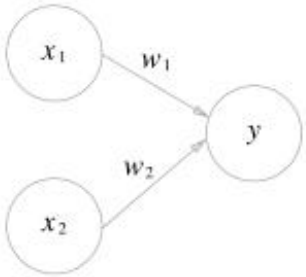
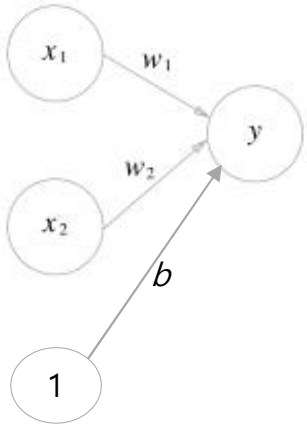
Multi-Layer Perceptron

Perceptron

- Perceptron (F.Rosenblatt, 1975)
  - 신경망의 기원





$$y = \begin{cases} 0\ (w_1x_1 + w_2x_2 \leq \theta) \\ 1\ (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

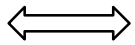
$$y = \begin{cases} 0\ (b + w_1x_1 + w_2x_2 \leq 0) \\ 1\ (b + w_1x_1 + w_2x_2 > 0) \end{cases}$$

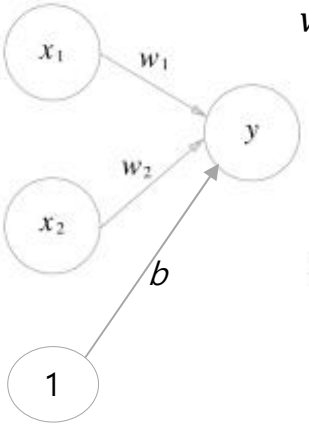
x_1, x_2 : input, y : output
w_1, w_2 : weight, b : bias,

Perceptron

- (Example) OR gate

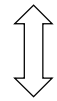
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

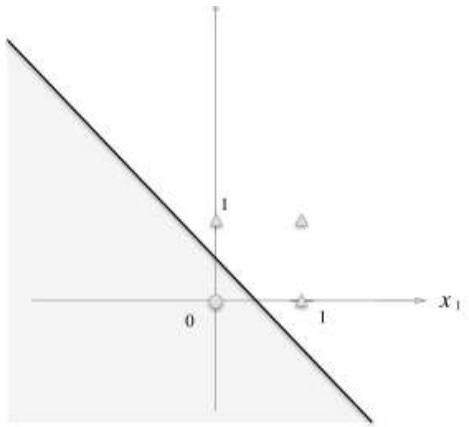




$w_1 = 1.0, w_2 = 1.0, b = -0.5$

$$y = \begin{cases} 0 \ (-0.5 + x_1 + x_2 \leq 0) \\ 1 \ (-0.5 + x_1 + x_2 > 0) \end{cases}$$

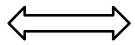


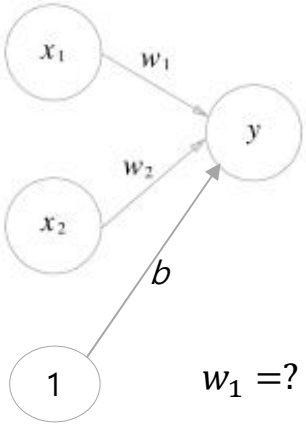


Perceptron

- (Example) XOR gate – single layer

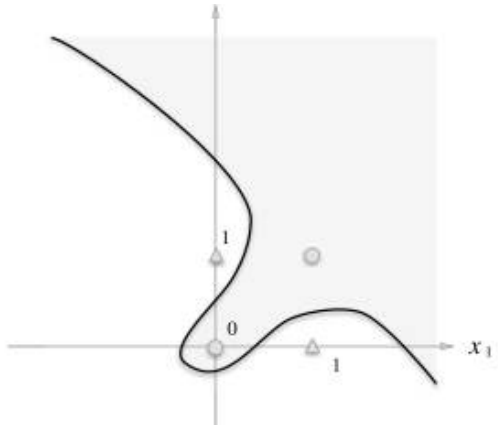
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0





