Neural Networks

Data Mining

Prof. Sujee Lee

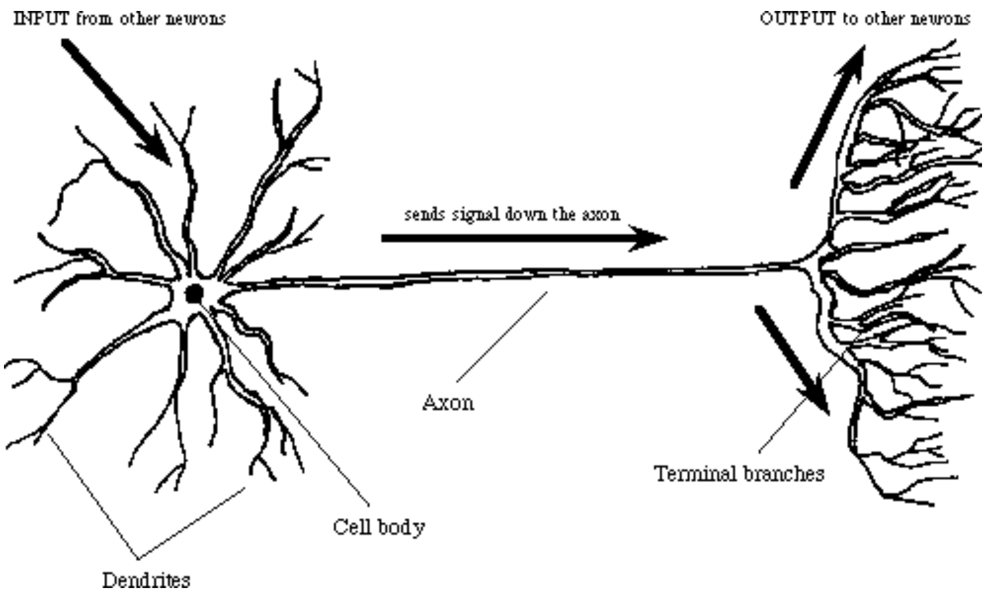
Department of Systems Management Engineering

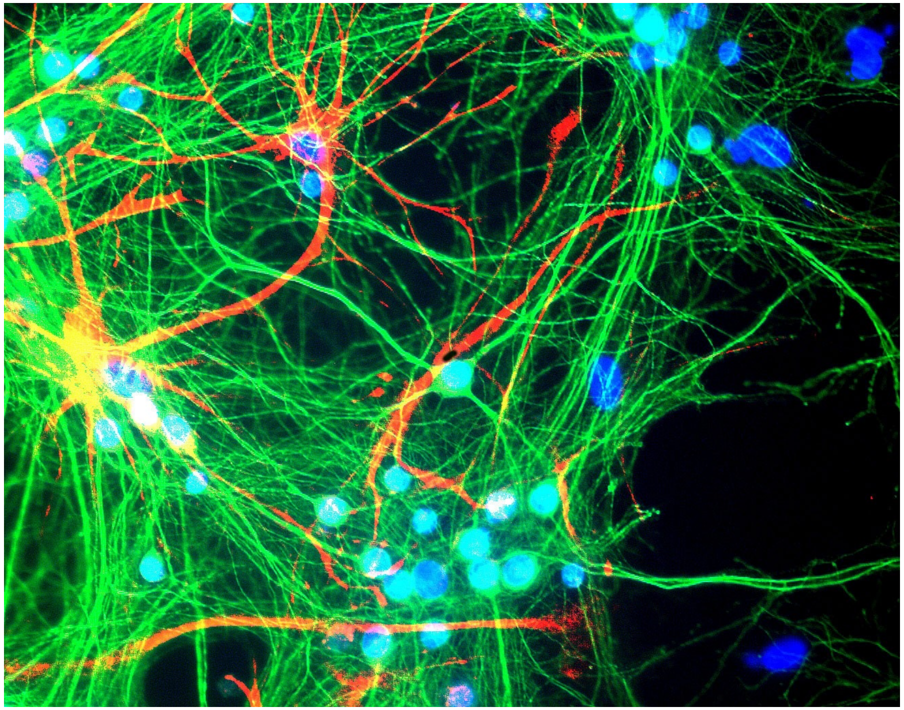
Sungkyunkwan University

Neural Networks

Neural Networks

- **Neural Networks**
  - a.k.a. artificial neural networks
  - inspired by interconnected neurons in biological systems
    - simple processing units
    - each unit receives a number of real-valued inputs
    - each unit produces a single real-valued output



