

Lecture II: Artificial Neural Networks

Pilsung Kang

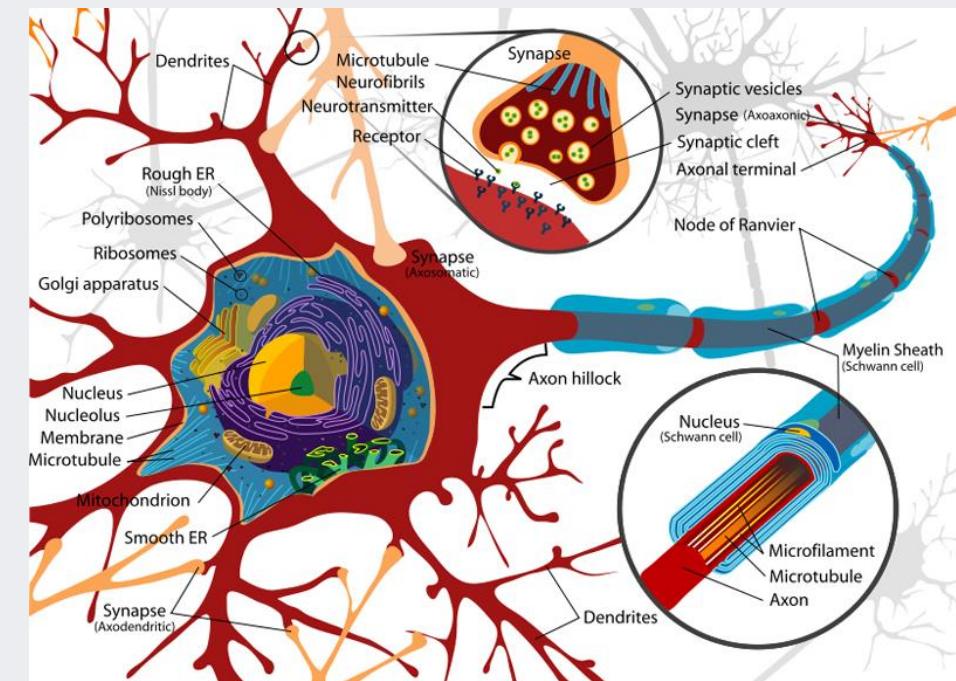
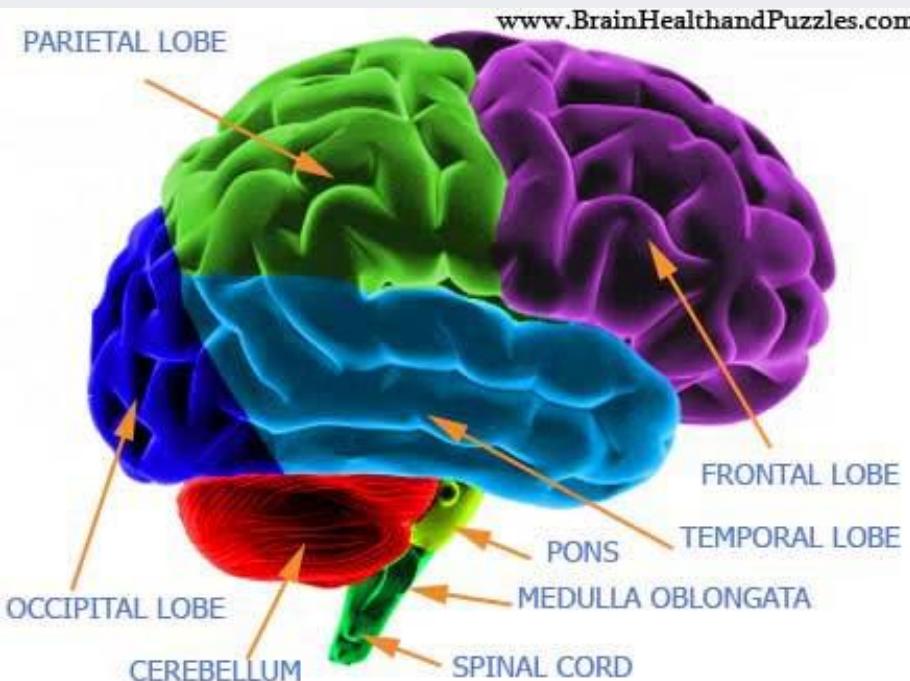
School of Industrial Management Engineering
Korea University

AGENDA

- 01 Artificial Neural Networks: Perceptron
- 02 Multi-layer Perceptron (MLP)
- 03 R Exercise

Brain Structure

- How our brain works...
 - ✓ Neurons transmit and analyze communication within the brain and other parts of the nervous system
 - ✓ A message within the brain is converted to electronic signs



Neuron Firing Off in Real-Time

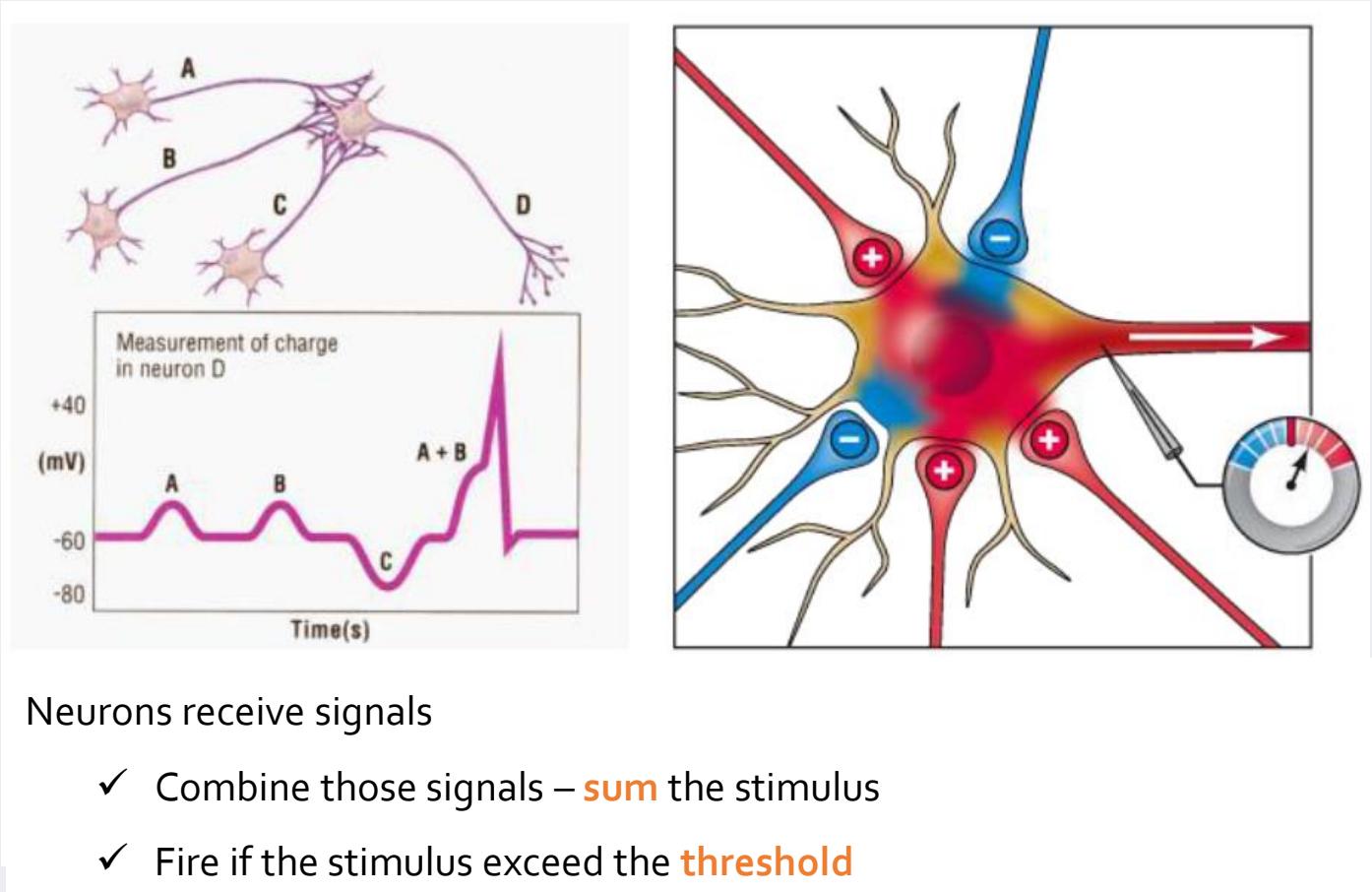


EEG powered by BCILAB | SIFT

<http://www.dailymail.co.uk/sciencetech/article-2581184/The-dynamic-mind-Stunning-3D-glass-brain-shows-neurons-firing-real-time.html>

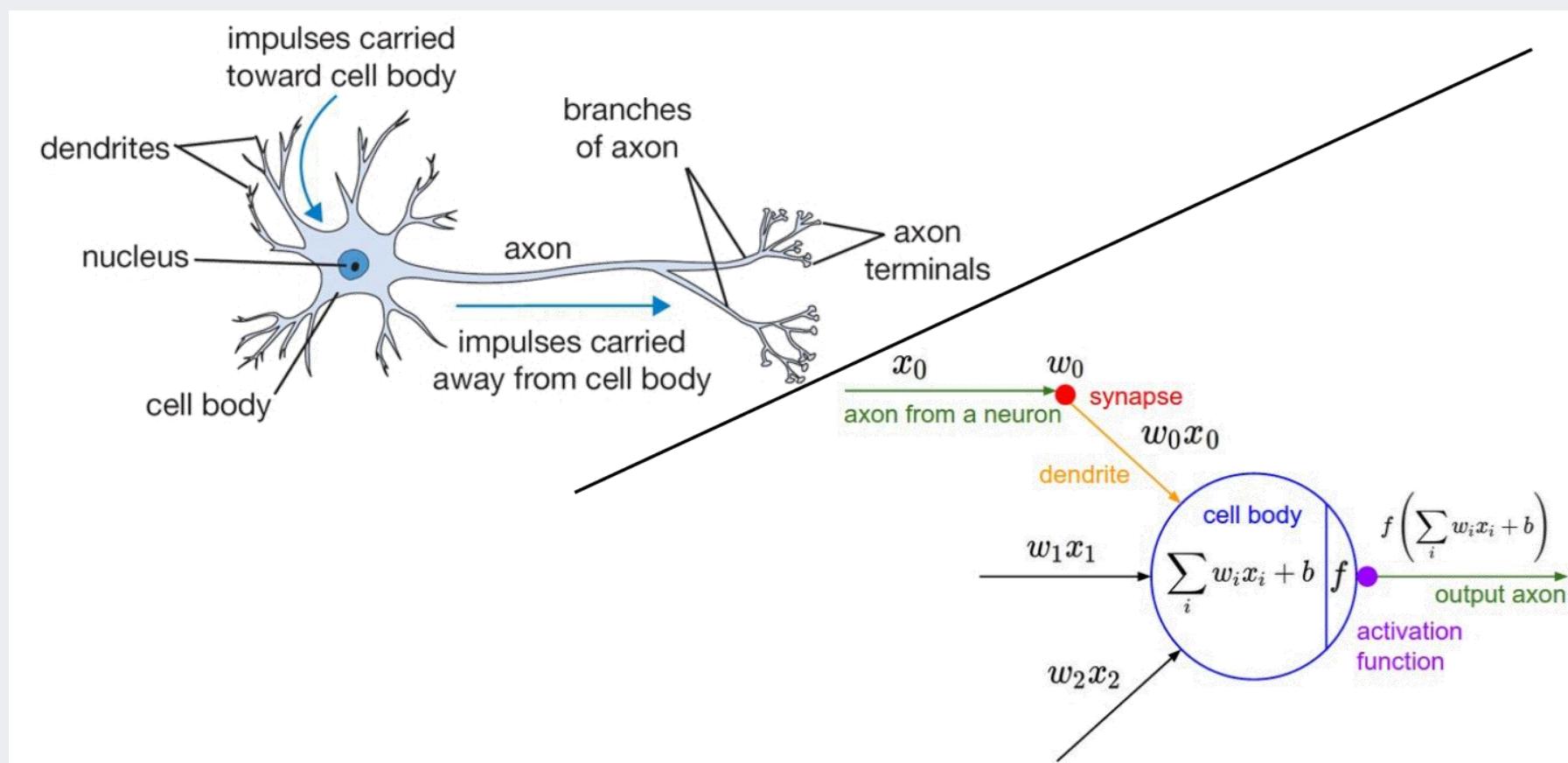
The Way Our Brain Works

- The way neurons work



Perceptron

- Imitate a single neuron

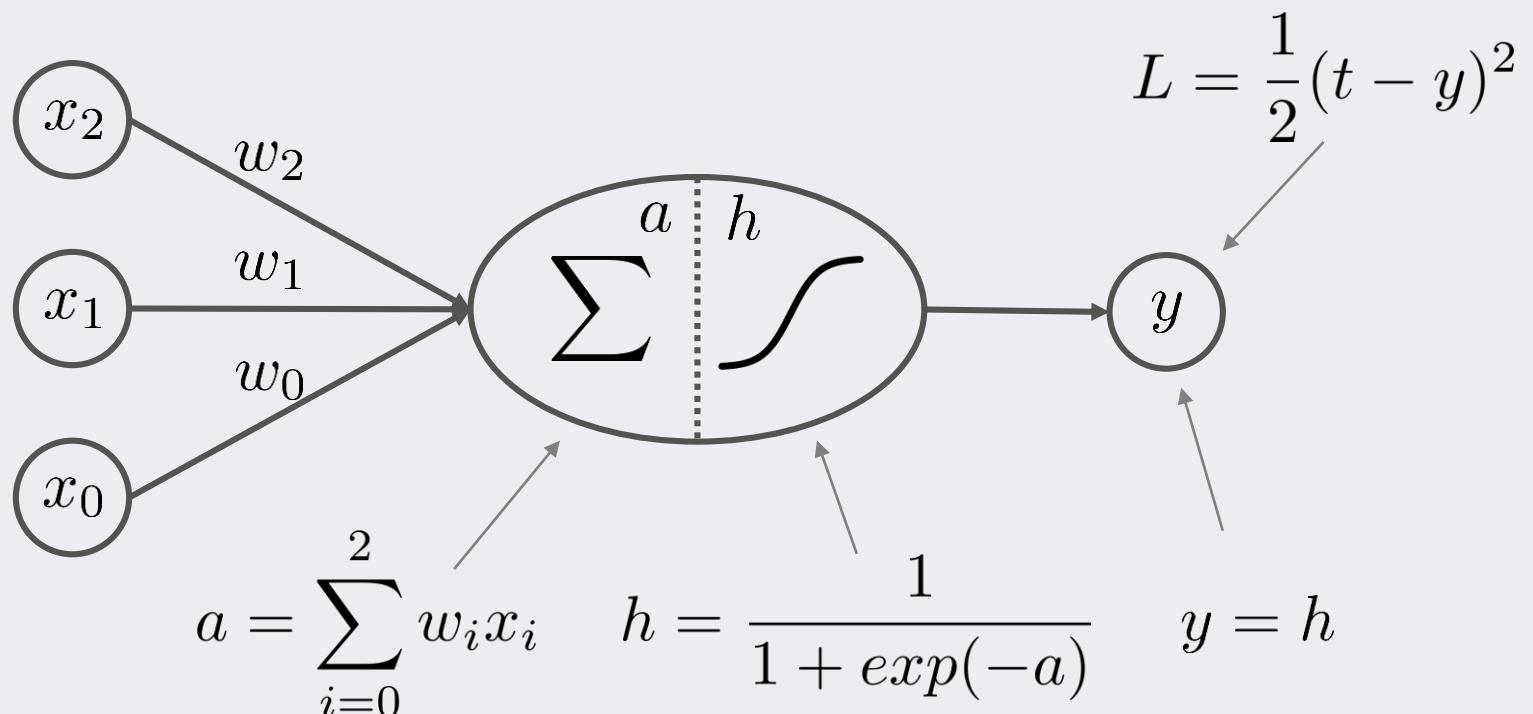


Perceptron

- Perceptron

- ✓ An organism with only 1 neuron

원하는 값(t)과
예측값(y)의 차이를
손실함수로 정의

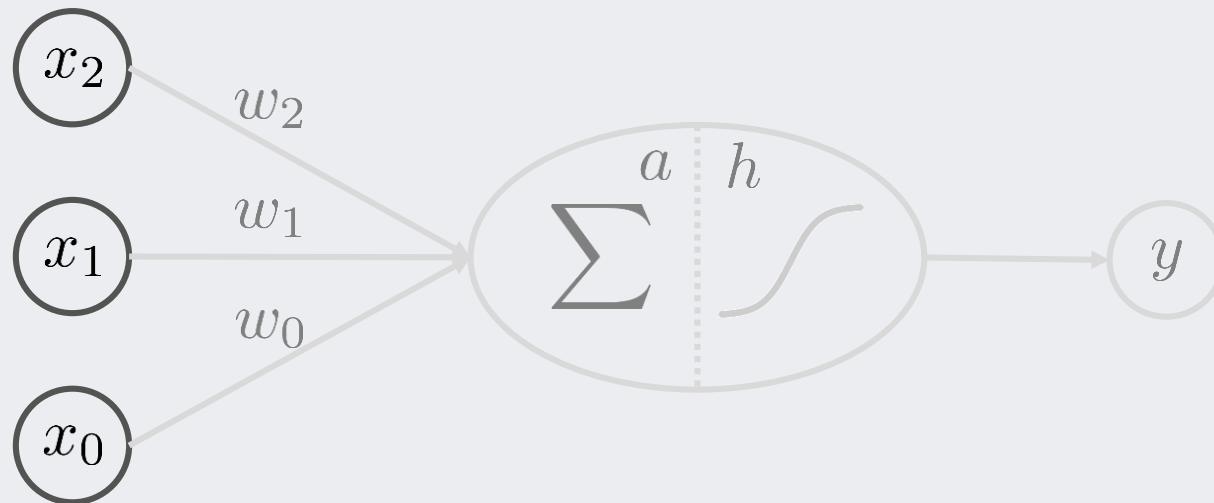


여러 변수들의 정보를
나름대로 취합해서

얼마만큼 다음 단계로
전달할지 결정한다
(활성화)

Perceptron

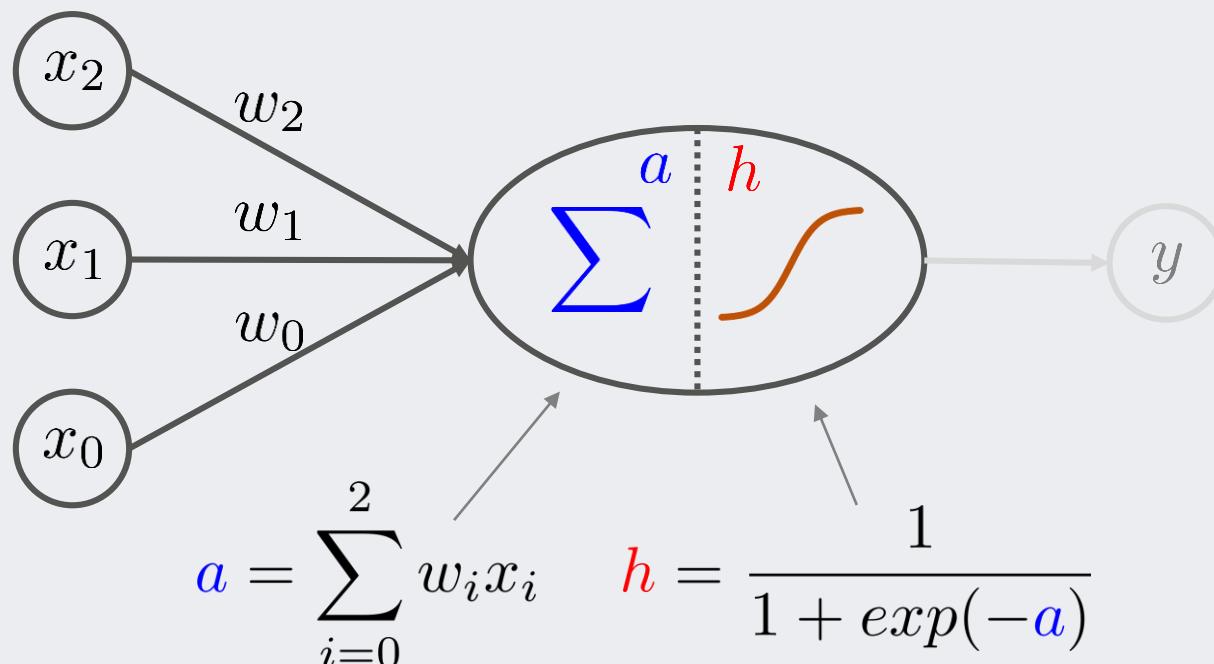
- Input node
 - ✓ Input (predictor, explanatory) variables



Perceptron

- Hidden node

✓ Take the weighted sum of input values and perform a non-linear activation



여러 변수들의 정보를
나름대로 취합해서

얼마만큼 다음 단계로
전달할지 결정한다
(활성화)

Perceptron

- Role of activation

- ✓ Determine how much information from the previous layer is forward to the next layer

