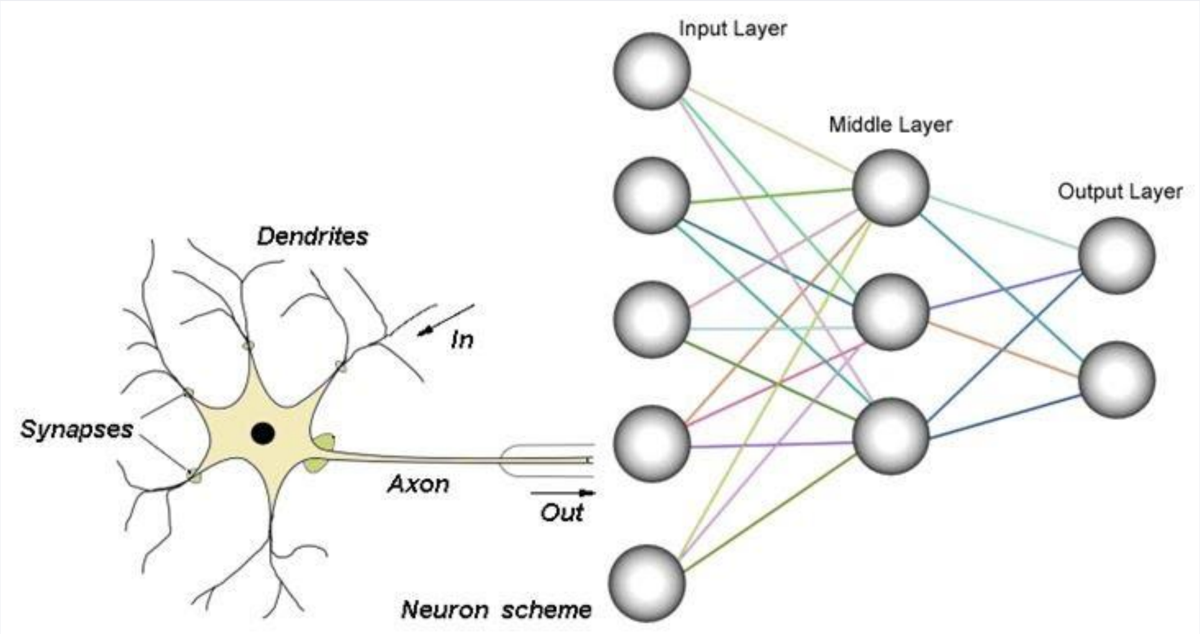


Lecture 10: Artificial Neural Networks

Pilsung Kang

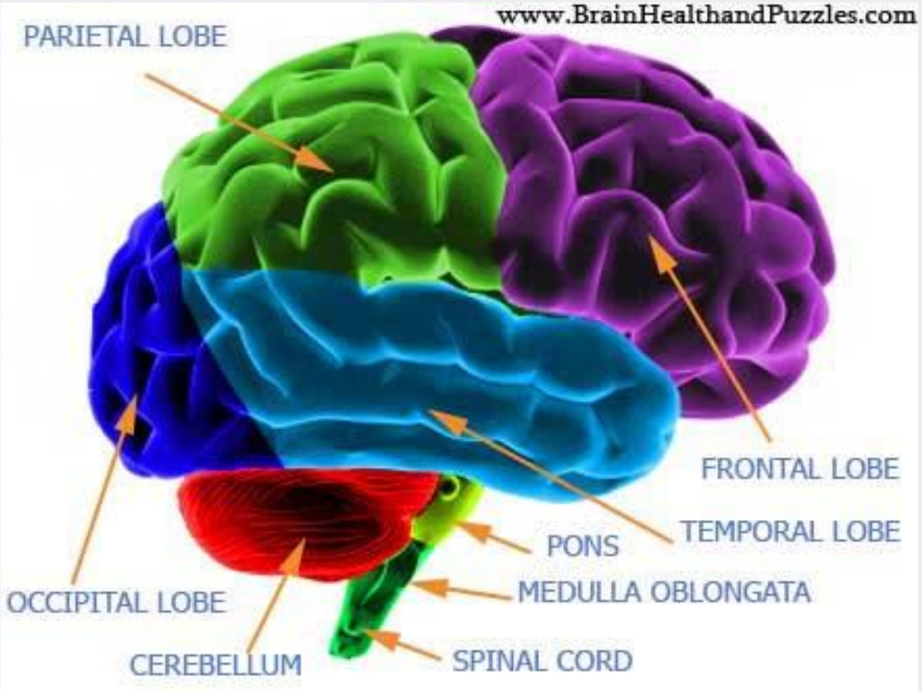
School of Industrial Management Engineering

