Multilayer Perceptron (MLP)

Seungjin Choi

Department of Computer Science and Engineering
Pohang University of Science and Technology
77 Cheongam-ro, Nam-gu, Pohang 37673, Korea
seungjin@postech.ac.kr

Outline

- Perceptron: A single layer neural network.
- Multilayer perceptron (MLP): A multilayer extension of perceptron.
  - Universal approximation
  - Error back-propagation (BP) algorithm

Linear Classification

- Let a real-valued function $f : \mathcal{X} \subseteq \mathbb{R}^m \to \mathbb{R}$ be a discriminant function.
- In binary classification, the input $\boldsymbol{x} \in \mathcal{X}$ is assigned to the positive class if $f(\boldsymbol{x}) \geq 0$, and otherwise to the negative class.
- Linear classification considers a linear discriminant function which has the form

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b,$$

  where $(\boldsymbol{w}, b) \in \mathbb{R}^m \times \mathbb{R}$ (weight vector, bias) are the parameters that control the function.
- Decision rule is given by $\operatorname{sgn}(f(\boldsymbol{x}))$,

$$\operatorname{sgn}(f(\boldsymbol{x})) = \begin{cases} 1 & \text{if } f(\boldsymbol{x}) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

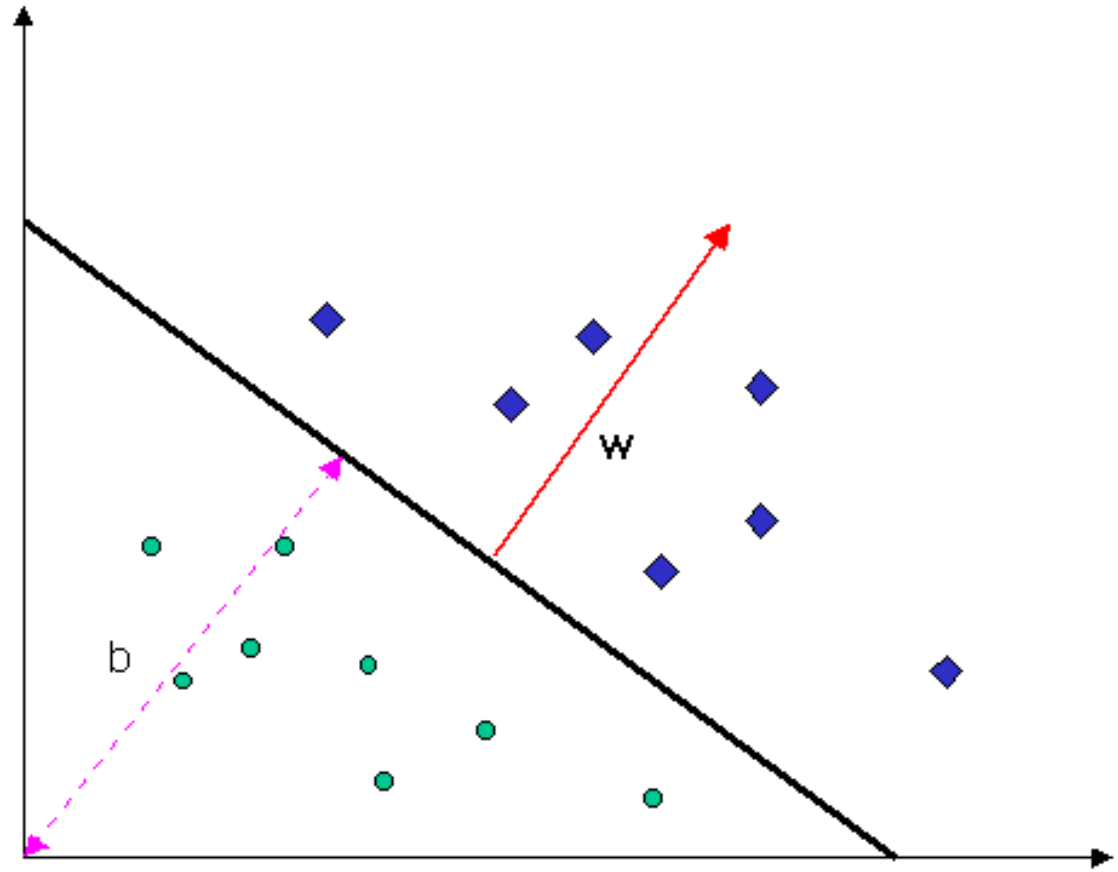
Separating Hyperplane

- A separating hyperplane is defined by

$$\boldsymbol{w}^\top \boldsymbol{x} + b = 0.$$

- The input space $\mathcal{X}$ is split into two parts by the hyperplane
- The separating hyperplane is an affine subspace of dimension $m - 1$ which divides the space into two half spaces which corresponds to the inputs of the two distinct classes.

Separating Hyperplane: Geometric View



Perceptron

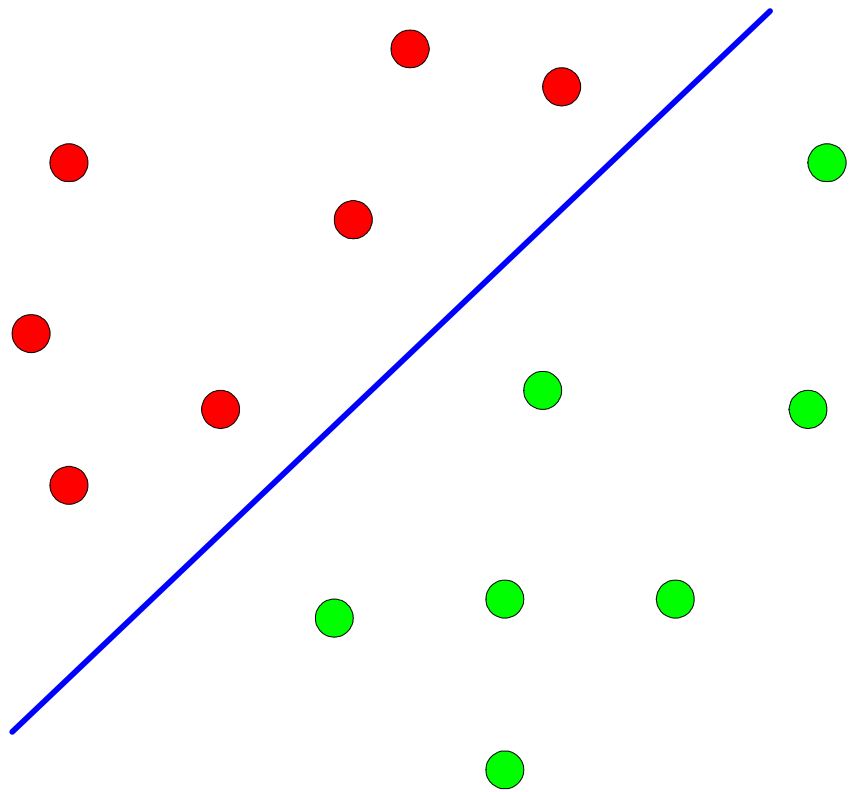
- Proposed by Rosenblatt in 1956
- The first iterative algorithm for learning linear classification
- A single-layer neural network with threshold activation functioin:

$$y = \text{sgn}\left(\boldsymbol{w}^\top \boldsymbol{x} + b\right)$$

- On-line and mistake-driven procedure: The weight vector is updated each time a training point is misclassified
- Perceptron Convergence: The algorithm is guaranteed to converge when data are linearly separable

Linearly Separable

Two classes of patterns are "linearly separable" if they can be separated by a linear hyperplane. In other words, there exists a hyperplane which separates two classes.