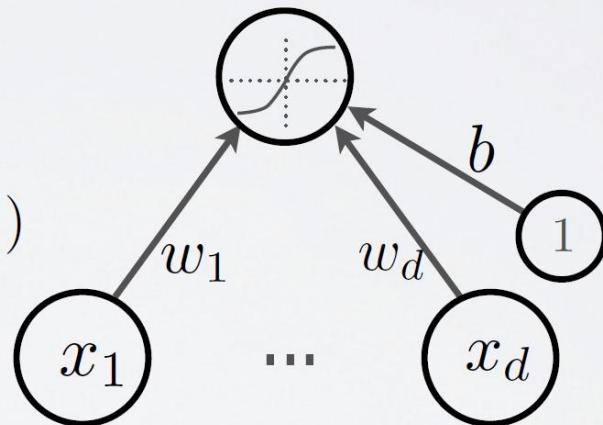
- Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron (output) activation

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

- \mathbf{w} are the connection weights
- b is the neuron bias
- $g(\cdot)$ is called the activation function

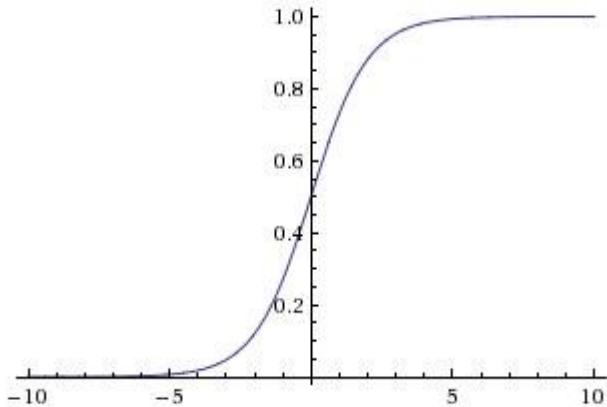


Perceptron

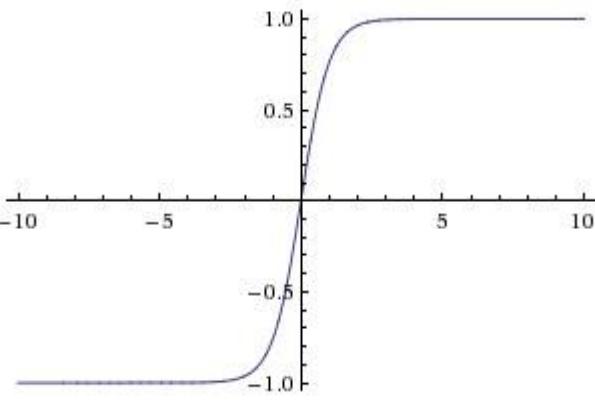
- Representative activation functions

- ✓ Sigmoid: the most commonly used activation, $[0, 1]$ range, learning speed is relatively slow
- ✓ Tanh: Similar to sigmoid but $[-1, 1]$ range, learning speed is relatively fast
- ✓ ReLU (Rectified linear unit): very fast learning speed, easy to compute (without exponential function)

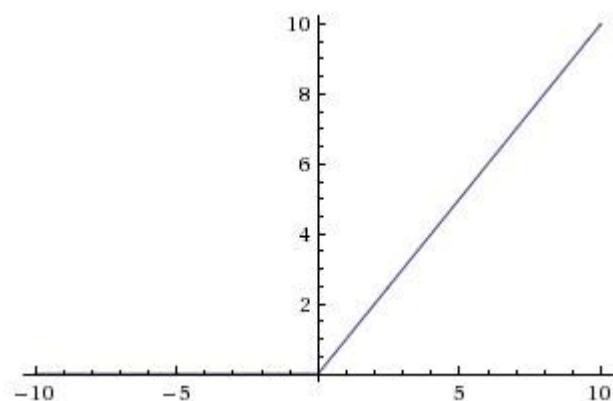
[Sigmoid]



[Tanh]



[ReLU]



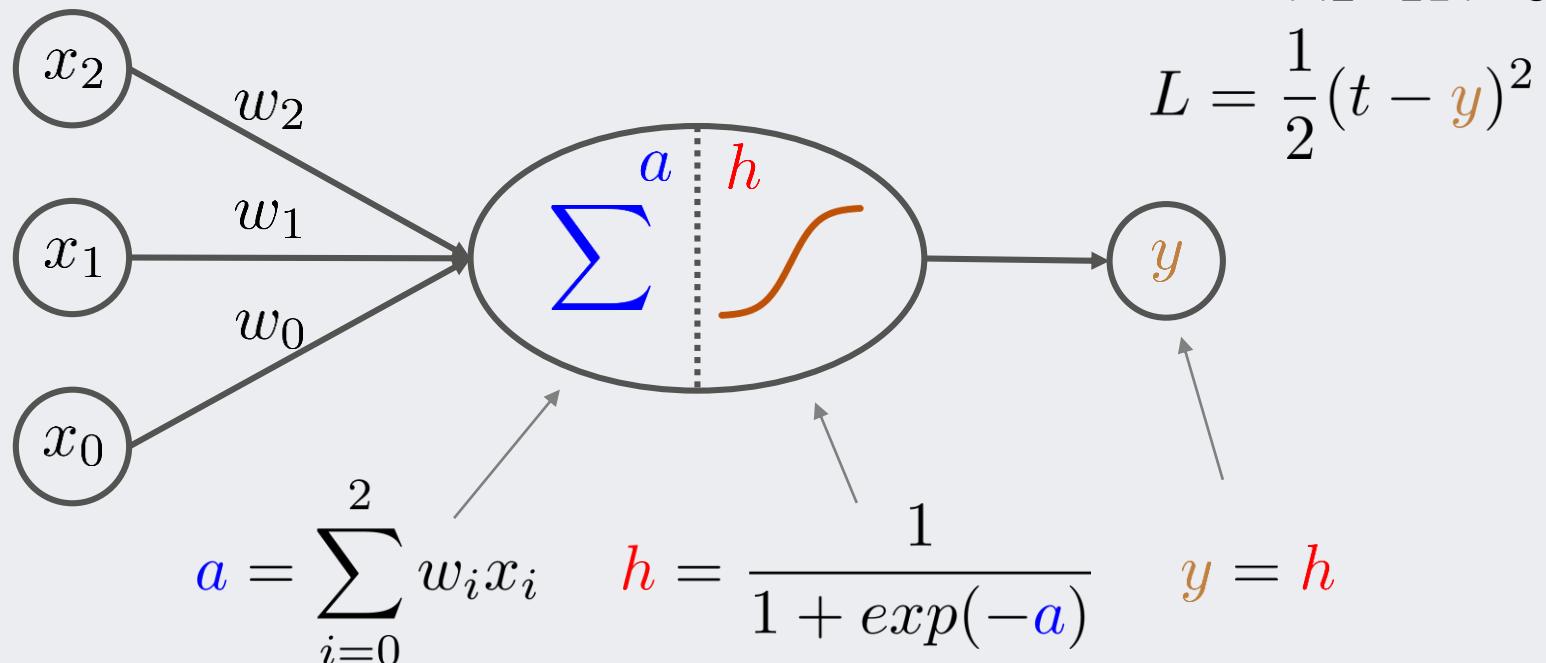
$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)} \quad g(a) = \tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} = \frac{\exp(2a)-1}{\exp(2a)+1} \quad g(a) = \text{reclin}(a) = \max(0, a)$$

Perceptron

- Output node

- ✓ Take the value from the hidden node (Perceptron has only one hidden node)
- ✓ It takes a weighted sum of hidden nodes in a multi-layer perceptron

원하는 값(t)과 예측값(y)의 차이를 손실함수로 정의



여러 변수들의 정보를
나름대로 취합해서

얼마만큼 다음 단계로
전달할지 결정한다
(활성화)

Perceptron

- Purpose of perceptron

- Purpose of perceptron
 - ✓ Find the weight w that can best match the input (x) and the target (t)

- How do we know that the relationship is accurately found?

- How do we know that the relationship is accurately found?
 - ✓ Use a loss function (how the output y is close to the target t)

- Regression: squared loss is commonly used

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

- Classification: cross-entropy is usually used (only binary classification is possible with perceptron)

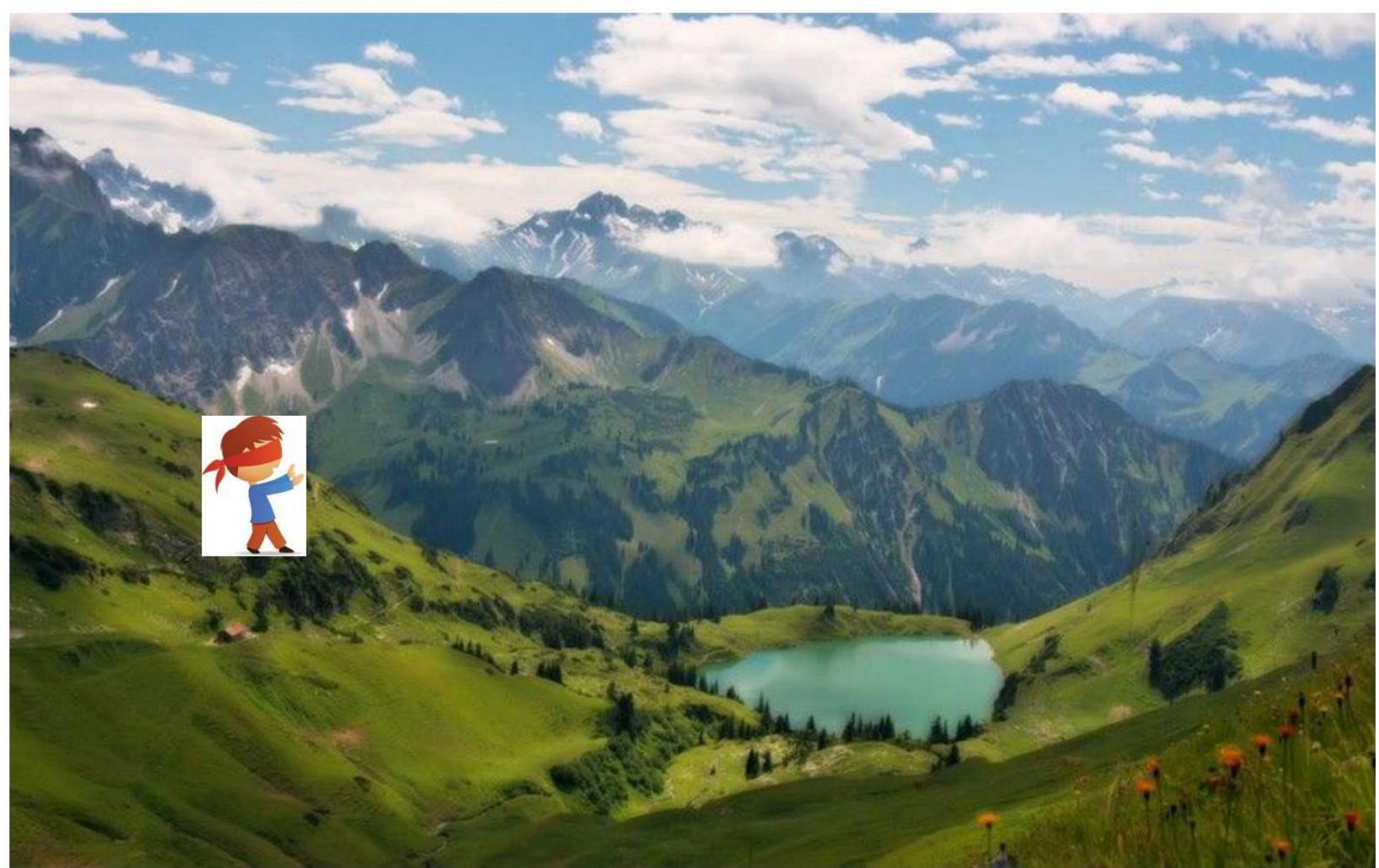
$$L = \sum_{i=1}^2 t_i \log p_i$$

- ✓ Cost function

- Cost function
 - ✓ Computes how inaccurate the current model is (the average of loss function values is commonly used)

Learning: Gradient Descent

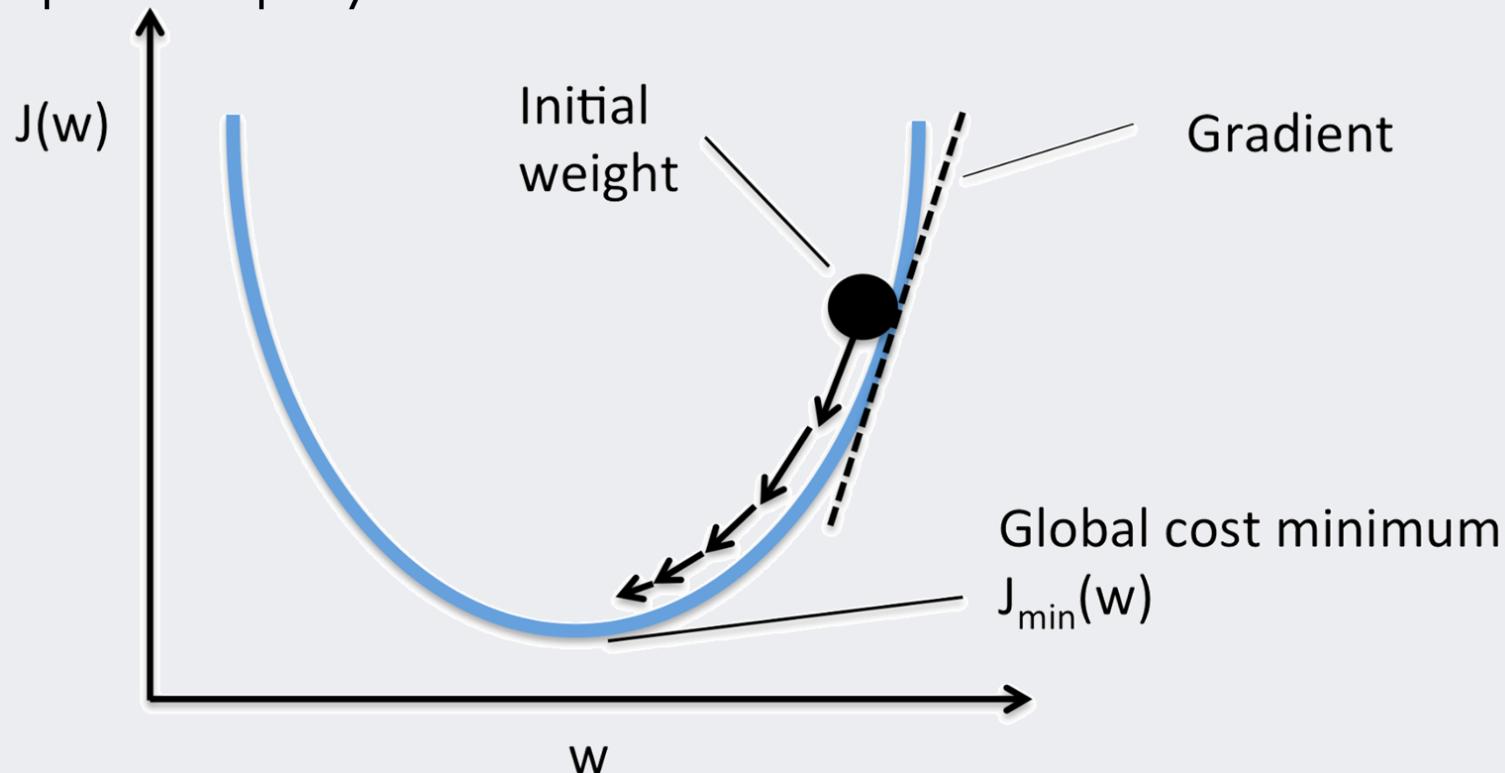
- Gradient Descent



Learning: Gradient Descent

- Gradient Descent Algorithm

- ✓ Blue line: the objective function to be minimized
- ✓ Black circle: the current solution
- ✓ Direction of the arrows: the direction that the current solution should move to improve the quality of solution



Learning: Gradient Descent

- Take the first derivative of the cost function w.r.t the current weight w
 - ✓ Is the gradient 0?
 - Yes: Current weights are the optimum! → end of learning
 - No: Current weights can be improved → learn more
 - ✓ How can we improve the current weights if the gradient is not 0?
 - Move the current weight toward to the opposite direction of the gradient
 - ✓ How much should the weights be moved?
 - Not sure
 - Move them a little and compute the gradient again
 - It will converge



Learning: Gradient Descent

- Theoretical Background

- ✓ Taylor expansion

$$f(x + \Delta x) = f(x) + \frac{f'(x)}{1!} \Delta x + \frac{f''(x)}{2!} \Delta x^2 + \dots$$

- ✓ If the first derivative is not zero, we can decrease the function value by moving x toward the opposite direction of its first derivative

$$x_{new} = x_{old} - \eta f'(x), \text{ where } 0 < \eta < 1$$

- ✓ Then the function value of the new x is always smaller than that of the old x

$$f(x_{new}) = f(x_{old} - \eta f'(x)) \cong f(x_{old}) - \eta |f'(x)|^2 < f(x_{old})$$



<https://www.youtube.com/watch?v=3d6DsjlBzJ4&t=322s>

Learning: Gradient Descent

- Use chain rule

$$\frac{\partial L}{\partial y} = y - t$$

$$\frac{\partial y}{\partial h} = \frac{exp(-h)}{(1 + exp(-h))^2} = \frac{1}{1 + exp(-h)} \cdot \frac{exp(-h)}{1 + exp(-h)} = y(1 - y)$$

$$\frac{\partial h}{\partial w_i} = x_i \quad \frac{\partial h}{\partial x_i} = w_i$$

- Gradients for w and x

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial w_i} = (y - t) \cdot y(1 - y) \cdot x_i$$

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial x_i} = (y - t) \cdot y(1 - y) \cdot w_i$$

Learning: Gradient Descent

- Weight update by Gradient Descent

$$w_i^{new} = w_i^{old} - \alpha \times (y - t) \cdot 1 \cdot h(1 - h) \cdot x_i$$

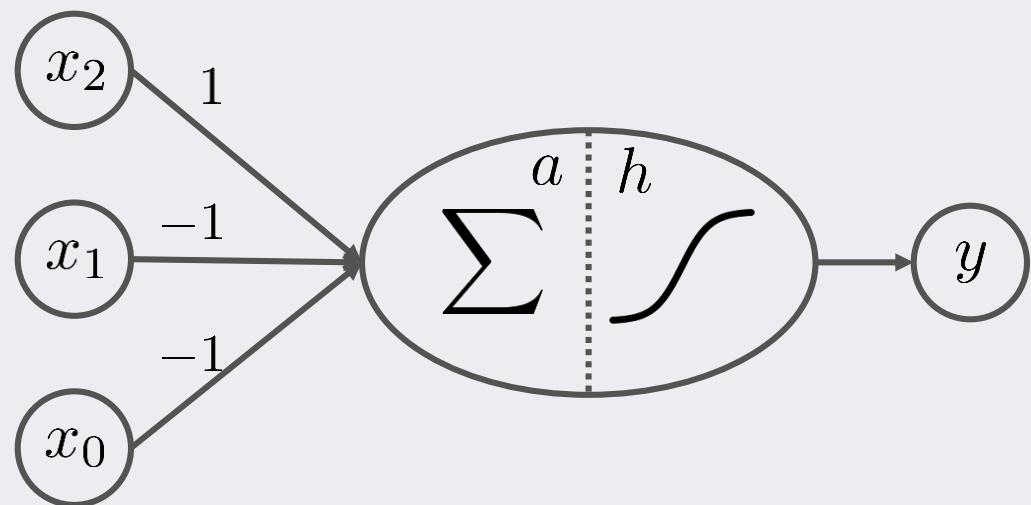
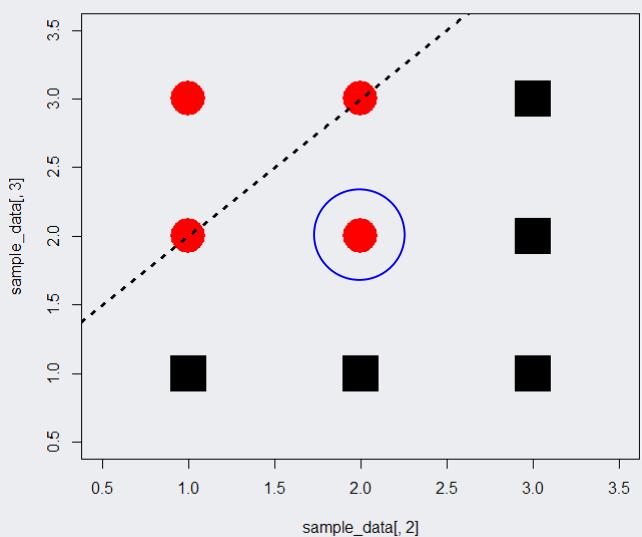