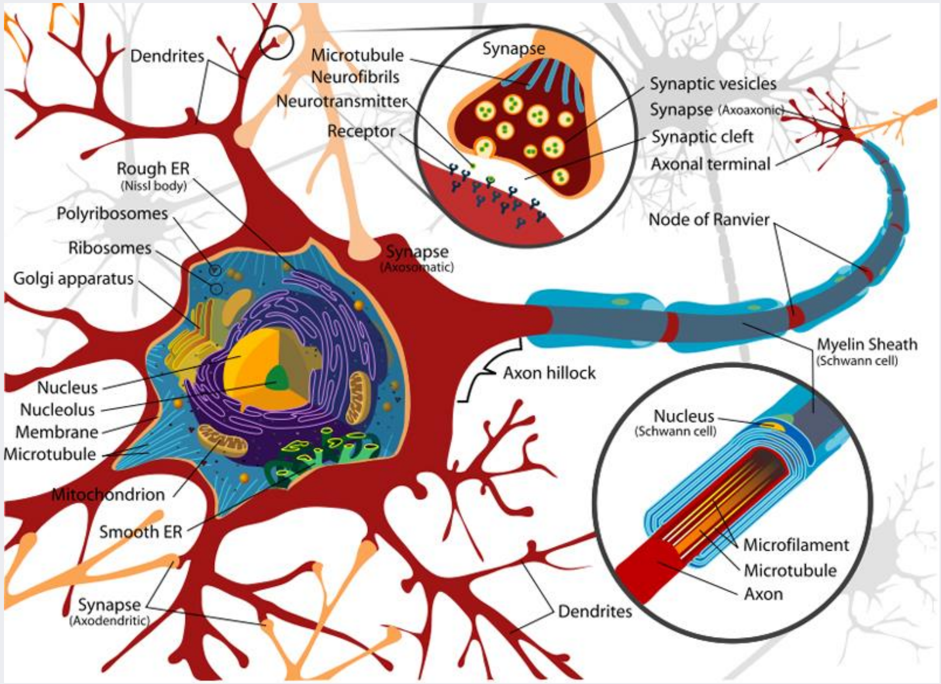
Korea University

AGENDA

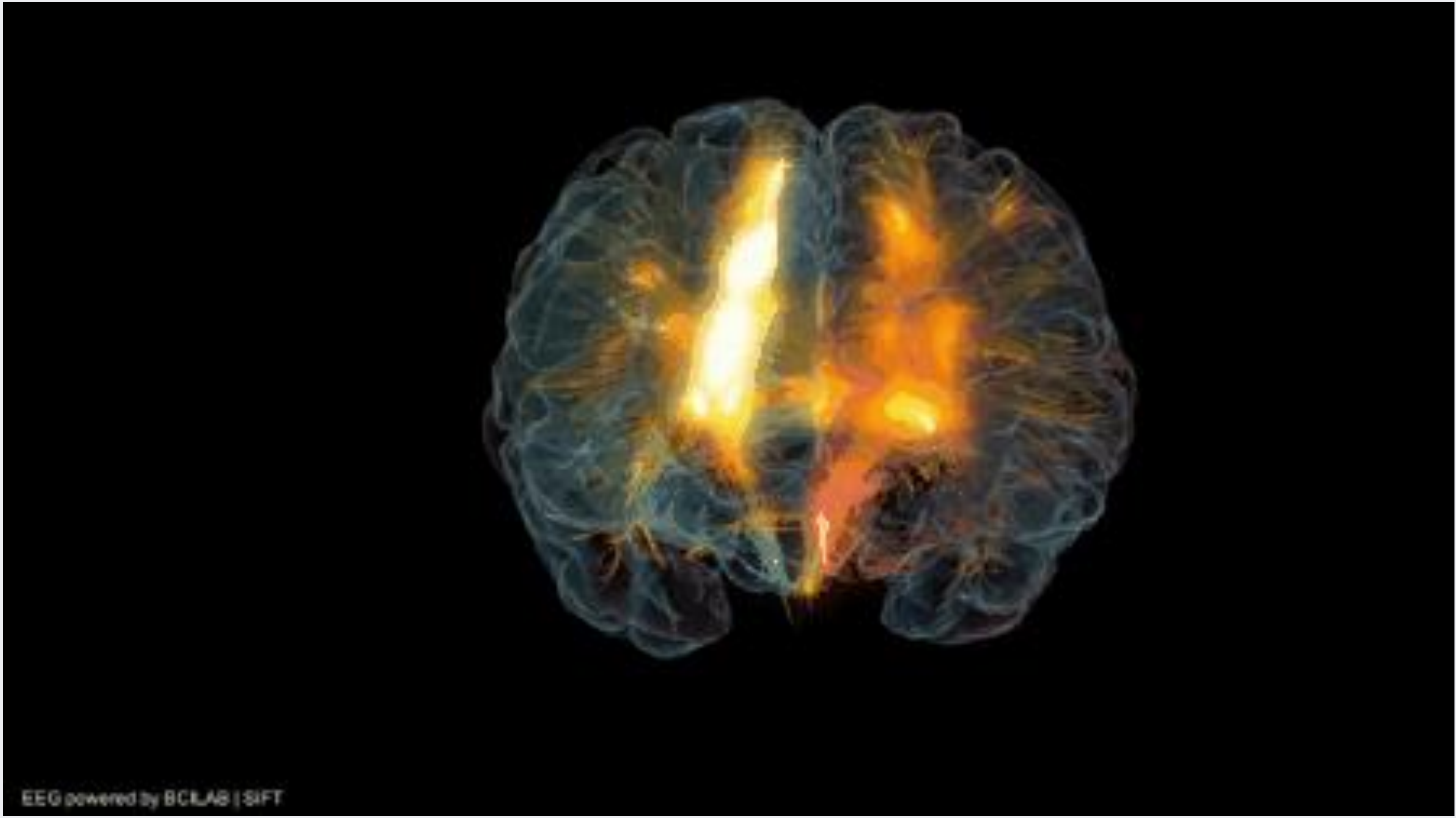
Brain Structure

- 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





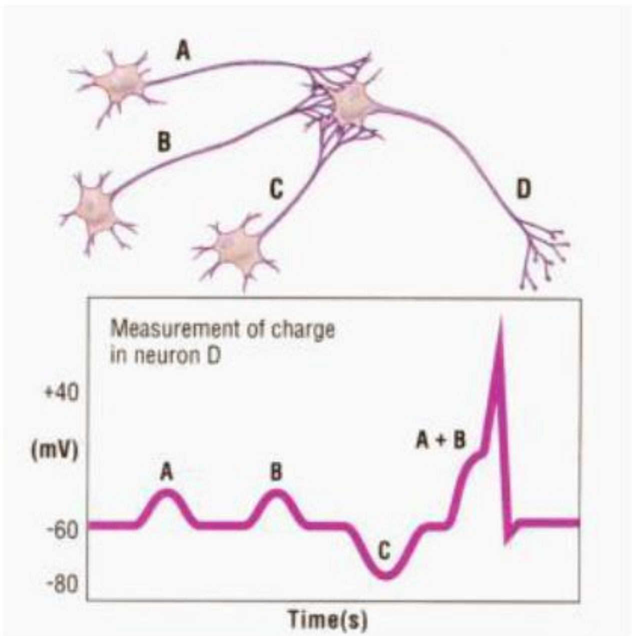
Neuron Firing Off in Real-Time

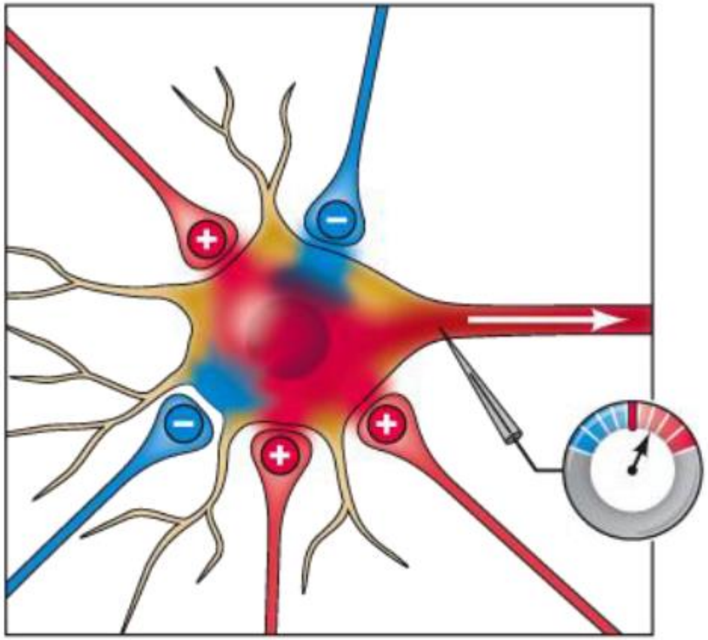


http://www.dailymail.co.uk/sciencetech/article-2581184/The-dynamic-mind-Stunning-3D-glass-brain-shows-neurons-firing-real-time.html

The Way Our Brain Works

- The way neurons work



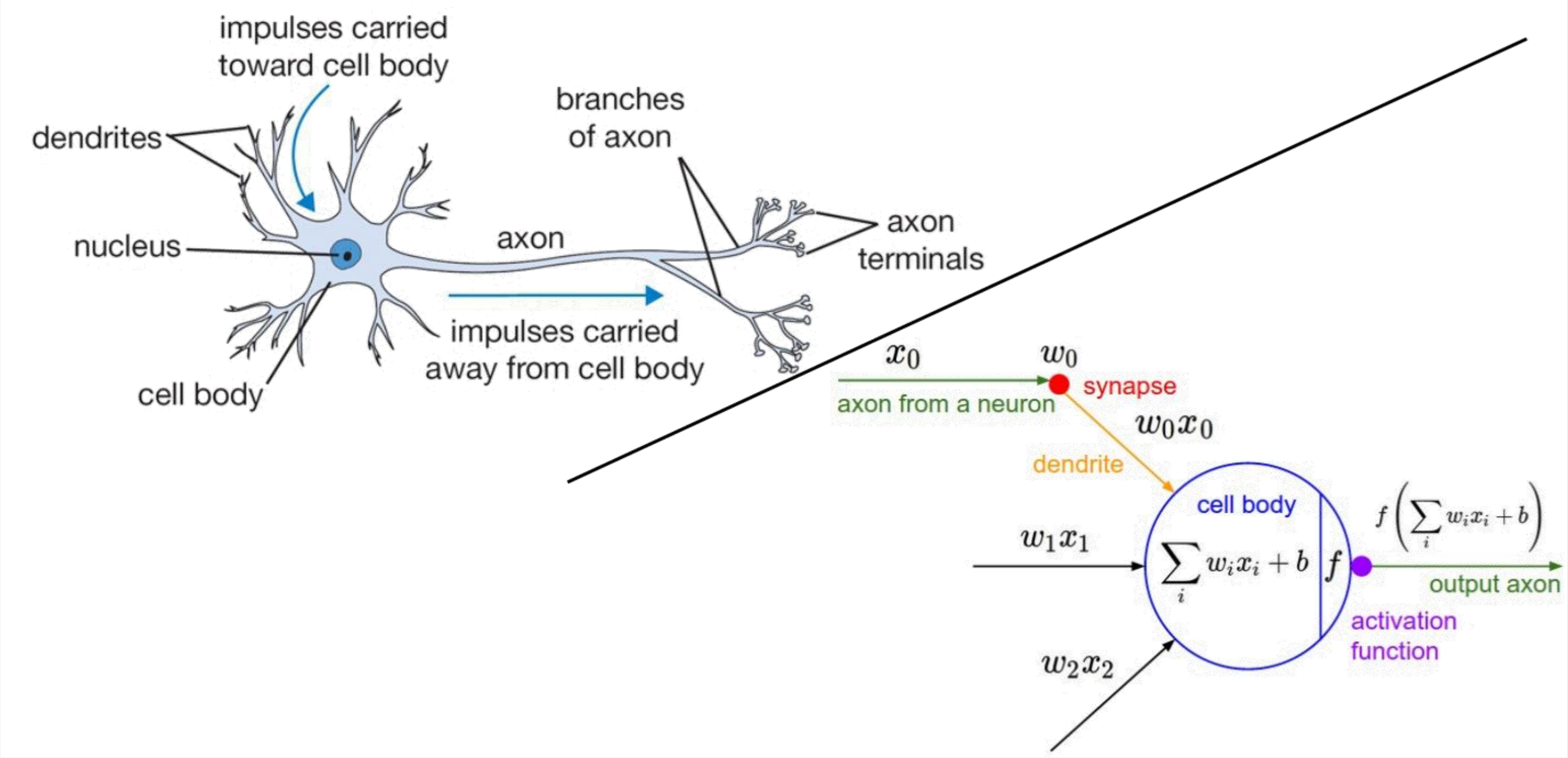


Neurons receive signals

- ✓ Combine those signals – **sum** the stimulus
- ✓ Fire if the stimulus exceed the **threshold**

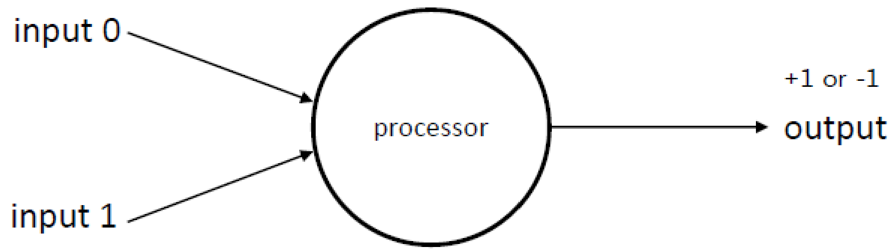
Perceptron

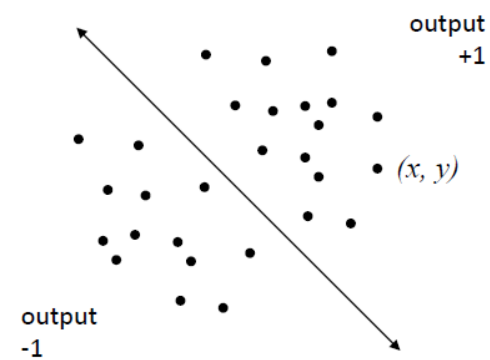
- Imitate a single neuron



Perceptron

- Perceptron
  - ✓ Accept all stimulus





A perceptron consists of one or more inputs, a processor, and a single output.

Step 1: **Receive** inputs.

Step 2: **Weight** inputs.

Step 3: **Sum** inputs.

Step 4: **Generate** output.

