Neural Network

Eun Yi Kim

Artificial Inteligence
& Computer Vision
Laboratory

I N D E X

What is a deep learning?

Perceptron

- Activation function
- LMS learning

Multiple Layer Perceptron

- Hidden units
- Backpropagation

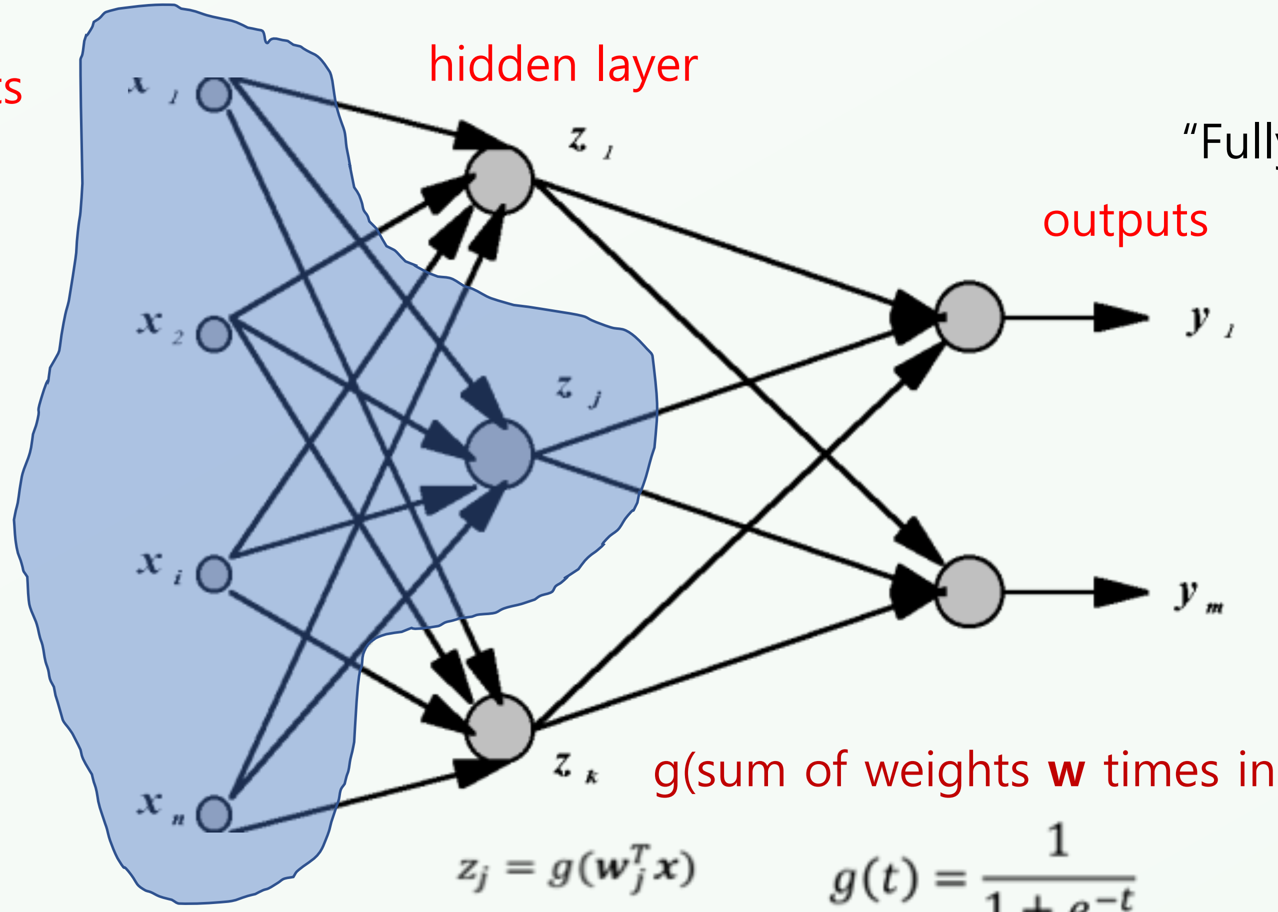
Summary

Artificial Inteligence
& Computer Vision
Laboratory

What is deep learning?

