 28 + x % 14 * 2];
        }
        m_B[1][j] += lr * (d - y2[j]) * y2[j] * (1 - y2[j]);
        m_B[2][j] += lr * (d - y3[j]) * y3[j] * (1 - y3[j]);
```

```
    }
    if (predict1) nTP1++;
    if (predict2) nTP2++;
    if (predict3) nTP3++;
}
Loss1 /= m_nBatch;
Loss2 /= m_nBatch;
Loss3 /= m_nBatch;
```

이후 각각의 출력 노드에 대해 예측값과 실제값을 비교한다. 먼저 이를 비교하여 Squared Error 방식으로 Loss를 구해주고, 10개의 모든 출력 노드에 대해 하나라도 다른 값이 있다면 True Positive가 아니라고 판단한다. 이후 가중치와 바이어스에 대해 Gradient Descent 방식으로 업데이트를 진행한다. 상술한 판단 방식으로 True Positive의 개수와 Loss를 구해준다.

```
int nTP1 = 0;
int nTP2 = 0;
int nTP3 = 0;
int i;

for (i = 0; i < m_nMnistTestTotal; i++)
{
    double* nResult1 = new double[10];
    double* nResult2 = new double[10];
    double* nResult3 = new double[10];
    double max1 = 0;
    double max2 = 0;
    double max3 = 0;
    int idx1 = -1;
    int idx2 = -1;
    int idx3 = -1;

    for (int j = 0; j < 10; j++) nResult1[j] = 0;

    for (int j = 0; j < 10; j++) {
        for (int x = 0; x < 28 * 28; x++) {
            nResult1[j] += m_MnistTestInput[i][x] * m_W[0][j][x];
        }
        nResult1[j] += m_B[0][j];
        nResult1[j] = 1. / (1 + exp(-nResult1[j]));

        for (int x = 0; x < 14 * 14; x++) {
            nResult2[j] += m_MnistTestInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_W[1][j][4 * x];
            nResult2[j] += m_MnistTestInput[i][(x / 14 * 2 + 1) * 28 + x % 14 * 2 + 1]
                * m_W[1][j][4 * x + 1];
            nResult2[j] += m_MnistTestInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_MnistTestInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_W[1][j][4 * x + 2];
            nResult2[j] += m_MnistTestInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_MnistTestInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_MnistTestInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_W[1][j][4 * x + 3];