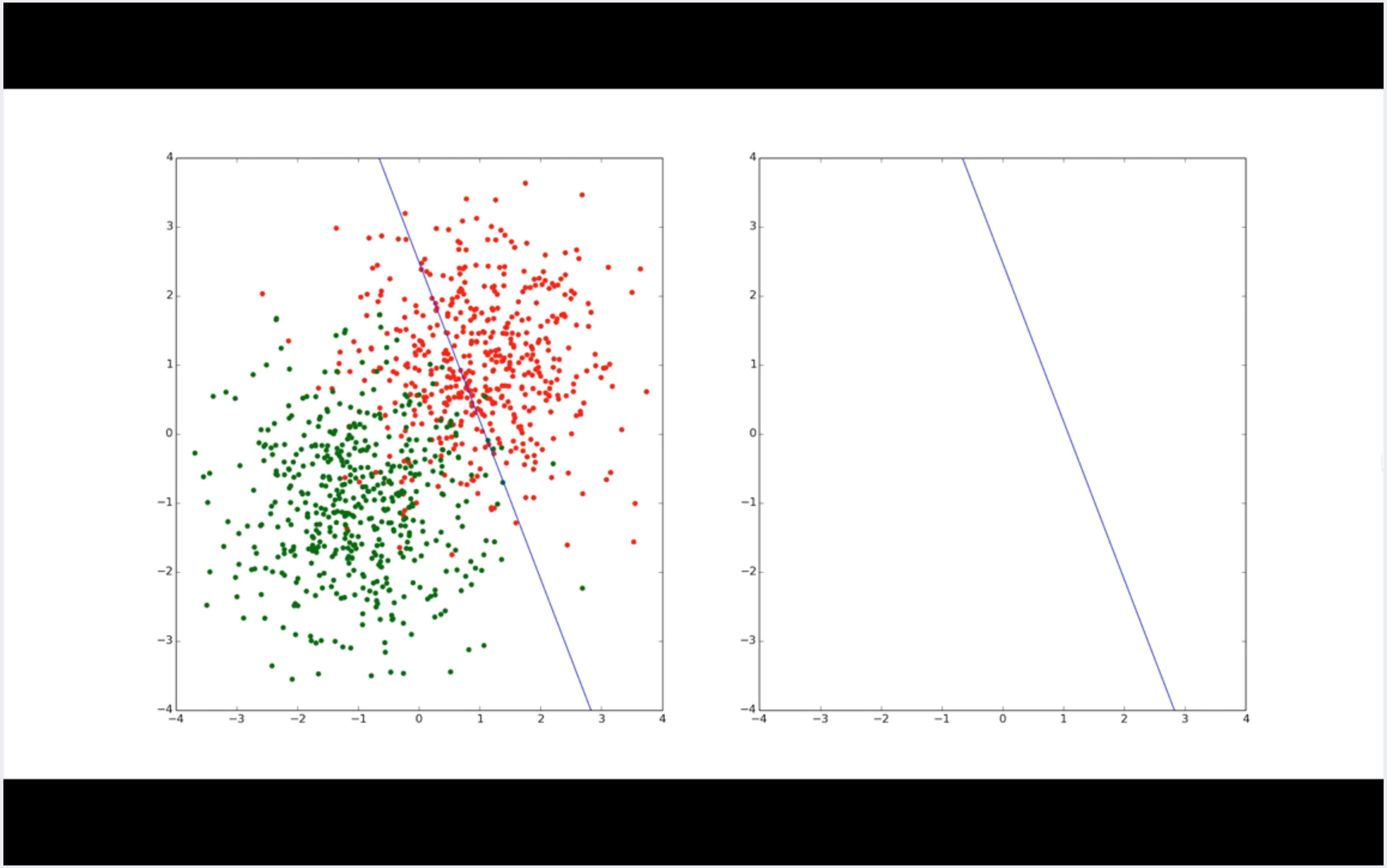
The Perceptron Algorithm:
1) For every input, multiply that input by its weight.
2) Sum all of the weighted inputs.
3) Compute the output of the perceptron based on that sum passed through an activation function (the sign of the sum).

$$\text{Sum} = W_0 * \text{input}_0 + W_1 * \text{input}_1$$

```
if (sum > 0) return 1; else return -1;
```