현재의 출력값(y)과 정답(t)이 차이가 많이 날 수록
가중치를 많이 업데이트 하라

대상 가중치와 연결된 입력
변수의 값이 클 수록
가중치를 많이 업데이트 하라

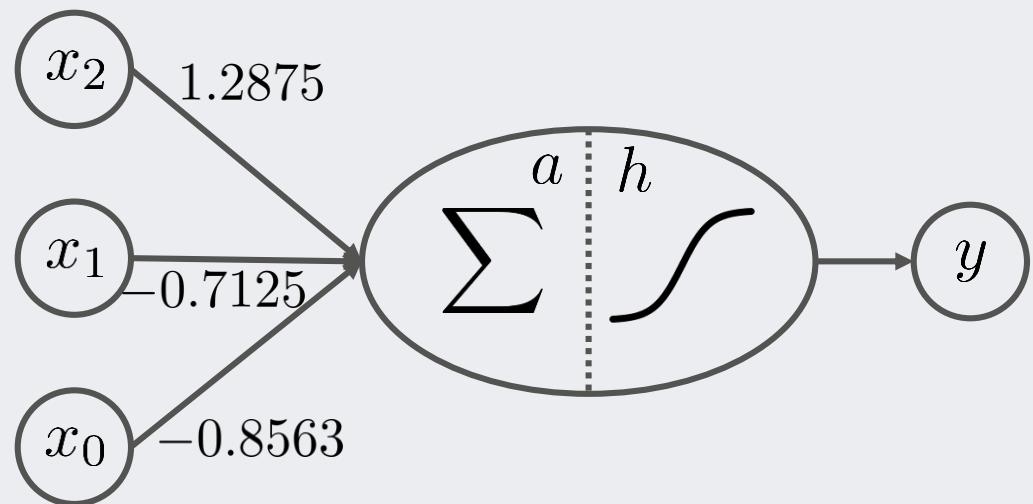
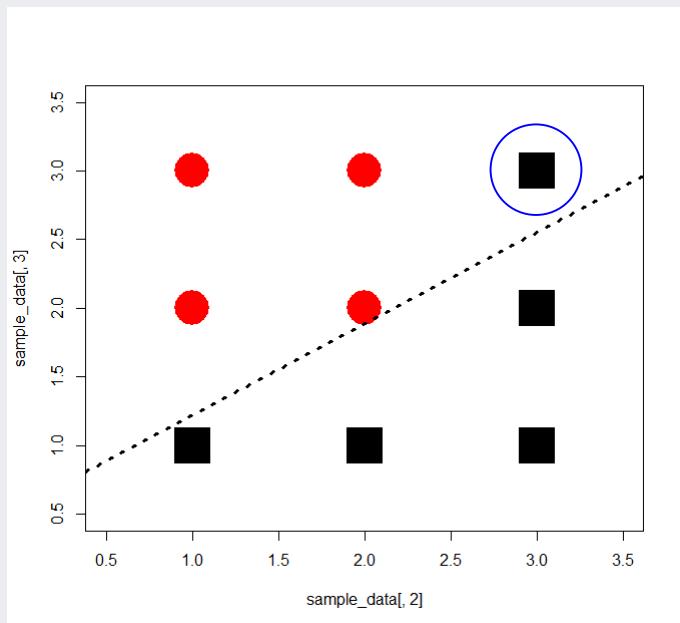
Training Perceptron: Example 1 ($\alpha = 1$)

- Initialize and select the first training example ($x_1=2, x_2=2, t=1$)



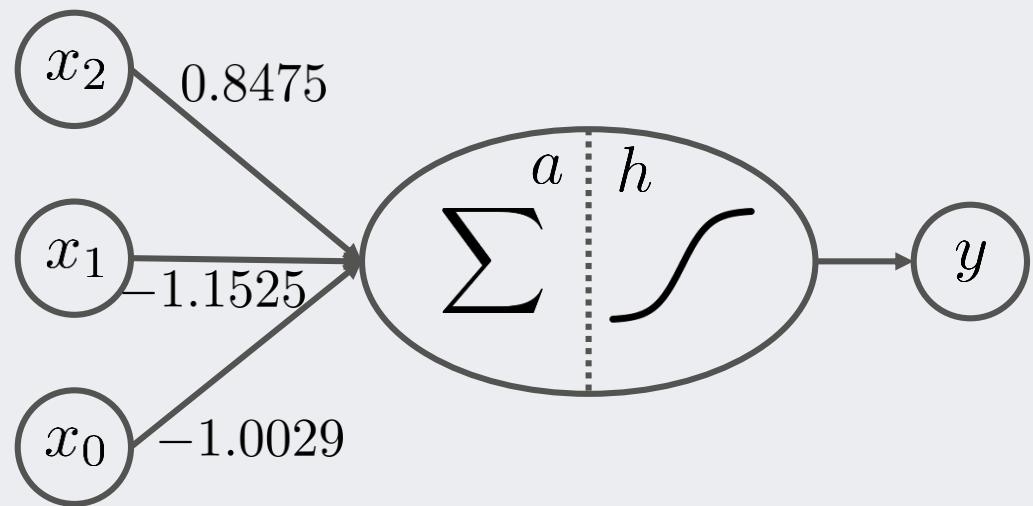
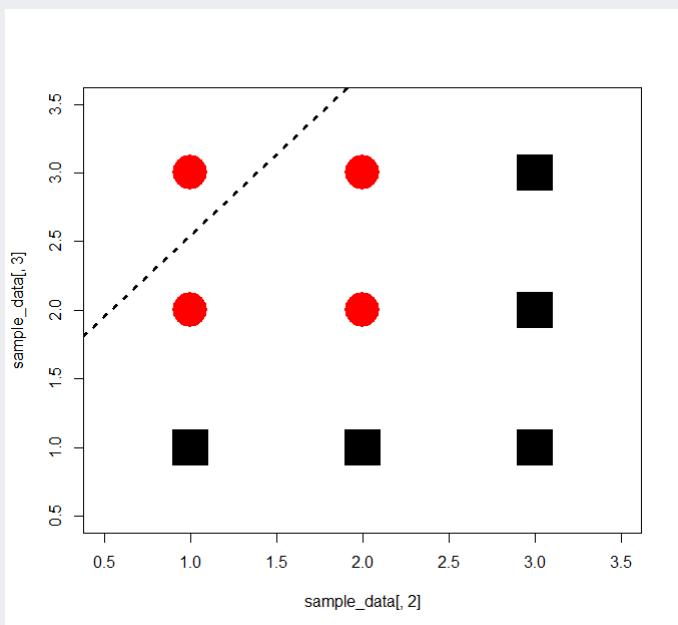
Training Perceptron: Example 1 ($\alpha = 1$)

- Training result and selection of the second training example ($x_1=3, x_2=3, t=0$)



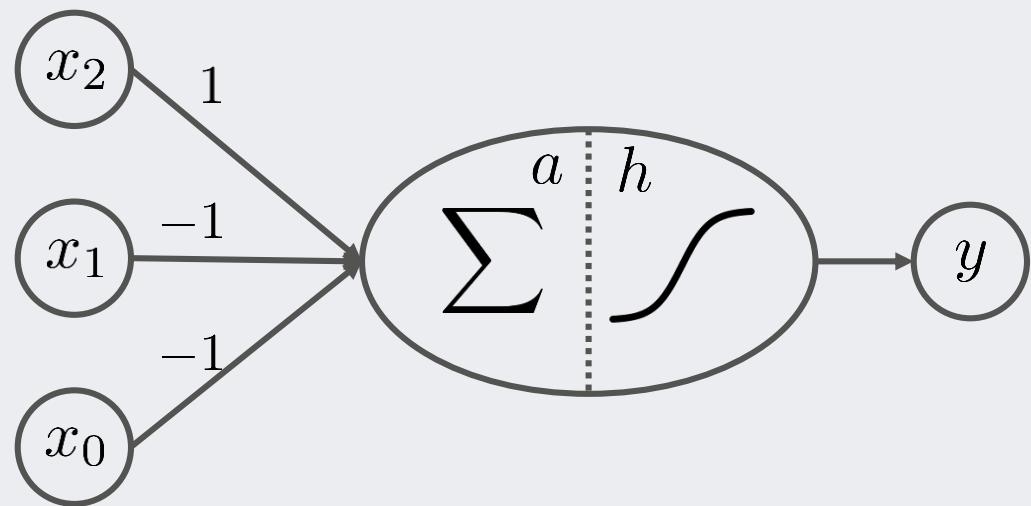
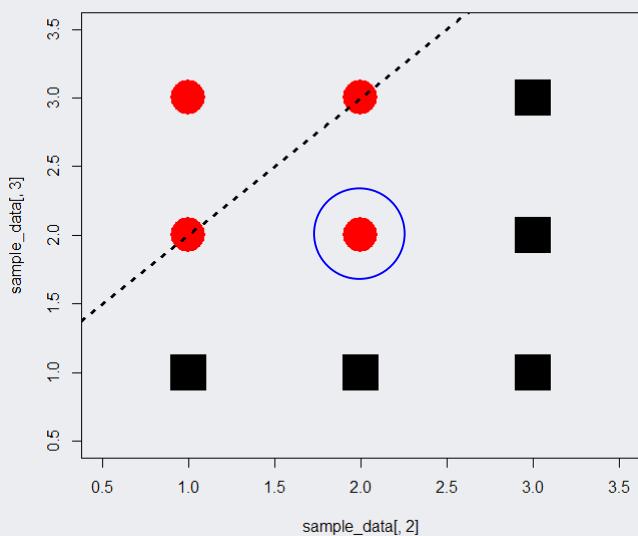
Training Perceptron: Example I (alpha = 1)

- Training result



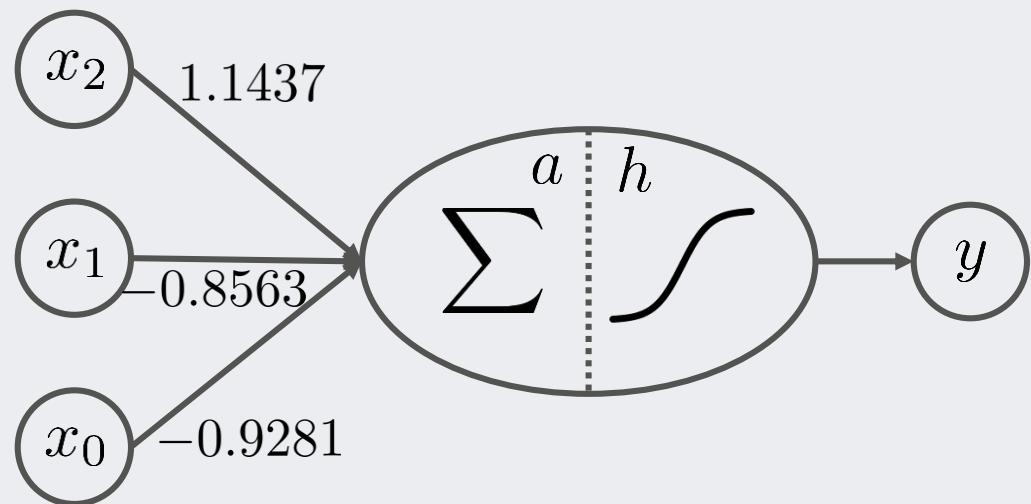
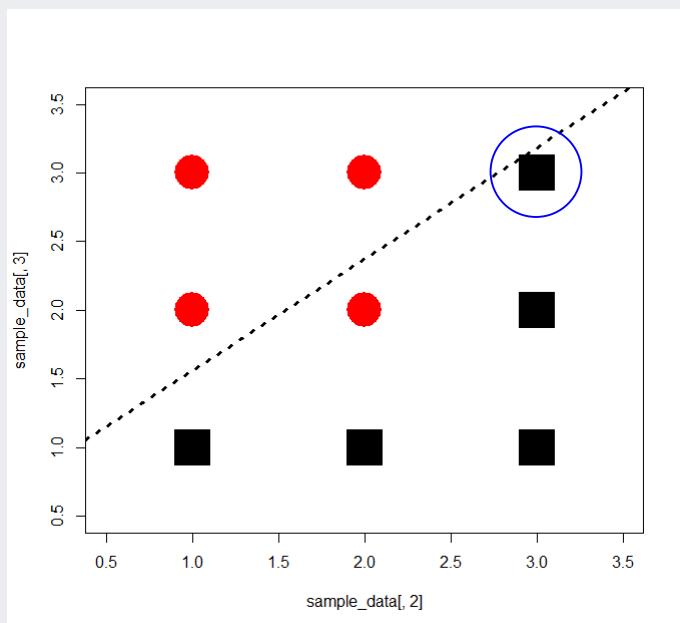
Training Perceptron: Example 1 ($\alpha = 0.5$)

- Initialize and select the first training example ($x_1=2, x_2=2, t=1$)



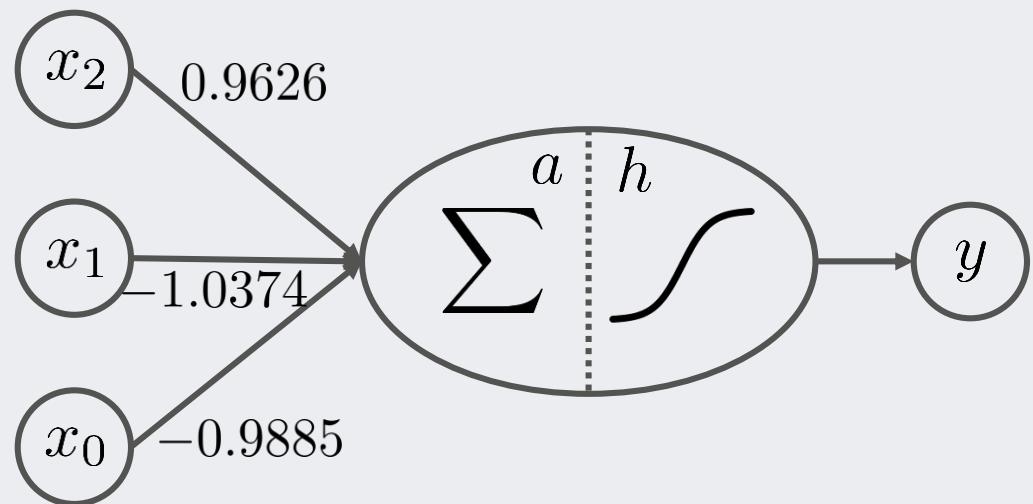
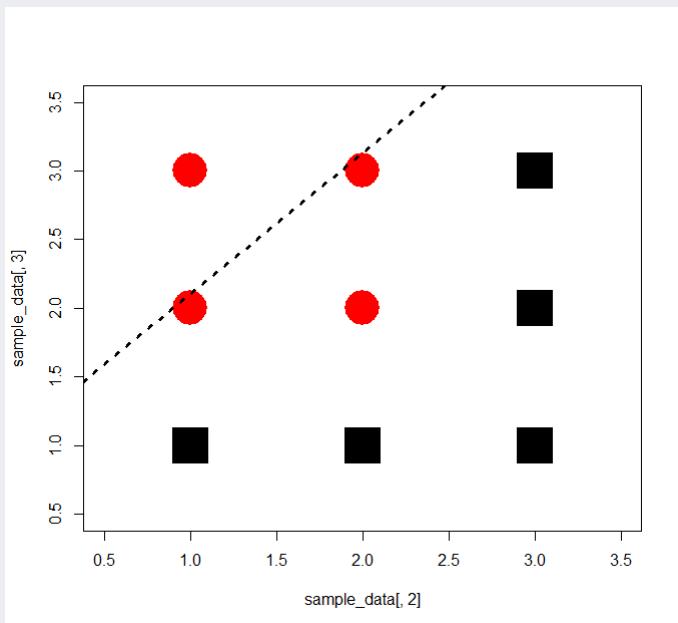
Training Perceptron: Example I ($\alpha = 0.5$)

- Training result and selection of the second training example ($x_1=3, x_2=3, t=0$)



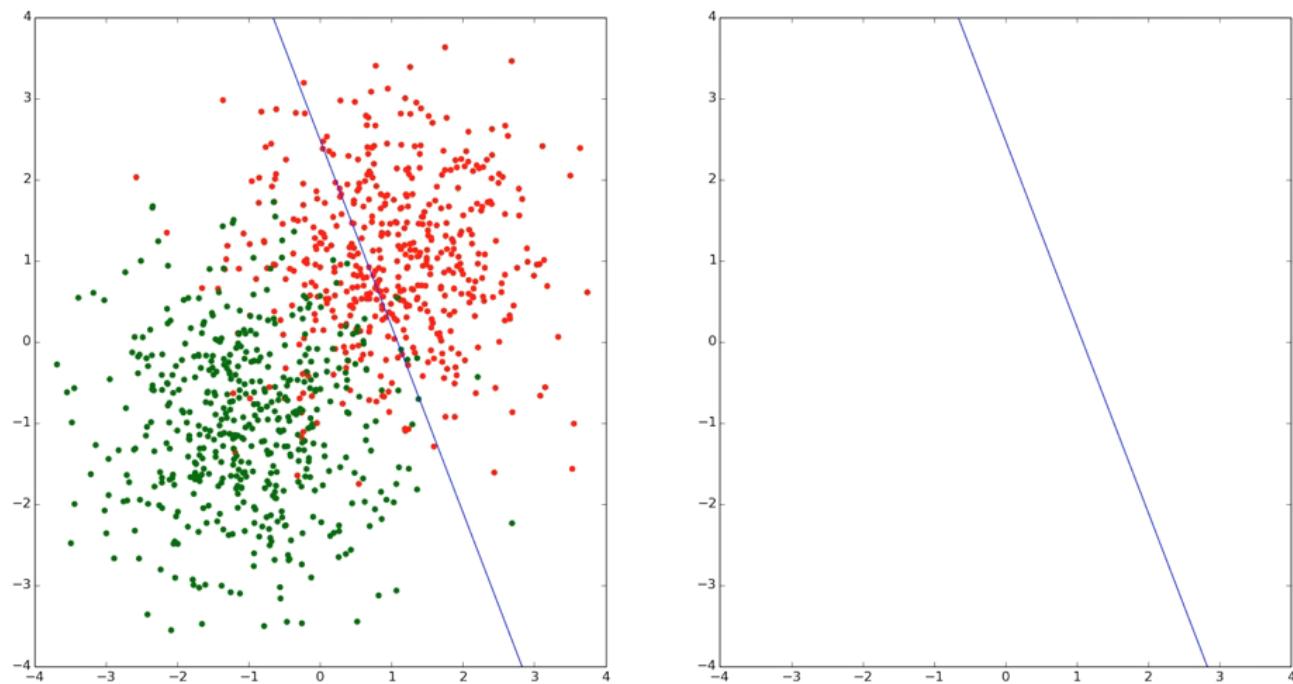
Training Perceptron: Example 1 (alpha = 0.5)

- Training result



Perceptron

- Training Perceptron

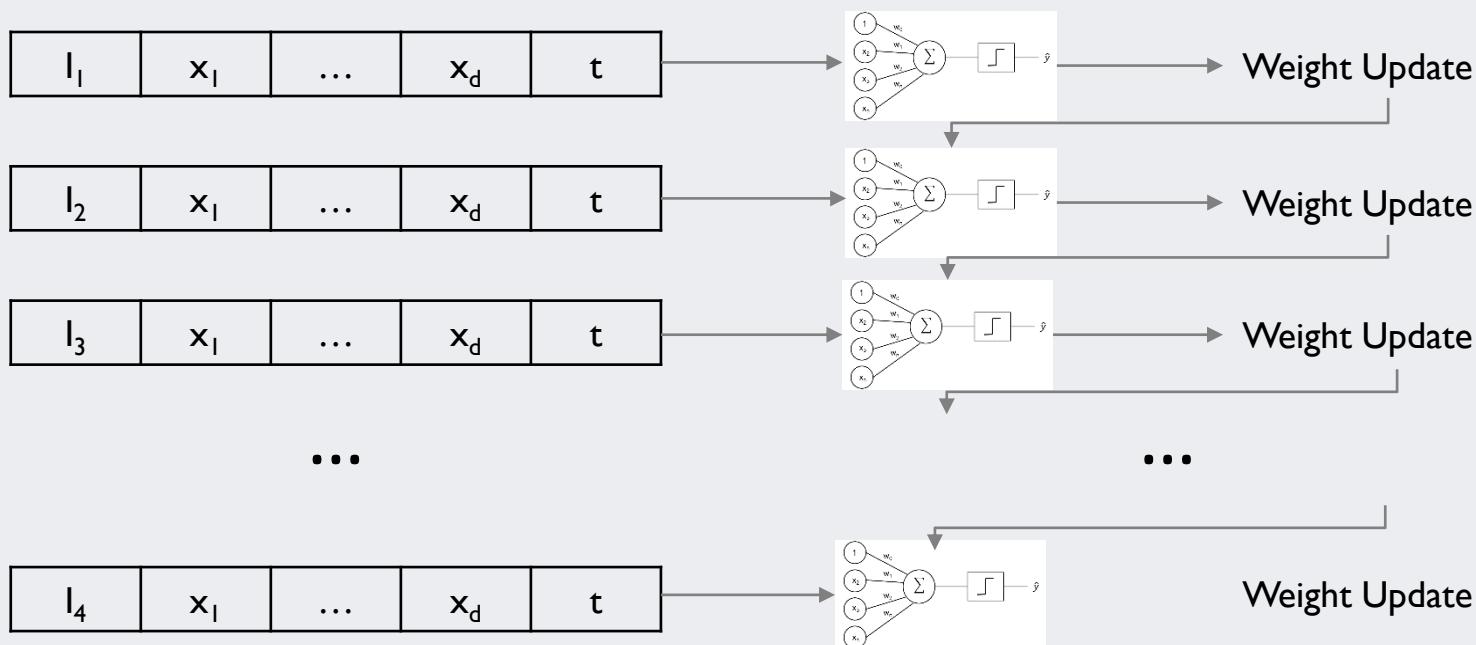


Issues on Gradient Descent

- Issue 1: How frequently should the weights be updated?