            nResult3[j] += m_MnistTestInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_W[2][j][4 * x];
            nResult3[j] += (m_MnistTestInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                + m_MnistTestInput[i][(x / 14 * 2) * 28 + x % 14 * 2 + 1]
```

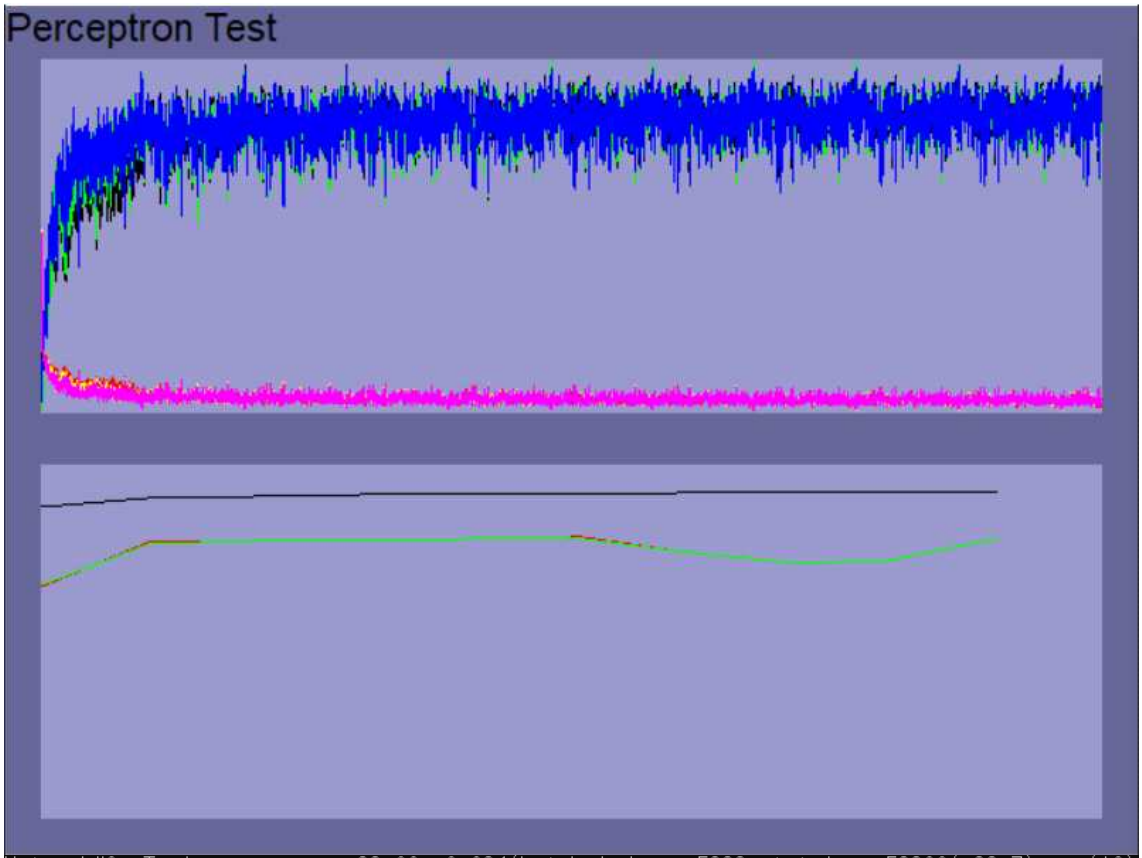
```
                + m_MnistTestInput[i][(x / 14 * 2 + 1) * 28 + x % 14 * 2]
                + m_MnistTestInput[i][(x / 14 * 2 + 1) * 28 + x % 14 * 2 + 1]) / 4
                * m_W[2][j][4 * x + 1];
            nResult3[j] += (m_MnistTestInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                - m_MnistTestInput[i][(x / 14 * 2 + 1) * 28 + x % 14 * 2 + 1])
                * m_W[2][j][4 * x + 2];
            nResult3[j] += m_MnistTestInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_MnistTestInput[i][(x / 14 * 2) * 28 + x % 14 * 2]
                * m_W[2][j][4 * x + 3];
        }
        nResult2[j] += m_B[1][j];
        nResult2[j] = 1. / (1 + exp(-nResult2[j]));

        nResult3[j] += m_B[2][j];
        nResult3[j] = 1. / (1 + exp(-nResult3[j]));

        if (nResult1[j] > max1) {
            max1 = nResult1[j];
            idx1 = j;
        }
        if (nResult2[j] > max2) {
            max2 = nResult2[j];
            idx2 = j;
        }
        if (nResult3[j] > max3) {
            max3 = nResult3[j];
            idx3 = j;
        }
    }

    if (idx1 == m_MnistTestOutput[i]) nTP1++;
    if (idx2 == m_MnistTestOutput[i]) nTP2++;
    if (idx3 == m_MnistTestOutput[i]) nTP3++;
}
```

마지막으로 실제 학습한 가중치와 바이어스로 모델의 실성능을 검증한다. Test Input을 Perceptron에 입력하여 nResult 값을 구한다. 이 값이 가장 큰 출력 노드의 인덱스를 구하면 (Argmax) 그 값이 이 퍼셉트론이 입력 이미지를 바탕으로 예측한 숫자이다. 만일 이 값이 Test Output과 같다면 True Positive라고 판단한다. True Positive로 이 모델의 Accuracy를 측정한다.