Perceptron

- Training Perceptron

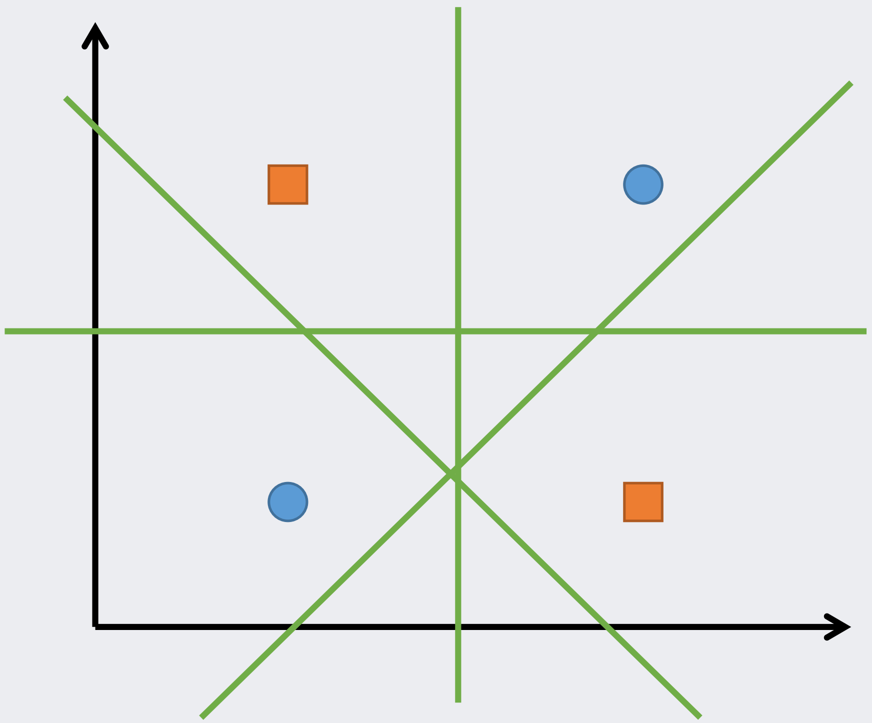


Limitation of Perceptron

- 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***

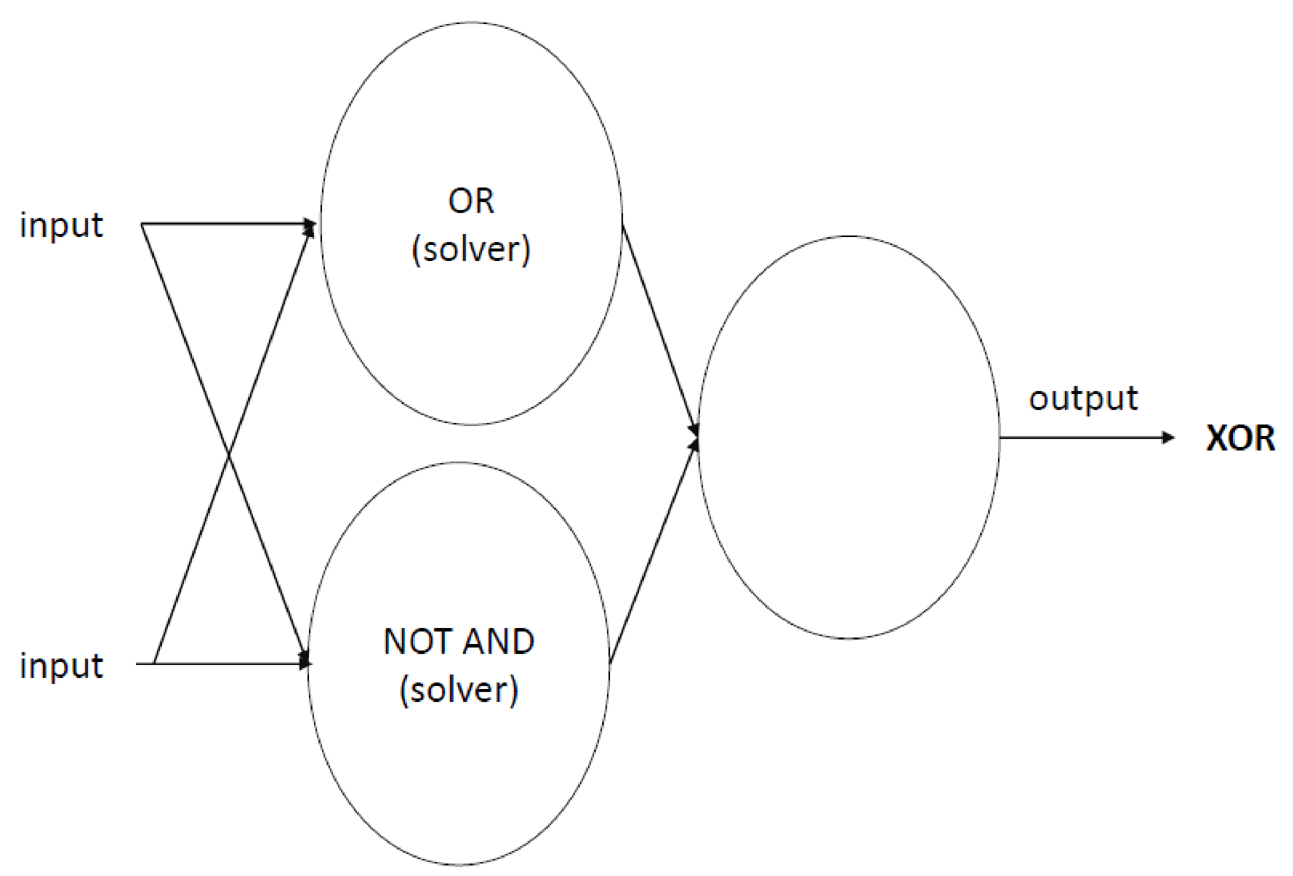
Limitation of Perceptron

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



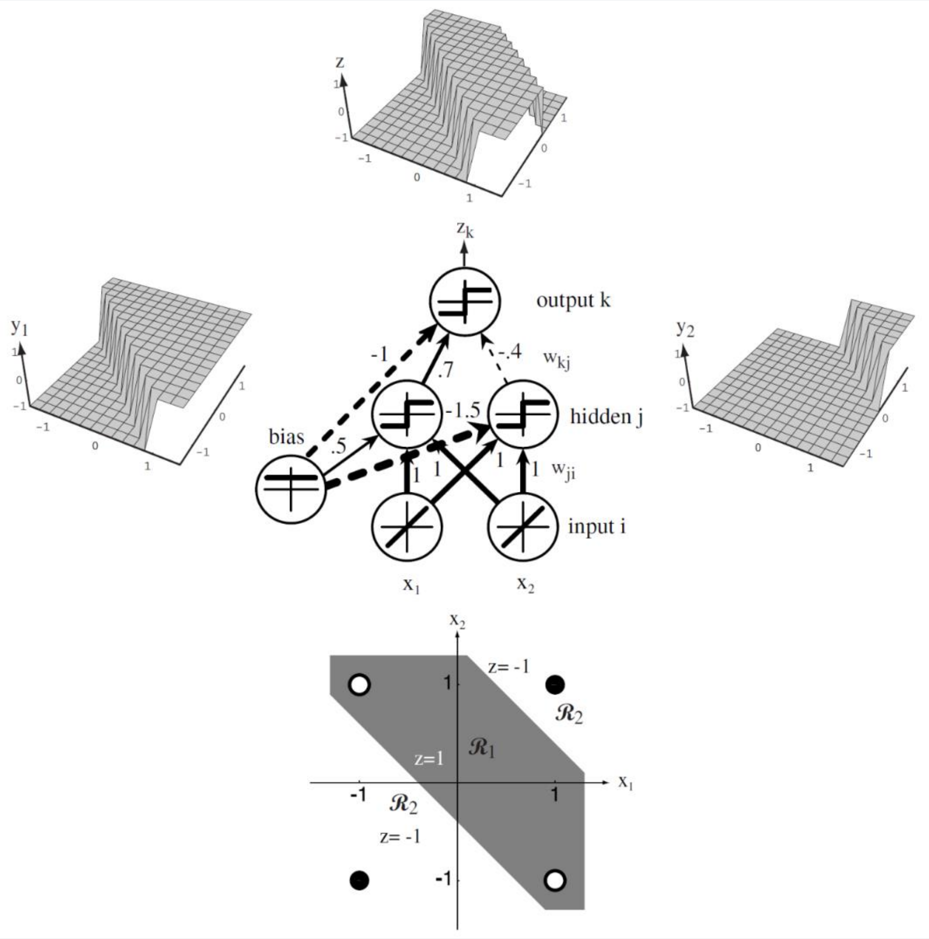
Multi-Layered Perceptron

- 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!



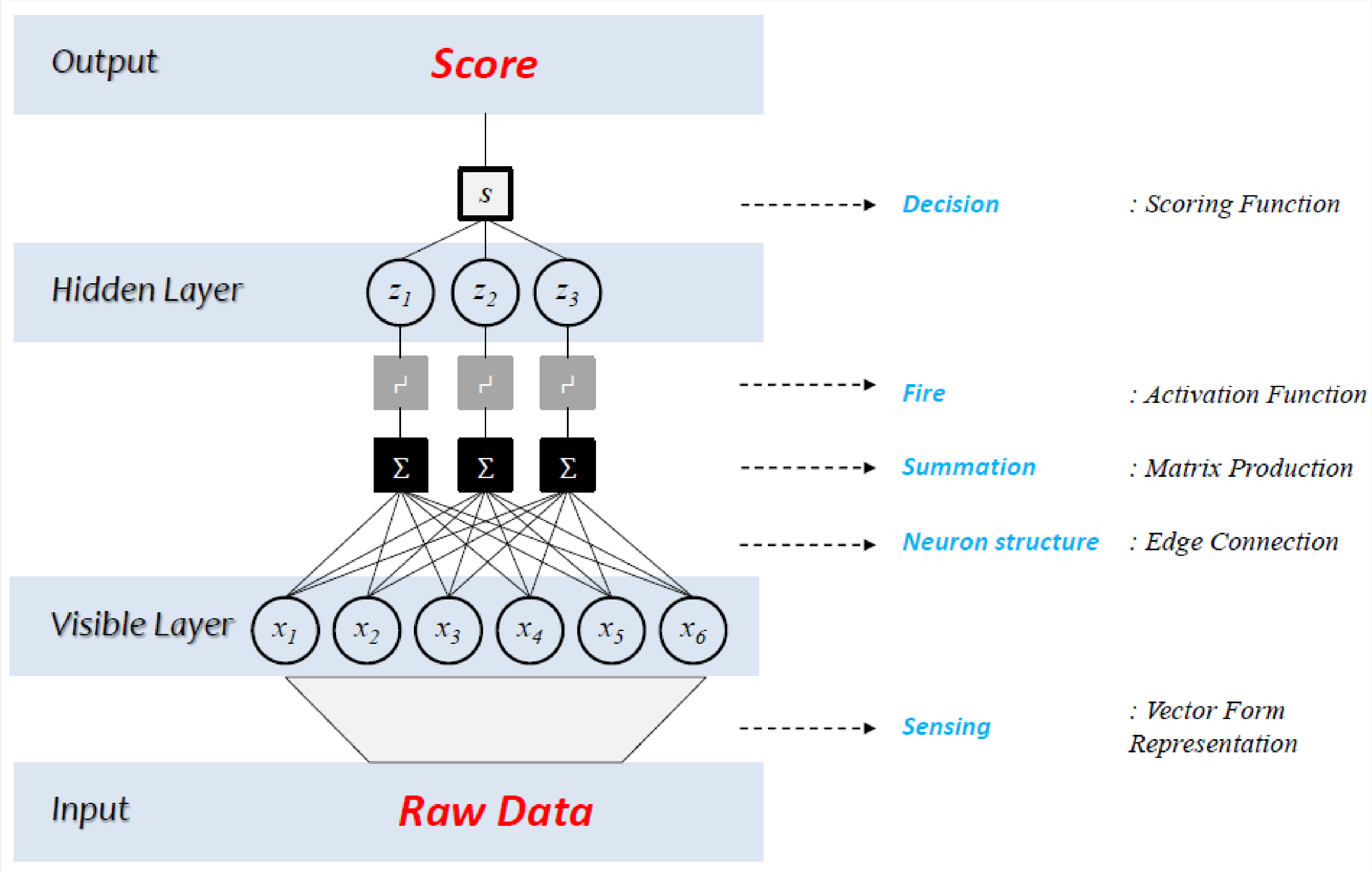
Multi-Layered Perceptron

- Non-linear model
  - ✓ Can find an arbitrary shape of class boundary or regression functions



Artificial Neural Network: Structure

- Feed-forward neural network with 1 hidden layer



Artificial Neural Network: Structure

- Feed-forward neural network with 1 hidden layer

- Hidden layer pre-activation:

$$\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

$$\left(a(\mathbf{x})_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$$

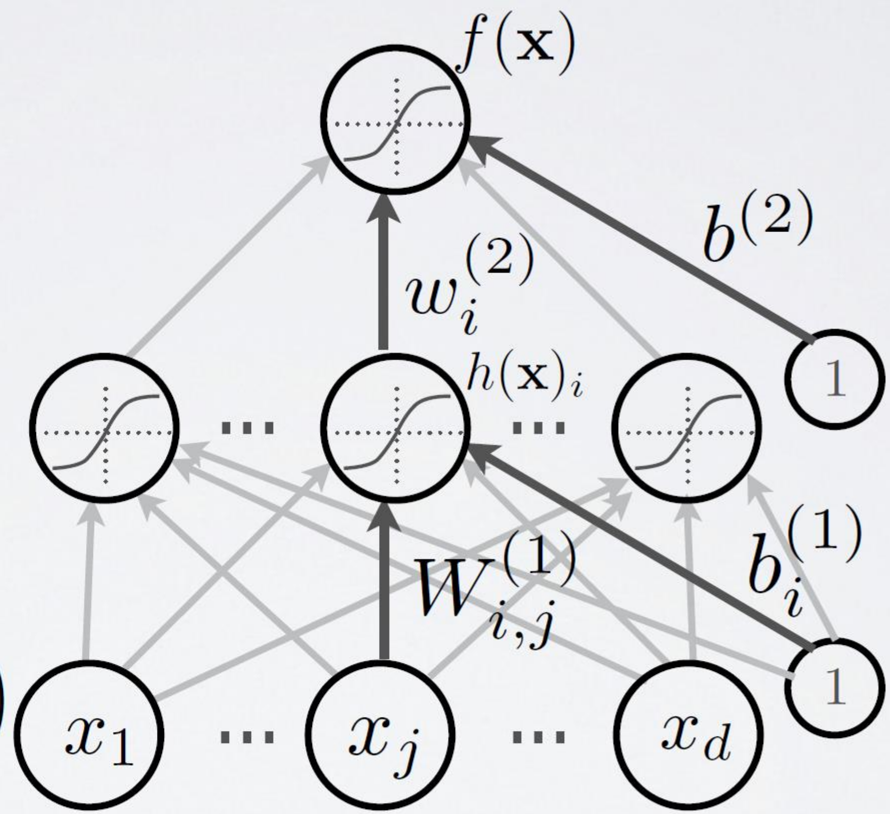
- Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

- Output layer activation:

$$f(\mathbf{x}) = o\left(b^{(2)} + \mathbf{w}^{(2)\top}\mathbf{h}^{(1)}\mathbf{x}\right)$$

output activation function



Artificial Neural Network: Activation Function

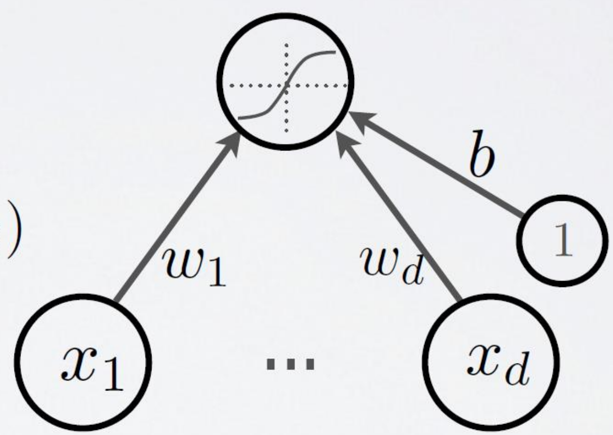
- Activation Function
  - ✓ Help to find non-linear decision boundary (classification) or regression function (regression)

