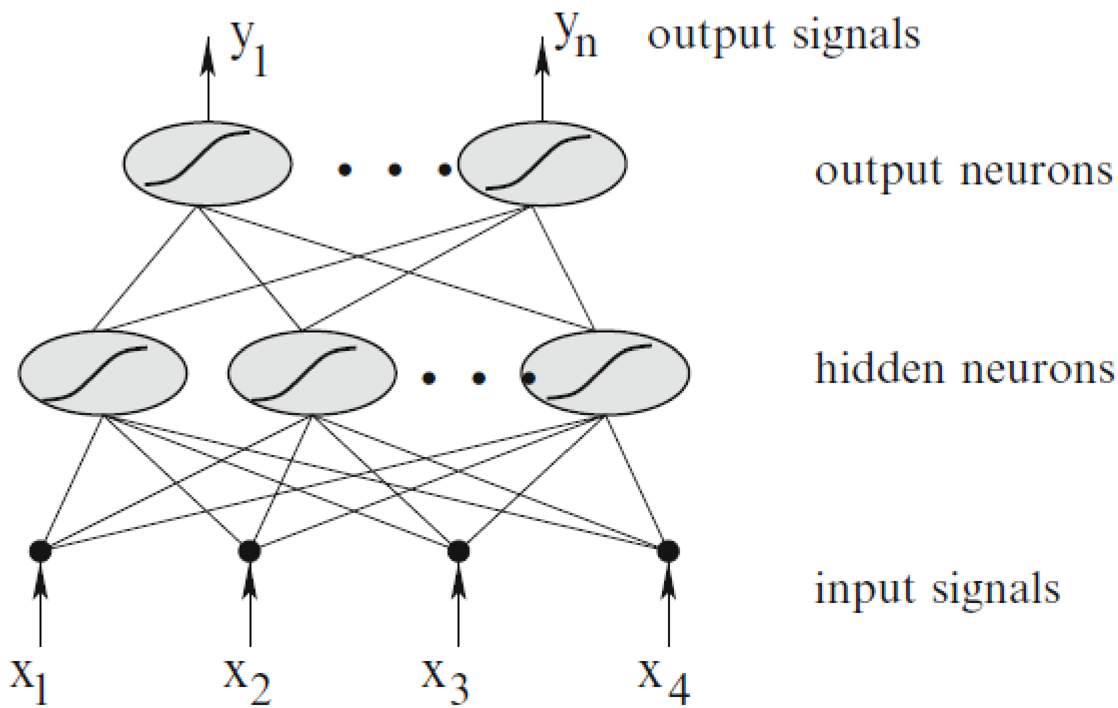
Programming Study
Artificial Neural Network