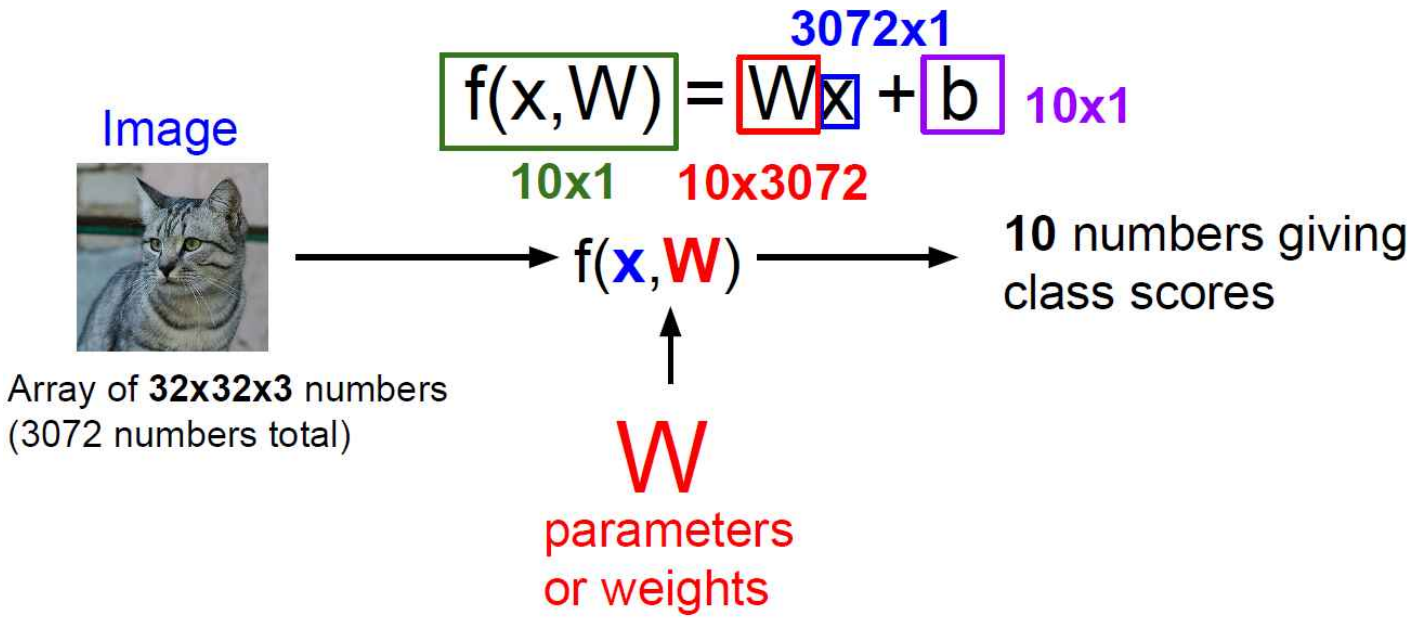
$w_1 = ??, w_2 = ??, b = ??$

- Single-layer perceptron 으로 구현 불가
  - Linear classification 만 가능



Linear Classifier

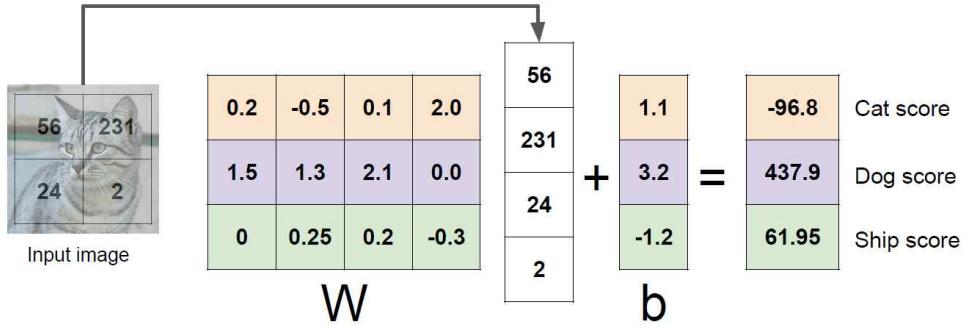
- Single layer perceptron



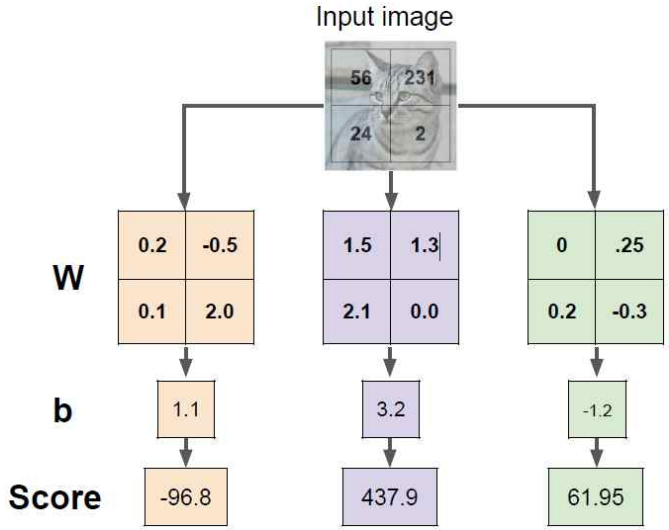
Linear Classifier

- Image with 4 pixels & 3 classes (Cat / Dog / Ship)