- ✓ Stochastic Gradient Descent (SGD)

- Compute the loss function for an individual training example and update the gradients

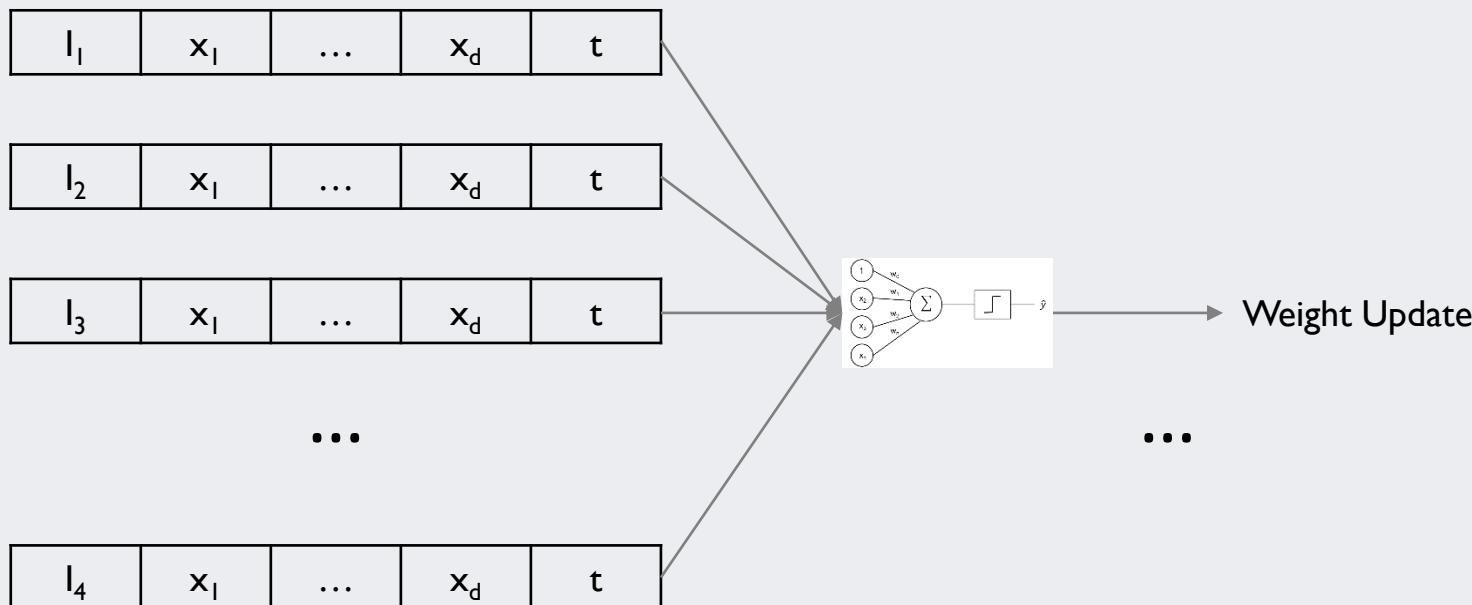


Issues on Gradient Descent

- Issue 1: How frequently should the weights be updated?

- ✓ Batch Gradient Descent (BGD)

- Fix the network weights, compute the cost function using all training examples, compute the gradient, and update the weights

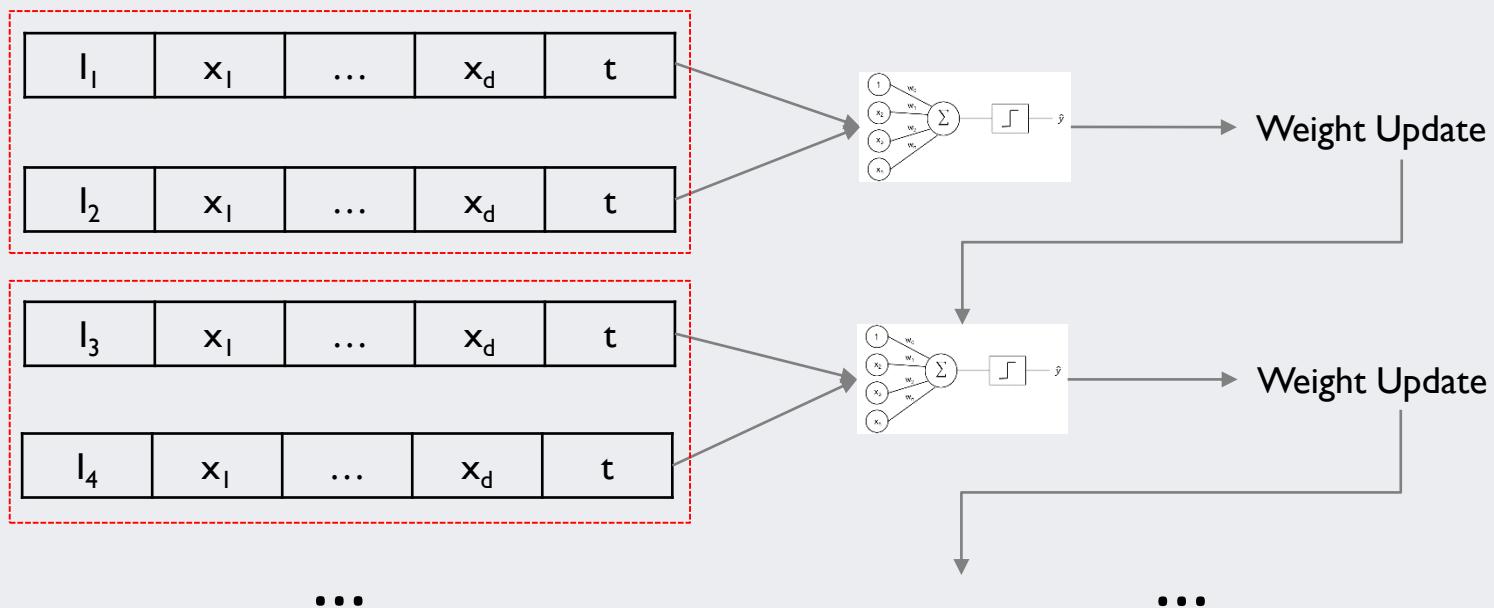


Issues on Gradient Descent

- Issue 1: How frequently should the weights be updated?

✓ Mini-Batch Gradient Descent

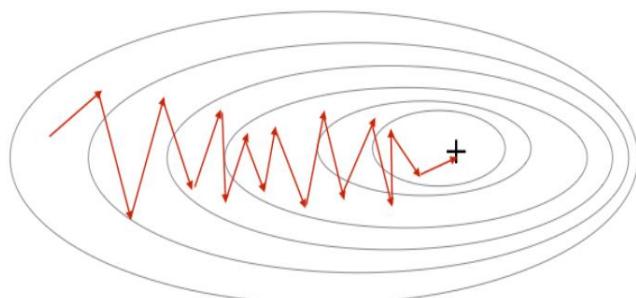
- A strategy between SGD and BGD
- Construct a mini-batch with n examples from N training examples and compute the gradient using the n examples (when the mini-batch size is set to 2)



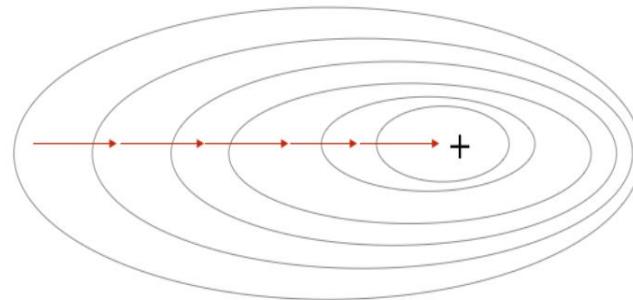
Issues on Gradient Descent

- Issue 1: How frequently should the weights be updated?

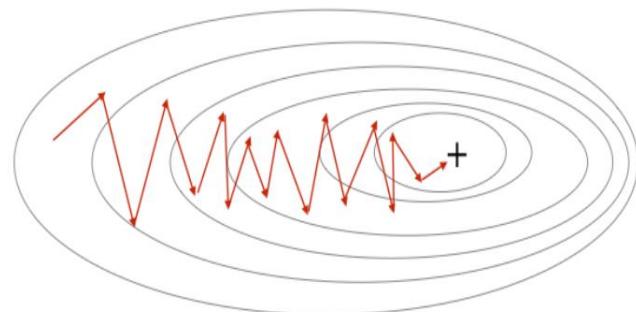
Stochastic Gradient Descent



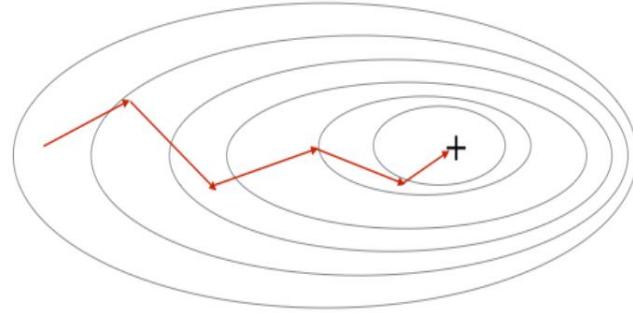
Gradient Descent



Stochastic Gradient Descent

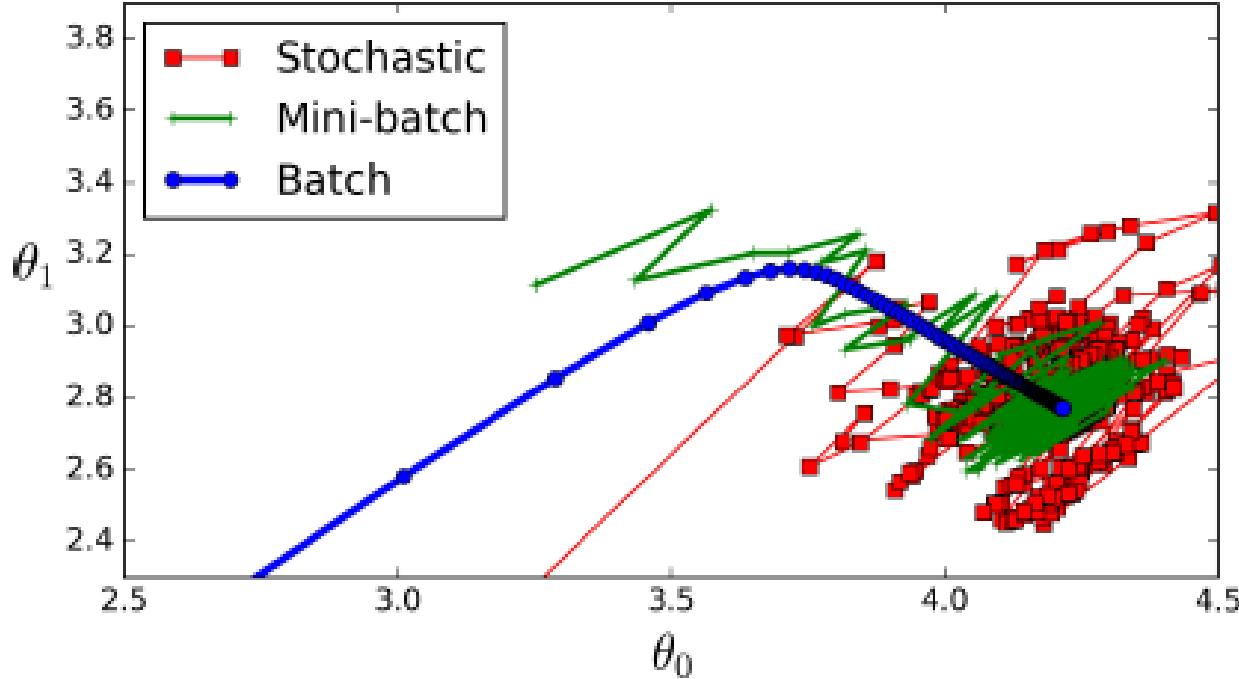


Mini-Batch Gradient Descent



Issues on Gradient Descent

- Issue 1: How frequently should the weights be updated?



Issues on Gradient Descent

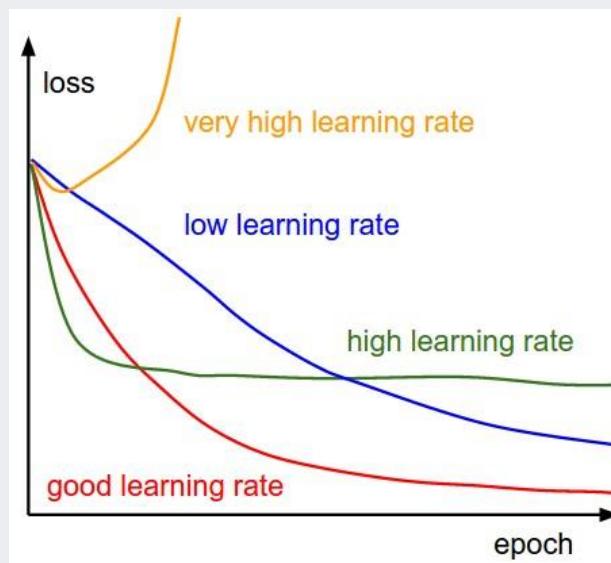
- Issue 2: How much should the weights be updated?

✓ A problem of deciding learning rate

$$w_{new} = w_{old} - \alpha f'(w)$$

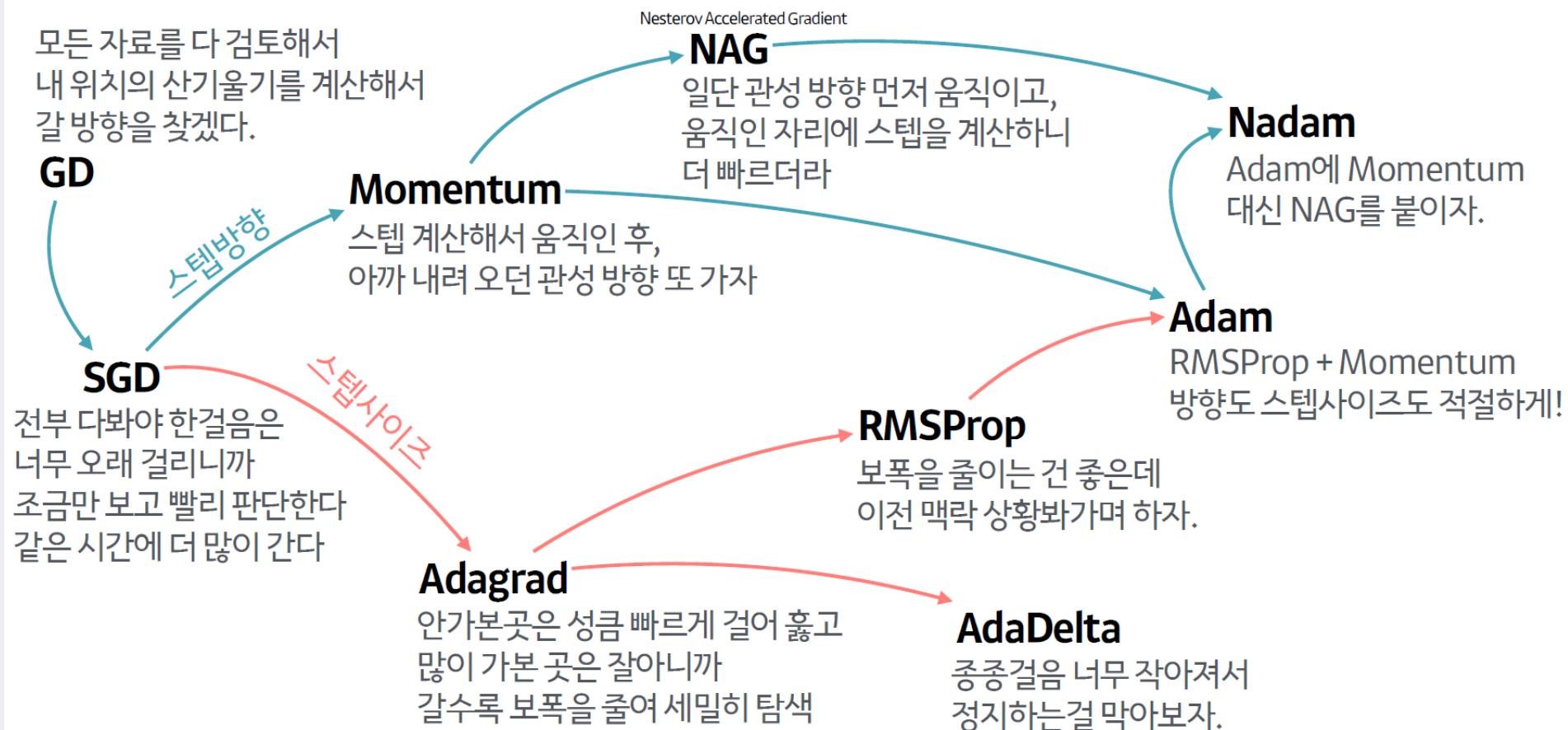
- If the learning rate is too large, the network will not converge
- If the learning rate is too small, it takes too long time to converge

✓ An appropriate (?) learning rate is required



Issues on Gradient Descent

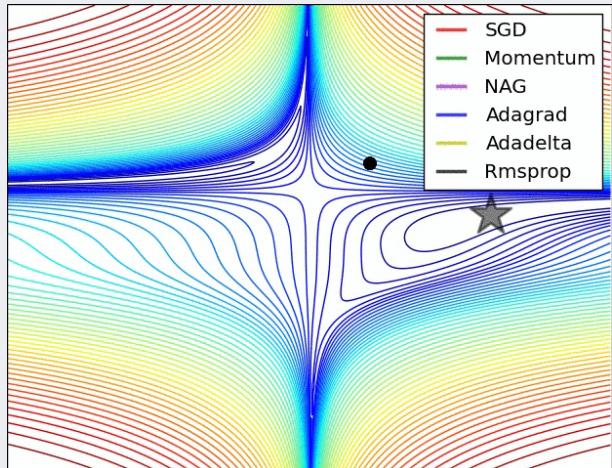
- Issue 2: How much should the weights be updated?



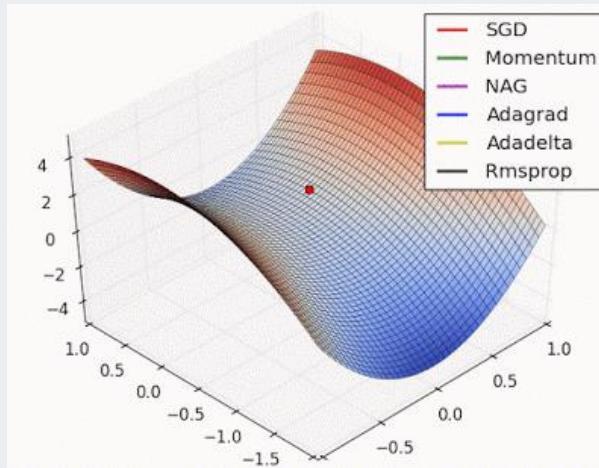
하용호. 자습해도 모르겠던 딥러닝, 머리속에 인스톨 시켜드립니다. (<https://www.slideshare.net/yongho/ss-79607172>)

Issues on Gradient Descent

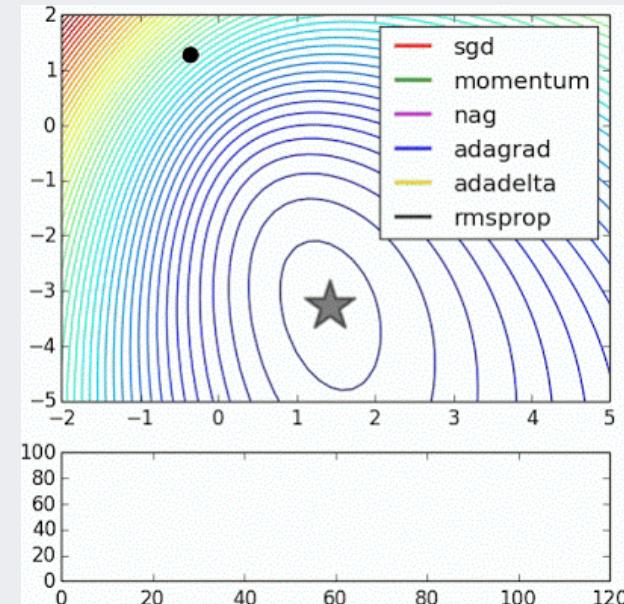
- Issue 2: How much should the weights be updated?