Sungwoo Nam

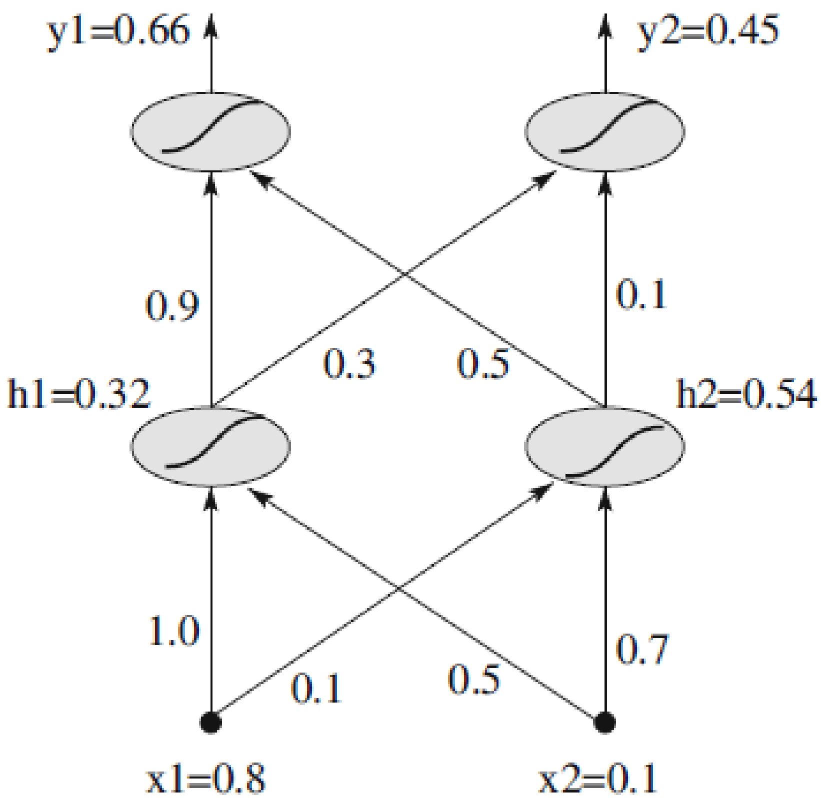
2018.1.9

Artificial Neural Network



An Introduction to Machine Learning – Miroslav Kubat, Springer

Forward Propagation



$$y_i = f(\sum_j w_{ji}^{(1)} f(\sum_k w_{kj}^{(2)} x_k))$$

$$f(\Sigma) = \frac{1}{1 + e^{-\Sigma}}$$

Forward Propagation In Code

```
Mat ForwardPropagation(const Mat& X, const Mat& HN, const Mat& ON, Mat& H)
{
    auto sigmoid = [](double &v, const int* pos) {
        v = 1.0 / (1.0 + exp(-v));
    };

    H = HN * X;
    H.forEach<double>(sigmoid);
    Mat O = ON * H;
    O.forEach<double>(sigmoid);

    return O;
}
```

$$f(\Sigma) = \frac{1}{1 + e^{-\Sigma}}$$

$$\sum_k w_{kj}^{(2)} x_k$$

Backward Propagation

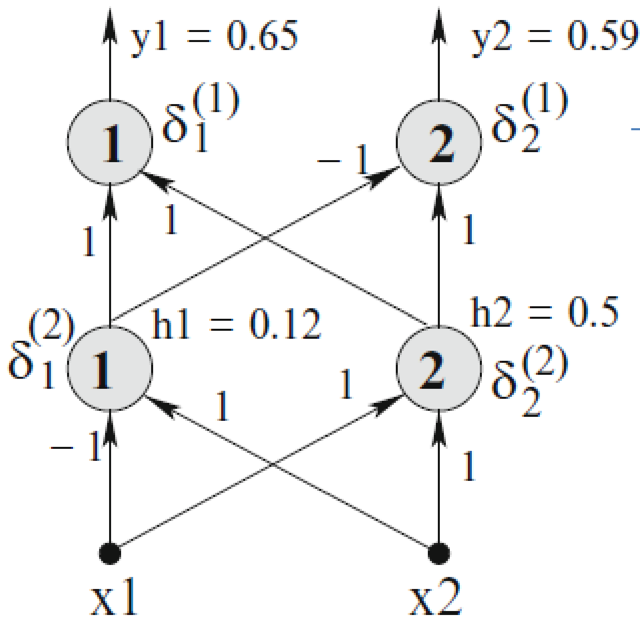
Table 5.2 *Backpropagation of error* in a neural network with one hidden layer

1. Present example $\mathbf{x}$ to the input layer and propagate it through the network.
2. Let $\mathbf{y} = (y_1, \ldots y_m)$ be the output vector, and let $\mathbf{t}(\mathbf{x}) = (t_1, \ldots t_m)$ be the target vector.
3. For each output neuron, calculate its responsibility, $\delta_i^{(1)}$, for the network's error:
   $\delta_i^{(1)} = y_i(1 - y_i)(t_i - y_i)$
4. For each hidden neuron, calculate its responsibility, $\delta_j^{(2)}$, for the network's error. While doing so, use the responsibilities, $\delta_i^{(1)}$, of the output neurons as obtained in the previous step.
   $\delta_j^{(2)} = h_j(1 - h_j) \sum_i \delta_i^{(1)} w_{ji}$
5. Update the weights using the following formulas, where η is the learning rate:

   output layer: $w_{ji}^{(1)} := w_{ji}^{(1)} + \eta\delta_i^{(1)} h_j$; h_j: the output of the j-th hidden neuron
   hidden layer: $w_{kj}^{(2)} := w_{kj}^{(2)} + \eta\delta_j^{(2)} x_k$; x_k: the value of the k-th attribute

6. Unless a termination criterion has been satisfied, return to step 1.

Backward Propagation



$$\delta_i^{(1)} = y_i(1 - y_i)(t_i - y_i)$$

$$\delta_j^{(2)} = h_j(1 - h_j) \sum_i \delta_i^{(1)} w_{ji}$$

$$w_{ji}^{(1)} := w_{ji}^{(1)} + \eta \delta_i^{(1)} h_j;$$

$$w_{kj}^{(2)} := w_{kj}^{(2)} + \eta \delta_j^{(2)} x_k;$$

Backward Propagation In Code

```
void BackwardPropagation(
    const Mat& X, Mat& HN, Mat& ON, const Mat& H,
    const Mat& Y, const Mat& T, double nu )
{
    Mat D1 = Y.clone();
    D1.forEach<double>([=](double &yi, const int* i) {
        double ti = T.at<double>(i);
        yi = yi * (1 - yi) * (ti - yi);
    });

    Mat P = ON.t() * D1;
    Mat D2 = H.clone();
    D2.forEach<double>([=](double &hi, const int*i) {
        double pi = P.at<double>(i);
        hi = hi * (1 - hi) * pi;
    });

    ON += nu * D1 * H.t();
    HN += nu * D2 * X.t();
}
```