Algebraic viewpoint

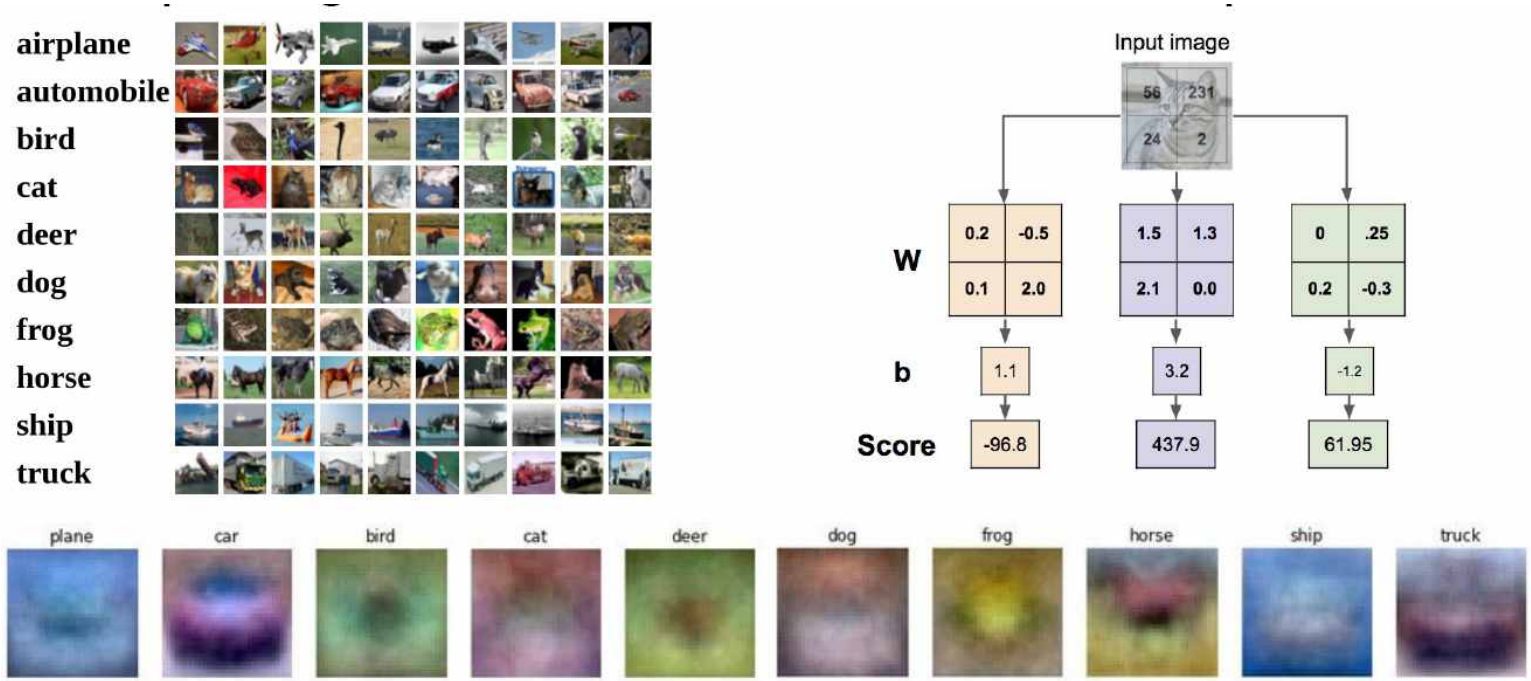


Visual viewpoint



Linear Classifier

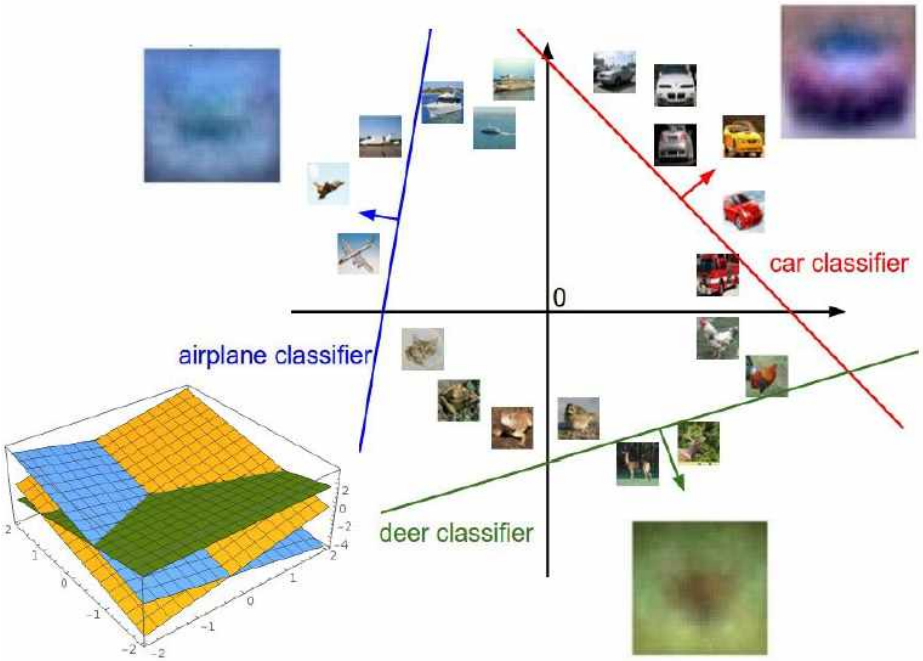
- Visual Viewpoint



One template per class

Linear Classifier

- Geometric Viewpoint



$$f(x,W) = Wx + b$$

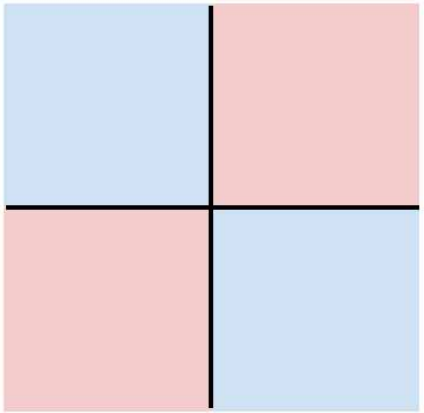
Array of **32x32x3** numbers
(3072 numbers total)

Linear Classifier

- Hard cases for a linear classifier

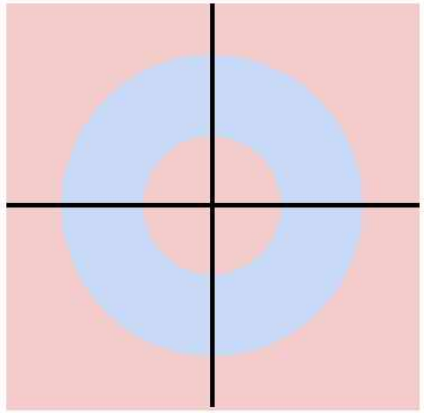
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



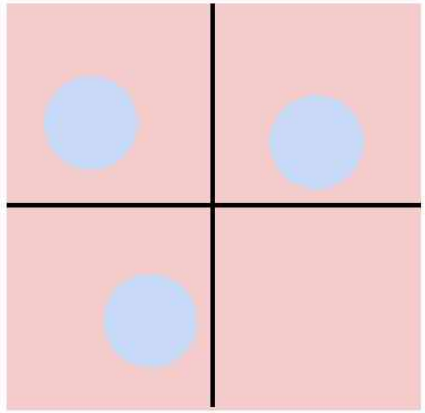
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

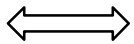
Class 2:
Everything else

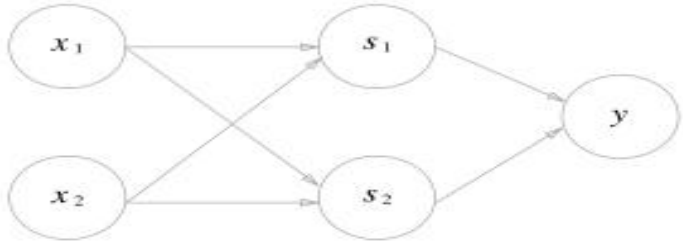


MLP

- (Example) XOR gate – 2 layers

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



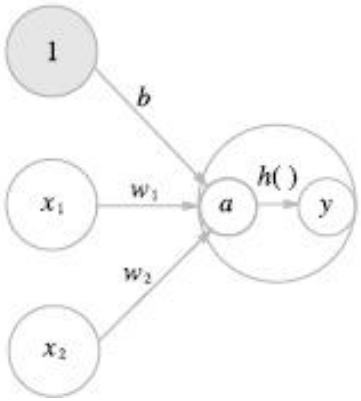



```
def XOR(x1, x2):
    s1 = NAND(x1, x2)
    s2 = OR(x1, x2)
    y = AND(s1, s2)
    return y
```

- multiple-layer perceptron 으로 nonlinear classification 구현 가능 !

MLP

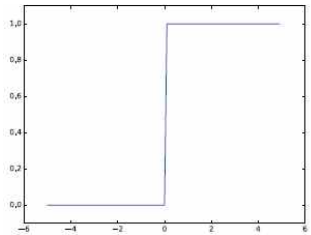
- Activation function (활성화 함수): $h(x)$



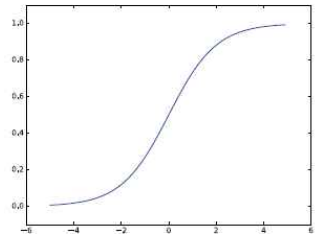
$$a = b + w_1 x_1 + w_2 x_2$$

$$y = h(a)$$

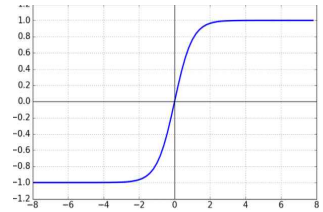
step $\quad h(x) = \begin{cases} 1, & x > 0 \\ 0, & x \le 0 \end{cases}$



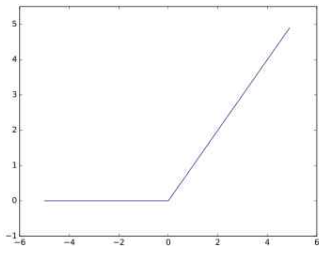
sigmoid $\quad h(x) = \dfrac{1}{1 + \exp(-x)}$



tanh $\quad h(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$

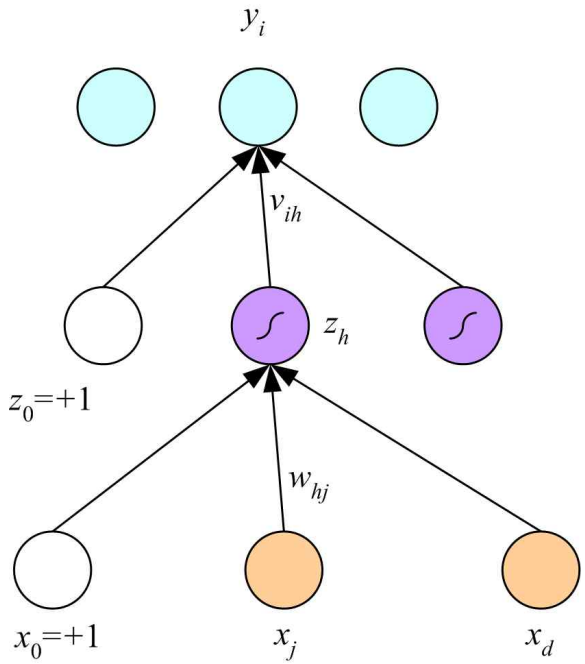


relu $\quad h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \le 0) \end{cases}$