Perceptron Criterion

Suppose that target values $\{y_t\}$ take either 1 or -1:

$$y_t = \begin{cases} 1 & \text{if } \boldsymbol{x} \in \mathcal{C}_1 \\ -1 & \text{if } \boldsymbol{x} \in \mathcal{C}_2 \end{cases}$$

What we want here is to find a $\boldsymbol{w}$ such that

$$\begin{cases} \boldsymbol{w}^\top \boldsymbol{x}_t > 0 & \text{for } \boldsymbol{x}_t \in \mathcal{C}_1 \\ \boldsymbol{w}^\top \boldsymbol{x}_t < 0 & \text{for } \boldsymbol{x}_t \in \mathcal{C}_2, \end{cases}$$

which is identical to

$$\boldsymbol{w}^\top \boldsymbol{x}_t y_t > 0 \quad \forall \boldsymbol{x}_t.$$

Perceptron Criterion (Cont'd)

The perceptron criterion leads to the following objective function

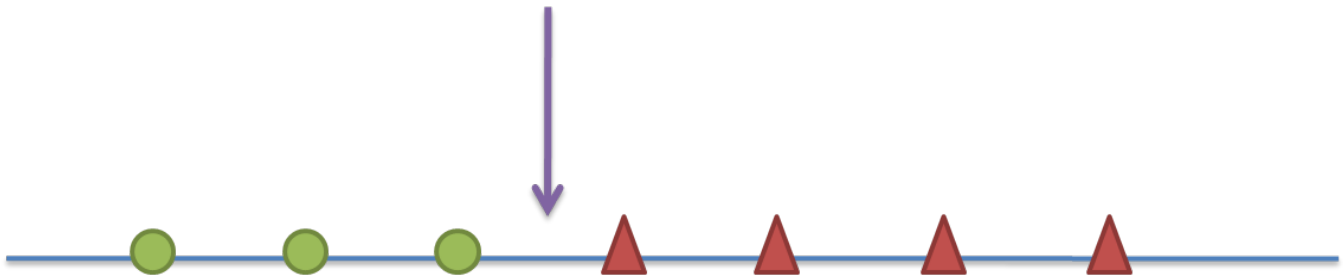
$$\mathcal{E}(\boldsymbol{w}) = -\sum_{\boldsymbol{x}_t \in \mathcal{M}} \boldsymbol{w}^\top \boldsymbol{x}_t y_t,$$

where $\mathcal{M}$ is the set of vectors $\boldsymbol{x}_t$ which are misclassified by the current weight vector.

The gradient of $\mathcal{E}(\boldsymbol{w})$ is

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{w}} = -\sum_{\boldsymbol{x}_t \in \mathcal{M}} \boldsymbol{x}_t y_t.$$

Perceptron Learning: A Basic Idea



If correct, do not move. If not correct, move it to the left.

Perceptron Learning

If the pattern is correctly classified, do nothing. If not,

$$\Delta \boldsymbol{w} = \eta \sum_{\boldsymbol{x}_t \in \mathcal{M}} \boldsymbol{x}_t y_t.$$

Perceptron Learning: Algorithm Outline

1. Get a training sample
2. Check to see if it is misclassified
   - 2.1 If classified correctly, do nothing
   - 2.2 If classified incorrectly, update $\boldsymbol{w}$ by