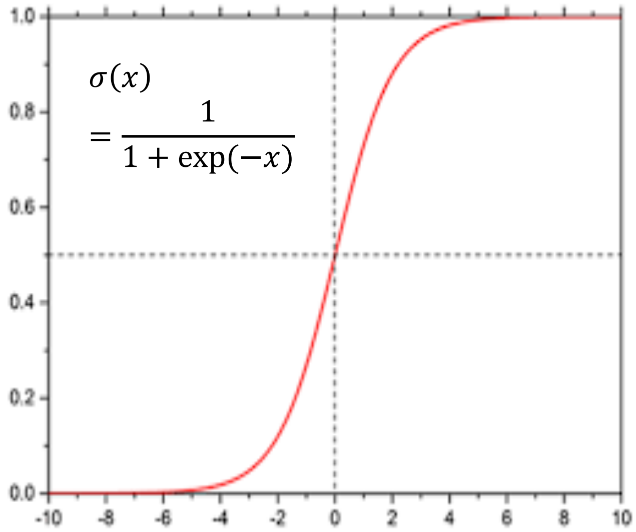


Logistic Regression

- **(Recap.) Logistic Regression**
  - Logistic Regression extends the ideas of linear regression for classification problem
    - i.e., the labels are binary $y = 0 \text{ or } 1$
  - we will use linear model $\mathbf{w}^T\boldsymbol{x} + b$, but $f(\boldsymbol{x})$ to be a probability

$$\hat{y} = f(\boldsymbol{x}) = \sigma(\mathbf{w}^T\boldsymbol{x} + b) = \frac{1}{1 + \exp(-\boldsymbol{w}^T\boldsymbol{x} - b)}$$

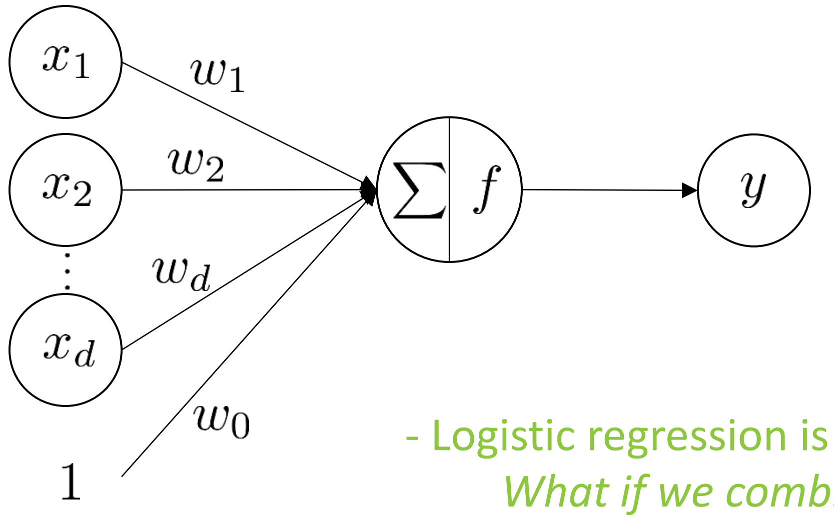
$$\boldsymbol{x} = (x_1, \dots, x_d) \in \mathbb{R}^d, \qquad y \in \mathbb{B}, \qquad 0 \le \hat{y} \le 1$$



Logistic Regression (cont.)