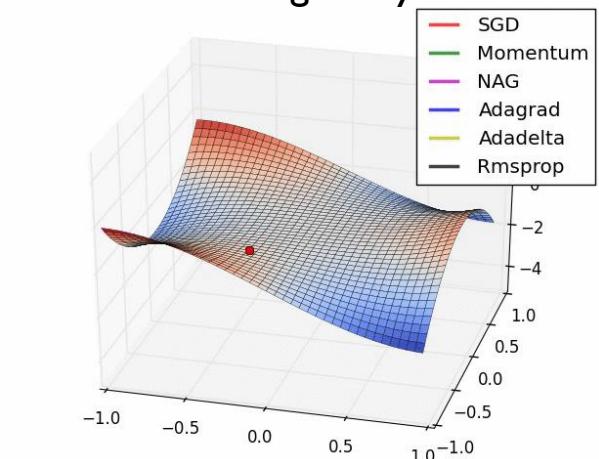
Beale's function



Long valley



Noisy moons



Saddle point



AGENDA

01 Artificial Neural Networks: Perceptron

02 Multi-layer Perceptron (MLP)

03 R Exercise

Limitation of Perceptron

- The Limitation of Linear Models

- ✓ **Classification:**

- Linear (Fisher) discriminant analysis, logistic regression, etc.
 - Can only produce a linear class boundary

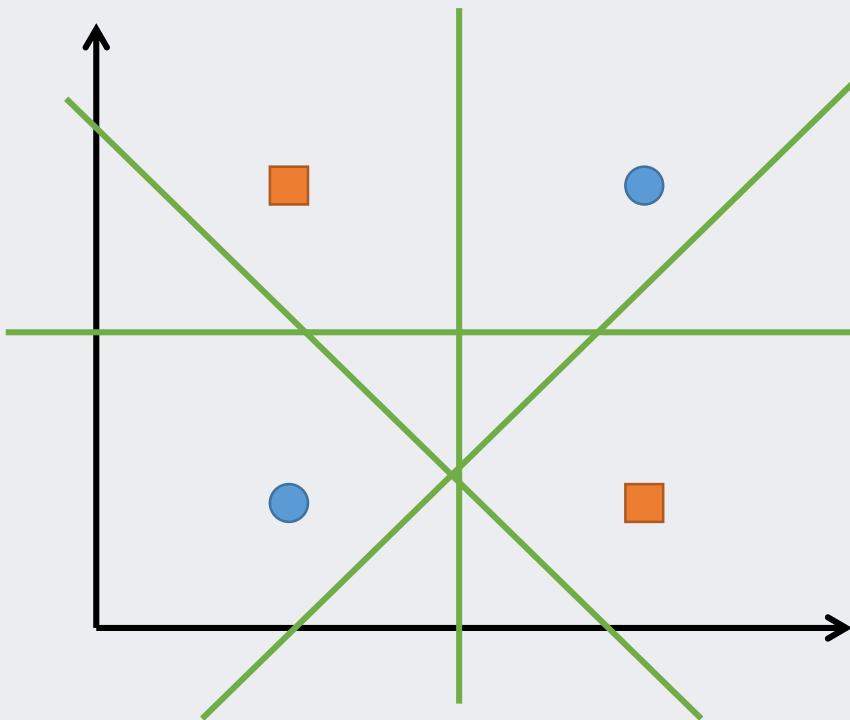
- ✓ **Regression:**

- Multiple linear regression
 - Can only capture the linear relationship between the predictors and the outcome

- ✓ Cannot results in good prediction performance *when the classification boundary or the predictor/outcome relationship is not linear*

Limitation of Perceptron

- The Limitation of Linear Models
 - ✓ Draw a straight line that perfectly separates the circles and crosses (XOR)

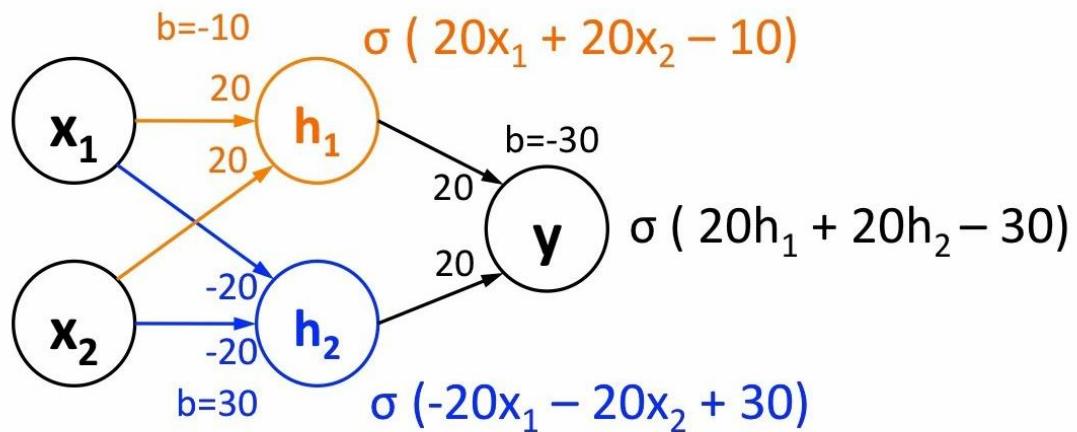
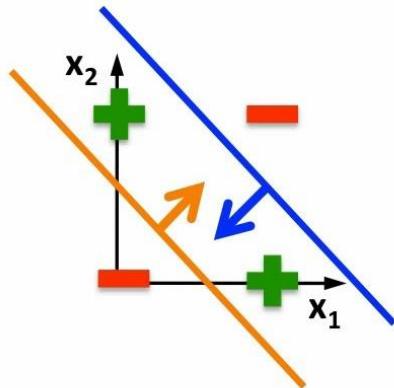


Multi-Layer Perceptron (MLP)

- Combine multiple perceptrons!

✓ If we cannot solve a complex problem directly, then it is better to **decompose** it into some small and simple problems that can be solved!

Linear classifiers
cannot solve this



$$\sigma(20*0 + 20*0 - 10) \approx 0$$

$$\sigma(20*1 + 20*1 - 10) \approx 1$$

$$\sigma(20*0 + 20*1 - 10) \approx 1$$

$$\sigma(20*1 + 20*0 - 10) \approx 1$$

$$\sigma(-20*0 - 20*0 + 30) \approx 1$$

$$\sigma(-20*1 - 20*1 + 30) \approx 0$$

$$\sigma(-20*0 - 20*1 + 30) \approx 1$$

$$\sigma(-20*1 - 20*0 + 30) \approx 1$$

$$\sigma(20*0 + 20*1 - 30) \approx 0$$

$$\sigma(20*1 + 20*0 - 30) \approx 0$$

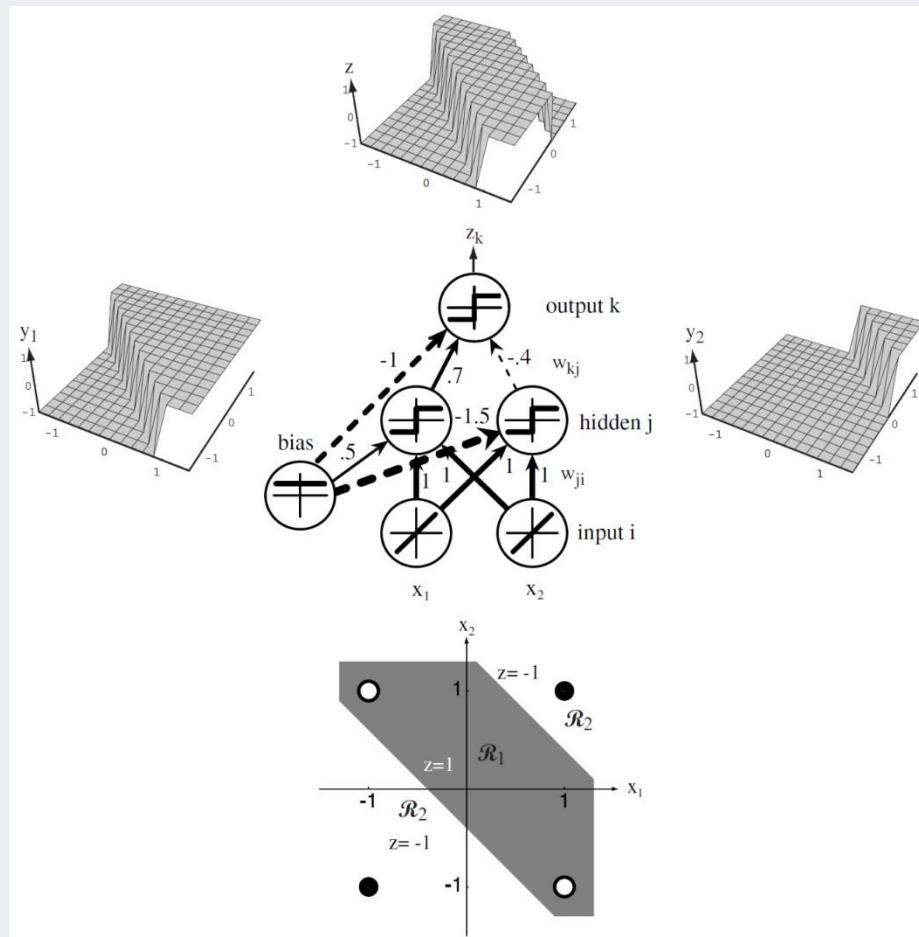
$$\sigma(20*1 + 20*1 - 30) \approx 1$$

$$\sigma(20*1 + 20*1 - 30) \approx 1$$

Multi-Layer Perceptron (MLP)

- Non-linear model

✓ Can find an arbitrary shape of class boundary or regression functions



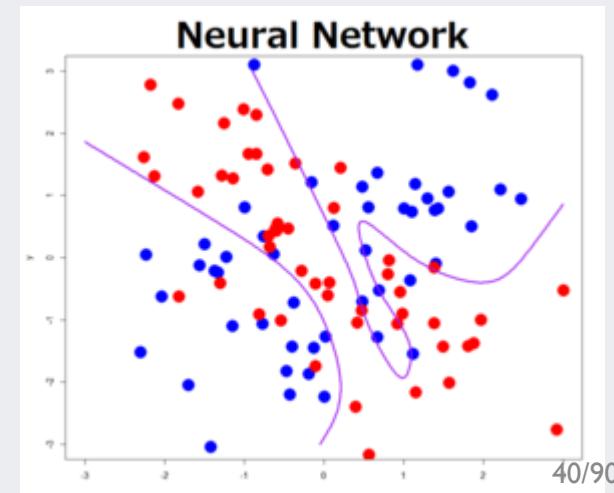
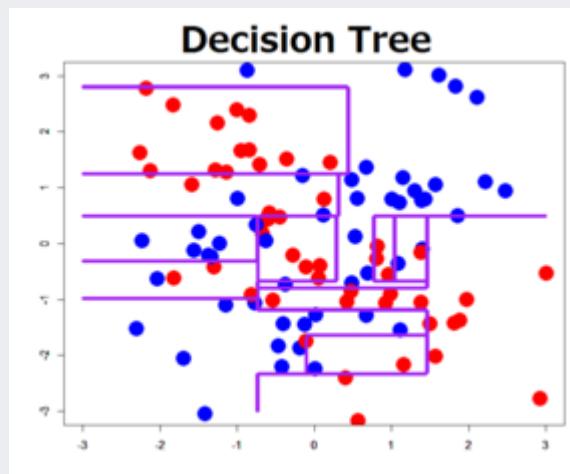
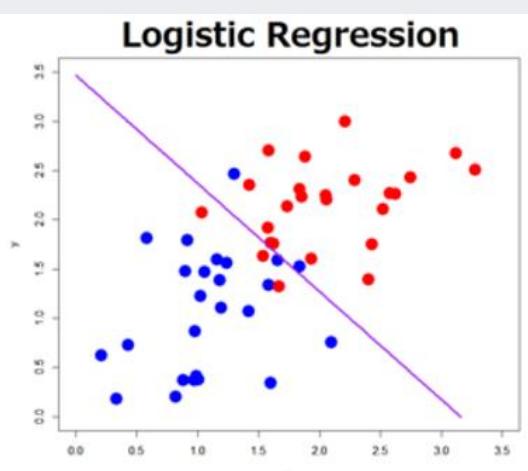
Multi-Layer Perceptron (MLP)

- Decision boundary of MLP

- ✓ Assume that the class decision boundary can be regarded as a combination of piecewise linear boundaries

	Logistic Regression	Decision Tree	MLP
No. of lines	1	No restriction	User defined (No of hidden layers and hidden nodes)
Direction of lines	No restriction	Vertical to an axis	No restriction

- MLP has the highest degree of freedom to define the decision boundary



Multi-Layer Perceptron (MLP)

- Various Structure of Artificial Neural Networks

A mostly complete chart of

Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

- Backfed Input Cell
- Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Different Memory Cell
- Kernel
- Convolution or Pool

Perceptron (P)



Feed Forward (FF)



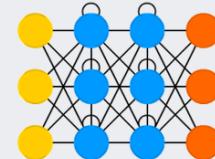
Radial Basis Network (RBF)



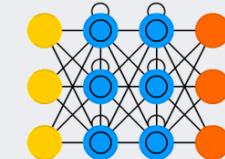
Deep Feed Forward (DFF)



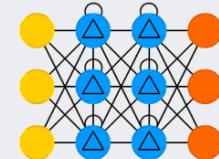
Recurrent Neural Network (RNN)



Long / Short Term Memory (LSTM)



Gated Recurrent Unit (GRU)



Auto Encoder (AE)



Variational AE (VAE)



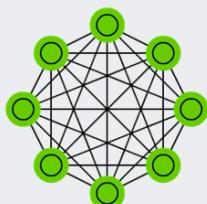
Denoising AE (DAE)



Sparse AE (SAE)



Markov Chain (MC)



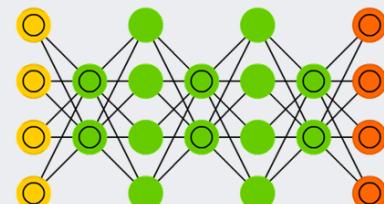
Hopfield Network (HN)



Restricted BM (RBM)

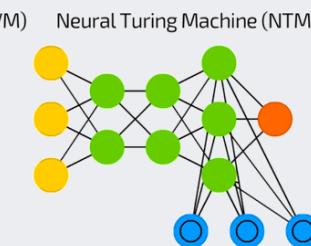
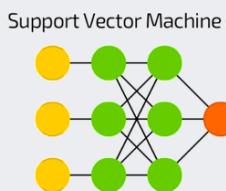
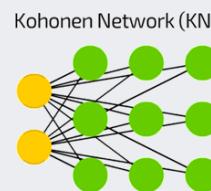
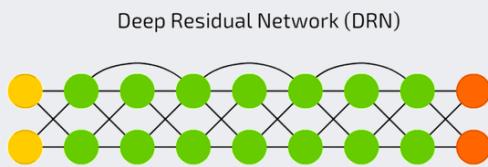
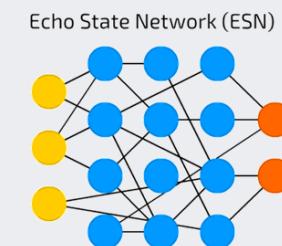
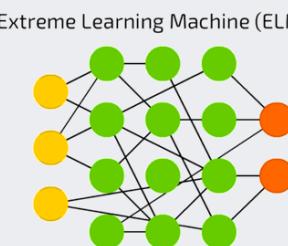
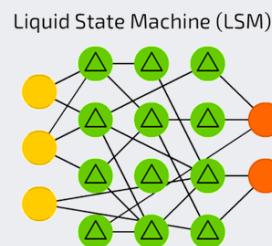
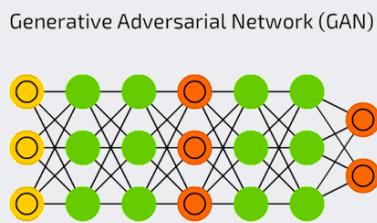
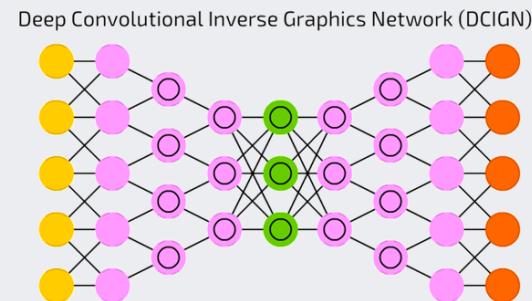
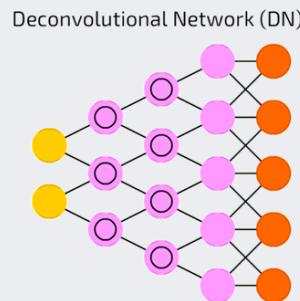
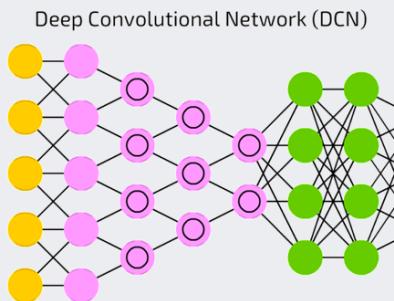


Deep Belief Network (DBN)



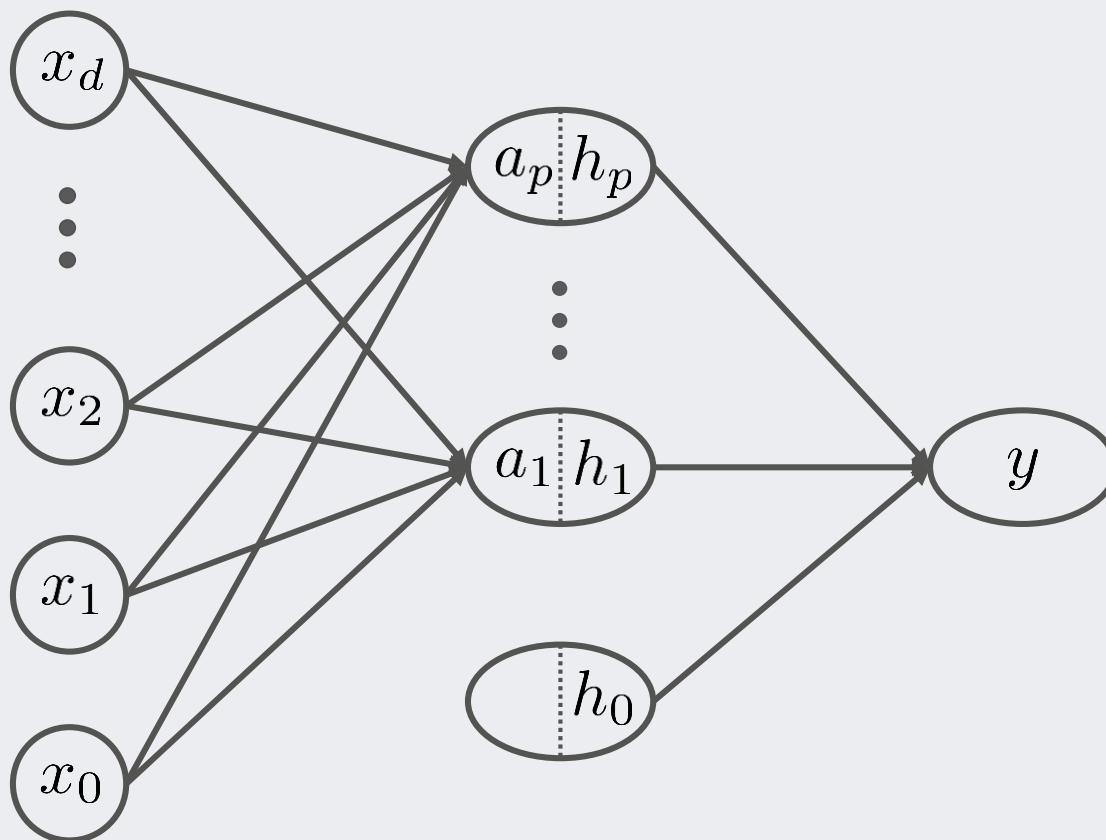
Multi-Layer Perceptron (MLP)