$$\boldsymbol{w}_{(k+1)} = \boldsymbol{w}_{(k)} + \eta \boldsymbol{x}_t y_t$$

3. Repeat steps 1 and 2 until convergence

Perceptron Convergence Theorem

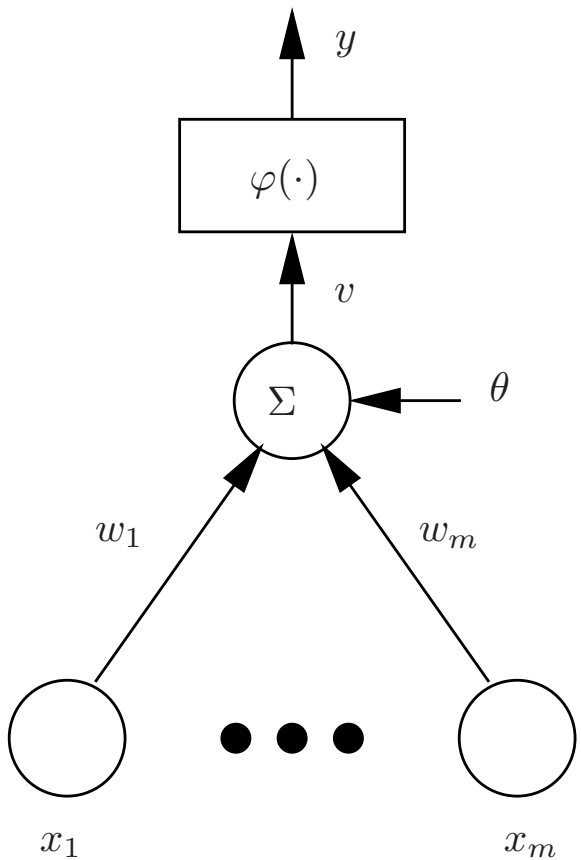
- The perceptron classifier minimizes the error probability, while MMSE classifier does not.
- One can easily see that the perceptron learning reduce the error

$$\begin{aligned} -\boldsymbol{w}_{(k+1)}^{\top}\boldsymbol{x}_t y_t &= -\boldsymbol{w}_{(k)}^{\top}\boldsymbol{x}_t y_t - \sum_{\boldsymbol{x}_t \in \mathcal{M}} \|\boldsymbol{x}_t y_t\|^2 \\ &\leq -\boldsymbol{w}_{(k)}^{\top}\boldsymbol{x}_t y_t. \end{aligned}$$

Theorem

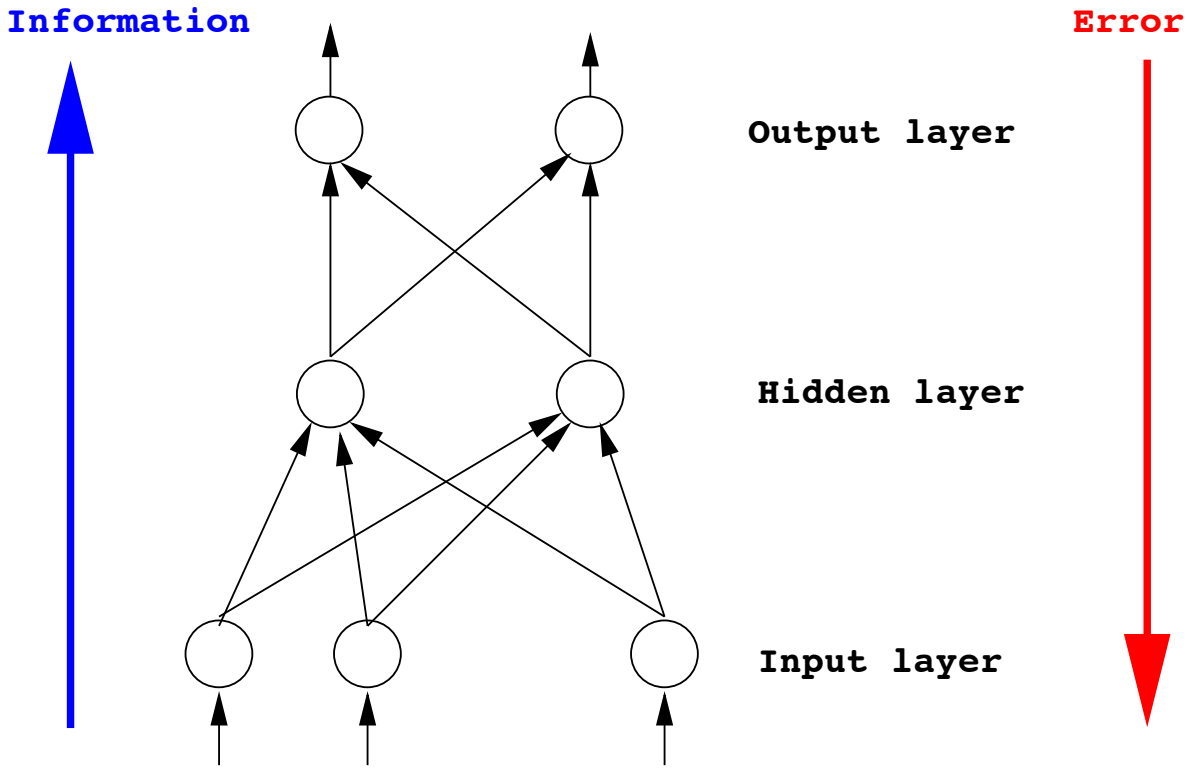
If classes $\mathcal{C}_1$ and $\mathcal{C}_2$ are linearly separable, then the perceptron rule converges in a finite number of steps to a separating hyperplane.