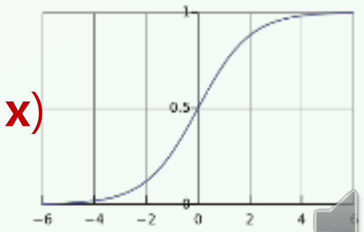
- Deep learning refers to training a neural network
  - A function that fits some data

$$x = (x_1, \dots, x_{784})^T$$



g(sum of weights **w** times inputs **x**)

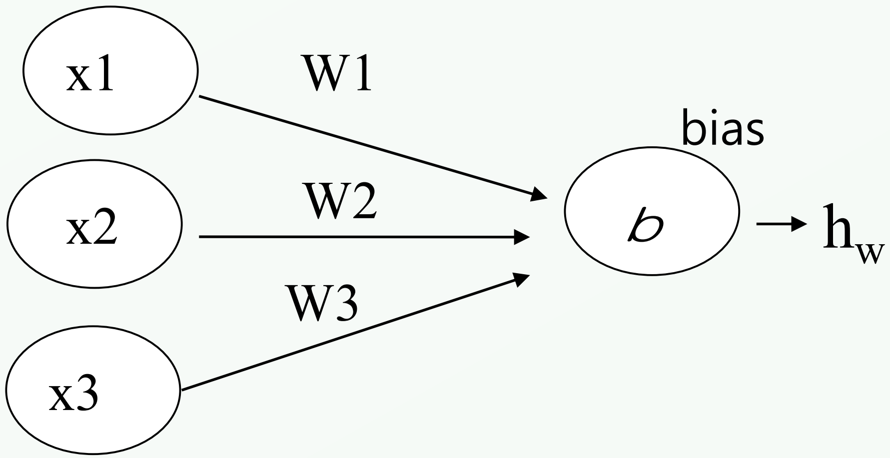
$$z_j = g(\mathbf{w}_j^T \mathbf{x}) \qquad g(t) = \frac{1}{1+e^{-t}}$$



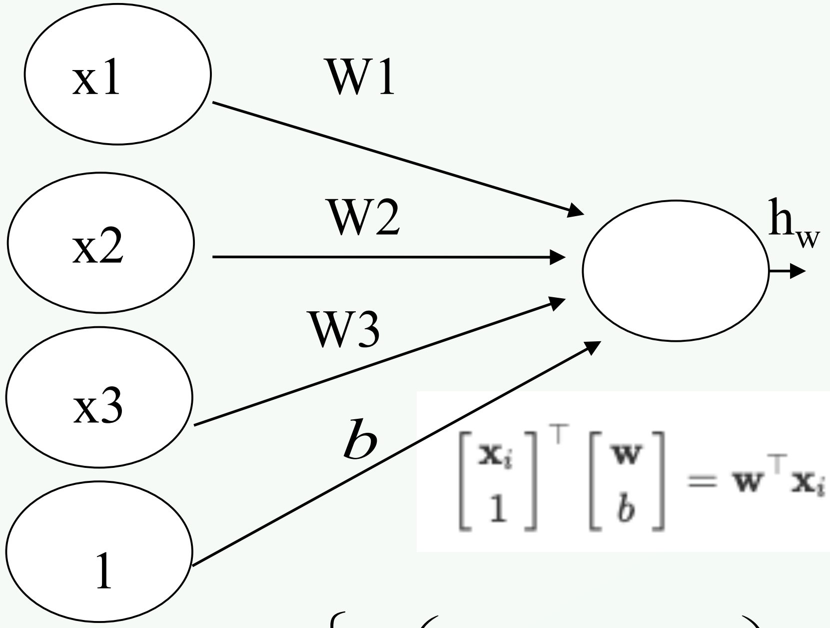
Perceptron

Perceptron

- Initial proposal of connectionist networks
- Rosenblatt, 50's and 60's
- Essentially a linear discriminant composed of nodes, weights