- Two steps for evaluation
  - Linear combination of inputs: $$s = \mathbf{w}^\top \mathbf{x} = \sum_{i=0}^{d} w_i x_i$$
  - Nonlinear transform of s : $$y = f(s) = \frac{1}{1 + \exp(-s)}$$

- Graphical representation (Single perceptron / Single neuron)

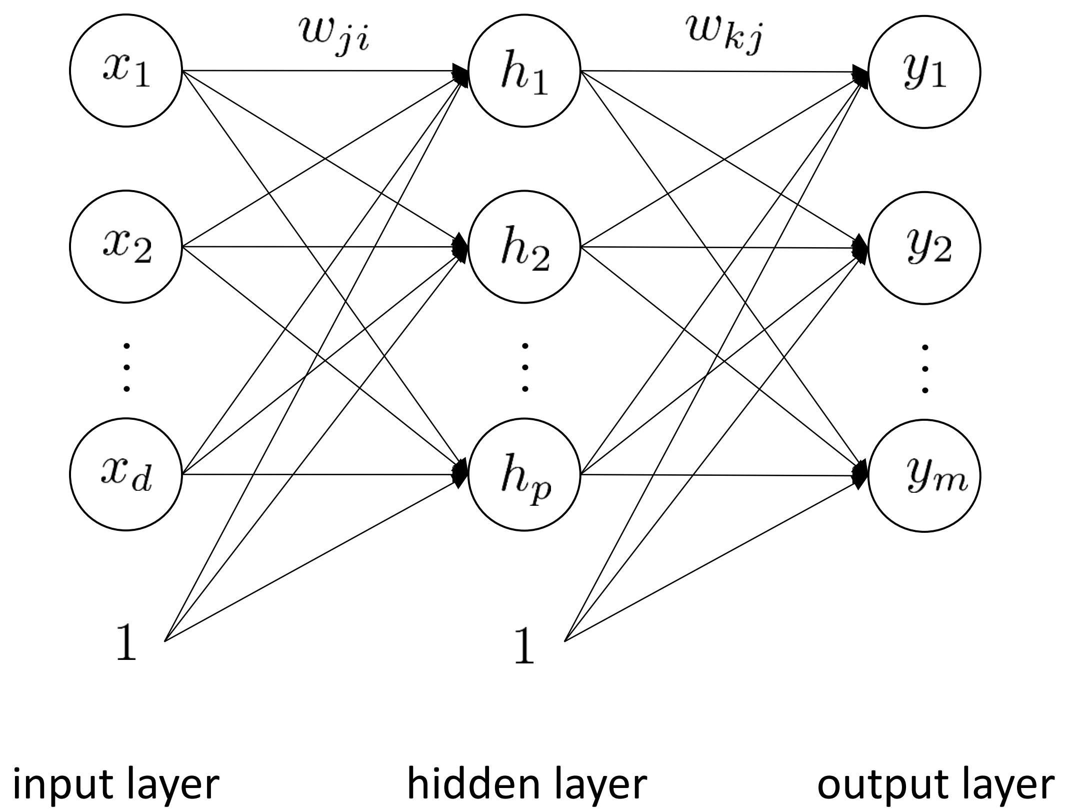


- Logistic regression is using ONE perceptron

What if we combine many perceptrons

Multi-layer Perceptron

- Let us cascade many perceptrons

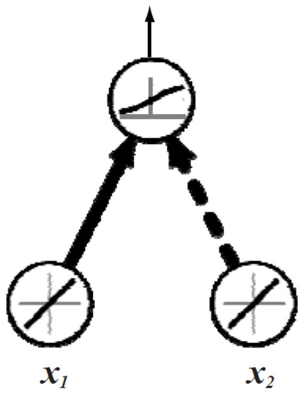


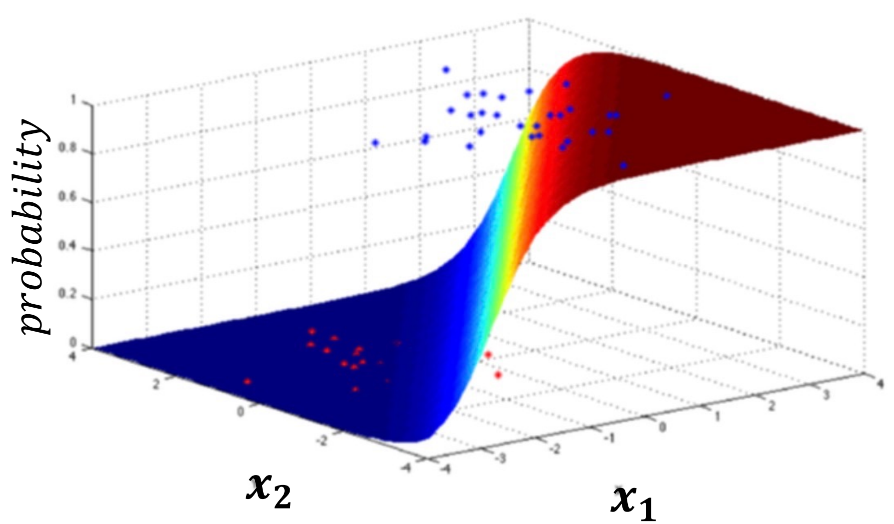
Multi-layer Perceptron

- **Structure of multi-layer perceptron**
  - **Input Layer**
    - Simply pass the input values to the next layer.