MLP

- Multilayer Perceptron (Rumelhart, 1986)
  - Input layer (layer 0)
  - Hidden layers (layer 1, 2, ….)
  - Output layer

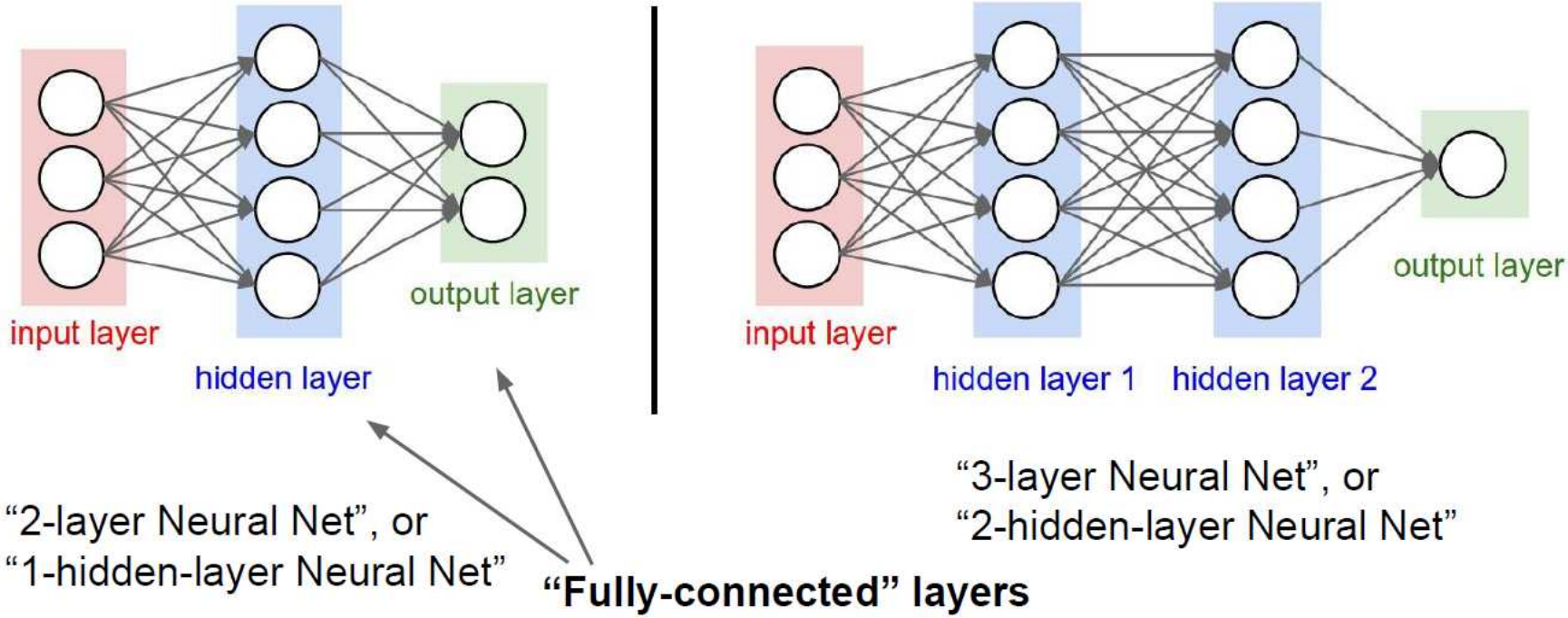


$$y_i = \boldsymbol{v}_i^T \boldsymbol{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

$$z_h = \text{sigmoid}\left(\boldsymbol{w}_h^T \boldsymbol{x}\right) = \frac{1}{1+\exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

