InfoLab

Deep Learning Seminar
CH 6. Deep Feedforward Networks

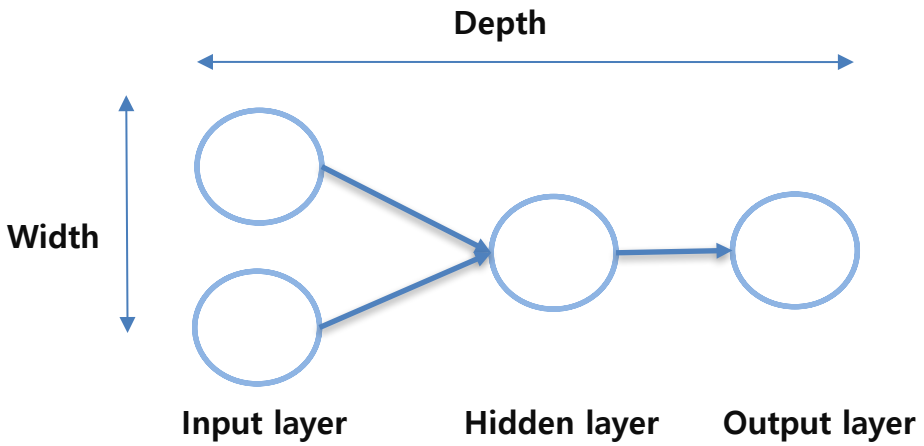
2017-07-26

Eunjeong Yi

Chapter 6. Deep Feedforward Networks

Deep feedforward network

- **No feedback connection**
- **Structure of model**
  - Input layer, hidden layer, output layer
  - The depth, width of model
  - Cost function



Example: Learning XOR

- **XOR function ("exclusive or")**

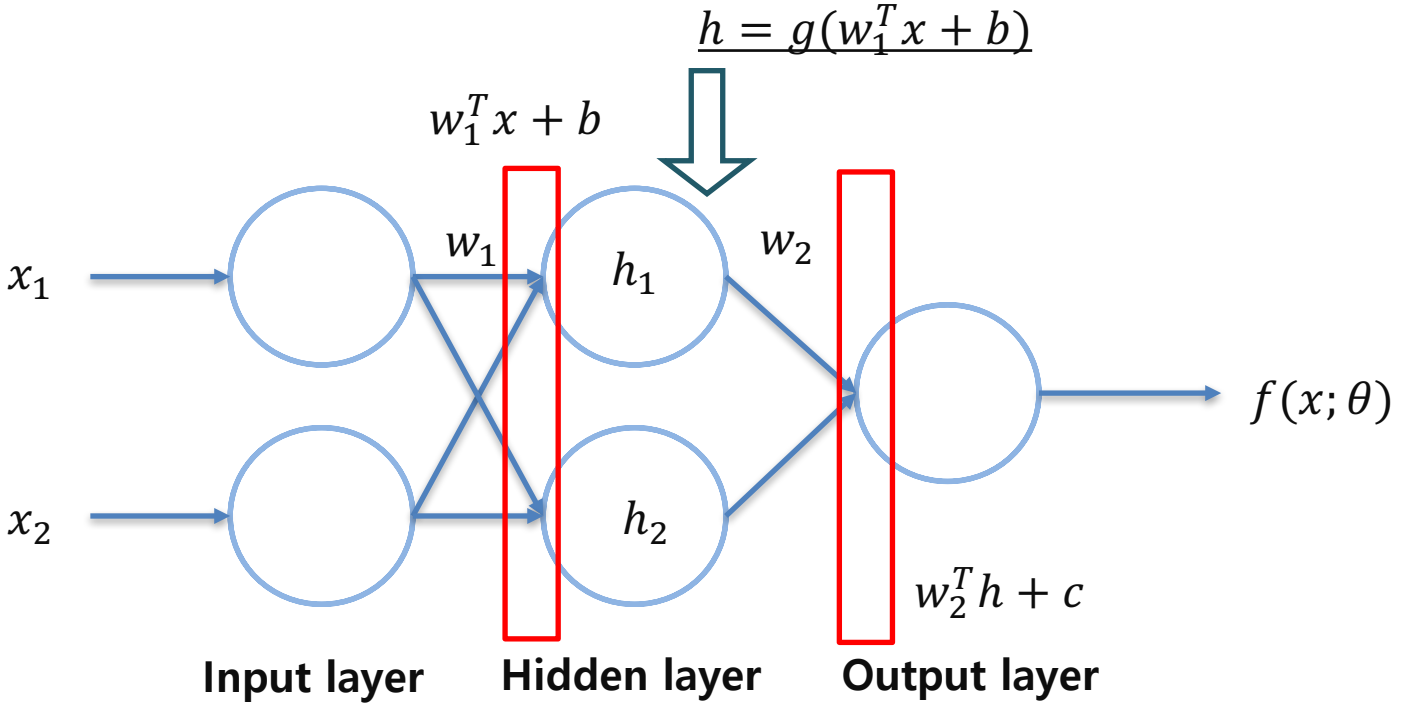
x_1	x_2	$x_1\ XOR\ x_2$
0	0	0
0	1	1
1	0	1
1	1	0