- Various Structure of Artificial Neural Networks



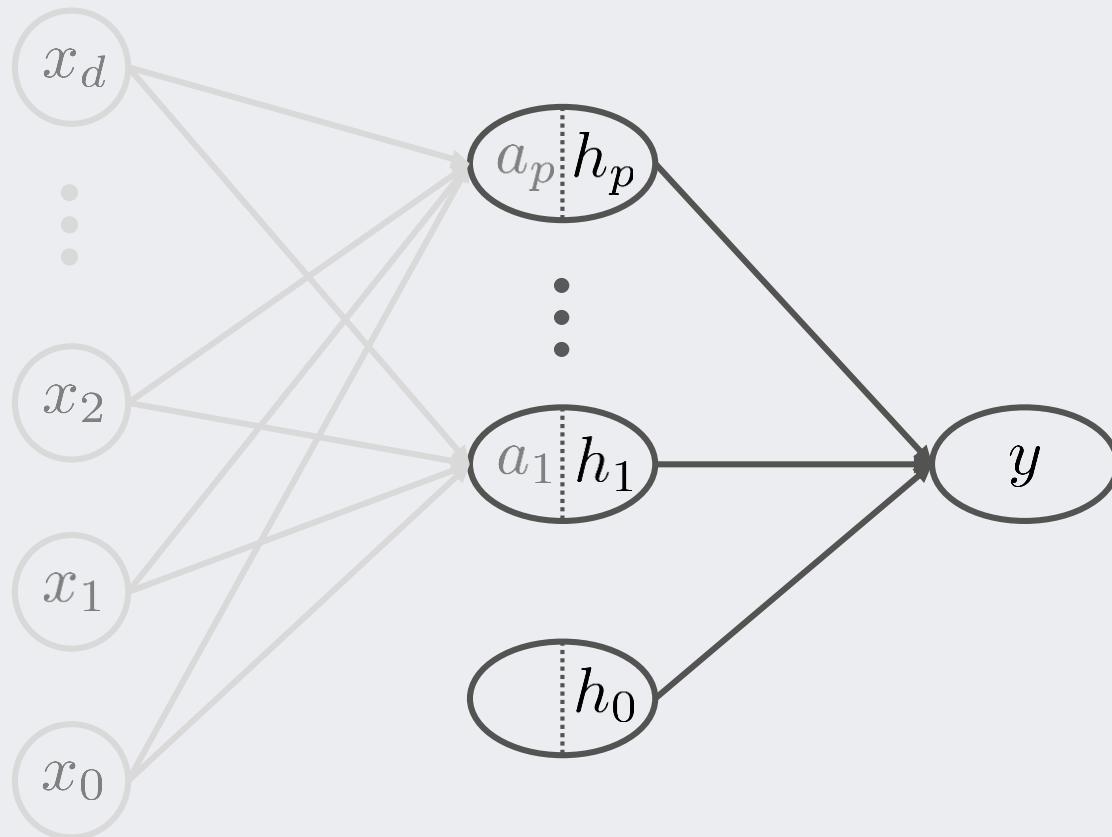
MLP with One Hidden Layer

- Basic Structure: Feed-forward Neural Network with One Hidden Layer
 - ✓ Each hidden node can be considered as an independent perceptron
 - ✓ The output node is a combination of all perceptrons



MLP with One Hidden Layer

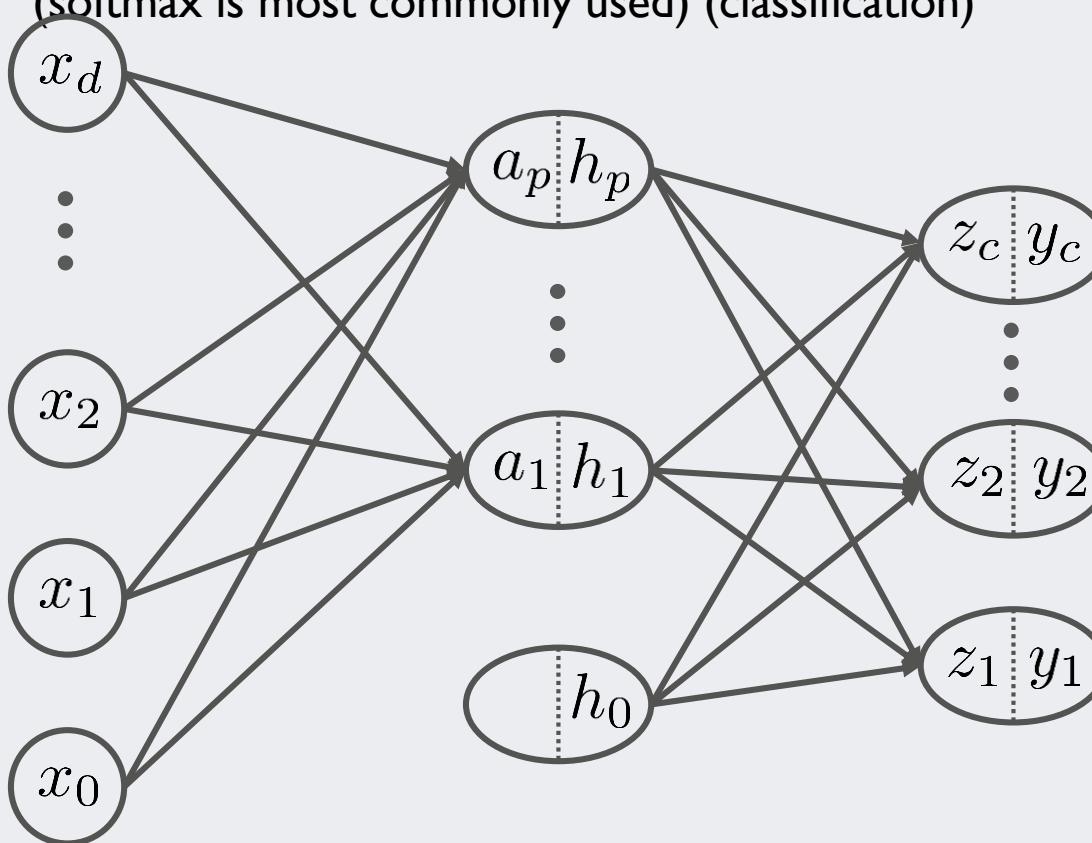
- Basic Structure: Feed-forward Neural Network with One Hidden Layer
 - ✓ Each hidden node can be considered as an independent perceptron
 - ✓ The output node is a linear combination of all perceptrons (regression)



$$y = \sum_{i=0}^p w_p^{(2)} h_p$$

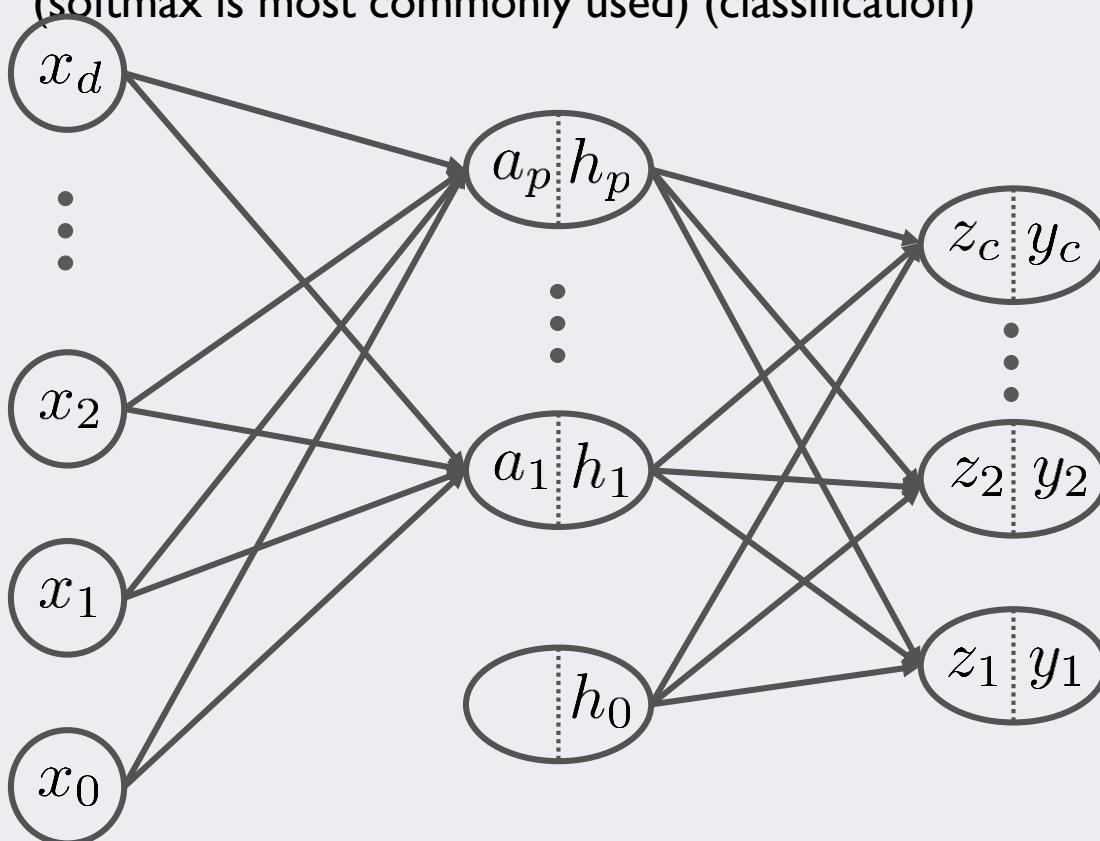
MLP with One Hidden Layer

- Basic Structure: Feed-forward Neural Network with One Hidden Layer
 - ✓ Each hidden node can be considered as an independent perceptron
 - ✓ The output node is a combination of all perceptrons with a non-linear transformation (softmax is most commonly used) (classification)



MLP with One Hidden Layer

- Basic Structure: Feed-forward Neural Network with One Hidden Layer
 - ✓ Each hidden node can be considered as an independent perceptron
 - ✓ The output node is a combination of all perceptrons with a non-linear transformation (softmax is most commonly used) (classification)



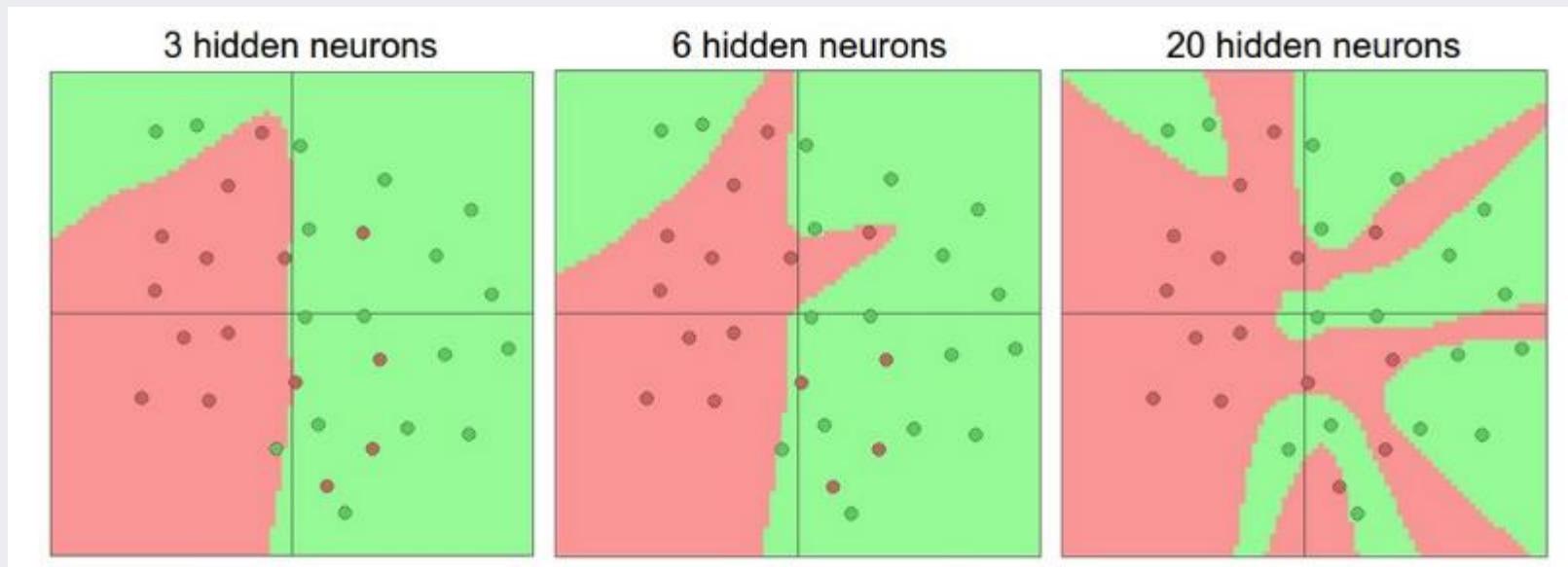
$$z_j = \sum_{i=0}^p w_{jp}^{(2)} h_p$$

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^c e^{z_k}}$$

각 출력 노드의 출력값을 해당 범주에 속하는 확률로 해석할 수 있음

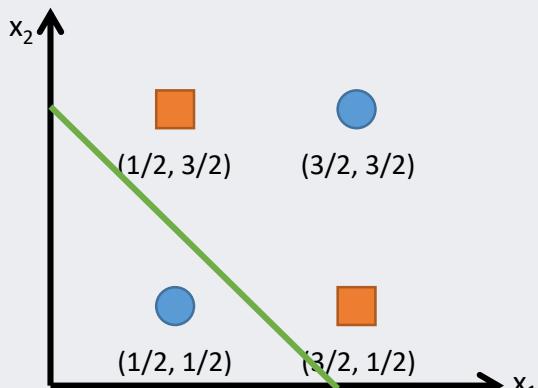
MLP: Hidden Nodes

- The role of hidden nodes
 - ✓ Determines the complexity of ANN
 - ✓ If we use more number of hidden nodes, we can find a more sophisticated decision boundary (classification) or an arbitrary shape of function (regression)



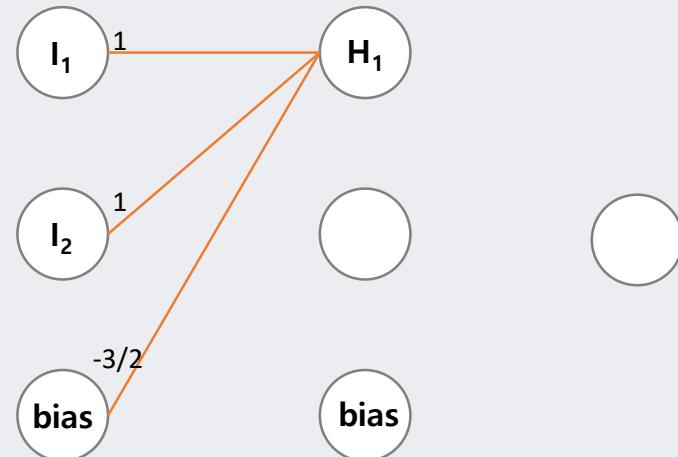
MLP: How it works?

- XOR problem revisited



$$h_1 = x_1 + x_2 - \frac{3}{2}$$

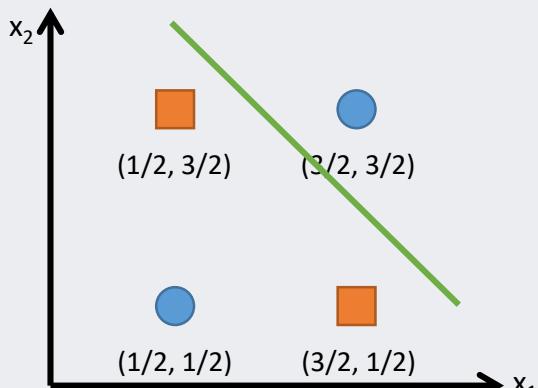
$$z_1 = g(h_1) = \begin{cases} 1 & \text{if } h_1 \geq 0 \\ -1 & \text{if } h_1 < 0 \end{cases}$$



	x_1	x_2	h_1	z_1
	1/2	1/2	-1/2	-1
	3/2	1/2	1/2	1
	1/2	3/2	1/2	1
	3/2	3/2	3/2	1

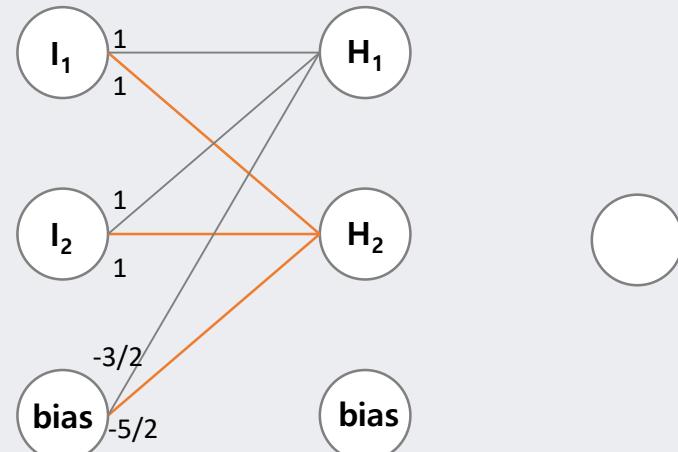
MLP: How it works?

- XOR problem revisited (cont')



$$h_2 = x_1 + x_2 - \frac{5}{2}$$

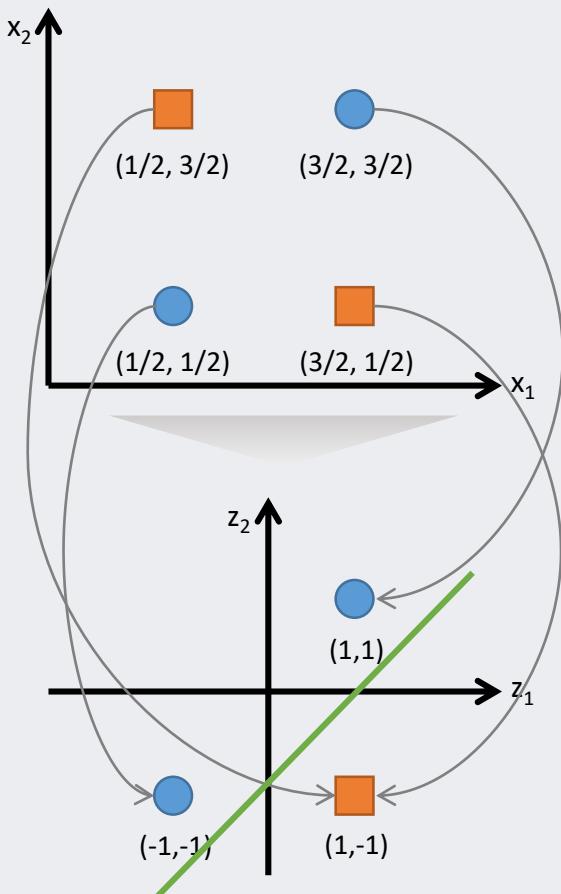
$$z_2 = g(h_2) = \begin{cases} 1 & \text{if } h_2 \geq 0 \\ -1 & \text{if } h_2 < 0 \end{cases}$$



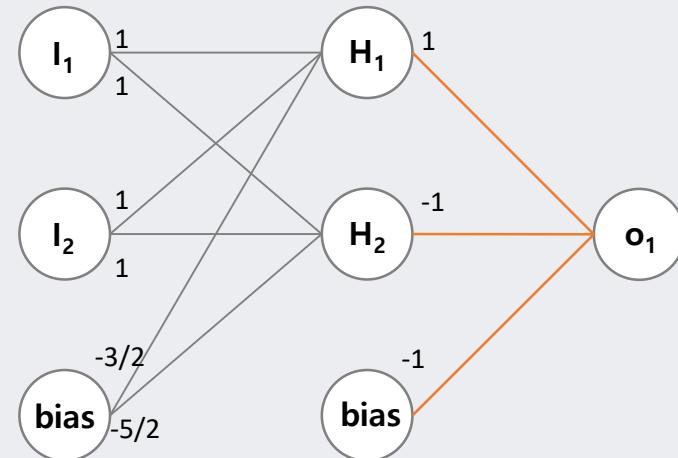
x_1	x_2	h_2	z_2
1/2	1/2	-3/2	-1
3/2	1/2	-1/2	-1
1/2	3/2	-1/2	-1
3/2	3/2	1/2	1

MLP: How it works?