$$\delta_i^{(1)} = y_i(1 - y_i)(t_i - y_i)$$

$$\delta_j^{(2)} = h_j(1 - h_j)\sum_i \delta_i^{(1)} w_{ji}$$

$$w_{ji}^{(1)} := w_{ji}^{(1)} + \eta\delta_i^{(1)} h_j;$$

$$w_{kj}^{(2)} := w_{kj}^{(2)} + \eta\delta_j^{(2)} x_k;$$

Training Data

- http://archive.ics.uci.edu/ml/datasets/steel+plates+faults

notes.txt | kNN.py | SteelPlateFaults.txt

```
42 50 270900 270944 267 17 44 24220 76 108 1687 1 0 80 0.0498 0.2415 0.1818 0.0047 0.4706 1 1 2.4265 0.9031 1.6435 0.8182 -0.2913 0.5822 1 0 0 0 0 0 0
645 651 2538079 2538108 108 10 30 11397 84 123 1687 1 0 80 0.7647 0.3793 0.2069 0.0036 0.6 0.9667 1 2.0334 0.7782 1.4624 0.7931 -0.1756 0.2984 1 0 0 0 0 0 0
829 835 1553913 1553931 71 8 19 7972 99 125 1623 1 0 100 0.971 0.3426 0.3333 0.0037 0.75 0.9474 1 1.8513 0.7782 1.2553 0.6667 -0.1228 0.215 1 0 0 0 0 0 0
853 860 369370 369415 176 13 45 18996 99 126 1353 0 1 290 0.7287 0.4413 0.1556 0.0052 0.5385 1 1 2.2455 0.8451 1.6532 0.8444 -0.1568 0.5212 1 0 0 0 0 0 0
1289 1306 498078 498335 2409 60 260 246930 37 126 1353 0 1 185 0.0695 0.4486 0.0662 0.0126 0.2833 0.9885 1 3.3818 1.2305 2.4099 0.9338 -0.1992 1 1 0 0 0 0 0 0 0
430 441 100250 100337 630 20 87 62357 64 127 1387 0 1 40 0.62 0.3417 0.1264 0.0079 0.55 1 1 2.7993 1.0414 1.9395 0.8736 -0.2267 0.9874 1 0 0 0 0 0 0
413 446 138468 138883 9052 230 432 1481991 23 199 1687 0 1 150 0.4896 0.339 0.0795 0.0196 0.1435 0.9607 1 3.9567 1.5185 2.6181 0.9205 0.2791 1 1 0 0 0 0 0 0
190 200 210936 210956 132 11 20 20007 124 172 1687 0 1 150 0.2253 0.34 0.5 0.0059 0.9091 1 1 2.1206 1 1.301 0.5 0.1841 0.3359 1 0 0 0 0 0 0
330 343 429227 429253 264 15 26 29748 53 148 1687 0 1 150 0.3912 0.2189 0.5 0.0077 0.8667 1 1 2.4216 1.1139 1.415 0.5 -0.1197 0.5593 1 0 0 0 0 0 0
74 90 779144 779308 1506 46 167 180215 53 143 1687 0 1 150 0.0877 0.4261 0.0976 0.0095 0.3478 0.982 1 3.1778 1.2041 2.2148 0.9024 -0.0651 1 1 0 0 0 0 0 0
106 118 813452 813500 442 13 48 50393 76 143 1687 0 1 150 0.1257 0.2326 0.25 0.0071 0.9231 1 1 2.6454 1.0792 1.6812 0.75 -0.1093 0.8612 1 0 0 0 0 0 0
505 515 106604 106668 284 42 69 31062 97 119 1687 0 1 150 0.5987 0.5562 0.1563 0.0059 0.2381 0.9275 1 2.4533 1 1.8062 0.8438 -0.1455 0.9048 1 0 0 0 0 0 0
46 58 179258 179312 480 15 54 61966 102 158 1687 0 1 150 0.0545 0.2593 0.2222 0.0071 0.8 1 1 2.6812 1.0792 1.7324 0.7778 0.0086 0.9093 1 0 0 0 0 0 0
581 590 230644 230704 433 22 60 38917 62 111 1687 0 1 150 0.6888 0.1981 0.15 0.0053 0.4091 1 1 2.6365 0.9542 1.7781 0.85 -0.2978 0.8299 1 0 0 0 0 0 0
451 466 368143 368208 728 30 68 69258 36 133 1687 0 1 150 0.5347 0.2533 0.2308 0.0089 0.5 0.9559 1 2.8621 1.1761 1.8129 0.7692 -0.2568 0.9888 1 0 0 0 0 0 0