    - Number of inputs (features) = Number of input nodes
  - **Hidden Layer(s)**
    - Each hidden unit represents an intermediate processing step.
    - There can be several hidden layers.
    - There can be several hidden nodes at each hidden layer.
  - **Output Layer**
    - The output unit represents the prediction of the target label.
    - Number of outputs = Number of output nodes

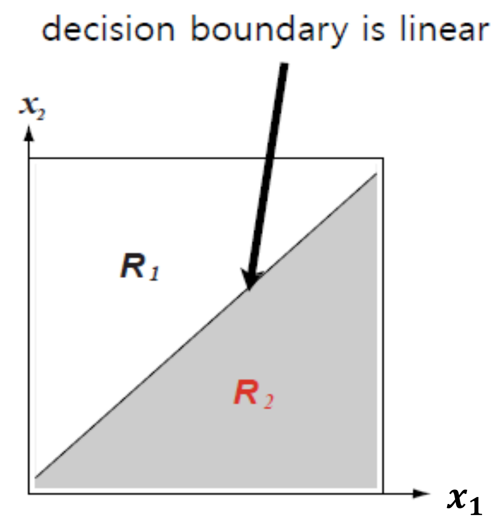
Representational Power of Perceptrons

- **One Perceptron**
  - Simple **logistic regression classifier**
    - If greater than 0.5, predict class 1
    - Otherwise, predict class 0
    - Can solve linearly separable problems