- $\mathbb{X} = \{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$
- **Mean squared error (MSE) loss function**

$$J(\boldsymbol{\theta}) = \frac{1}{4}\sum_{\mathbf{x}\in\mathbb{X}}\left(f^*(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta})\right)^2$$

- $f^*(x)$: $correct\ answer$
- $f(x;\theta)$: $calculated\ result\ by\ neural\ network$

Example: Learning XOR



(g = activation function, b, c = bias)

$$f(x;\theta) = f(x;w_1,b,w_2,c) = w_2^T g(w_1^T x + b) + c$$

Gradient-based Learning

- **Loss function of neural network is non-convex**
  - Trained by using iterative, gradient-based optimizers
- **Cost function**
  - Learning Conditional Distribution
  - Learning Conditional Statistics
- **Output layers**
  - Linear Units for Gaussian Output Distributions
  - Sigmoid Units
  - Softmax Units

Learning Conditional Distribution

- **Negative log-likelihood as cross-entropy between training data and model distribution**

$$J(\boldsymbol{\theta}) = H\left(\hat{p}_{data}, p_{model}(y|x)\right) = -\sum_{x,y\sim\hat{p}_{data}} \hat{p}_{data} \log p_{model}(y|x)$$

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{x,y\sim\hat{p}_{data}} \log p_{model}(y|x)$$

  - $\hat{p}_{data}$: $training\ data - generating\ distribution$
  - $p_{model}(y|x)$: $probability\ distribution\ estimating\ \hat{p}_{data}$
  - $H\left(\hat{p}_{data}, p_{model}(y|x)\right)$: $cross\ entropy\ between\ \hat{p}_{data}\ and\ p_{model}$

- ***log*** **function undoes the exp of some output units**

Learning Conditional Statistics

- **Mean Square Error (MSE)**

$$f^* = \arg\min_f \mathbb{E}_{x,y\sim p_{data}} ||y - f(x)||^2$$

- **Mean Absolute Error (MAE)**

$$f^* = \arg\min_f \mathbb{E}_{x,y\sim p_{data}} ||y - f(x)||_1$$

- **MSE, MAE often lead to poor results when used with gradient-based learning**

Linear Units for Gaussian Output Distributions

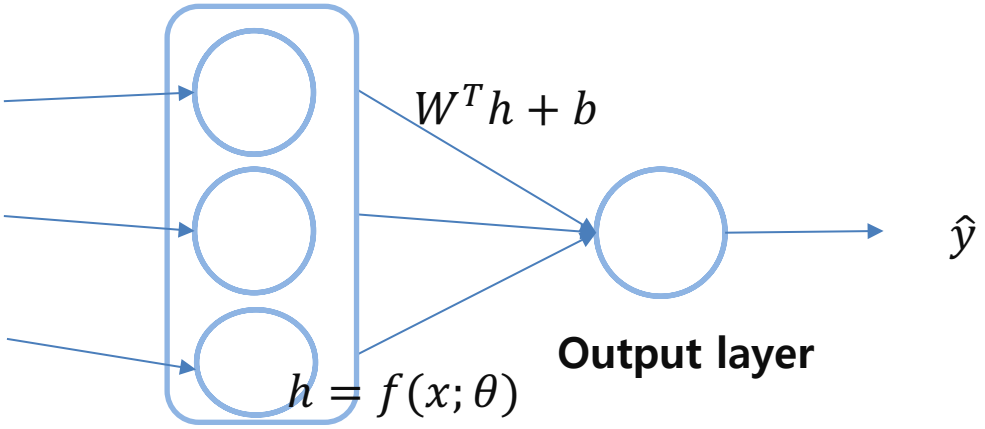
- **Feature *h*, produce a vector $\hat{y}$**