```
Network#0, Train accuracy:  96.00, 0.024(batch index:  5992, total :  59200( 98.7), ep(10)
Network#1, Train accuracy:  94.00, 0.028(batch index:  5992, total :  59200( 98.7), ep(10)
Network#2, Train accuracy:  95.00, 0.028(batch index:  5992, total :  59200( 98.7), ep(10)
Network#0, Train accuracy:  87.00, 0.055(batch index:  5993, total :  59300( 98.8), ep(10)
Network#1, Train accuracy:  86.00, 0.058(batch index:  5993, total :  59300( 98.8), ep(10)
Network#2, Train accuracy:  87.00, 0.060(batch index:  5993, total :  59300( 98.8), ep(10)
Network#0, Train accuracy:  76.00, 0.109(batch index:  5994, total :  59400( 99.0), ep(10)
Network#1, Train accuracy:  75.00, 0.111(batch index:  5994, total :  59400( 99.0), ep(10)
Network#2, Train accuracy:  76.00, 0.114(batch index:  5994, total :  59400( 99.0), ep(10)
Network#0, Train accuracy:  78.00, 0.093(batch index:  5995, total :  59500( 99.2), ep(10)
Network#1, Train accuracy:  79.00, 0.087(batch index:  5995, total :  59500( 99.2), ep(10)
Network#2, Train accuracy:  78.00, 0.092(batch index:  5995, total :  59500( 99.2), ep(10)
Network#0, Train accuracy:  95.00, 0.027(batch index:  5996, total :  59600( 99.3), ep(10)
Network#1, Train accuracy:  95.00, 0.025(batch index:  5996, total :  59600( 99.3), ep(10)
Network#2, Train accuracy:  95.00, 0.026(batch index:  5996, total :  59600( 99.3), ep(10)
Network#0, Train accuracy:  90.00, 0.044(batch index:  5997, total :  59700( 99.5), ep(10)
Network#1, Train accuracy:  93.00, 0.036(batch index:  5997, total :  59700( 99.5), ep(10)
Network#2, Train accuracy:  92.00, 0.041(batch index:  5997, total :  59700( 99.5), ep(10)
Network#0, Train accuracy:  78.00, 0.109(batch index:  5998, total :  59800( 99.7), ep(10)
Network#1, Train accuracy:  79.00, 0.105(batch index:  5998, total :  59800( 99.7), ep(10)
Network#2, Train accuracy:  80.00, 0.109(batch index:  5998, total :  59800( 99.7), ep(10)
Network#0, Train accuracy:  98.00, 0.012(batch index:  5999, total :  59900( 99.8), ep(10)
Network#1, Train accuracy:  98.00, 0.011(batch index:  5999, total :  59900( 99.8), ep(10)
Network#2, Train accuracy:  98.00, 0.013(batch index:  5999, total :  59900( 99.8), ep(10)
Network#0, Train accuracy:  86.00, 0.057(batch index:  6000, total :  60000(100.0), ep(10)
Network#1, Train accuracy:  88.00, 0.059(batch index:  6000, total :  60000(100.0), ep(10)
Network#2, Train accuracy:  88.00, 0.062(batch index:  6000, total :  60000(100.0), ep(10)
Network#0, Test accuracy:  91.720
Network#1, Test accuracy:  78.580
Network#2, Test accuracy:  78.520
```

다음은 이 퍼셉트론으로 대략 10 Cycle 정도 학습을 진행해본 결과이다. 검정색이 1번 분류기, 빨간색이 2번 분류기, 초록색이 3번 분류기이다. 모델 학습 단계에서의 Accuracy는 대략적으로 전부 비슷한 값을 가졌지만, 모델 검증 단계에서의 Accuracy는 1번 분류기의 값이

압도적으로 높았다. 이는 2번 분류기와 3번 분류기에서는 모든 화소의 값을 이용하여 예측을 진행하지 않아 데이터 중 일부만을 이용하여 예측을 했기 때문에 정보량의 차이로 발생한 간극이라고 예상된다. 또한 2번 분류기와 3번 분류기의 성능은 1번 분류기의 입력값보다 절반 정도의 정보를 입력 받아 둘은 비슷한 성능을 내는 것으로 예상된다.