Logistic Regression

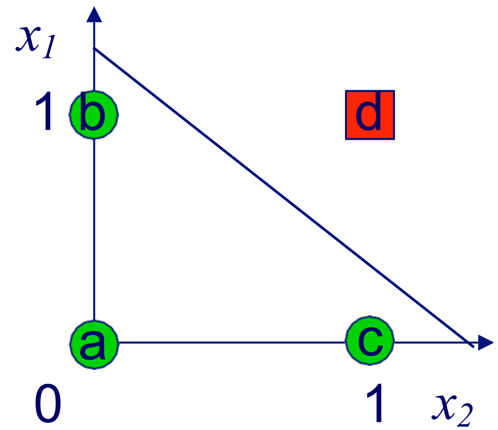


Representational Power of Perceptrons

- **Some linearly separable functions**

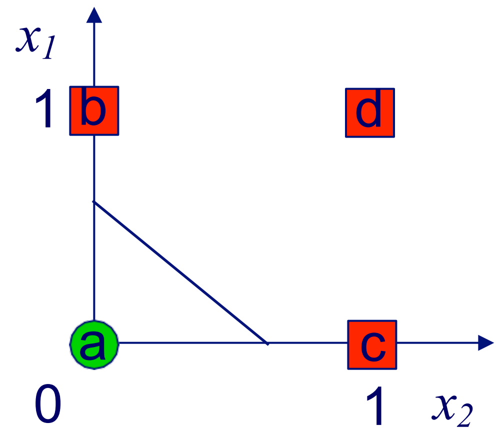
AND

	x_1	x_2	y
a	0	0	0
b	0	1	0
c	1	0	0
d	1	1	1



OR

	x_1	x_2	y
a	0	0	0
b	0	1	1
c	1	0	1
d	1	1	1

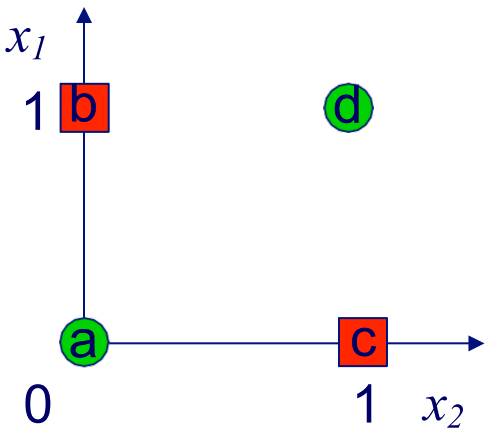


Representational Power of Perceptrons

- **XOR is not linearly separable**

XOR

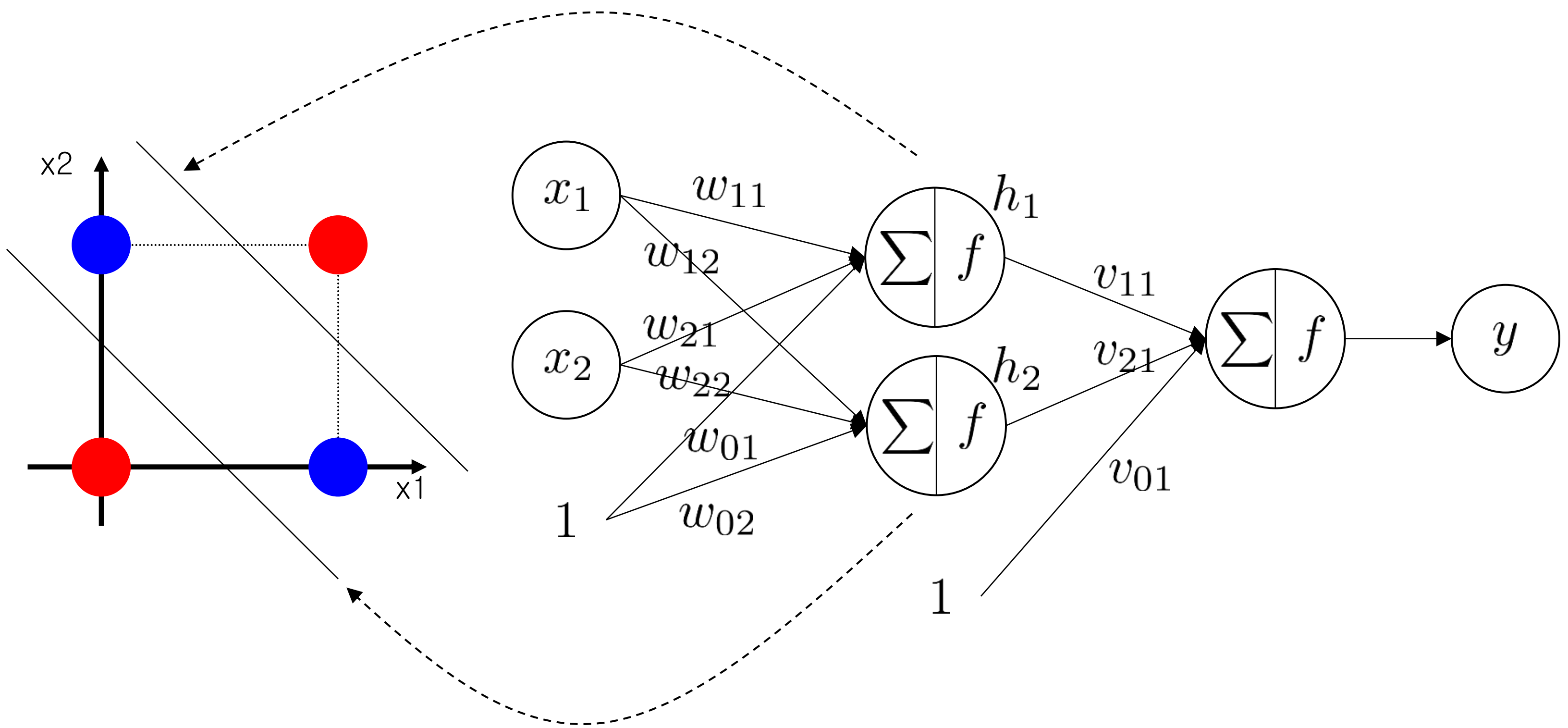
	x_1	x_2	y
a	0	0	0
b	0	1	1
c	1	0	1
d	1	1	0