$$\hat{y} = W^T h + b$$

- **To produce the mean of a conditional Gaussian distribution**

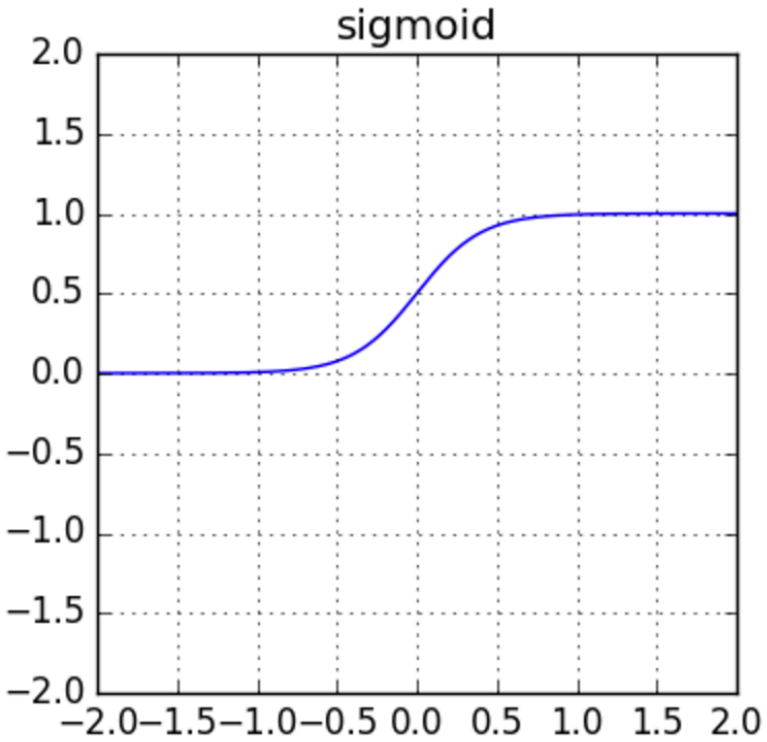
$$p(y|x) = \mathcal{N}(y; \hat{y}, I)$$

- **No saturation → little difficult to gradient-based optimization**



Sigmoid Units

- **Binary classification**
- **Output:** $\hat{y} = \sigma(w^T h + b)$
  - $\sigma(z) = \frac{1}{1+e^{-z}}$
- **Saturate to 0 and 1**



http://neat-python.readthedocs.io/en/latest/activation.html

Softmax Units

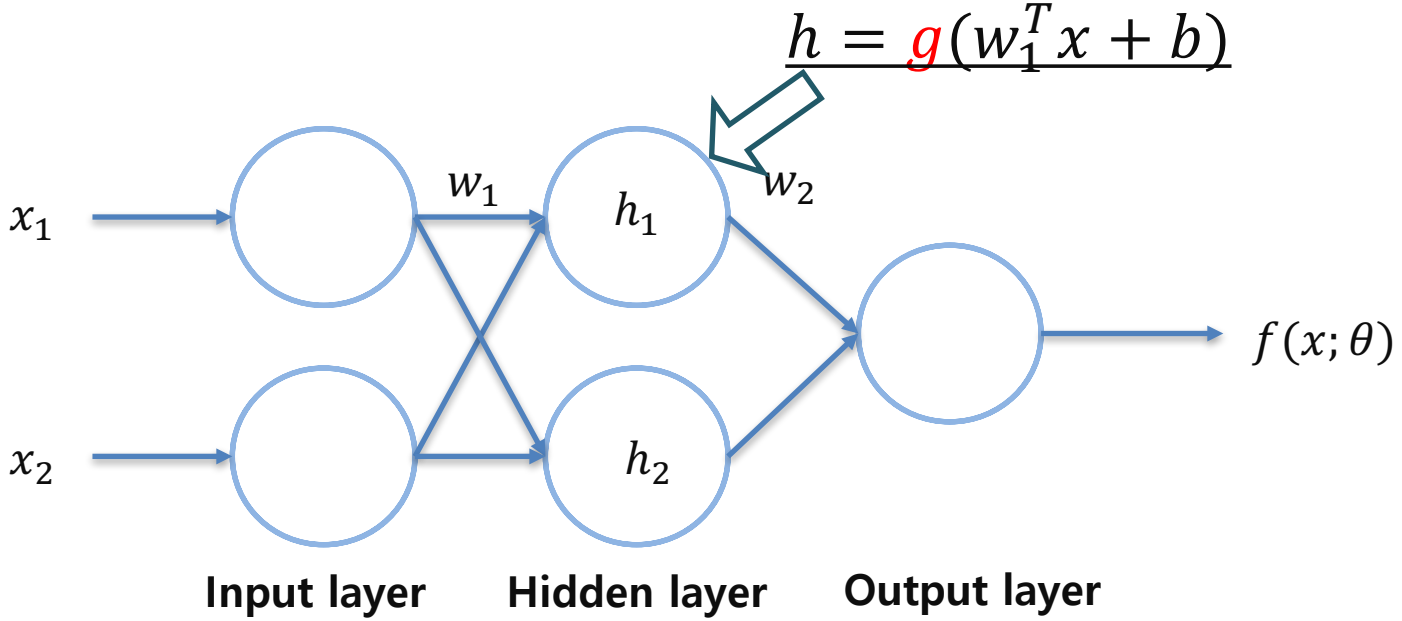
- **Multiclass classification**
- **To generalize to the case of a discrete variable with n values, vector $\hat{y}, with\ \hat{y}_i = P(y = i|x)$**