or



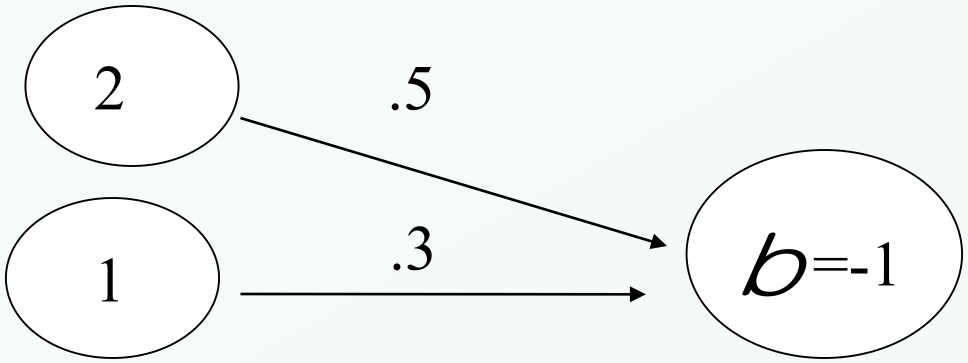
$$\begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix}^\top \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} = \mathbf{w}^\top \mathbf{x}_i + b$$

Hypothesis $h_w = \left\{ \begin{array}{c} 1 : \left(\sum_i w_i x_i \right) + b > 0 \\ 0 : otherwise \end{array} \right\}$

$$h_w = \left\{ \begin{array}{l} 1 : \left(\sum_i w_i x_i + b \right) > 0 \\ 0 : otherwise \end{array} \right\}$$

Perceptron Example

Artificial Inteligence & Computer Vision Laboratory



$$2(0.5) + 1(0.3) + -1 = 0.3 > 0 \text{ , so } h_w=1$$

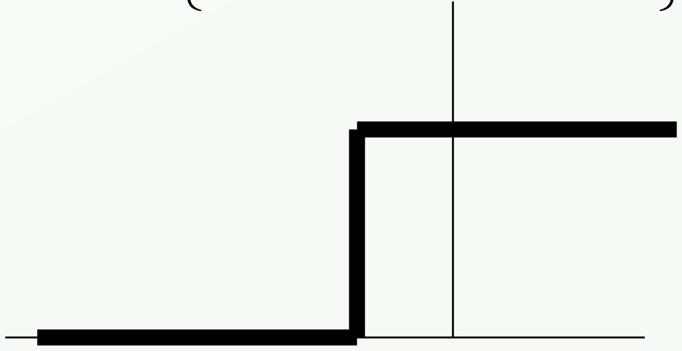
Learning Procedure:

- Randomly assign weights (between 0 and1)
- Present inputs from training data
- Get output hw, nudge weights to gives results toward our desired output T
- Repeat; stop when no errors, or enough epochs completed

Perceptron with Activation Function

Old:

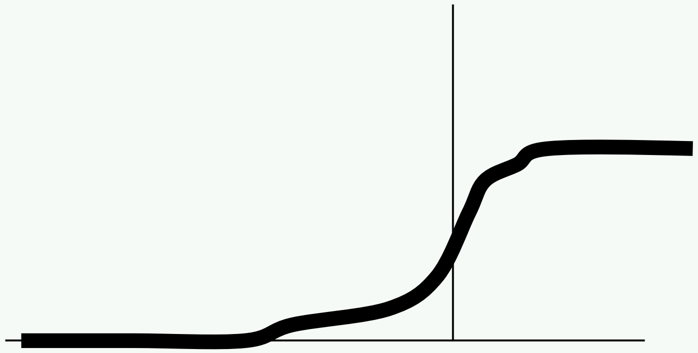
$$h_w = \begin{cases} 1 : \sum_j w_j x_j + b > 0 \\ 0 : otherwise \end{cases}$$