- 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

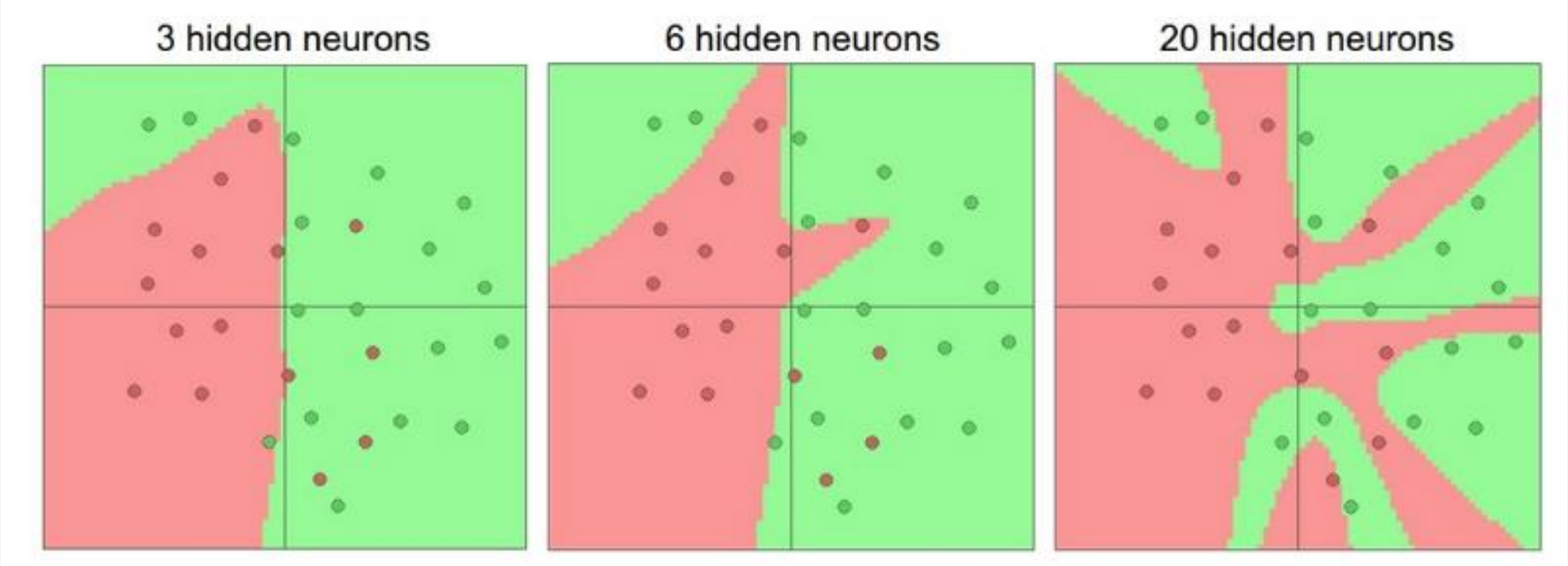
$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$



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

Artificial Neural Network: Hidden Nodes

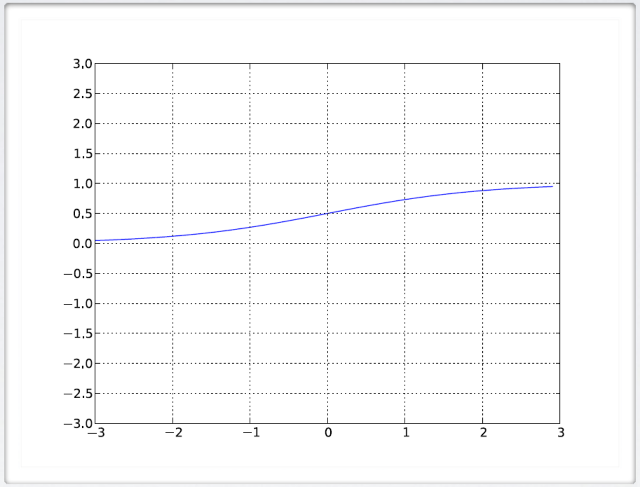
- The role of hidden nodes
  - ✓ Determines the complexity of ANN
  - ✓ If we use more number of hidden nodes, we can find a more sophisticated decision boundary (classification) or an arbitrary shape of function (regression)



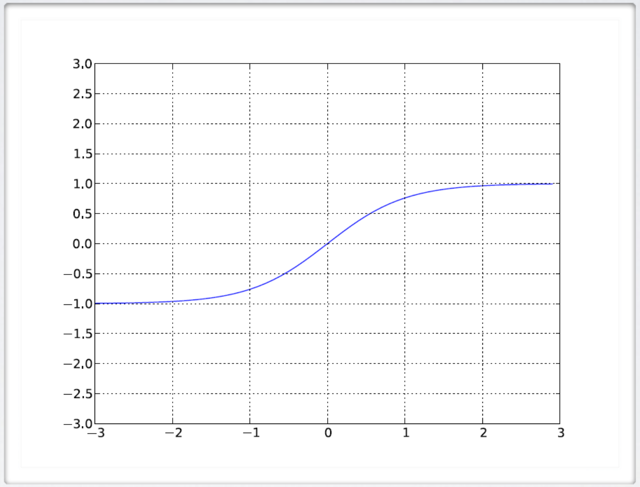
Artificial Neural Network: Activation Function

- Activation Function
  - ✓ Help to find non-linear decision boundary (classification) or regression function (regression)

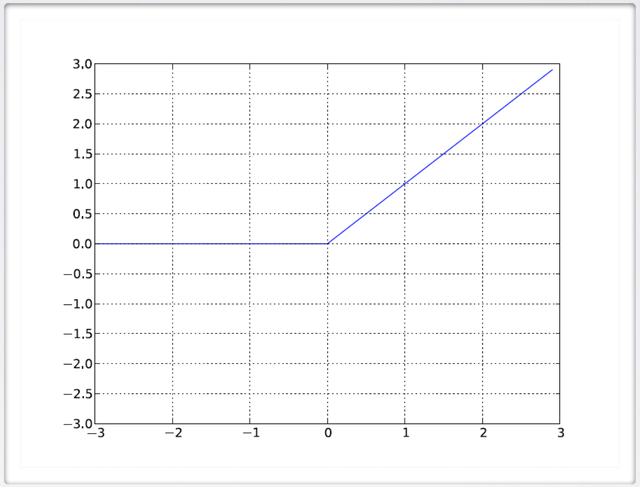
$g(a) = \mathrm{sigm}(a) = \frac{1}{1+\exp(-a)}$



$g(a) = \tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} = \frac{\exp(2a)-1}{\exp(2a)+1}$

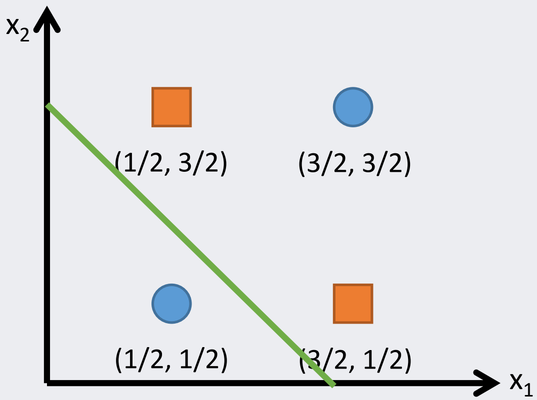


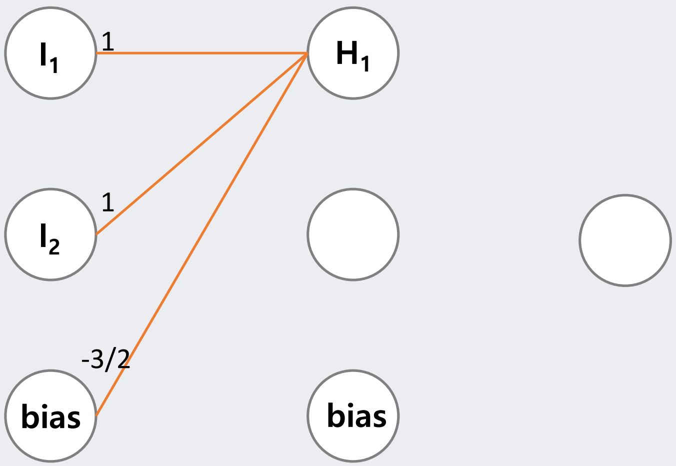
$g(a) = \mathrm{reclin}(a) = \max(0, a)$



Neural Networks: How it works?

- XOR problem revisited





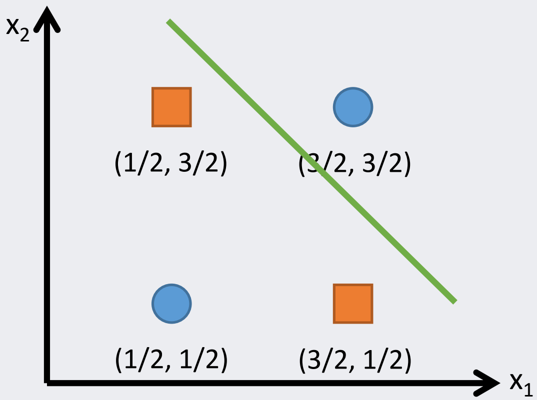
$$h_1 = x_1 + x_2 - \frac{3}{2}$$

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

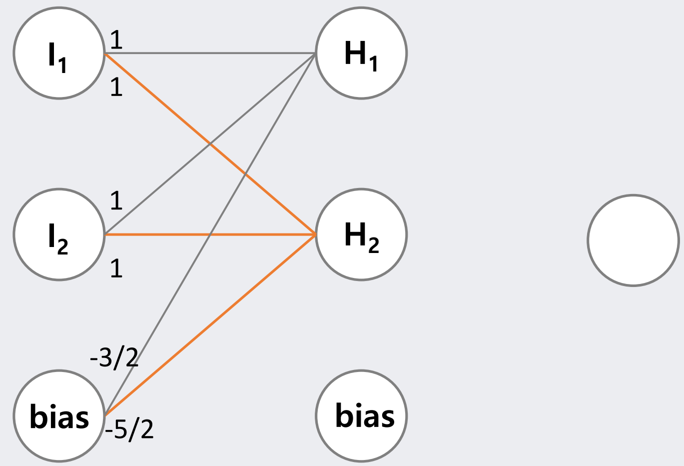
	x_1	x_2	h_1	z_1
●	1/2	1/2	-1/2	-1
■	3/2	1/2	1/2	1
■	1/2	3/2	1/2	1
●	3/2	3/2	3/2	1

Neural Networks: How it works?