$$z = W^T h + b \left(z_i = log\ \tilde{P}(y = i|x)\right)$$

$$softmax(z)_i = \frac{exp(z_i)}{\Sigma_j exp(z_j)}$$

Hidden Units

- How to choose the type of hidden unit to use in the hidden layers of the model
- Input: $z = W^T x + b$
- Activation function $g(z)$
- Ex) Rectified Linear Units (ReLU), Logistic Sigmoid and Hyperbolic Tangent



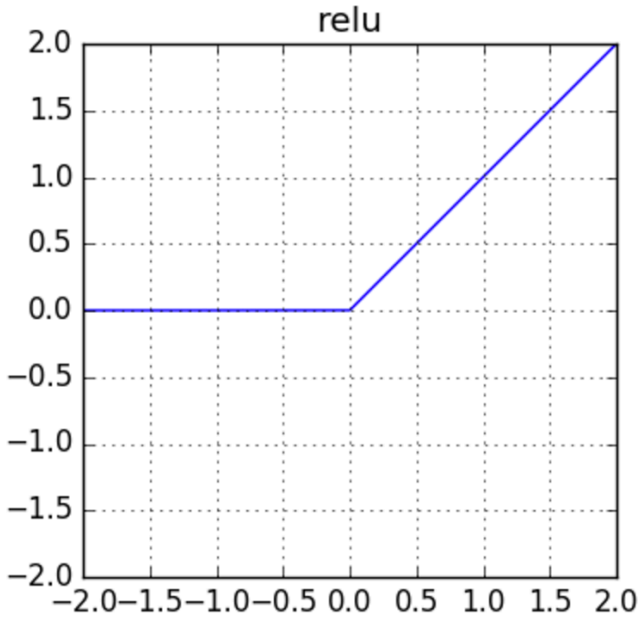
(g = activation function, b, c = bias)

Rectified Linear Units (ReLU)

- **Activation function $g(z)$**

$$g(z) = \max(0, z)$$

- **If model's behavior is closer to linear, models are easier to optimize**



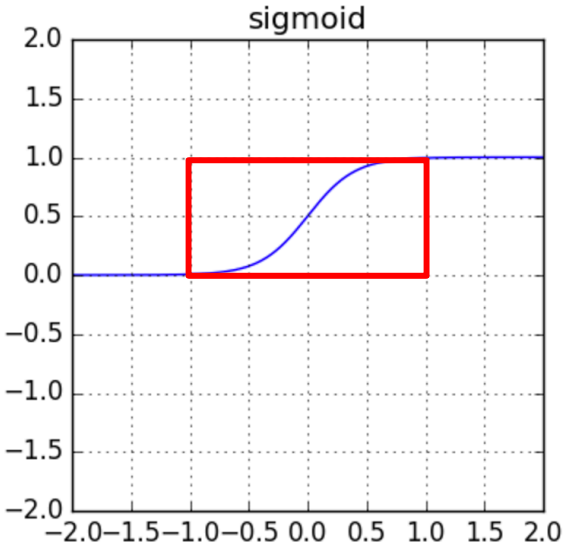
http://neat-python.readthedocs.io/en/latest/activation.html