  - a multilayer perceptron can represent XOR

Representational Power of Perceptrons

- **Solving XOR problem using MLP**



Representational Power of Perceptrons

- **Solving XOR problem using MLP**

$w_{11} = 1.0,\ w_{21} = 1.0,$
$w_{01} = -1.5$

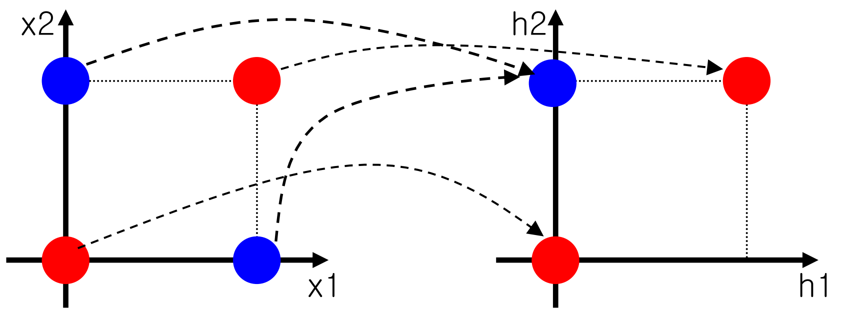
x_1	x_2	Σ	h_1
0	0	−1.5	0
0	1	−0.5	0
1	0	−0.5	0
1	1	0.5	1