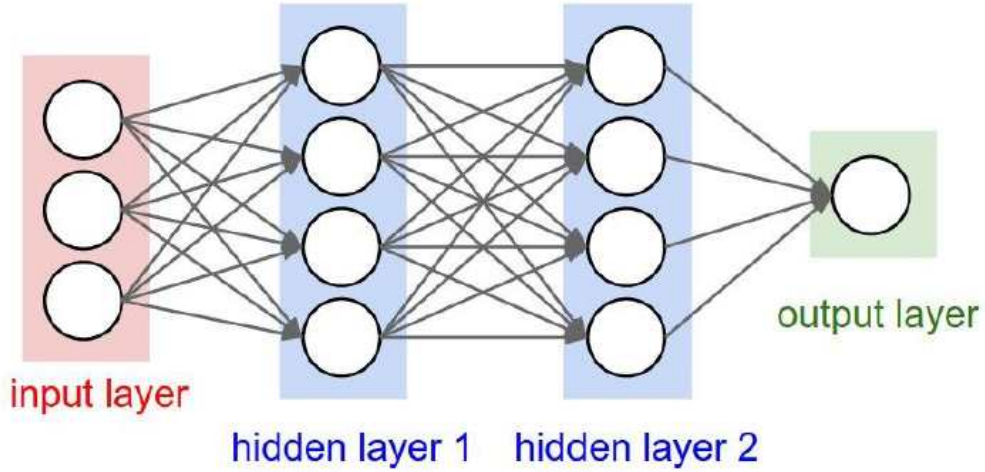
MLP

- Architecture



MLP

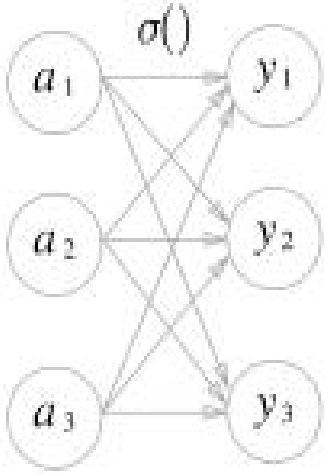
- Feedforward computation




```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

MLP

- Softmax function
  - Classification network 의 출력



$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

MLP

- Loss Function
  1) Mean square error (MSE)

$$E = \frac{1}{2}\sum_k (y_k - t_k)^2$$

y_k: 신경망 출력, t_k: 정답 레이블

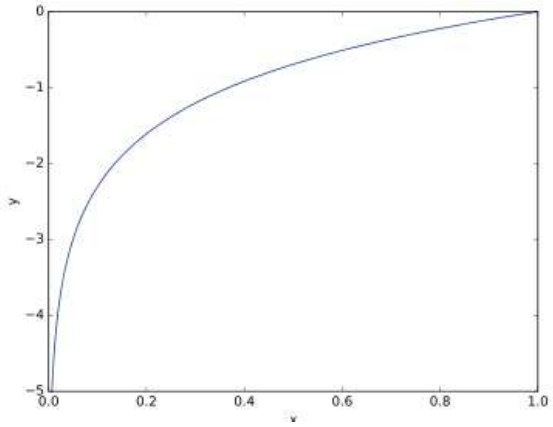
  2) Cross entropy error (CEE)

$$E = -\sum_k t_k \log y_k$$

=> Classification 문제에 많이 활용

* Mini-batch 학습

$$E = -\frac{1}{N}\sum_n \sum_k t_{nk} \log y_{nk}$$

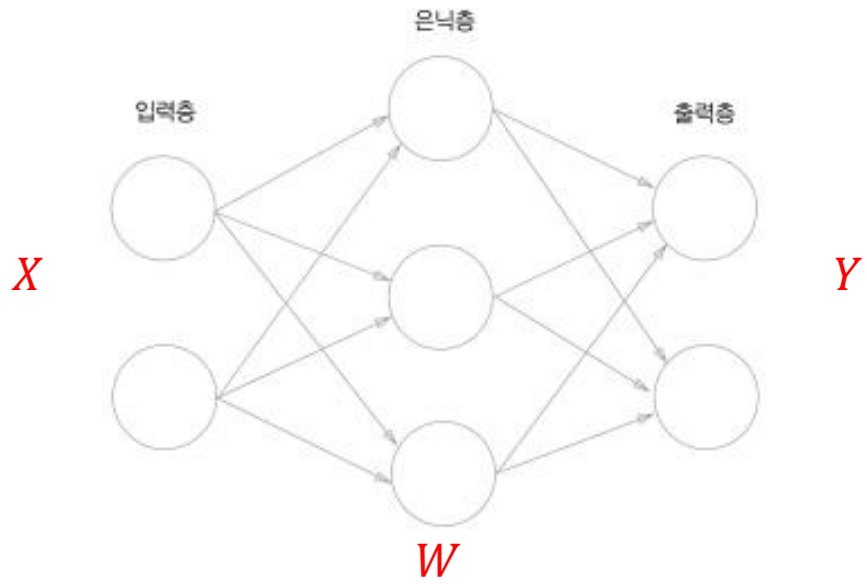


MLP

- Learning algorithm
  - Loss function 을 최소화 하는 W 를 구하는 문제
  - Optimization 문제

$$W^* = \arg\min \frac{1}{2} \sum_k (y_k - t_k)^2$$

$$\text{s.t} \qquad y_k = f(X, W), \qquad k = 1, \ldots, N$$



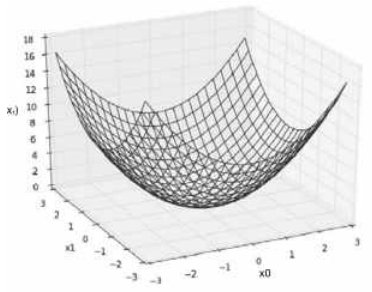
Training MLP

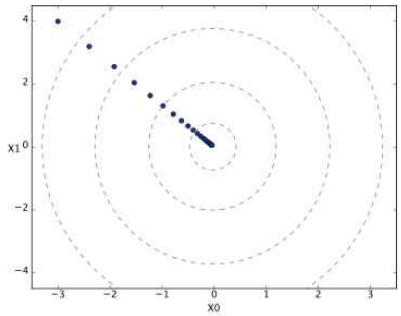
- Approach
  - Non-convex problem
  - Local search technique
- Gradient descent method
  - Hill-climbing 방법
    ① 초기 값을 임의로 설정하고
    ② 현재 값 주변을 탐색하여, 주변에서 가장 좋은 값을 선택. 개선이 없으면 종료 => 기울기 (gradient) 값 사용
    ③ 선택된 값을 현재 값으로 하여 더 이상 개선이 없을 때 반복
  - 문제
    - 초기 값에 의존 => global 최적을 구하기 어려움
  - 예

$$f(x_o, x_1) = x_0^2 + x_1^2$$

$$x_o = x_o - \eta \frac{\partial f}{\partial x_o}$$

$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$





Training MLP

- Learning a Perceptron
  - Error function

$$E = \frac{1}{2}(t - y)^2 = \frac{1}{2}(t - W^T X)^2$$

  - $W^* = \arg\min E(W, X)$
  - Update rule

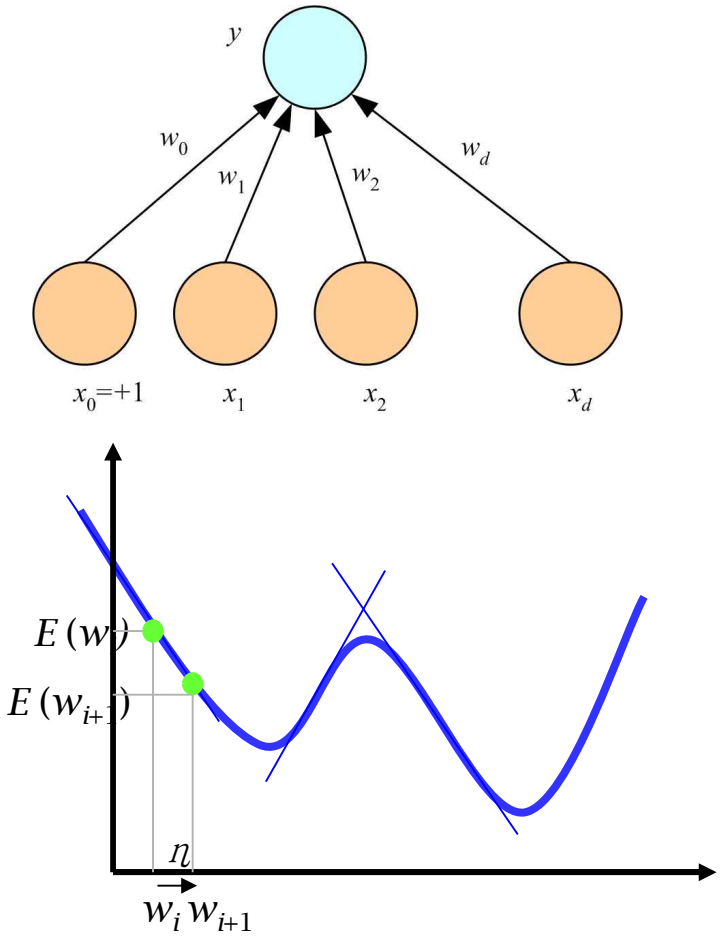
$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \forall i$$

$$w_{i+1} = w_i + \Delta w_i$$

η: learning factor

$$\Delta w_i = \eta(t - y)x_i, \qquad i = 0, \ldots, d$$

Update = LearningFactor · (DesiredOutput - ActualOutput) · Input



Gradient descent

Training MLP

(Example)

학습하는 과정에 있어서 현 단계의 연결 강도가 [0.2 0.5 0.3]인 경우, 목표치가 1인 학습 패턴 X = [0 0 1]을 신경망에 입력한 결과, −1이 출력되었다. 이때의 연결 강도 변화량 ⊿W와 다음 학습 단계에서의 연결 강도 W를 구하라. 단, 학습률은 1이고, 학습 신호는 목표치와 출력의 차이이다.

$$\Delta W = \alpha\gamma X = 1 \cdot \big(1 - (-1)\big) \cdot [0 \quad 0 \quad 1] = [0 \quad 0 \quad 2]$$

$$W_{k+1} = W_k + \Delta W = [0.2 \quad 0.5 \quad 0.3] + [0 \quad 0 \quad 2] = [0.2 \quad 0.5 \quad 2.3]$$

Training MLP

- Learning MLP

$$y_k = \sum_{j=0}^{H} v_j z_j$$

Forward

$$z_j = sigmoid \quad (\boldsymbol{w}_j^T \boldsymbol{x})$$

$$\boldsymbol{x}$$

$$E = \frac{1}{2}(t-y)^2$$

error

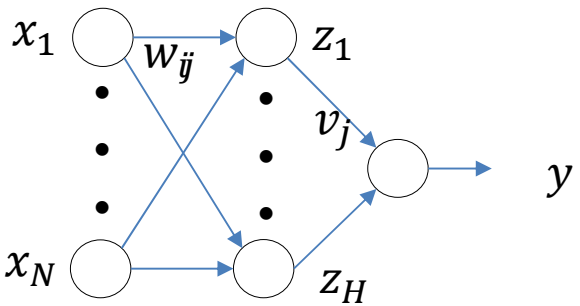
$$\Delta v_j = \eta(t-y)z_j$$

Backward

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

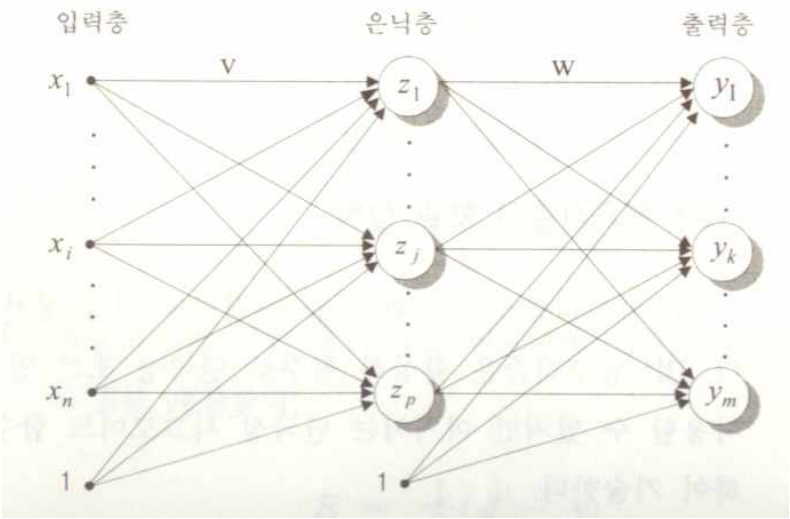
$$= -\eta \frac{\partial E}{\partial y}\frac{\partial y}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}}$$