- XOR problem revisited (cont')



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

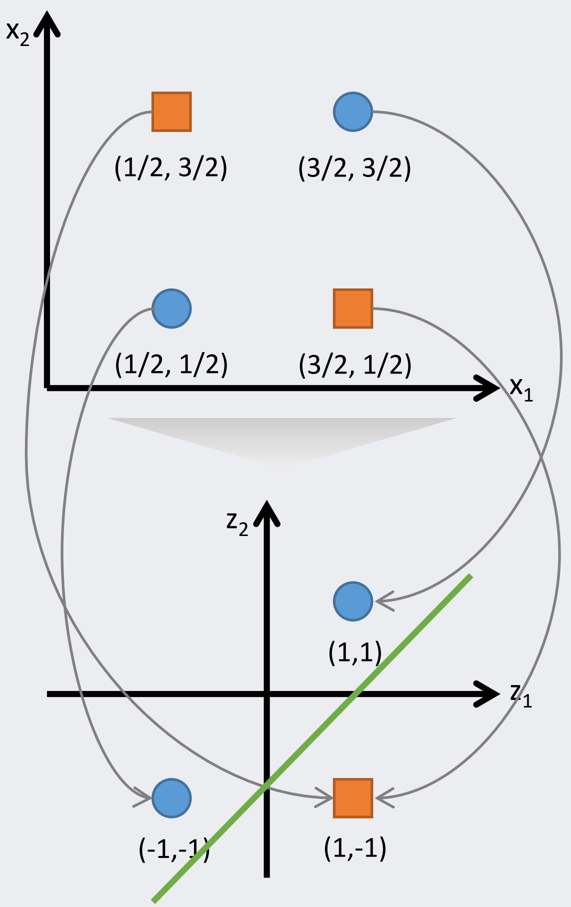


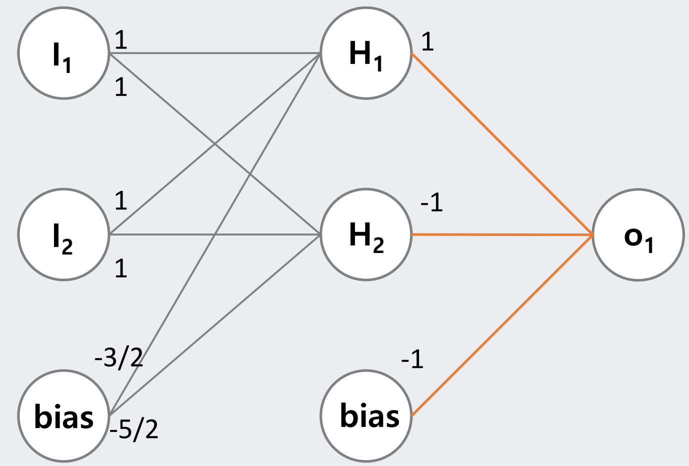
$$z_2 = g(h_2) = \begin{cases} 1 & if\ h_2 \geq 0 \\ -1 & if\ h_2 < 0 \end{cases}$$

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

Neural Networks: How it works?

- XOR problem revisited (cont')





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

$$o_1 = z_1 - z_2 - 1$$

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

Neural Networks: 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:

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

Neural Network: Training

- Gradient descent algorithm
  - ✓ Taylor expansion of a function

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

  - ✓ 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

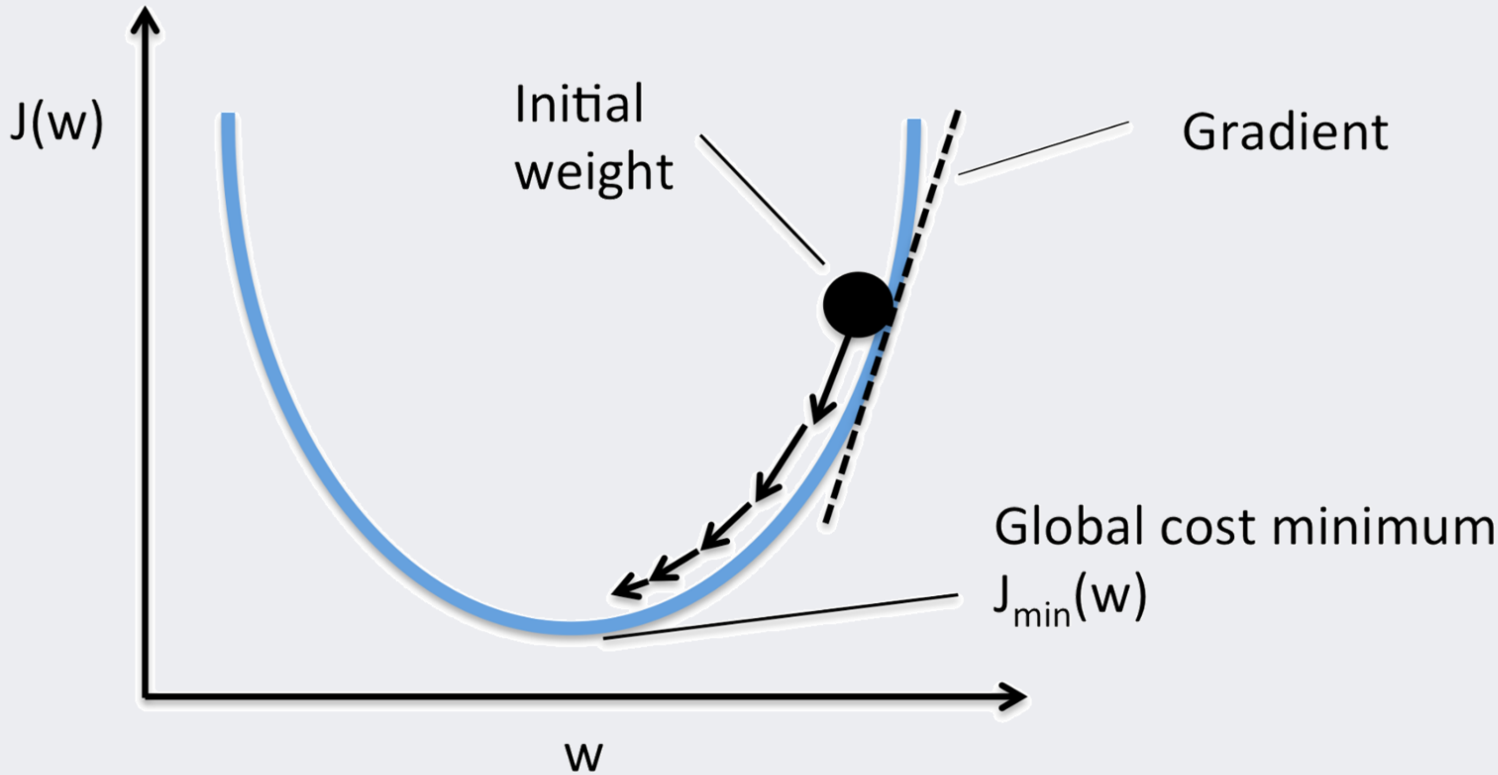
$$x_{new} = x_{old} - \eta f'(x), \quad where \;\; 0 < \eta < 1$$

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

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

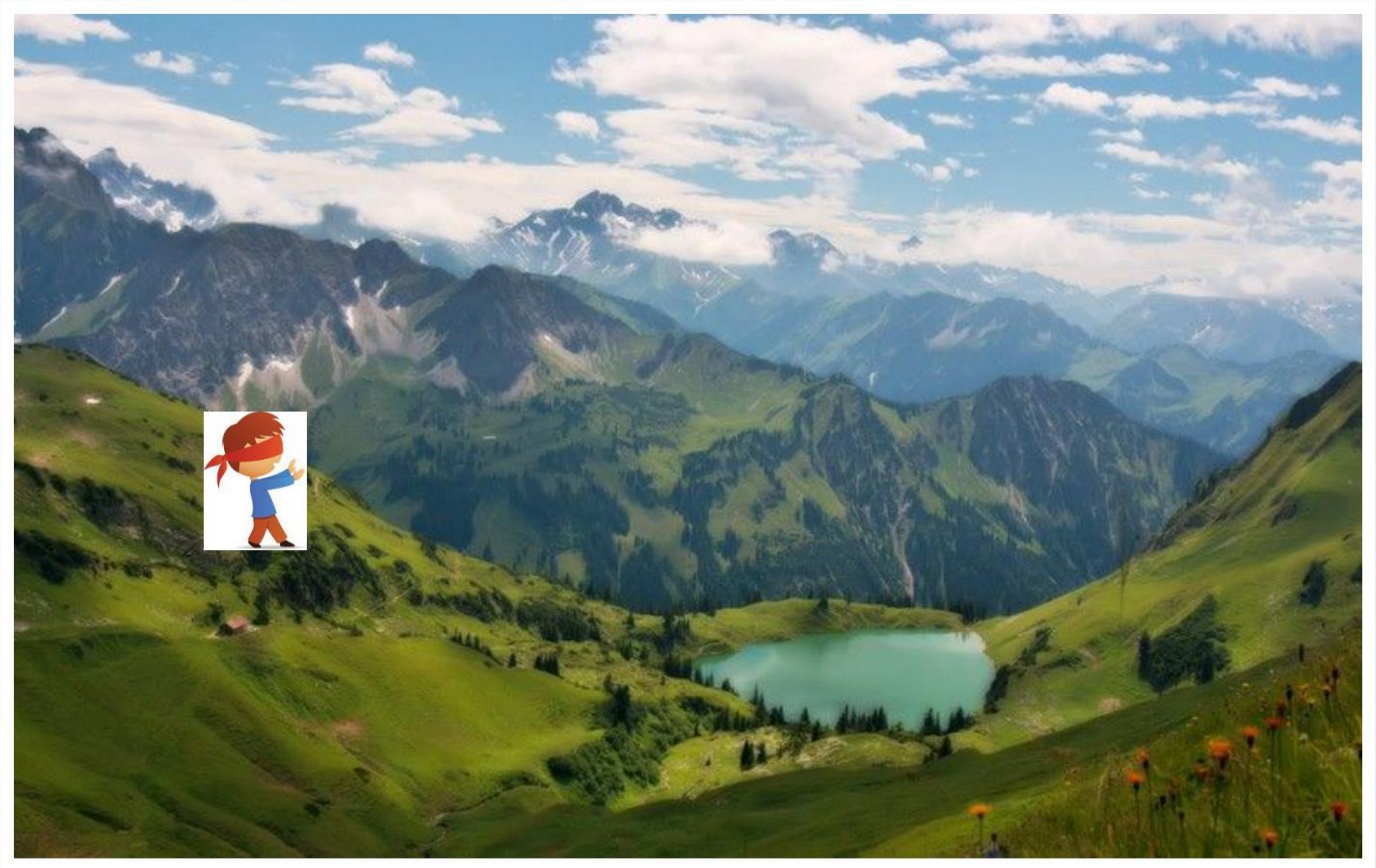
Neural Network: Training

- Gradient Descent Algorithm
  - ✓ Illustrative Example
    - Blue line: the value of the objective function w.r.t. w
    - Black point: current position
    - Arrow: the direction that w should move toward to minimize the objective function