$-b$

New:

$$h_w = \frac{1}{1 + e^{-\sum_j w_j x_l + b}}$$



Activation Function: g

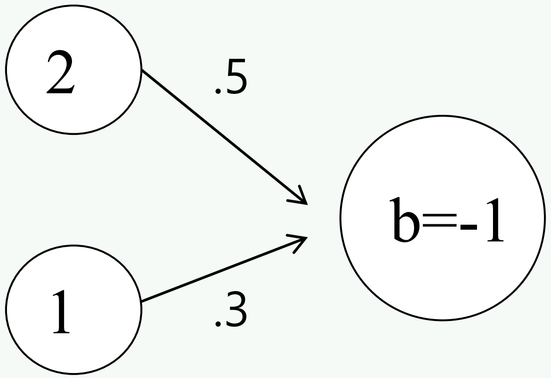
$$h_w = \begin{cases} 1 : \left(\sum_i w_i x_i + b\right) > 0 \\ 0 : otherwise \end{cases}$$

Perceptron is essentially linear classifier

Simple Perception Training

$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

$$\Delta w_i(t) = (y - h_w)I_i$$



Weights include Threshold. y=Desired, h_w=Actual output.

Example: y=0, h_w=1, W1=0.5, W2=0.3, I1=2, I2=1, θ=-1

$$w_1(t+1) = 0.5 + (0-1)(2) = -1.5$$
$$w_2(t+1) = 0.3 + (0-1)(1) = -0.7$$
$$w_\theta(t+1) = -1 + (0-1)(1) = -2$$

If we present this input again, we'd output 0 instead

A learning rule for Perceptron

Artificial Inteligence
& Computer Vision
Laboratory

- Threshold perceptrons have some advantages , in particular
  - Simple learning algorithm that fits a threshold perceptron to any linearly separable training set.

- Key idea: Learn by adjusting weights to reduce error on training set.
  - update weights repeatedly (epochs) for each example.

- We'll use Sum of squared errors
  - Learning is an optimization search problem in weight space.

A learning rule for Perceptron

Artificial Inteligence & Computer Vision Laboratory

- Let S = {(x_i, y_i): i = 1, 2, ..., N} be a training set. (Note, x is a vector of inputs, and y is the vector of the true outputs.)

- Let h_w be the perceptron classifier represented by the weight vector w.

- Definition:

$$E(\mathbf{x}) = Squared\ Error(\mathbf{x}) = \frac{1}{2}(y - h_{\mathbf{w}}(\mathbf{x}))^2$$

A learning rule for Perceptron

- The squared error for a single training example with input x and true output y is:

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2 \,,$$

$$h_w = g(\sum_j w_j I_j + \Theta) = \frac{1}{1 + e^{-\sum_j w_j I_j + \Theta}}$$

- Where $h_w(x)$ is the output of the perceptron on the example and y is the true output value.
- We can use the gradient descent to reduce the squared error by calculating the partial derivatives of E with respect to each weight.

$$\begin{aligned} \frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}(y - g(\Sigma_{j=0}^{n} W_j x_j)) \\ &= -Err \times g'(in) \times x_j \end{aligned}$$

  - Note: g’(in) derivative of the activation function. For sigmoid g’=g(1-g).
  - For threshold perceptrons, g’(n) is undefined, the original perceptron rule simply omitted it.

A learning rule for Perceptron

$$\frac{\partial E}{\partial W_j} = -Err \times g'(in) \times x_j$$

- Gradient descent algorithm → we want to reduce , E, for each weight w_i , change weight in direction of steepest descent:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

α - learning rate

$$W_j \leftarrow W_j + \alpha \times I_j \times Err$$

- Intuitively:
  - Err = y – $h_W(x)$ positive
    → weights are increased for positive inputs and decreased for negative inputs.
  - Err = y – $h_W(x)$ negative
    → opposite

$W_j \leftarrow W_j + \alpha \times I_j \times Err$

Perceptron learning rule:

1. Start with random weights, $\mathbf{w} = (w_1, w_2, \ldots, w_n)$.
2. Select a training example $(\mathbf{x}, y) \in S$.
3. Run the perceptron with input $\mathbf{x}$ and weights $\mathbf{w}$ to obtain g
4. Let α be the training rate (a user-set parameter).

$$\forall w_i, w_i \leftarrow w_i + \Delta w_i,$$

where

$$\Delta w_i = \alpha(y - g(in))g'(in)x_i$$

5. Go to 2.

Epoch → cycle through the examples

Epochs are repeated until some stopping criterion is reached—typically, that the weight changes have become very small.