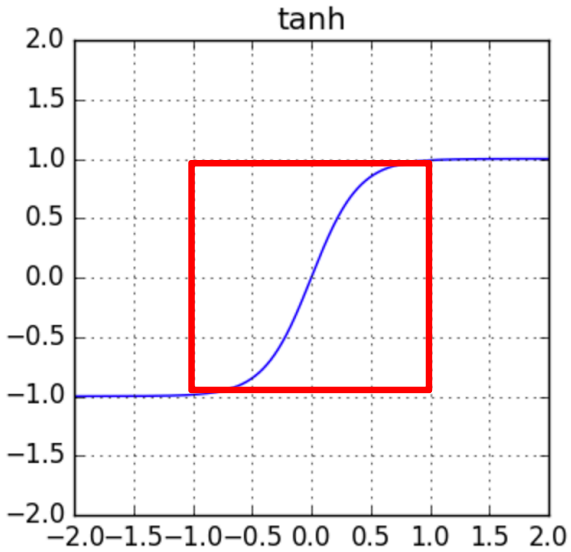
Logistic Sigmoid and Hyperbolic Tangent

- **Activation function**

$$g(z) = \sigma(z)$$
$$or$$
$$g(z) = tanh(z) = 2\sigma(2z) - 1$$

- **Saturation of sigmoidal units make gradient-based learning difficult**

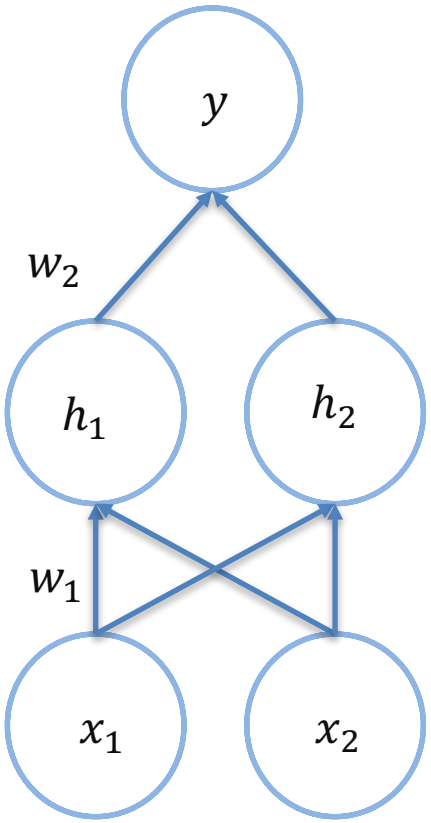




http://neat-python.readthedocs.io/en/latest/activation.html

Back-Propagation

- **Method to calculate the gradient of the loss function with respect to the weights in an artificial neural network**