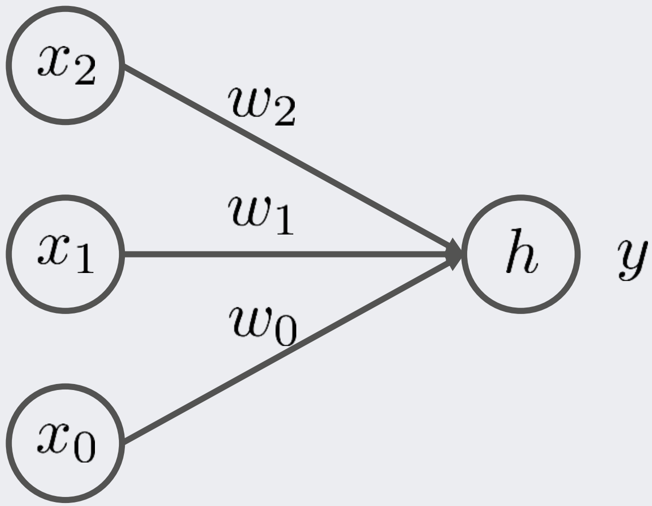
Neural Network: Training

- Gradient Descent Algorithm



Neural Network: Training

- A Simple Example (Logistic Regression with two input variables)



$$h = \sum_{i=0}^{2} w_i x_i$$

$$y = \frac{1}{1 + exp(-h)}$$

- Let's define the squared loss function $L = \frac{1}{2}(t - y)^2$
- How to find the gradient w.r.t. w or x?

Neural Network: Training

- Use chain rule

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

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

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

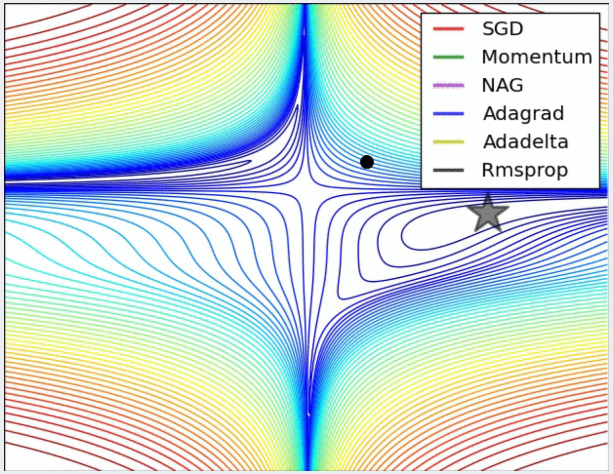
- Gradients for w and x

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

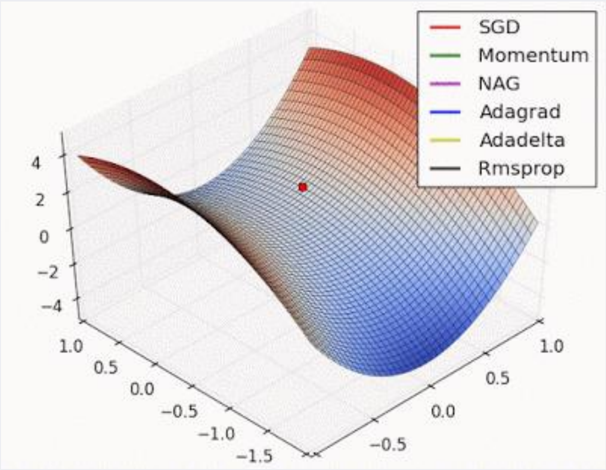
$$\frac{L}{\partial x_i} = \frac{L}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial x_i} = (y - t) \cdot y(1 - y) \cdot w_i$$

Neural Network: Training

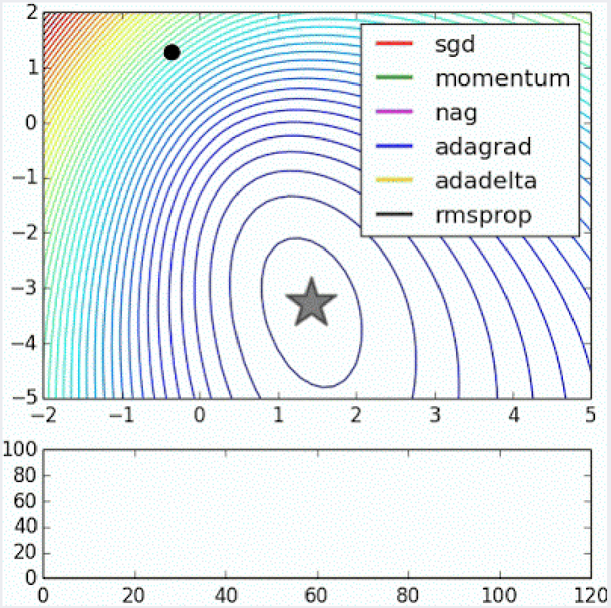
- Convergence



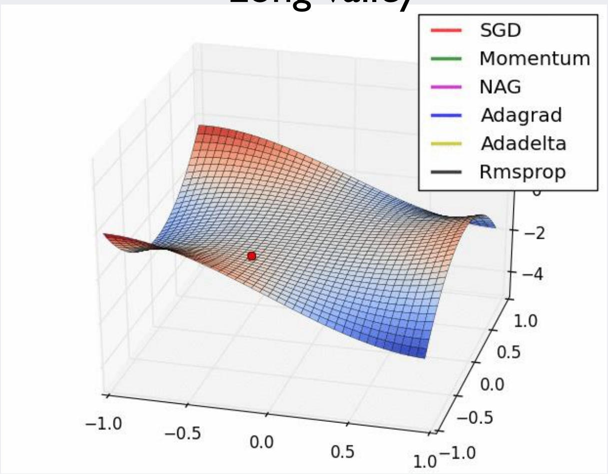
Beale's function



Long valley



Noisy moons



Saddle point

http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Neural Network: Training

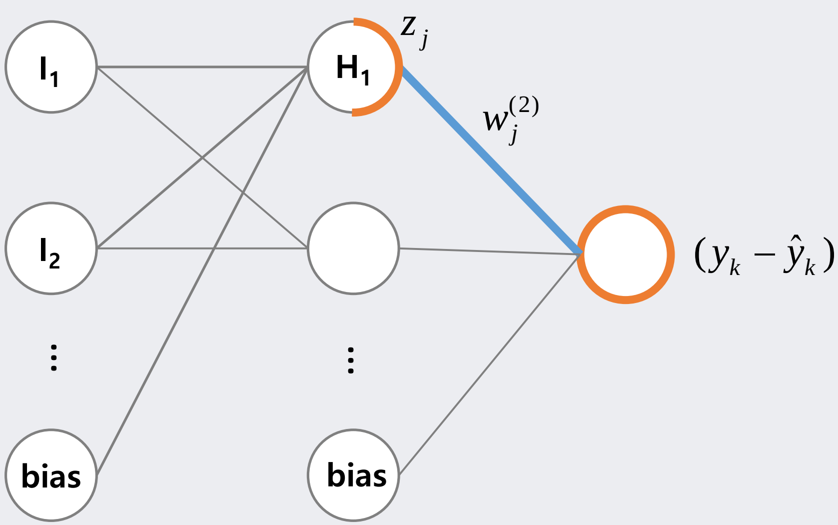
- Error Back-Propagation
  - ✓ The error of k^{th} observation

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

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

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

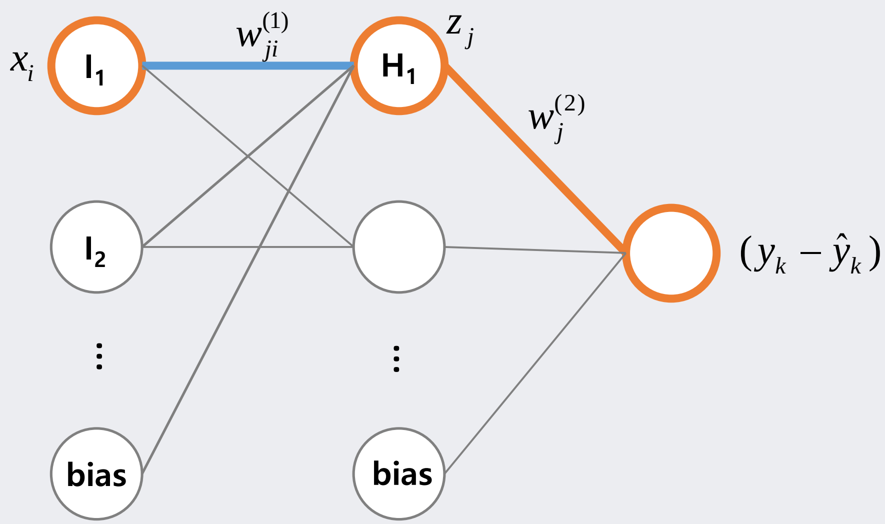
최종 결과물을 얻고	**Feed Forward and Prediction**
그 결과물과 우리가 원하는 결과물 과의 차이점을 찾은 후	**Cost Function**
그 차이가 무엇으로 인해 생기는 지	**Differentiation (미분)**
역으로 내려가면서 추정하여	**Back Propagation**
새로운 Parameter 값을 배움	**Weight Update**