The stochastic gradient method selects examples randomly from the training set rather than cycling through them.

Perceptron Learning (details)

Update the weights by minimizing the errors

$$E_d = \frac{1}{2}(y^{(d)} - f^{(d)})^2$$

$$f^{(d)} = f(x^{(d)}; w) = \Sigma_i w_i x_i$$

$$w_i \leftarrow w_i + \Delta w_i$$
$$\Delta w_i = -\eta \frac{\partial E_d}{\partial w_i}$$

$\eta \in (0,1)$ 은 학습률 (learning rate)

$$\frac{\partial E_d}{\partial w_i} = \frac{\partial E_d}{\partial f^{(d)}} \frac{\partial f^{(d)}}{\partial w_i} = \frac{\partial}{\partial f^{(d)}} \frac{1}{2} \left(y^{(d)} - f^{(d)}\right)^2 \frac{\partial f^{(d)}}{\partial w_i}$$

$$= \frac{1}{2}(-2)\left(y^{(d)} - f^{(d)}\right)x_i^{(d)} = -\left(y^{(d)} - f^{(d)}\right)x_i^{(d)}$$

$$\boxed{w_i \leftarrow w_i + \eta\left(y^{(d)} - f^{(d)}\right)x_i^{(d)}}$$

Learning of Sigmoid Neuron (details)

- Output of sigmoid unit

$$s^{(d)} = \sum_i w_i x_i^{(d)} \qquad f^{(d)} = f(\mathbf{x}^{(d)}; \mathbf{w}) = \frac{1}{1 + \exp(-s^{(d)})}$$

- Weights

$$\begin{aligned}
\frac{\partial E_d}{\partial w_i} &= \frac{\partial E_d}{\partial f^{(d)}} \frac{\partial f^{(d)}}{\partial w_i} = \frac{\partial}{\partial f^{(d)}} \frac{1}{2} \left(y^{(d)} - f^{(d)}\right)^2 \frac{\partial f^{(d)}}{\partial s^{(d)}} \frac{\partial s^{(d)}}{\partial w_i} \\
&= \frac{1}{2}(-2)\left(y^{(d)} - f^{(d)}\right) f^{(d)} \left(1 - f^{(d)}\right) x_i^{(d)} \\
&= -\left(y^{(d)} - f^{(d)}\right) f^{(d)} \left(1 - f^{(d)}\right) x_i^{(d)}
\end{aligned}$$

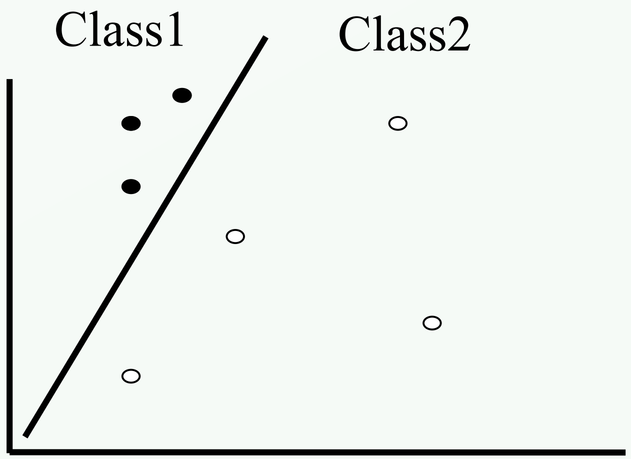
$$\boxed{w_i \leftarrow w_i + \eta\left(y^{(d)} - f^{(d)}\right) \boxed{f^{(d)}\left(1 - f^{(d)}\right)} x_i^{(d)}}$$