- XOR problem revisited (cont')



$$o_1 = z_1 - z_2 - 1$$



x_1	x_2	h_1	z_1	h_2	z_2	o_1	z
1/2	1/2	-1/2	-1	-3/2	-1	-1	-1
3/2	1/2	1/2	1	-1/2	-1	1	1
1/2	3/2	1/2	1	-1/2	-1	1	1
3/2	3/2	3/2	1	1/2	1	-1	-1

$$z = g(o_1) = \begin{cases} 1 & \text{if } o_1 \geq 0 \\ -1 & \text{if } o_1 < 0 \end{cases}$$

MLP: Formulation

- General formulation

- ✓ The output of the hidden node j (when the activation function is sigmoid):

$$h_j = \sum_{i=1}^{d+1} w_{ji}^{(1)} x_i, \quad z_j = g(h_j) = \frac{1}{1 + \exp(-h_j)}$$

- ✓ The output of the output node (when the activation function is sigmoid):

$$o = \sum_{j=1}^{p+1} w_j^{(2)} g(h_j), \quad z = g(o) = \frac{1}{1 + \exp(-o)}$$

- ✓ The final outcome of the neural network:

$$\hat{y} = g\left(\sum_{j=1}^{p+1} w_j^{(2)} g\left(\sum_{i=1}^{d+1} w_{ji}^{(1)} x_i\right)\right)$$

MLP: Training

- Gradient descent algorithm

- ✓ Taylor expansion of a function

$$f(x + \Delta x) = f(x) + \frac{f'(x)}{1!} \Delta x + \frac{f''(x)}{2!} \Delta x^2 + \dots$$

- ✓ For the minimization problem, we can reduce the objective function value by moving the current solution to the opposite direction of the first derivative if it is not zero

$$x_{new} = x_{old} - \eta f'(x), \quad \text{where } 0 < \eta < 1$$

- ✓ The objective function value become lower compared to the previous solution

$$f(x_{new}) = f(x_{old} - \eta f'(x)) \cong f(x_{old}) - \eta |f'(x)|^2 < f(x_{old})$$

MLP: Training

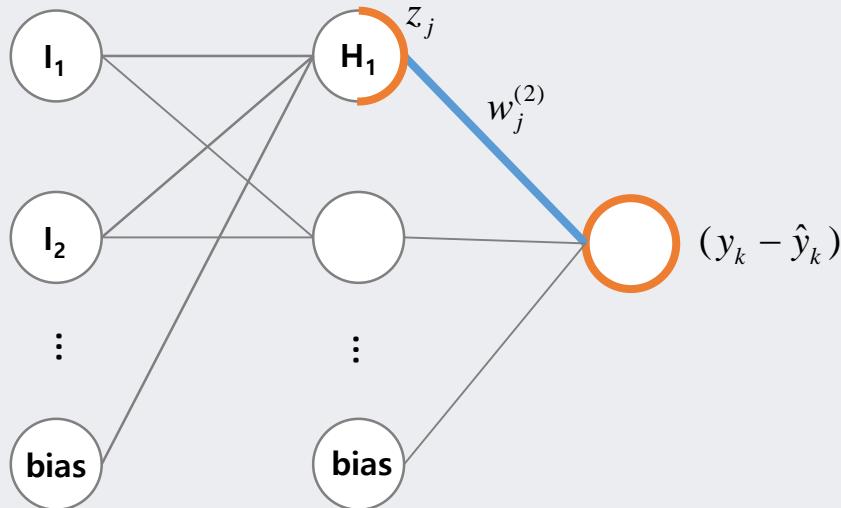
- Error Back-Propagation

- ✓ The error of kth observation

$$Err_k = \frac{1}{2} (y_k - \hat{y}_k)^2, \quad \hat{y}_k = \sum_{j=1}^{p+1} w_j^{(2)} g\left(\sum_{i=1}^{d+1} w_{ji}^{(1)} x_i\right)$$

- ✓ The weight $w_j^{(2)}$ which connects the jth hidden node

$$\frac{\partial Err_k}{\partial w_j^{(2)}} = \frac{\partial Err_k}{\partial \hat{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial w_j^{(2)}} = (y_k - \hat{y}_k) \cdot z_j$$



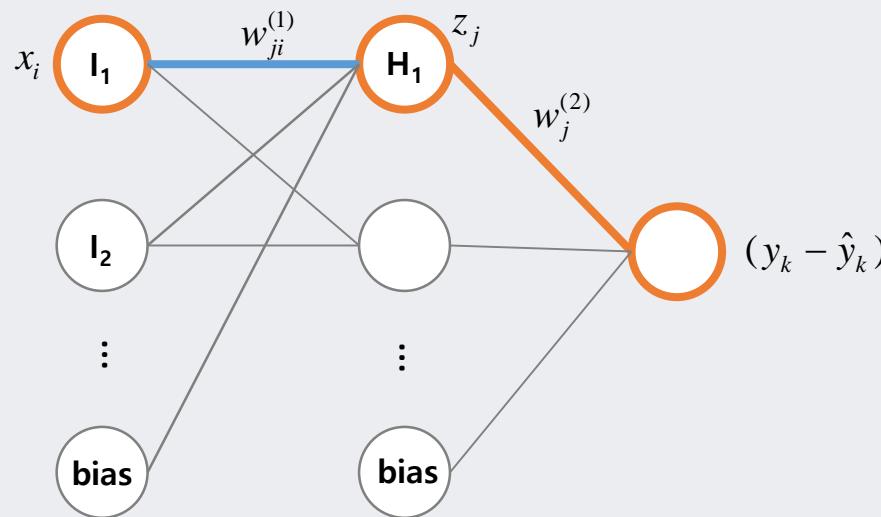
최종 결과물을 얻고	Feed Forward and Prediction
그 결과물과 우리가 원하는 결과물 과의 차이점을 찾은 후	Cost Function
그 차이가 무엇으로 인해 생기는지	Differentiation (미분)
역으로 내려가면서 추정하여	Back Propagation
새로운 Parameter 값을 배움	Weight Update

MLP: Training

- Error Back-Propagation

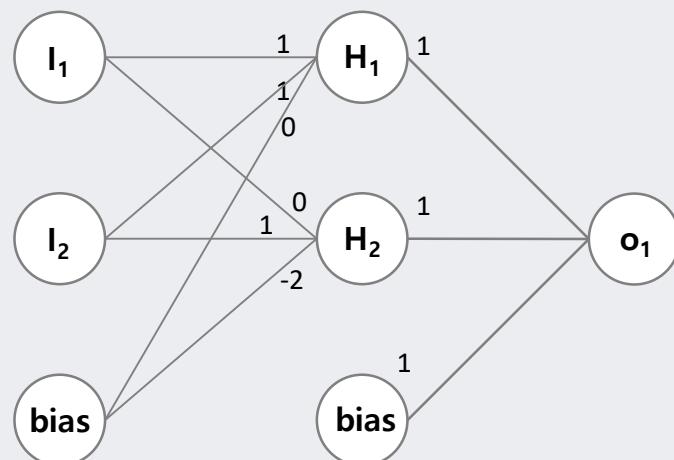
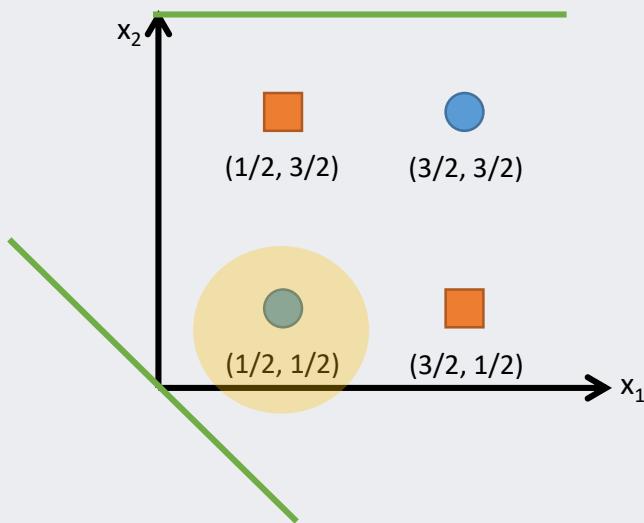
✓ The weight $w_{ji}^{(1)}$ which connects the i^{th} input node and j^{th} hidden node

$$\frac{\partial Err_k}{\partial w_{ji}^{(1)}} = \frac{\partial Err_k}{\partial \hat{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial z_j} \cdot \frac{\partial z_j}{\partial h_j} \cdot \frac{\partial h_j}{\partial w_{ji}^{(1)}} = (y_k - \hat{y}_k) \cdot w_j^{(2)} \cdot z_j \cdot (1 - z_j) \cdot x_i$$



MLP: Training

- Error Back-Propagation: Example
- ✓ Initial weight: Random generation



$$h_1 = \sum w_{1i}^{(1)} x_i = 1 \times 0.5 + 1 \times 0.5 + 0 \times 1 = 1$$

$$z_1 = \frac{1}{1 + \exp(1)} = 0.269$$

$$h_2 = \sum w_{2i}^{(1)} x_i = 0 \times 0.5 + 1 \times 0.5 + (-2) \times 1 = -1.5$$

$$z_2 = \frac{1}{1 + \exp(-1.5)} = 0.818$$

$$\hat{y} = \sum w_j^{(2)} z_j = 1 \times 0.269 + 1 \times 0.818 + 1 \times 1 = 2.087$$

MLP: Training

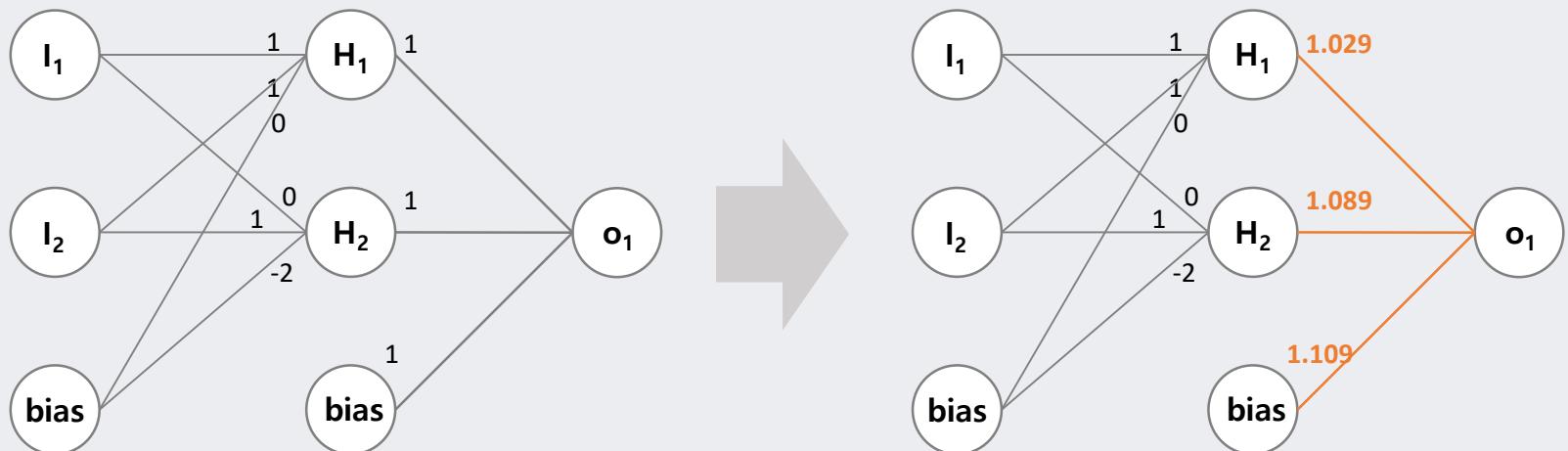
- Error Back-Propagation: Example

✓ Update the weights between the output and the hidden nodes

$$w_1^{(2)}(\text{new}) = w_1^{(2)}(\text{old}) - \eta \times (y - \hat{y}) \times z_1 = 1 - 0.1 \times (1 - 2.087) \times 0.269 = 1.029$$

$$w_2^{(2)}(\text{new}) = w_2^{(2)}(\text{old}) - \eta \times (y - \hat{y}) \times z_2 = 1 - 0.1 \times (1 - 2.087) \times 0.818 = 1.089$$

$$w_0^{(2)}(\text{new}) = w_0^{(2)}(\text{old}) - \eta \times (y - \hat{y}) \times b^{(2)} = 1 - 0.1 \times (1 - 2.087) \times 1 = 1.109$$



MLP: Training

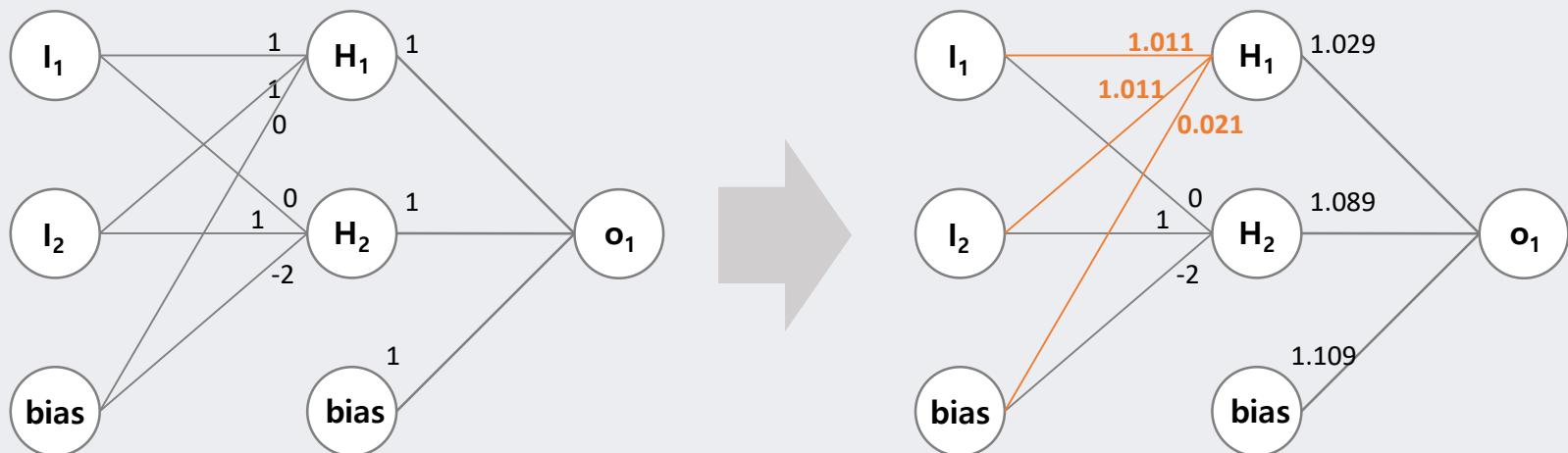
- Error Back-Propagation: Example

✓ Update the weights between the H_1 and the input nodes

$$w_{11}^{(1)}(\text{new}) = w_{11}^{(1)}(\text{old}) - \eta \times (y - \hat{y}) \times w_1^{(2)} \times z_1 \times (1 - z_1) \times x_1 = 1.011$$

$$w_{12}^{(1)}(\text{new}) = w_{12}^{(1)}(\text{old}) - \eta \times (y - \hat{y}) \times w_1^{(2)} \times z_1 \times (1 - z_1) \times x_2 = 1.011$$

$$w_{10}^{(1)}(\text{new}) = w_{10}^{(1)}(\text{old}) - \eta \times (y - \hat{y}) \times w_1^{(2)} \times z_1 \times (1 - z_1) \times b^{(1)} = 0.021$$



MLP: Training

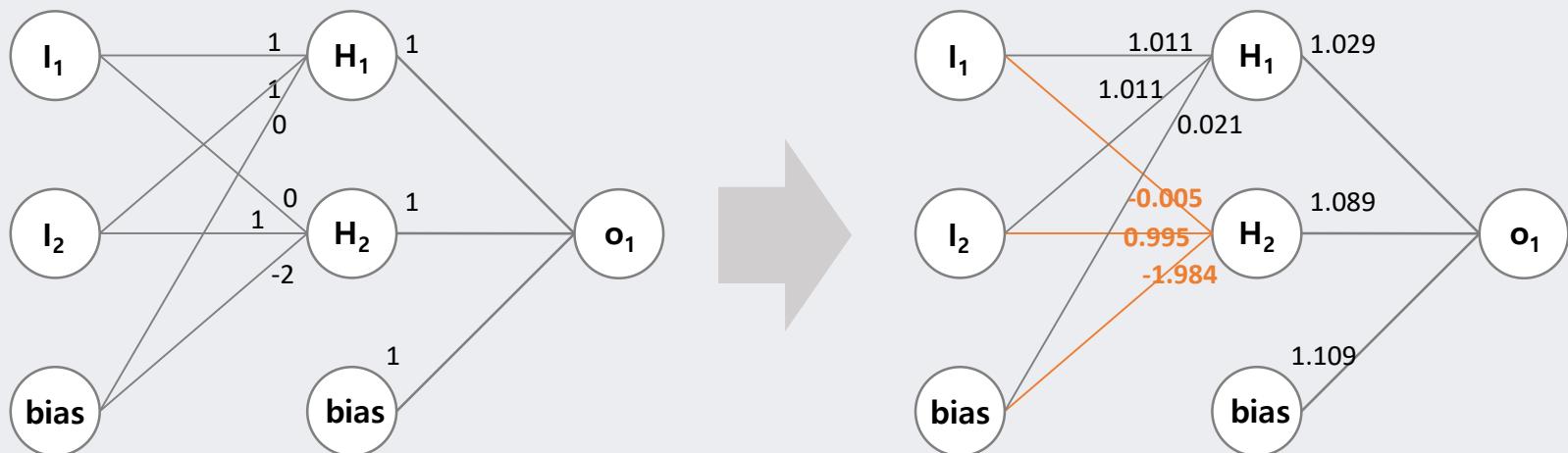
- Error Back-Propagation: Example

✓ Update the weights between the H_1 and the input nodes

$$w_{21}^{(1)}(\text{new}) = w_{21}^{(1)}(\text{old}) - \eta \times (y - \hat{y}) \times w_2^{(2)} \times z_2 \times (1 - z_2) \times x_1 = -0.005$$

$$w_{22}^{(1)}(\text{new}) = w_{22}^{(1)}(\text{old}) - \eta \times (y - \hat{y}) \times w_2^{(2)} \times z_2 \times (1 - z_2) \times x_2 = 0.995$$

$$w_{20}^{(1)}(\text{new}) = w_{20}^{(1)}(\text{old}) - \eta \times (y - \hat{y}) \times w_2^{(2)} \times z_2 \times (1 - z_2) \times b^{(1)} = -1.984$$



MLP: Training

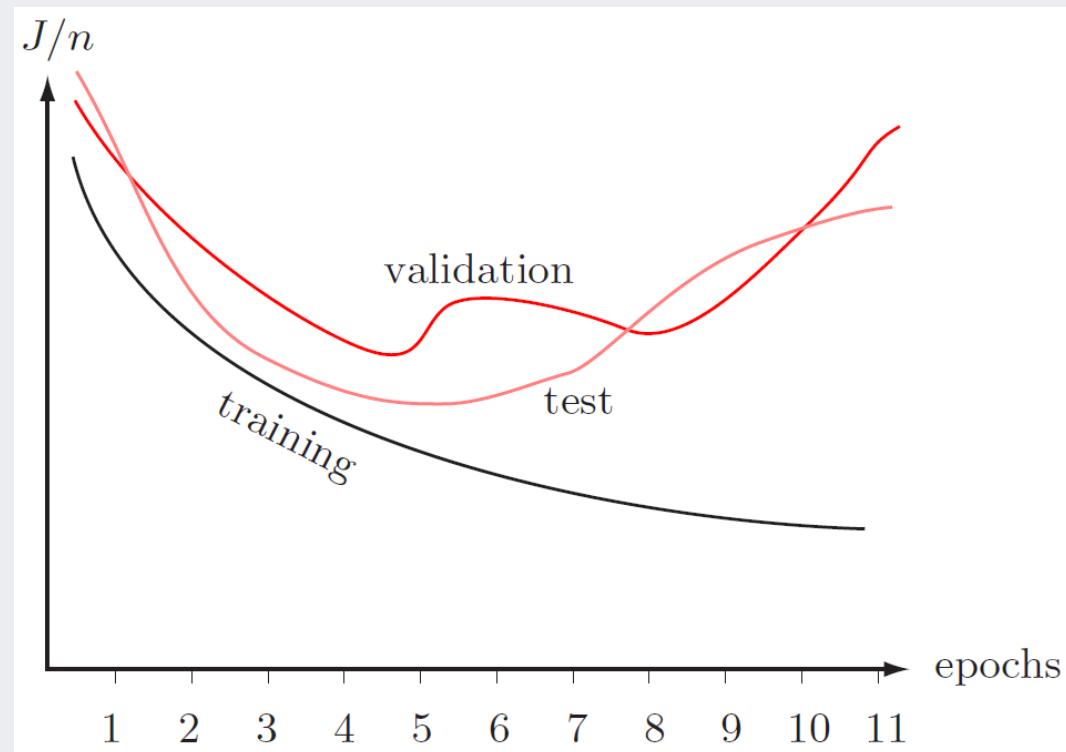
- Goal
 - ✓ Find the weights that yield best predictions
- Features
 - ✓ The process described before is repeated for all records
 - ✓ At each record, compare the prediction to the actual target
 - ✓ Difference is the error for the output node
 - ✓ Error is propagated back and distributed to all the hidden nodes and used to update their weights

MLP: Training

- Why it works
 - ✓ Big errors lead to big changes in weights
 - ✓ Small errors leave weights relatively unchanged
 - ✓ Over thousand of updates, a given weight keeps changing until the error associated with it is negligible
- Common criteria to stop updating
 - ✓ When weights change very little from one epoch to the next
 - ✓ When the misclassification rate reaches a required threshold
 - ✓ When a limit on runs is reached

MLP: Training

- With sufficient iterations, neural networks can easily over-fit the data.
- To avoid over-fitting,
 - ✓ Track error in validation data
 - ✓ Limit iterations
 - ✓ Limit complexity of network
 - ✓ N. of hidden layers, nodes, etc.



Recommended Video Lectures

- 유튜브 3Blue1Brown Neural Network 강좌

✓ https://www.youtube.com/channel/UCYO_jab_esuFRV4bI7AjtAw



The screenshot shows a YouTube video player. The main video frame displays a close-up of a human eye with a neural network diagram overlaid. Below the video, the channel name '3Blue1Brown' is visible. At the bottom of the player, there is a control bar with icons for navigation, volume, and settings, along with a progress bar indicating the video is at 0:02 / 19:13.

Neural networks

3Blue1Brown - 1 / 4

But what *is* a Neural Network? | Chapter 1, deep learning
3Blue1Brown

How machines learn
Gradient descent, how neural networks learn | Chapter 2, deep learning
3Blue1Brown

Backpropagation
What is backpropagation really doing? | Chapter 3, deep learning
3Blue1Brown

Backpropagation calculus
Backpropagation calculus | Appendix to deep learning chapter 3
3Blue1Brown

But what *is* a Neural Network? | Chapter 1, deep learning

조회수 853,260회

3Blue1Brown 2017. 10. 5.