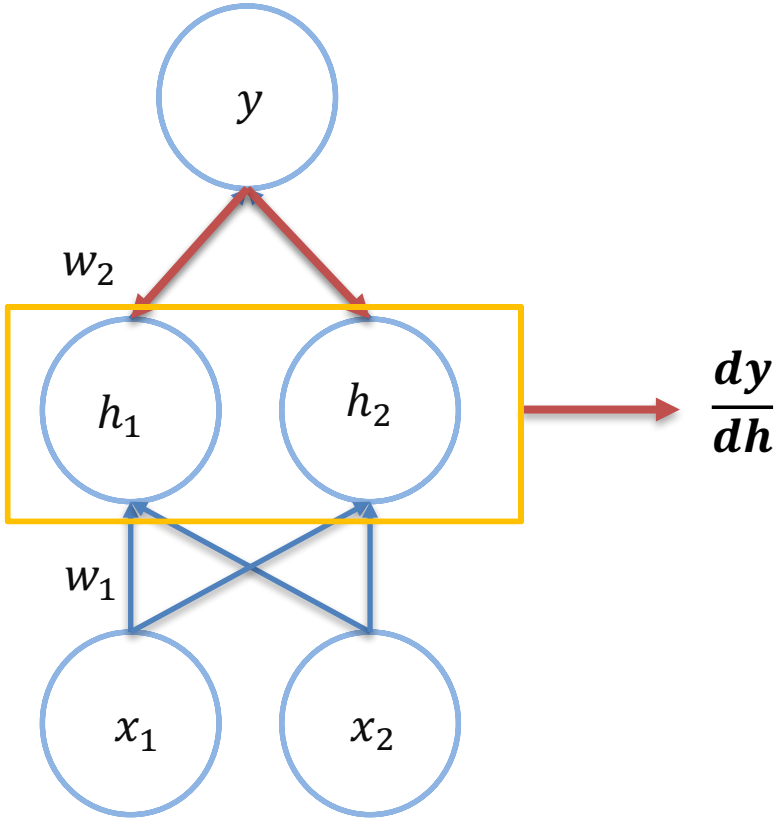
Forward propagation result

$$\boldsymbol{y = w_2^T h + b = w_2^T g\left(w_1^T x + c\right) + b}$$

(g = activation function, b, c = bias)

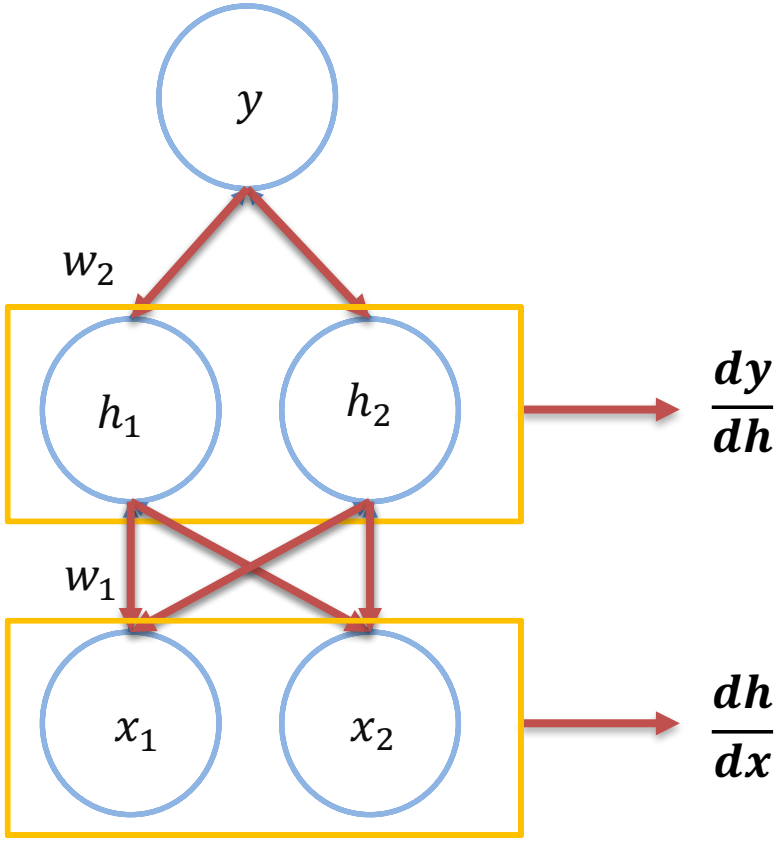
Back-Propagation

- **Method to calculate the gradient of the loss function with respect to the weights in an artificial neural network**