$w_{12} = 1.0,\ w_{22} = 1.0,$
$w_{02} = -0.5$

x_1	x_2	Σ	h_2
0	0	−0.5	0
0	1	0.5	1
1	0	0.5	1
1	1	1.5	1

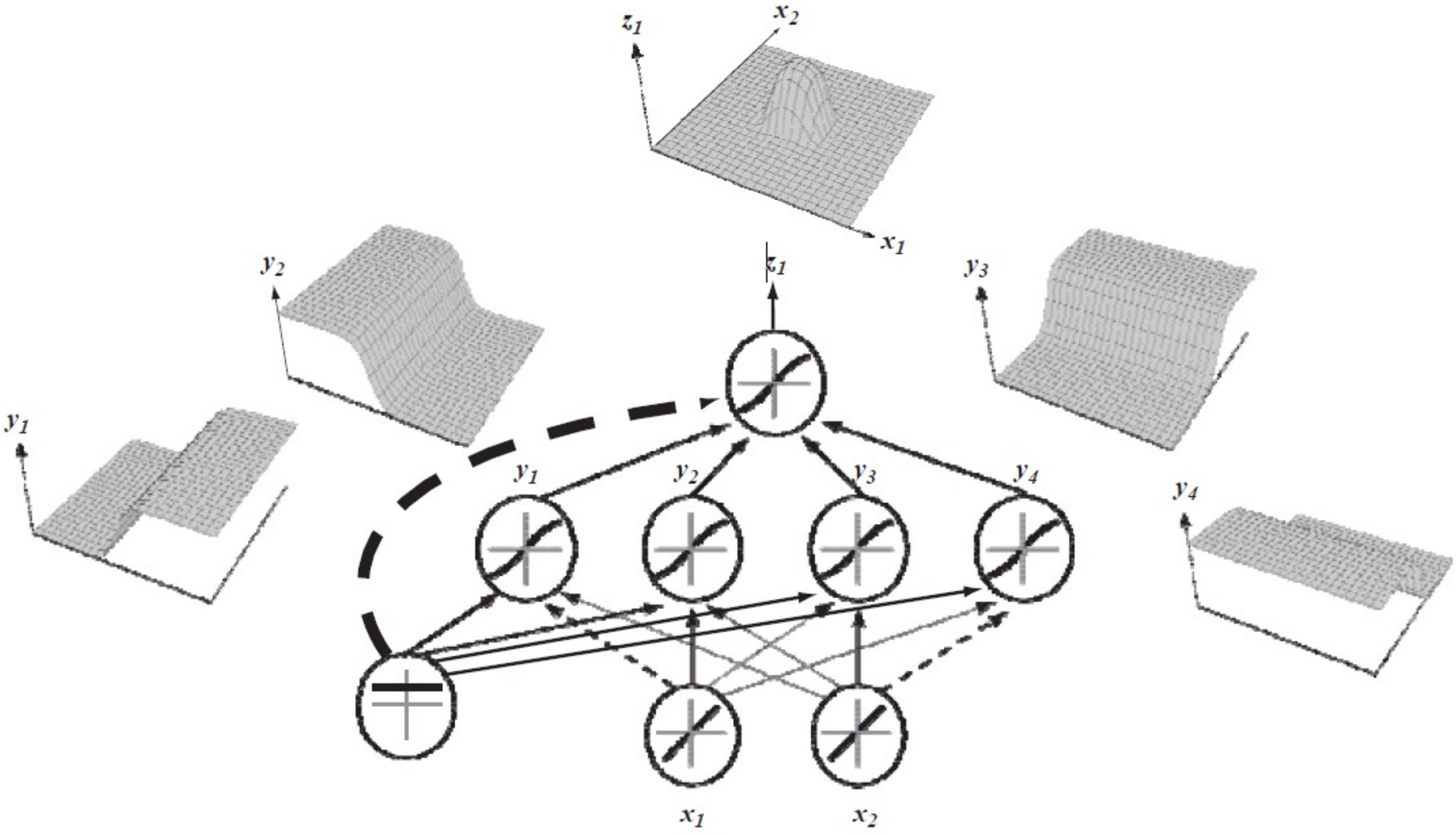
$v_{11} = -1.0,\ v_{21} = 1.0,$
$v_{01} = -0.5$

h_1	h_2	Σ	y
0	0	−0.5	0
0	1	0.5	1
0	1	0.5	1
1	1	−0.5	0



Representational Power of Perceptrons

- **Capacity of neural network**

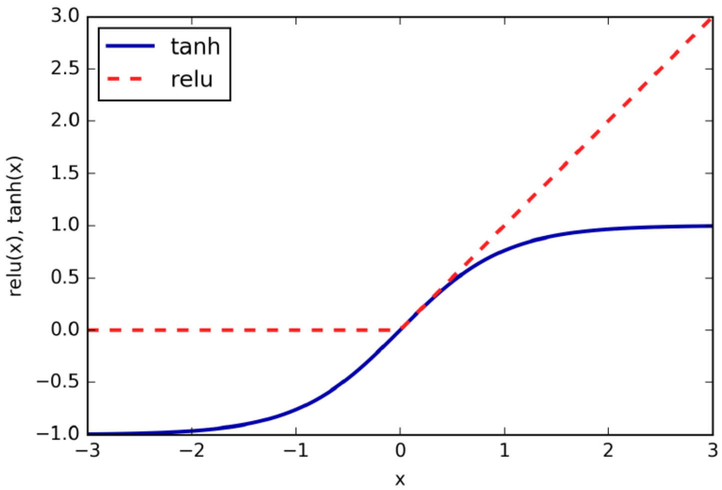


Structure of multi-layer perceptron

- **Hidden Layers: Exploiting non-linearity to capture complex patterns**
  - A hidden layer transforms inputs using weights, biases, and an activation function, allowing the network to learn complex patterns
  - Other examples of nonlinear **activation functions**
    - rectifying nonlinear unit (**relu**) $g(z) = \max(0, z)$

      : cuts off values below zero,
    - hyperbolic tangent (**tanh**) $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

      : saturates to –1 for low input values and +1 for high input values.