$$= \eta\,(t-y) v_j\, z_j\,(1-z_j)\, x_i$$



Training MLP

- Error Back Propagation Algorithm
  - 오류 역전파 알고리즘
  - D.Parker, 1982



Step 1 : Initialize weights and counter

$V, W \leftarrow$ *small random value*

$p \leftarrow$ *number of training pattern pairs*

$k \leftarrow 1$

$E \leftarrow 0$

Step 2 : Set learning rate α *and* E_{max}

Step 3 : For each training pattern pair

do Step 4-10 until $k = p$

Step 4 : Compute output of hidden layer

$$NET_z = XV^T$$

$$Z = \frac{1}{1 + e^{-NET_z}}$$

Step 5 : Compute output

$$NET_y = ZW^T$$

$$y = \frac{1}{1 + e^{-NET_y}}$$

Step 6 : Compute output error

$$E \leftarrow \frac{1}{2}(d-y)^2 + E$$

Step 7 : Compute error signal of output layer

$$\delta_y = (d-y)\,y(1-y)$$

Step 8 : Compute error signal of hidden layer

$$\delta_z = z(1-z)\sum_{i=1}^{m}\delta_y W$$

Step 9 : Update weights

$$W = W + \alpha\delta_y Z$$

$$V = V + \alpha\delta_z X$$

Step 10 : Increase counter and goto Step 3

$$k \leftarrow k+1$$

Step 11 : Test stop condition

If $E < E_{max}$ *stop*

else, $E \leftarrow 0$ *and goto Step 3*

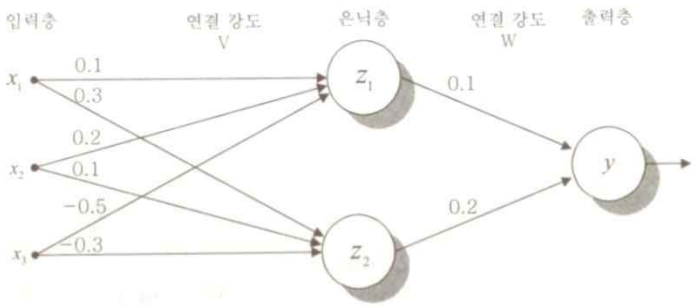
Training MLP

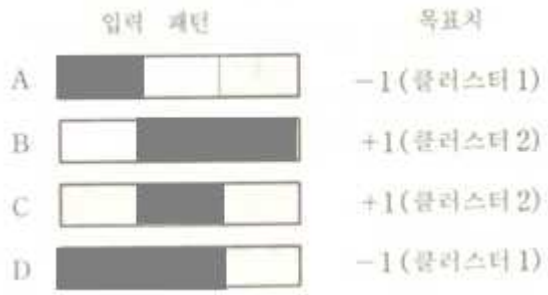
(Example)

그림과 같은 3계층 신경망을 BP 알고리즘으로 학습시켜 4개의 패턴을 2개의 클러스터로 분류하기 위해 초기 연결 강도 V와 W를 아래와 같이 설정하였다. 목표치가 −1인 학습 패턴 A를 입력할 경우의 오차는 얼마인가? 단, 단극성 시그모이드 함수를 활성화 함수로 사용한다.

$$V = \begin{bmatrix} 0.1 & 0.2 & -0.5 \\ 0.3 & 0.1 & -0.3 \end{bmatrix}$$

$$W = [0.1 \quad 0.2]$$





$$NET_z = XV^T = [1 \quad 0 \quad 0]\begin{bmatrix} 0.1 & 0.3 \\ 0.2 & 0.1 \\ -0.5 & -0.3 \end{bmatrix} = [0.1 \quad 0.3]$$

$$Z = \frac{1}{1+e^{-NET_z}} = [0.525 \quad 0.574]$$

$$NET_y = ZW^T = [0.525 \quad 0.574]\begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} = 0.167$$

$$y = \frac{1}{1+e^{-NET_y}} = 0.542$$

$$E = \frac{1}{2}(d-y)^2 = 0.5 \cdot (-1-0.542)^2 = 1.189$$

Training MLP

(Example)

예제 13.1과 동일한 조건에서 학습 패턴 A를 입력하여 신경망을 학습시키는 경우, 은닉층과 출력층간의 연결 강도 변화량 ΔW를 구하라. 단, 학습률 α는 1이다.

$$\delta_y = (d - y)y(1 - y) = (-1 - 0.542)0.542(1 - 0.542) = -0.383$$

$$\Delta W = \alpha\, \delta_y\, Z = 1 \cdot (-0.383) \cdot [0.525 \quad 0.574] = [-0.201 \quad -0.220]$$

예제 13.1과 동일한 조건에서 학습 패턴 A를 입력하여 신경망을 학습시키는 경우, 입력층과 은닉층간의 연결 강도 변화량 ΔV를 구하라. 단, 학습률 α는 1이다.

$$\delta_{z,1} = z_1(1 - y)\, \delta_y\, w_1 = 0.525 \cdot (1 - 0.542) \cdot (-0.383) \cdot 0.1 = -0.009$$
$$\delta_{z,2} = z_2(1 - y)\, \delta_y\, w_2 = 0.574 \cdot (1 - 0.542) \cdot (-0.383) \cdot 0.2 = -0.020$$

$$\Delta V = \alpha\, \delta_z\, X = 1 \cdot \begin{bmatrix} -0.009 \\ -0.020 \end{bmatrix} [1 \quad 0 \quad 0] = \begin{bmatrix} -0.009 & 0 & 0 \\ -0.020 & 0 & 0 \end{bmatrix}$$

Training MLP

- Training a 2-layer network => only needs 20 lines