구독중 60.8만명

Subscribe to stay notified about new videos: <http://3b1b.co/subscribe>
Support more videos like this on Patreon: <https://www.patreon.com/3blue1brown>
Special thanks to these supporters: <http://3b1b.co/nn1-thanks>

더보기

Fast campus

한글의 특성 + 텍스트 분석 알고리즘
모두 다루는 국내유일 10주 과정

TEXT MINING TEXT MINING TEXT MINING TEXT MINING

파이썬을 활용한 텍스트 마이닝 캠프 >

Recommended Video Lectures

- 유튜브 Brandon Rohrer 강좌

- ✓ <https://www.youtube.com/watch?v=ILsA4nyG7I0&list=PLVZqlMpoM6kbaeySxhdtgQPFEc5nV7Faa&index=2>

The screenshot shows a YouTube search results page for 'Brandon Rohrer'. The main video thumbnail on the left is titled 'How neural networks work' by Brandon Rohrer, showing a blue background with white text. Below it is a video player with a progress bar at 0:02 / 24:37. To the right is a sidebar titled 'Talks' showing five video thumbnails:

- 1. How Deep Neural Networks Work (24:38)
- 2. How Convolutional Neural Networks work (26:14)
- 3. How Data Science Works (49:48)
- 4. Deep Learning Demystified (22:19)

Below the sidebar, there are two more video cards:

- How Deep Neural Networks Work** by Brandon Rohrer (565,188 views, 1 like, 223 dislikes)
- Neural Network 3D Simulation** by Denis Dmitriev (9.8 million views)

At the bottom, there is a description of the first video: 'A gentle introduction to the principles behind neural networks, including backpropagation. Rated G for general audiences.'

AGENDA

- 01 Artificial Neural Networks: Perceptron
- 02 Multi-layer Perceptron (MLP)
- 03 R Exercise

R Exercise: Classification

- Dataset: Cardiotocography Data Set
 - ✓ <https://archive.ics.uci.edu/ml/datasets/Cardiotocography>

UCI 

Machine Learning Repository
Center for Machine Learning and Intelligent Systems

[About](#) [Citation Policy](#) [Donate a Data Set](#) [Contact](#)

Repository Web Google [Search](#)

[View ALL Data Sets](#)

Cardiotocography Data Set

Download: [Data Folder](#) [Data Set Description](#)

Abstract: The dataset consists of measurements of fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms classified by expert obstetricians.

Data Set Characteristics:	Multivariate	Number of Instances:	2126	Area:	Life
Attribute Characteristics:	Real	Number of Attributes:	23	Date Donated	2010-09-07
Associated Tasks:	Classification	Missing Values?	N/A	Number of Web Hits:	96768

Source:

Marques de Sá, J.P., [jpmdesa '@' gmail.com](mailto:jpmdesa@gmail.com), Biomedical Engineering Institute, Porto, Portugal.
Bernardes, J., [joaobern '@' med.up.pt](mailto:joaobern@med.up.pt), Faculty of Medicine, University of Porto, Portugal.
Ayres de Campos, D., [sisporto '@' med.up.pt](mailto:sisporto@med.up.pt), Faculty of Medicine, University of Porto, Portugal.

Data Set Information:

2126 fetal cardiotocograms (CTGs) were automatically processed and the respective diagnostic features measured. The CTGs were also classified by three expert obstetricians and a consensus classification label assigned to each of them. Classification was both with respect to a morphologic pattern (A, B, C, ...) and to a fetal state (N, S, P). Therefore the dataset can be used either for 10-class or 3-class experiments.

R Exercise: Classification

- Dataset: Cardiotocography Data Set

Input variables

LB - FHR baseline (beats per minute)
AC - # of accelerations per second
FM - # of fetal movements per second
UC - # of uterine contractions per second
DL - # of light decelerations per second
DS - # of severe decelerations per second
DP - # of prolonged decelerations per second
ASTV - percentage of time with abnormal short term variability
MSTV - mean value of short term variability
ALTV - percentage of time with abnormal long term variability
MLTV - mean value of long term variability
Width - width of FHR histogram
Min - minimum of FHR histogram
Max - Maximum of FHR histogram
Nmax - # of histogram peaks
Nzeros - # of histogram zeros
Mode - histogram mode
Mean - histogram mean
Median - histogram median
Variance - histogram variance
Tendency - histogram tendency
CLASS - FHR pattern class code (1 to 10)

Target variable

NSP - fetal state class code (N=normal; S=suspect; P=pathologic)

R Exercise: Classification

- Performance evaluation function

```
# Part 1: Multi-class classification with ANN & Multinomial logistic regression
# Performance evaluation function for multi-class classification -----
perf_eval_multi <- function(cm){
  # Simple Accuracy
  ACC = sum(diag(cm))/sum(cm)
  # Balanced Correction Rate
  BCR = 1
  for (i in 1:dim(cm)[1]){
    BCR = BCR*(cm[i,i]/sum(cm[i,]))
  }
  BCR = BCR^(1/dim(cm)[1])
  return(c(ACC, BCR))
}
```

✓ perf_eval_multi

- Argument: confusion matrix (cm)
- Outputs: simple accuracy (ACC), balanced correction rate (BCR)

R Exercise: Classification

- Install package

```
# Install nnet package and prepare it
install.packages("nnet")
library(nnet)
```

✓ “nnet” package

- Many packages provide neural network module
- “nnet” is one of the most widely used packages

R Exercise: Classification

- Data loading and preprocessing

```
# Multi-class classification (ctgs dataset)
ctgs_data <- read.csv("ctgs.csv")
n_instance <- dim(ctgs_data)[1]
n_var <- dim(ctgs_data)[2]

# Conduct normalization
ctgs_input <- ctgs_data[,-n_var]
ctgs_target <- ctgs_data[,n_var]
ctgs_input <- scale(ctgs_input, center = TRUE, scale = TRUE)
ctgs_target <- as.factor(ctgs_target)
ctgs_data_normalized <- data.frame(ctgs_input, Class = ctgs_target)
```

✓ Data loading

- `read.csv()` function, use `header = TRUE` option because the first row is the variable name
- Use `dim()` function to check the number of rows and columns in the dataset

✓ Data normalization

- Make the average and standard deviation of each variable to 0 and 1, respectively
- Convert the target variable type to “factor”

R Exercise: Classification

- Data loading and preprocessing

```
# Initialize performance matrix
perf_summary <- matrix(0, nrow = 2, ncol = 2)
colnames(perf_summary) <- c("ACC", "BCR")
rownames(perf_summary) <- c("Multi_Logit", "ANN")

# Split the data into the training/validation sets
set.seed(12345)
trn_idx <- sample(1:n_instance, round(0.8*n_instance))
ctgs_trn <- ctgs_data_normalized[trn_idx,]
ctgs_tst <- ctgs_data_normalized[-trn_idx,]
```

- ✓ Initialize the performance summary matrix
- ✓ Data partition
 - set.seed(): set the seed of random initialization
 - Use 80% for training and 20% for test

R Exercise: Classification

- Training the Multinomial Logistic Regression

```
# Multinomial logistic regression -----
# Train multinomial logistic regression
ml_logit <- multinom(Class ~ ., data = ctgs_trn)

# Check the coefficients
summary(ml_logit)
t(summary(ml_logit)$coefficients)
```

✓ Estimated regression coefficients

```
> t(summary(ml_logit)$coefficients)
      2          3
(Intercept) -4.694977008 -11.58608545
LB           -1.136169974   2.94827810
AC           -3.747793932  -3.44072166
FM            0.469688181   1.07087196
UC           -0.907147801  -1.07874916
DL            0.028822895   0.15122342
DS           -0.433749154   0.23945220
DP            1.479317937   1.36253728
ASTV          1.376230801   3.46304432
MSTV          -0.269653587  -1.37476523
ALTV          0.413875887   1.48408278
```

MLTV	-0.056153529	0.68602348
Width	-0.007595327	0.13962712
Min	0.327512978	0.55153110
Max	0.523037010	1.21168222
Nmax	0.328949181	-0.92033096
Nzeros	-0.154133544	0.36098681
Mode	-1.041006093	-0.05063308
Mean	4.336477367	-1.34224223
Median	-0.604921319	-4.03591917
Variance	1.178814582	1.99001476
Tendency	0.117217134	0.16513548

R Exercise: Classification

- Classify the Test Examples using the Multinomial Logistic Regression

```
# Predict the class label  
ml_logit_prey <- predict(ml_logit, newdata = ctgs_tst)  
cfmatrix <- table(ctgs_tst$Class, ml_logit_prey)  
cfmatrix  
  
perf_summary[1,] <- perf_eval_multi(cfmatrix)  
perf_summary
```

✓ Simple accuracy: 0.8941, Balanced correction rate: 0.7468

R Exercise: Classification

- Data transformation for MLR

```
# Artificial Neural Network -----  
# Train ANN  
ann_trn_input <- ctgs_trn[,-n_var]  
ann_trn_target <- class.ind(ctgs_trn[,n_var])
```

- ✓ For multi-class classification, the number of output nodes is the same as the number of classes
- ✓ Use class.ind() function to convert the target variable (factor type) to one-hot (1-of-C coding) vector

Class	C_1	C_2	C_3
1	1	0	0
2	0	1	0
3	0	0	1
1	1	0	0
2	0	1	0

R Exercise: Classification

- Search the best number of hidden nodes

```
# Find the best number of hidden nodes in terms of BCR
# Candidate hidden nodes
nH <- seq(from=5, to=30, by=5)
# 5-fold cross validation index
val_idx <- sample(c(1:5), dim(ann_trn_input)[1], replace = TRUE, prob = rep(0.2,5))
val_perf <- matrix(0, length(nH), 3)
```

- ✓ The best number of hidden node depends on the dataset
- ✓ Perform 5-fold cross validation
 - The range of hidden nodes: 5 ~ 30 (step by 5)
 - Initialize validation index
 - val_perf: store the performance for each number of hidden node

R Exercise: Classification

- Search the best number of hidden nodes

```
for (i in 1:length(nH)) {  
  cat("Training ANN: the number of hidden nodes:", nH[i], "\n")  
  eval_fold <- c()  
  for (j in c(1:5)) {  
    # Training with the data in (k-1) folds  
    tmp_trn_input <- ann_trn_input[which(val_idx != j),]  
    tmp_trn_target <- ann_trn_target[which(val_idx != j),]  
    tmp_nnet <- nnet(tmp_trn_input, tmp_trn_target, size = nH[i],  
                      decay = 5e-4, maxit = 500)
```

- ✓ Repeat the process for all candidate number of hidden node (i) and five folds (j)
 - If the val_idx is j, use the example for validation, otherwise use the example for training
- ✓ nnet(): function for training the neural network
 - Arg 1: input (X) of training data
 - Arg 2: target (y) of training data
 - Arg 3: number of hidden nodes
 - Arg 4 & 5: weight decay threshold, maximum number of epochs

R Exercise: Classification

- Search the best number of hidden nodes

```
# Evaluate the model with the remaining 1 fold
tmp_val_input <- ann_trn_input[which(val_idx == j),]
tmp_val_target <- ann_trn_target[which(val_idx == j),]
eval_fold <- rbind(eval_fold, cbind(max.col(tmp_val_target),
max.col(predict(tmp_nnet, tmp_val_input)))))

# Confusion matrix
cfm <- table(eval_fold[,1], eval_fold[,2])
# nH
val_perf[i,1] <- nH[i]
# Record the validation performance
val_perf[i,2:3] <- t(perf_eval_multi(cfm))
```

- ✓ Combine the prediction results for 5 validation sets using rbind() and evaluate it together

R Exercise: Classification

- Search the best number of hidden nodes

```
ordered_val_perf <- val_perf[order(val_perf[,3], decreasing = TRUE),]  
colnames(ordered_val_perf) <- c("nH", "ACC", "BCR")  
ordered_val_perf  
# Find the best number of hidden node  
best_nH <- ordered_val_perf[1,1]
```

- ✓ Find the best hidden node in terms of BCR

- Different results can be obtained for different trials

```
> ordered_val_perf  
    nH      ACC      BCR  
[1,] 30 0.9112287 0.8260219  
[2,] 25 0.9118166 0.8226941  
[3,] 20 0.9135802 0.8094907  
[4,] 15 0.9012346 0.8008131  
[5,] 10 0.8924162 0.7813366  
[6,]  5 0.8847737 0.7753846
```

R Exercise: Classification

- Model training with the best number of hidden nodes

```
# Test the ANN
ann_tst_input = ctgs_tst[,-n_var]
ann_tst_target = class.ind(ctgs_tst[,n_var])
ctgs_nnet <- nnet(ann_trn_input, ann_trn_target, size = best_nH,
                  decay = 5e-4, maxit = 500)
```

- ✓ Use the best number of hidden nodes determined by the 5-fold cross validation to train the entire training dataset

R Exercise: Classification

- Evaluate the MLR and compare the results with the multinomial logistic regression

```
# Performance evaluation
prey <- predict(ctgs_nnet, ann_tst_input)
tst_cm <- table(max.col(ann_tst_target), max.col(prey))
tst_cm
perf_summary[2,] <- perf_eval_multi(tst_cm)
perf_summary
```

```
> cfmatrix
  m1_logit_prey
    1   2   3
1 318 14  0
2 18  41  1
3  4   8  21
```

```
> tst_cm
      1   2   3
1 321 11  0
2 20  40  0
3  4   5  24
```

```
> perf_summary
          ACC      BCR
Multi_Logit 0.8941176 0.7468081
ANN         0.9058824 0.7768270
```

R Exercise: Regression

- Dataset: Concrete Compressive Strength Data Set
 - ✓ Predict the strength of concrete with different proportions of ingredients
 - ✓ <https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength>

UCI 

Machine Learning Repository
Center for Machine Learning and Intelligent Systems

[About](#) [Citation Policy](#) [Donate a Data Set](#) [Contact](#)

Search Repository Web [View ALL Data Sets](#)

[Google](#)

Concrete Compressive Strength Data Set

Download: [Data Folder](#) [Data Set Description](#)

Abstract: Concrete is the most important material in civil engineering. The concrete compressive strength is a highly nonlinear function of age and ingredients.



Data Set Characteristics:	Multivariate	Number of Instances:	1030	Area:	Physical
Attribute Characteristics:	Real	Number of Attributes:	9	Date Donated	2007-08-03
Associated Tasks:	Regression	Missing Values?	N/A	Number of Web Hits:	115632

Source:

Original Owner and Donor
Prof. I-Cheng Yeh
Department of Information Management
Chung-Hua University,
Hsin Chu, Taiwan 30067, R.O.C.
e-mail:icyeh@chu.edu.tw
TEL:886-3-5186511

Date Donated: August 3, 2007

Data Set Information:

Number of instances 1030
Number of Attributes 9
Attribute breakdown 8 quantitative input variables, and 1 quantitative output variable
Missing Attribute Values None

R Exercise: Regression

- Dataset: Concrete Compressive Strength Data Set

- ✓ Input and target variables

Cement (component 1) -- quantitative -- kg in a m3 mixture -- Input Variable
Blast Furnace Slag (component 2) -- quantitative -- kg in a m3 mixture -- Input Variable
Fly Ash (component 3) -- quantitative -- kg in a m3 mixture -- Input Variable
Water (component 4) -- quantitative -- kg in a m3 mixture -- Input Variable
Superplasticizer (component 5) -- quantitative -- kg in a m3 mixture -- Input Variable
Coarse Aggregate (component 6) -- quantitative -- kg in a m3 mixture -- Input Variable
Fine Aggregate (component 7) -- quantitative -- kg in a m3 mixture -- Input Variable
Age -- quantitative -- Day (1~365) -- Input Variable
Concrete compressive strength -- quantitative -- MPa -- Output Variable

R Exercise: Regression

- Compare MLR, k-NN, & ANN

- ✓ Performance evaluation function

```
# Part 2: Regression with MLR, k-NN, and ANN
# Performance evaluation function for regression -----
perf_eval_reg <- function(tgt_y, pre_y){
  # RMSE
  rmse <- sqrt(mean((tgt_y - pre_y)^2))
  # MAE
  mae <- mean(abs(tgt_y - pre_y))
  # MAPE
  mape <- 100*mean(abs((tgt_y - pre_y)/tgt_y))
  return(c(rmse, mae, mape))
}
```

- ✓ Args: target value, predicted value
 - ✓ Outputs: RMSE, MAE, MAPE

R Exercise: Regression

- Data loading and preprocessing

```
# Concrete strength data
concrete <- read.csv("concrete.csv", header = FALSE)
n_instance <- dim(concrete)[1]
n_var <- dim(concrete)[2]
RegX <- concrete[,-n_var]
RegY <- concrete[,n_var]
# Data Normalization
RegX <- scale(RegX, center = TRUE, scale = TRUE)
# Combine X and Y
RegData <- as.data.frame(cbind(RegX, RegY))
```

- ✓ Use `read.csv()` function
 - Use `header = FALSE` option because the first row is not the name of variable
- ✓ Store the number of instances and variables
- ✓ Perform normalization for input variables
- ✓ Combine the normalized input variables and target variable for modeling

R Exercise: Regression

- Data loading and preprocessing

```
# Split the data into the training/test sets
set.seed(12345)
trn_idx <- sample(1:n_instance, round(0.7*n_instance))
trn_data <- RegData[trn_idx,]
tst_data <- RegData[-trn_idx,]
perf_summary_reg <- matrix(0,3,3)
rownames(perf_summary_reg) <- c("MLR", "k-NN", "ANN")
colnames(perf_summary_reg) <- c("RMSE", "MAE", "MAPE")
```

- ✓ Data partitioning: 70% for training 30% for test
- ✓ Initialize the performance summary table
 - Algorithms: MLR, k-NN, ANN
 - Metrics: RMSE, MAE, MAPE

R Exercise: Regression

- Training and Evaluating MLR

```
# Multiple linear regression
full_model <- lm(RegY ~ ., data = trn_data)
mlr_prey <- predict(full_model, newdata = tst_data)
perf_summary_reg[1,] <- perf_eval_reg(tst_data$RegY, mlr_prey)
perf_summary_reg
```

- ✓ Train the MLR with all variables

```
> perf_summary_reg
      RMSE      MAE      MAPE
MLR  10.44926 8.166065 30.85209
k-NN  0.00000 0.000000  0.00000
ANN   0.00000 0.000000  0.00000
```

R Exercise: Regression

- Training and Evaluating the k-NN

```
# Evaluate the k-NN with the test data
# k-Nearest Neighbor Learning (Regression) -----
install.packages("FNN", dependencies = TRUE)
library(FNN)

knn_reg <- knn.reg(trn_data[,-n_var], test = tst_data[,-n_var], trn_data$RegY, k=3)
knn_prey <- knn_reg$pred
perf_summary_reg[2,] <- perf_eval_reg(tst_data$RegY, knn_prey)
perf_summary_reg
```

✓ MAE is decreased by 1.7%p, MAPE is decreased by 7.5%p compared to MLR

	> perf_summary_reg		
	RMSE	MAE	MAPE
MLR	10.449259	8.166065	30.85209
k-NN	8.452958	6.462481	23.30499
ANN	0.000000	0.000000	0.00000

R Exercise: Regression

- Search the best number of hidden nodes

```
# Find the best number of hidden nodes in terms of BCR
# Candidate hidden nodes
nH <- seq(from=2, to=20, by=2)
# 5-fold cross validation index
val_idx <- sample(c(1:5), length(trn_idx), replace = TRUE, prob = rep(0.2,5))
val_perf <- matrix(0, length(nH), 4)
...
for (i in 1:length(nH)) {
  ...
  for (j in c(1:5)) {
    tmp_nnet <- nnet(RegY ~ ., data = tmp_trn_data, size = nH[i],
                      linout = TRUE, decay = 5e-4, maxit = 500)
  ...
}
```

- ✓ Perform the 5-fold cross validation by varying the number of hidden nodes from 2 to 20 with the step size of 2
- ✓ (Note) linout = TRUE option must be set to solve a regression problem

R Exercise: Regression

- Search the best number of hidden nodes

```
ordered_val_perf <- val_perf[order(val_perf[,3], decreasing = FALSE),]  
colnames(ordered_val_perf) <- c("nH", "RMSE", "MAE", "MAPE")  
ordered_val_perf  
  
# Find the best number of hidden node  
best_nH <- ordered_val_perf[1,1]
```

```
> ordered_val_perf  
    nH      RMSE      MAE      MAPE  
[1,] 14 7.258657 5.028841 17.65126  
[2,] 20 6.933016 5.090388 17.76599  
[3,] 12 6.952557 5.093377 18.49242  
[4,] 16 6.975732 5.152586 18.37121  
[5,] 10 6.839858 5.161052 18.68690  
[6,]  8 7.298235 5.253384 18.67945  
[7,] 18 7.966798 5.494715 20.98212  
[8,]  4 7.290364 5.613038 19.26955  
[9,]  6 7.325283 5.643885 19.42178  
[10,] 2 9.277461 6.734610 22.96352
```

R Exercise: Regression

- Training and Evaluating ANN

```
# Test the model and compare the performance
ann_prey <- predict(best_nnet, tst_data[,-n_var])
perf_summary_reg[3,] <- perf_eval_reg(tst_data$RegY, ann_prey)
perf_summary_reg
```

- ✓ ANN resulted in the lowest error rate among the three regression algorithms

```
> perf_summary_reg
      RMSE      MAE      MAPE
MLR  10.449259 8.166065 30.85209
k-NN  8.452958 6.462481 23.30499
ANN   7.056971 4.950896 16.08717
```



ANY
questions?