669 684 491552 491684 1097 59 133 119540 50 134 1687 0 1 150 0.7931 0.446 0.1136 0.0089 0.2542 0.9925 1 3.0402 1.1761 2.1206 0.8864 -0.1487 1 1 0 0 0 0 0 0
156 192 713788 714056 5044 167 282 570911 11 143 1687 0 1 150 0.1849 0.4772 0.1343 0.0213 0.2156 0.9503 1 3.7028 1.5563 2.4281 0.8657 -0.1157 1 1 0 0 0 0 0 0
90 104 751059 751132 552 38 76 59750 79 134 1687 0 1 150 0.1067 0.4599 0.1918 0.0083 0.3684 0.9605 1 2.7419 1.1461 1.8633 0.8082 -0.1543 0.9918 1 0 0 0 0 0 0
82 89 844704 844729 137 8 25 14907 92 126 1687 0 1 150 0.0972 0.2171 0.28 0.0041 0.875 1 1 2.1367 0.8451 1.3979 0.72 -0.1499 0.2998 1 0 0 0 0 0 0
1601 1613 21349 21376 209 15 27 24807 96 141 1687 0 1 200 0.0877 0.3549 0.4444 0.0071 0.8 1 1 2.3201 1.0792 1.4314 0.5556 -0.0727 0.5362 1 0 0 0 0 0 0
17 28 42683 42717 284 18 34 32604 87 141 1687 0 1 200 0.0202 0.2406 0.3235 0.0065 0.6111 1 1 2.4533 1.0414 1.5315 0.6765 -0.1031 0.6173 1 0 0 0 0 0 0
43 52 86284 86313 153 13 29 17753 101 134 1687 0 1 200 0.051 0.4138 0.3103 0.0053 0.6923 1 1 2.1847 0.9542 1.4624 0.6897 -0.0935 0.4317 1 0 0 0 0 0 0
63 72 115485 115504 106 10 19 13829 107 150 1687 0 1 200 0.0747 0.3801 0.4737 0.0053 0.9 1 1 2.0253 0.9542 1.2787 0.5263 0.0192 0.2942 1 0 0 0 0 0 0
82 92 149044 149083 264 15 39 32175 92 141 1687 0 1 200 0.0972 0.3231 0.2564 0.0059 0.6667 1 1 2.4216 1 1.5911 0.7436 -0.0479 0.6422 1 0 0 0 0 0 0
75 84 184350 184383 201 17 35 27349 123 148 1687 0 1 200 0.0889 0.3232 0.2727 0.0053 0.5294 0.9429 1 2.3032 0.9542 1.5185 0.7273 0.063 0.4913 1 0 0 0 0 0 0
1363 1372 2128884 2128907 171 9 23 16645 77 111 1373 0 1 40 0.0015 0.1739 0.3913 0.0066 1 1 1 2.233 0.9542 1.3617 0.6087 -0.2395 0.3464 1 0 0 0 0 0 0
1358 1372 2356396 2356435 395 19 39 38722 75 118 1373 0 1 40 0.0015 0.2766 0.359 0.0102 0.7368 1 1 2.5966 1.1461 1.5911 0.641 -0.2341 0.8355 1 0 0 0 0 0 0
1404 1410 5202306 5202324 85 9 18 8948 97 119 1627 0 1 40 0.2668 0.213 0.3333 0.0037 0.6667 1 1 1.9294 0.7782 1.2553 0.6667 -0.1776 0.215 1 0 0 0 0 0 0
```

Training

```
void testSteelPlateDefects0()
{
    Mat D = ReadMat("SteelPlateFaults.txt", " \t");
    D = D.rowRange(0, 340);

    Mat X = D.colRange(4, 6).t();
    Normalize<double>(X);
    Mat T = D.colRange(27, 29).t();

    int M = T.rows, N = X.rows;
    Mat HN(N, N, CV_64FC1, Scalar(0.01));
    Mat ON(M, N, CV_64FC1, Scalar(0.01));

    Mat H, Y;
    for (int i = 0; i < 1000; ++i)
    {
        Y = ForwardPropagation(X, HN, ON, H );
        BackwardPropagation(X, HN, ON, H, Y, T, 0.1);

        printf("iteration %d, Error %.3f\n", i, AverageDistance(T-Y));
    }

    printf( "Trained Error %.3f\n", AverageDistance(T - Y));
}
```