```
1  import numpy as np
2  from numpy.random import randn
3
4  N, D_in, H, D_out = 64, 1000, 100, 10
5  x, y = randn(N, D_in), randn(N, D_out)
6  w1, w2 = randn(D_in, H), randn(H, D_out)
7
8  for t in range(2000):
9    h = 1 / (1 + np.exp(-x.dot(w1)))
10   y_pred = h.dot(w2)
11   loss = np.square(y_pred - y).sum()
12   print(t, loss)
13
14   grad_y_pred = 2.0 * (y_pred - y)
15   grad_w2 = h.T.dot(grad_y_pred)
16   grad_h = grad_y_pred.dot(w2.T)
17   grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19   w1 -= 1e-4 * grad_w1
20   w2 -= 1e-4 * grad_w2
```

Define the network

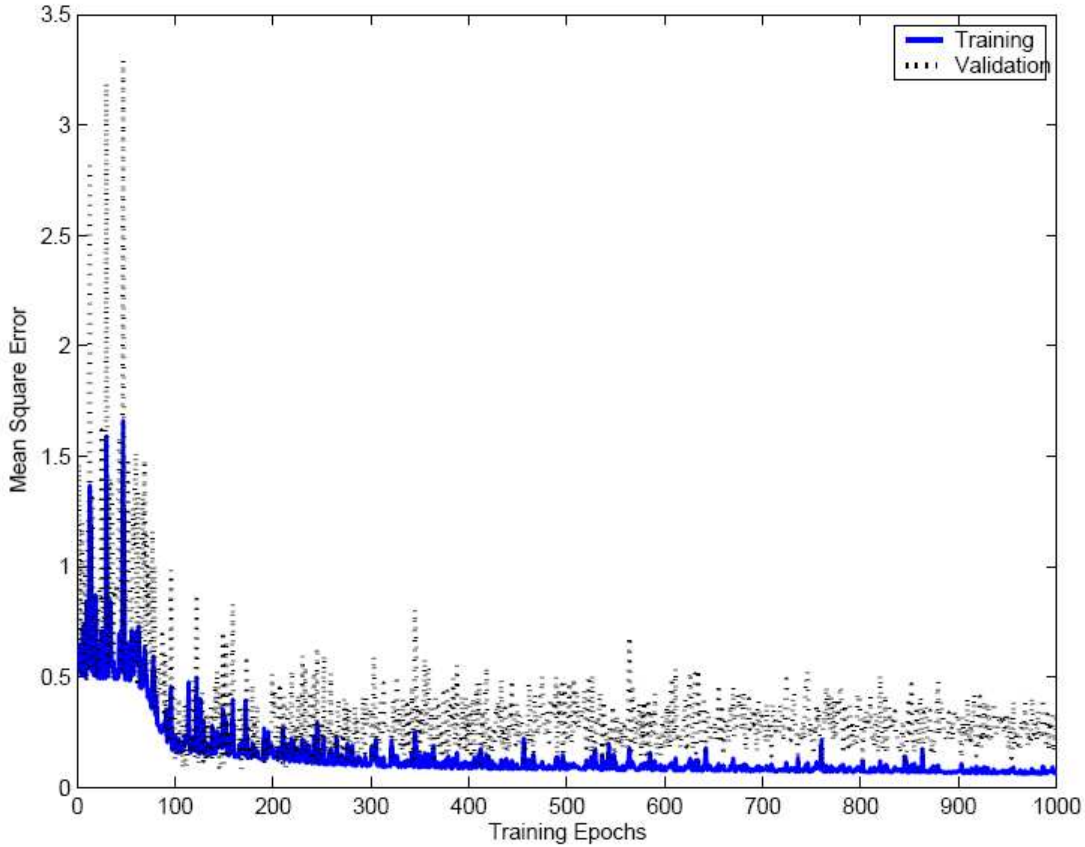
Forward pass

Calculate the analytical gradients

Gradient descent

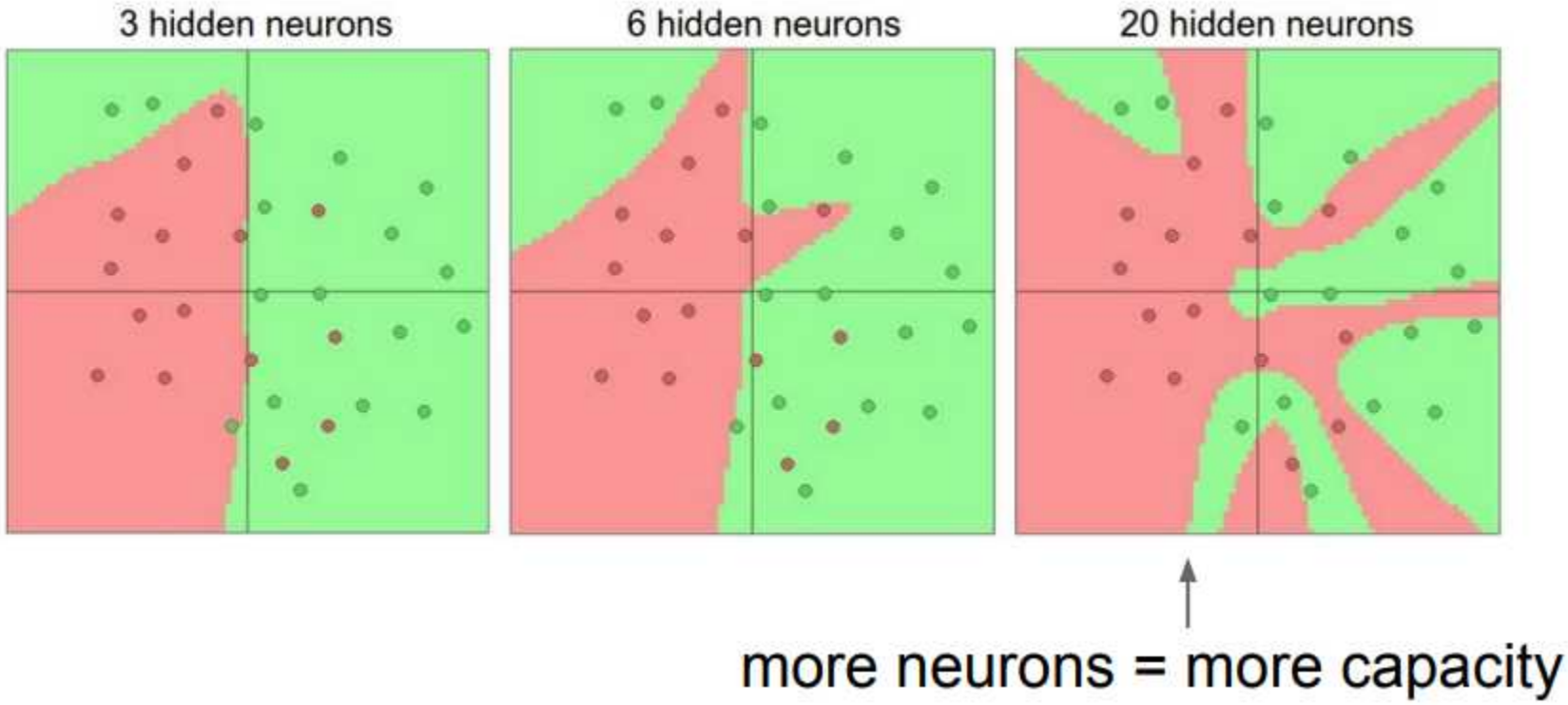
Training MLP

- Training Error / Validation Error



Training MLP

- Setting the number of layers



Training MLP

- Hyperparameter
  1) 초기 연결 강도 (weight)
     - 연결강도는 국지적 최적값에 수렴
     - 초기 연결강도에 따라 결과 달라짐
     - -0.5 - 0.5
  2) 신경망 구조
     - 입력층 / 출력층 수: 문제에 따라 결정
     - 은닉층 수: 적절히 결정하여야 함
       - 적은 경우 - 학습이 안 되는 현상 (에러가 줄어들지 않음)
       - 많은 경우 - 학습 속도가 느림, 에러가 줄어들지 않음
  3) 학습률
     - 큰 경우: 학습 속도 빠름, 학습 이루어지지 않을 수 있음 (overshoot)
     - 작은 경우: 학습속도 느림
  4) 모멘텀
     - 학습 초반에는 변화량을 크게, 학습 후반에는 변화량을 작게 => 학습 속도 향상

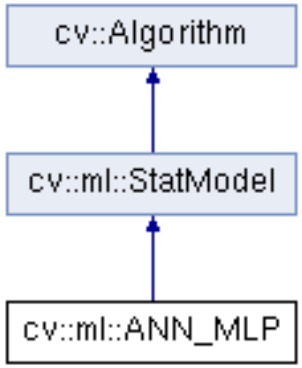
$$\Delta V_k = \alpha\delta_x + \beta\Delta V_{k-1}$$

$$\Delta W_k = \alpha\delta_y + \beta\Delta W_{k-1}$$

β 모멘텀상수

OpenCV

- ANN_MLP Class



Public Types

enum **ActivationFunctions** {
IDENTITY = 0,
SIGMOID_SYM = 1,
GAUSSIAN = 2,
RELU = 3,
LEAKYRELU = 4
}

enum **TrainFlags** {
UPDATE_WEIGHTS = 1,
NO_INPUT_SCALE = 2,
NO_OUTPUT_SCALE = 4
}

enum **TrainingMethods** {
BACKPROP =0,
RPROP = 1,
ANNEAL = 2
}

Letter_recog.cpp