McCulloch-Pitts Model



- $v = w_1 x_1 + \cdots + w_m x_m - \theta$
- $y = \varphi(v)$
- $\varphi(\cdot)$: squashing function (hard-limiter, logistic, tanh)

MLP: Structure



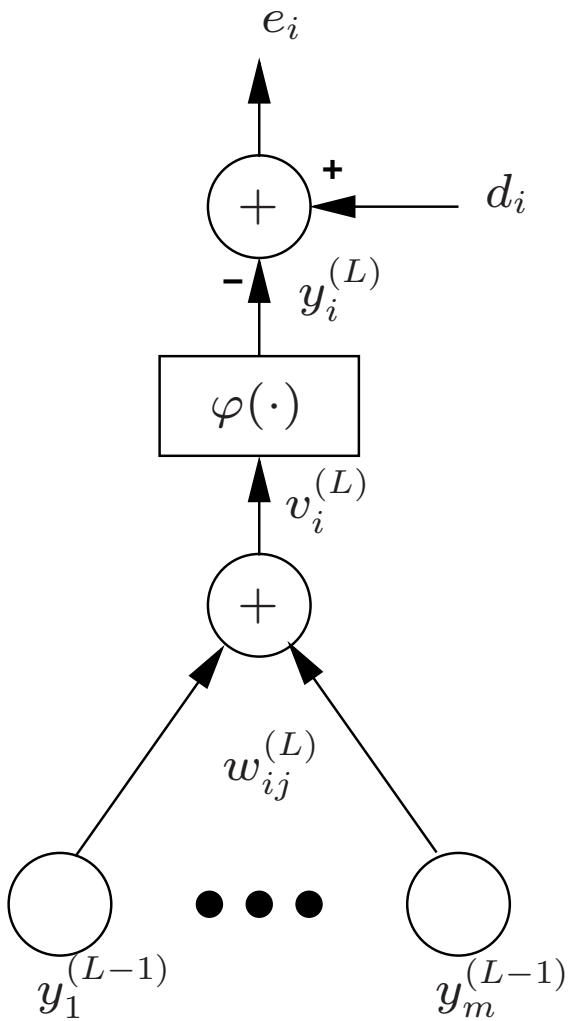
Train MLP: Error Back-propagation

- Error function (sum of squared errors) is given by

$$\mathcal{E} = \frac{1}{2} \sum_j e_j^2,$$

  where $e_j = d_j - y_j^{(L)}$.
- We can use the gradient method to derive an updating algorithm for training MLPs. Any problem?
- Weights connecting hidden layer and output layer, can be easily updated, using the gradient method. What about the rest of weights?
- Use the idea of error back-propagation!