Structure of multi-layer perceptron

- **Output Layer: Making predictions for the target task**
  - The output of the last hidden layer $\mathbf{h}$ becomes the input for the output layer $\hat{y} = f^{(l)}(\mathbf{h})$
    - **Linear Units** (for regression, $y \in \mathbb{R}$) : $\hat{y} = \mathbf{w}^T\mathbf{h} + b$
    - **Sigmoid Units** (for binary classification, $y \in \{0,1\}$) :

$$\hat{y} = P(y = 1|\boldsymbol{x}) = \sigma(z) = \sigma(\mathbf{w}^T\mathbf{h} + b)$$

    - **Softmax Units** (for multi-class classification, $y \in \{1,2, \dots, c\}$)

$$\hat{\boldsymbol{y}} = (\hat{y}_1, \dots, \hat{y}_c), \text{ where } \hat{y}_k = p(y = k|\boldsymbol{x})$$

$$\boldsymbol{z} = (z_1, \dots, z_c) = \mathbf{W}^T\mathbf{h} + \boldsymbol{b}$$

$$\hat{y}_k = \mathrm{softmax}(\boldsymbol{z})_k = \frac{\exp(z_k)}{\sum_j \exp(z_j)} \text{ so that } \sum_k \hat{y}_k = 1$$

Learning a Neural Network

Learning a Multi-layer Perceptron

- **Find the optimal parameters $\boldsymbol{\theta}^*$ (which include all weights and biases)**
  - Given a (training) dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}$ such that $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id}) \in \mathbb{R}^d$ is the i-th input vector of d features and y_i is the corresponding target label.
  - The model: $\hat{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$
  - The cost function (to be minimized, usually non-convex)

$$L(\boldsymbol{\theta}) = \sum_{(\boldsymbol{x}_i, y_i) \in D} L_i(y_i, \hat{y}_i)$$

  - For training, any gradient-based optimization algorithms can be used (**backpropagation**).
    - e.g., simple gradient descent $\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$

Learning a Multi-layer Perceptron

- **Typical choice of the loss function $\boldsymbol{L_i(y_i, \hat{y}_i)}$**
  - For **regression** ($y_i \in \mathbb{R}$), use squared error

$$L_i(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

  - For **binary classification** ($y_i \in \{0,1\}$), use binary cross-entropy

$$L_i(y_i, \hat{y}_i) = [-y_i \mathrm{log} \hat{y}_i - (1 - y_i) \mathrm{log}(1 - \hat{y}_i)]$$

  - For **multi-class classification** ($y_i \in \{1,2, \dots, c\}, \boldsymbol{y}_i = \mathrm{one\_hot}(y_i) = (y_{i1}, \dots, y_{ic})$), use categorical

cross-entropy

$$L_i(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i) = -\sum_{k=1}^{c} y_{ik} \log \hat{y}_{ik}$$

Design Issues for MLP

- **Issues in Designing Architecture**
  - **Number of units in input layer**
    - One input unit per binary/continuous attribute
    - k units for each categorical attribute with k values
  - **Number of units in output layer**
    - One output unit for 2-class problem
    - c output units for c-class problem
  - **Number of hidden layers (depth)**
  - **Number of units per hidden layer (width)**

Characteristics of MLP

- **Advantages**
  - Multi-layer neural networks are universal approximators
  - Well-suited for continuous-valued inputs and outputs
  - Can handle redundant and irrelevant attributes because weights are automatically learnt for all attributes
- **Disadvantages**
  - Can suffer from overfitting if the network is too large. (regularization needed)
  - Training is computationally intensive and may converge to local minimum
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network
    - Techniques have recently been developed for the extraction of rules from trained neural networks