$$W_j \leftarrow W_j + \alpha \times I_j \times Err$$

Perceptrons are not powerful

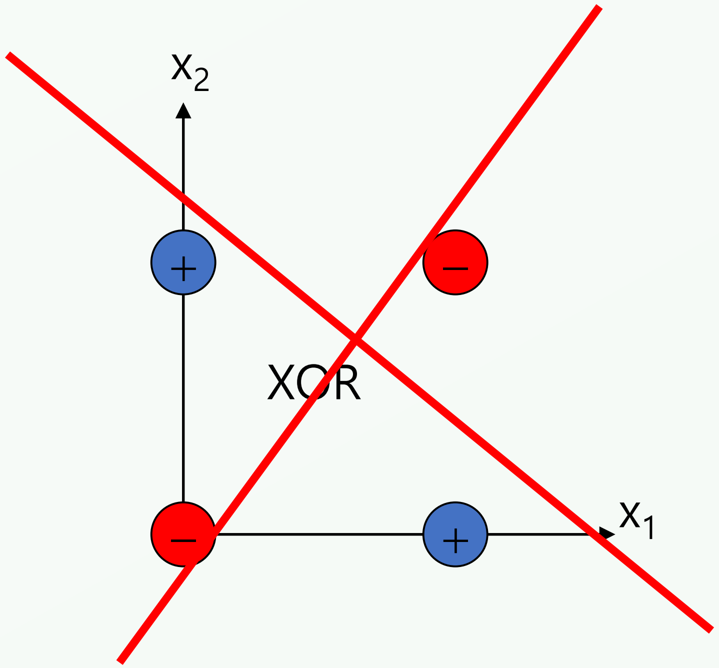
Artificial Inteligence & Computer Vision Laboratory

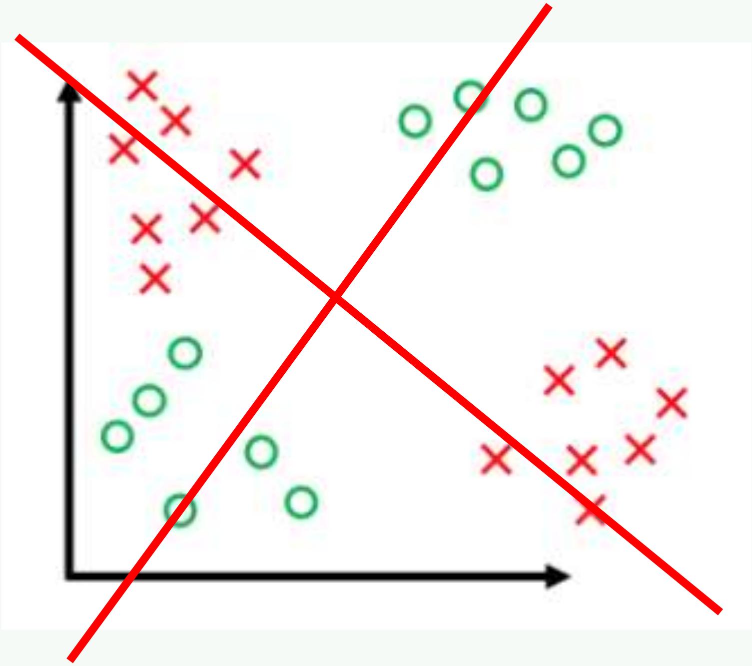
- Essentially a linear discriminant
- Perceptron theorem: If a linear discriminant exists that can separate the classes without error, the training procedure is guaranteed to find that li ne or plane.



Linear Separable Functions

- Minsky & Papert (1969): Perceptrons can only represent linearly separable functions





XOR problem
Not linearly separable

Linear Separable Functions

- Minsky & Papert (1969)
  - Perceptrons can only represent linearly separable functions
  - But, adding hidden layer allows more target functions to be represented