```
static bool
build_mlp_classifier( const string& data_filename,const string& filename_to_save, const string& filename_to_load )
{
    const int class_count = 26;
    Mat data;
    Mat responses;

    bool ok = read_num_class_data( data_filename, 16, &data, &responses );
    if( !ok )
        return ok;

    Ptr<ANN_MLP> model;

    int nsamples_all = data.rows;
    int ntrain_samples = (int)(nsamples_all*0.8);

    // Create or load MLP classifier
    if( !filename_to_load.empty() )
    {
        model = load_classifier<ANN_MLP>(filename_to_load);
        if( model.empty() )
            return false;
        ntrain_samples = 0;
    }
    else
    {
        Mat train_data = data.rowRange(0, ntrain_samples);
        Mat train_responses = Mat::zeros( ntrain_samples, class_count, CV_32F );

        // 1. unroll the responses
        cout << "Unrolling the responses...₩n";
        for( int i = 0; i < ntrain_samples; i++ )
        {
            int cls_label = responses.at<int>(i) - 'A';
            train_responses.at<float>(i, cls_label) = 1.f;
        }

        // 2. train classifier
        int layer_sz[] = { data.cols, 100, 100, class_count };
        int nlayers = (int)(sizeof(layer_sz)/sizeof(layer_sz[0]));
        Mat layer_sizes( 1, nlayers, CV_32S, layer_sz );
```

```
#if 1
        int method = ANN_MLP::BACKPROP;
        double method_param = 0.001;
        int max_iter = 300;
#else
        int method = ANN_MLP::RPROP;
        double method_param = 0.1;
        int max_iter = 1000;
#endif

        Ptr<TrainData> tdata = TrainData::create(train_data, ROW_SAMPLE, train_responses);

        cout << "Training the classifier (may take a few minutes)...₩n";
        model = ANN_MLP::create();
        model->setLayerSizes(layer_sizes);
        model->setActivationFunction(ANN_MLP::SIGMOID_SYM, 0, 0);
        model->setTermCriteria(TC(max_iter,0));
        model->setTrainMethod(method, method_param);
        model->train(tdata);
        cout << endl;
    }

    test_and_save_classifier(model, data, responses, ntrain_samples, 'A', filename_to_save);
    return true;
}
```