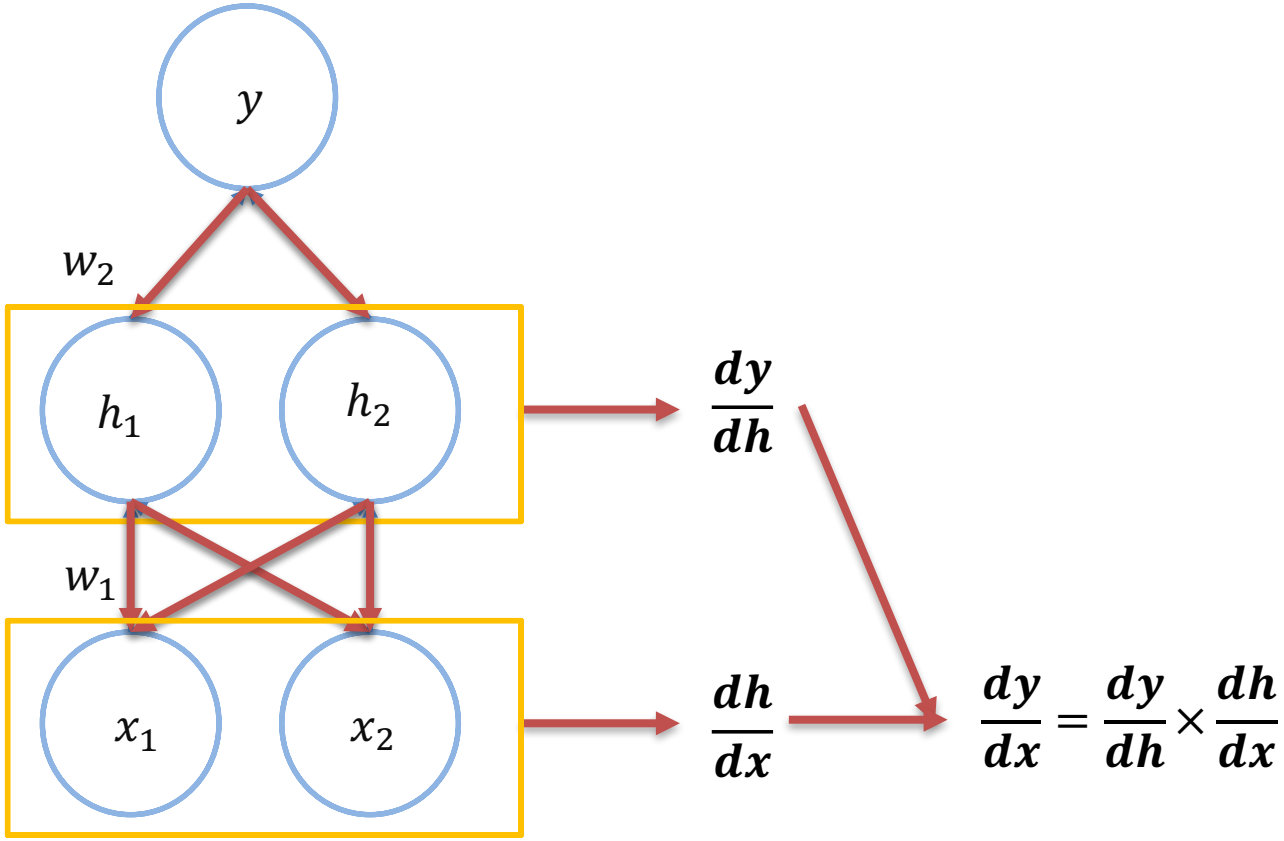
Back-Propagation

- **Method to calculate the gradient of the loss function with respect to the weights in an artificial neural network**

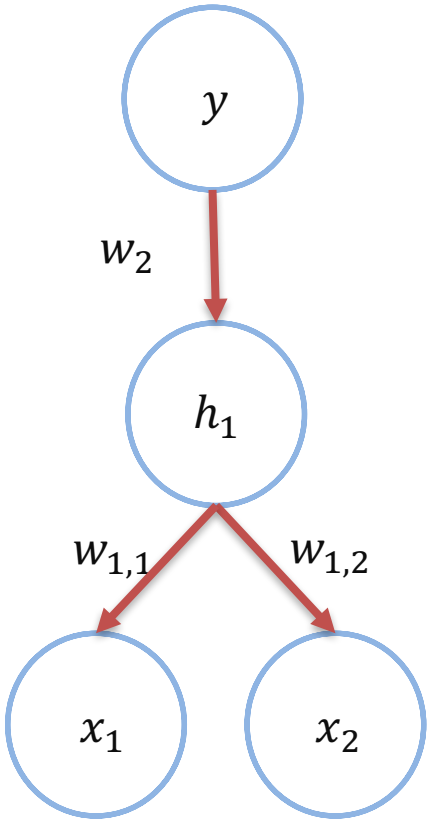


Back-Propagation

- **Method to calculate the gradient of the loss function with respect to the weights in an artificial neural network**



Back-Propagation



$$\frac{\delta y}{\delta w_{1,1}} = \frac{\delta y}{\delta x_1} \times \frac{\delta x_1}{\delta h_1} \times \frac{\delta h_1}{\delta w_{1,1}}$$

Update $w_{1,1}$

$$w_{1,1} = w_{1,1} - \eta \frac{\delta y}{\delta w_{1,1}}$$

(η: learning weight)

Using same way,
all weight update

Chapter 7. Regularization for Deep Learning

Q&A

Thank you