MLP Output Layer



Updating Final Layer Weights

The input-output relation is given by

$$v_i^{(L)} = \sum_j w_{ij}^{(L)} y_j^{(L-1)}, \quad y_i^{(L)} = \varphi(v_i^{(L)}).$$

Compute the gradient

$$\frac{\partial \mathcal{E}}{\partial w_{ij}^{(L)}} = \frac{\partial \mathcal{E}}{\partial v_i^{(L)}} \frac{\partial v_i^{(L)}}{\partial w_{ij}^{(L)}},$$

where

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial v_i^{(L)}} &= \frac{\partial \mathcal{E}}{\partial e_i} \frac{\partial e_i}{\partial y_i^{(L)}} \frac{\partial y_i^{(L)}}{\partial v_i^{(L)}} = -e_i \, \varphi' \left(v_i^{(L)} \right), \\ \frac{\partial v_i^{(L)}}{\partial w_{ij}^{(L)}} &= y_j^{(L-1)}. \end{aligned}$$

Updating Final Layer Weights (Cont'd)

Define the local gradient

$$\delta_i^{(L)} = -\frac{\partial \mathcal{E}}{\partial v_i^{(L)}} = e_i \, \varphi' \left(v_i^{(L)} \right).$$

Then the updating algorithm for $w_{ij}^{(L)}$ is given by

$$\begin{aligned} w_{ij}^{(L)}(t+1) &= w_{ij}^{(L)}(t) - \eta \frac{\partial \mathcal{E}}{\partial w_{ij}^{(L)}} \\ &= w_{ij}^{(L)}(t) + \eta \, \delta_i^{(L)}(t) \, y_j^{(L-1)}(t). \end{aligned}$$

This updating rule is true for every layer, but the problem will lie in computing the local gradient, except for the final layer.

We need a recursion for the local gradient $\rightarrow$ the key idea of BP

Recursion for the local gradient

$$\delta_i^{(L-1)} = -\frac{\partial \mathcal{E}}{\partial v_i^{(L-1)}} = -\frac{\partial \mathcal{E}}{\partial y_i^{(L-1)}} \frac{\partial y_i^{(L-1)}}{\partial v_i^{(L-1)}} = -\frac{\partial \mathcal{E}}{\partial y_i^{(L-1)}} \varphi' \left(v_i^{(L-1)} \right).$$

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial y_i^{(L-1)}} &= \sum_k e_k \frac{\partial e_k}{\partial y_i^{(L-1)}} = \sum_k e_k \frac{\partial e_k}{\partial v_k^{(L)}} \frac{\partial v_k^{(L)}}{\partial y_i^{(L-1)}} \\ &= -\sum_k e_k \varphi' \left(v_k^{(L)} \right) w_{ki}^{(L)} = -\sum_k \delta_k^{(L)} w_{ki}^{(L)}. \end{aligned}$$

Then, the recursion for the local gradient is described by

$$\begin{aligned} \delta_i^{(L-1)} &= -\frac{\partial \mathcal{E}}{\partial y_i^{(L-1)}} \varphi' \left(v_i^{(L-1)} \right) \\ &= \left(\sum_k \delta_k^{(L)} w_{ki}^{(L)} \right) \varphi' \left(v_i^{(L-1)} \right). \end{aligned}$$

Algorithm Outline: BP

$$w_{ij}^{(l)}(t+1) \quad = \quad w_{ij}^{(l)}(t) + \eta\, \delta_i^{(l)}(t)\, y_j^{(l-1)}(t).$$

$$\delta_i^{(l)}(t) \quad = \quad \begin{cases} \varphi'\left(v_i^{(l)}(t)\right) \sum_k \delta_k^{(l+1)}(t) w_{ki}^{(l+1)}(t) & \text{hidden neuron} \\ e_i^{(L)}(t)\, \varphi'\left(v_i^{(L)}(t)\right) & \text{output neuron} \end{cases}$$