Neural Network: Training

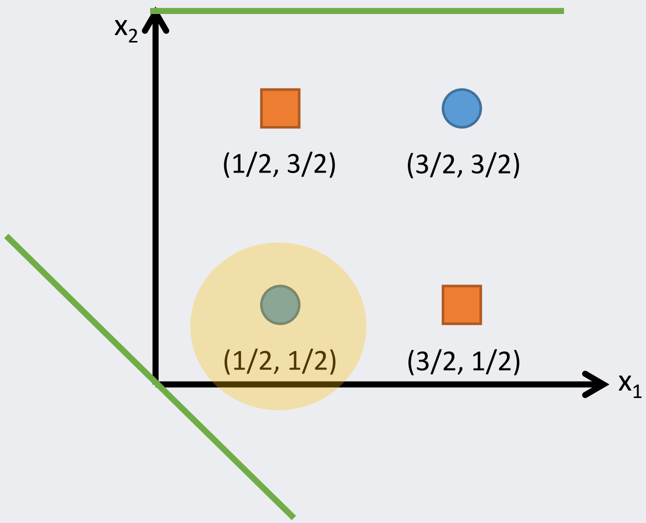
- Error Back-Propagation
  - ✓ The weight $w_{ji}^{(1)}$ which connects the i[th] input node and j[th] hidden node

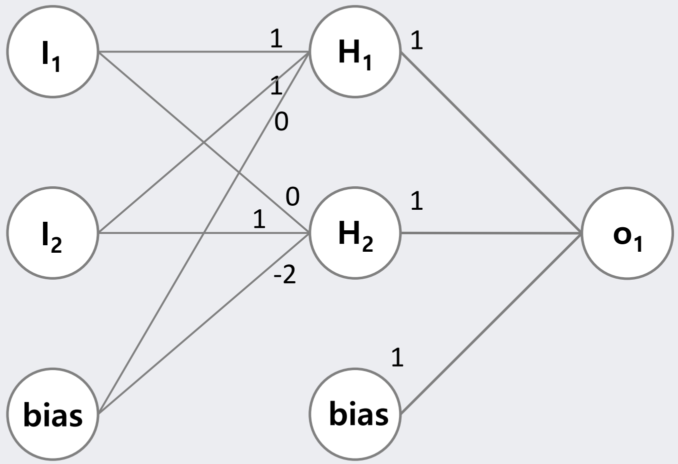
$$\frac{\partial Err_k}{\partial w_{ji}^{(1)}} = \frac{\partial Err_k}{\partial \hat{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial z_j} \cdot \frac{\partial z_j}{\partial h_j} \cdot \frac{\partial h_j}{\partial w_{ji}^{(1)}} = (y_k - \hat{y}_k) \cdot w_j^{(2)} \cdot z_j \cdot (1 - z_j) \cdot x_i$$



Neural Network: Training

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





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

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

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

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

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

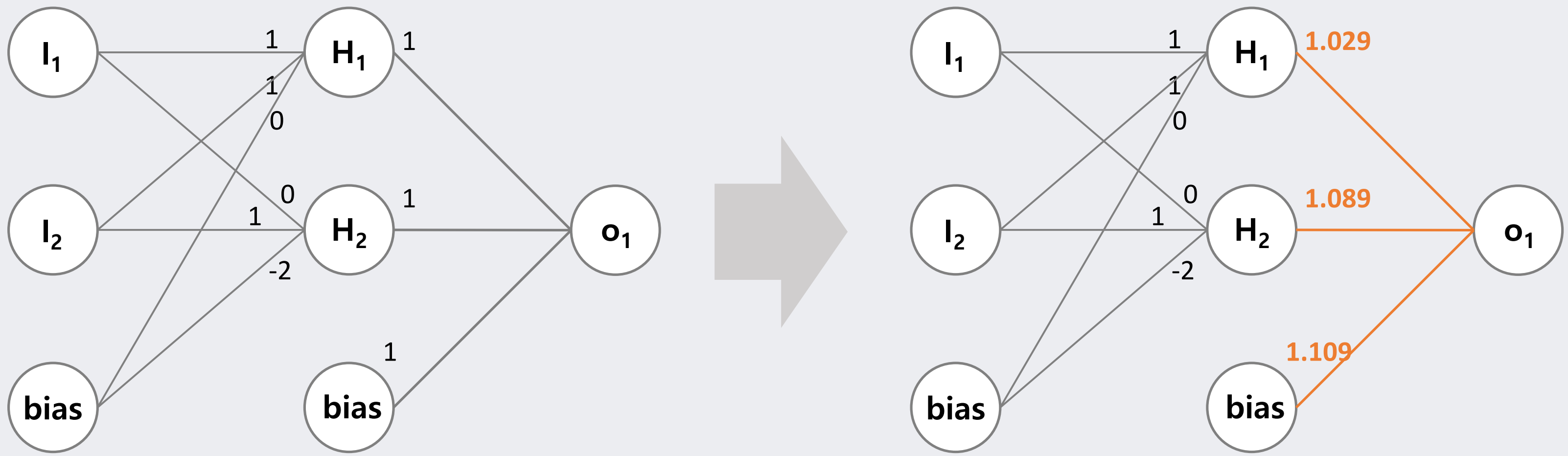
Neural Network: Training

- Error Back-Propagation: Example
  - ✓ Update the weights between the output and the hidden nodes

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

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

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



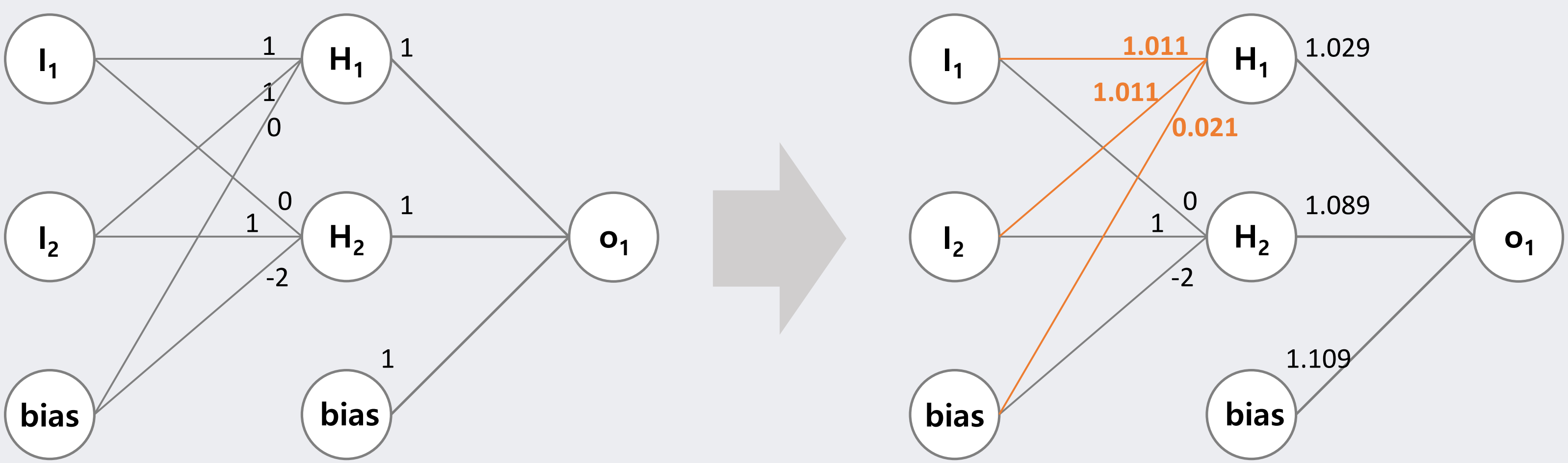
Neural Network: Training

- Error Back-Propagation: Example
  - ✓ Update the weights between the H_1 and the input nodes

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

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

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



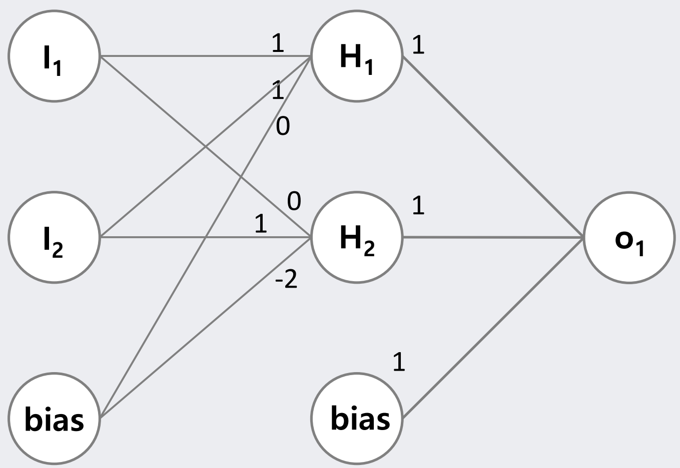
Neural Network: Training

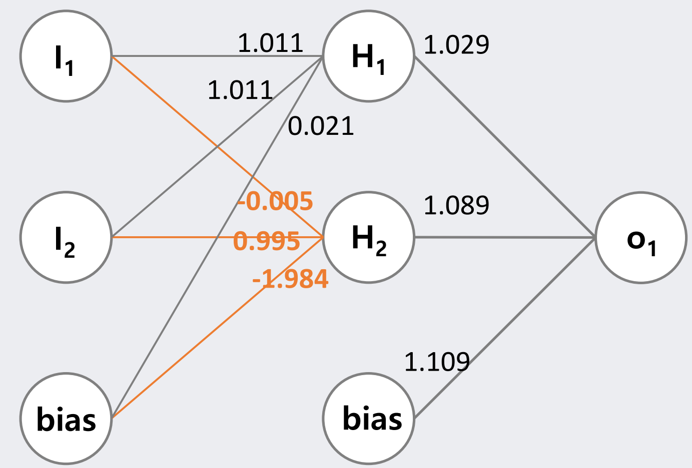
- Error Back-Propagation: Example
  - ✓ Update the weights between the H_1 and the input nodes

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

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

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





Neural Networks: Learning

- 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

Neural Networks: Learning

- Case updating
  - ✓ Weights are updated after each record is run through the network
  - ✓ Completion of all records through the network is one 'epoch'
  - ✓ After one epoch is completed, return to first record and repeat the process

- Batch updating
  - ✓ All records in the training set are fed to the network before updating takes place
  - ✓ The error used for updating is the sum of all errors from all records

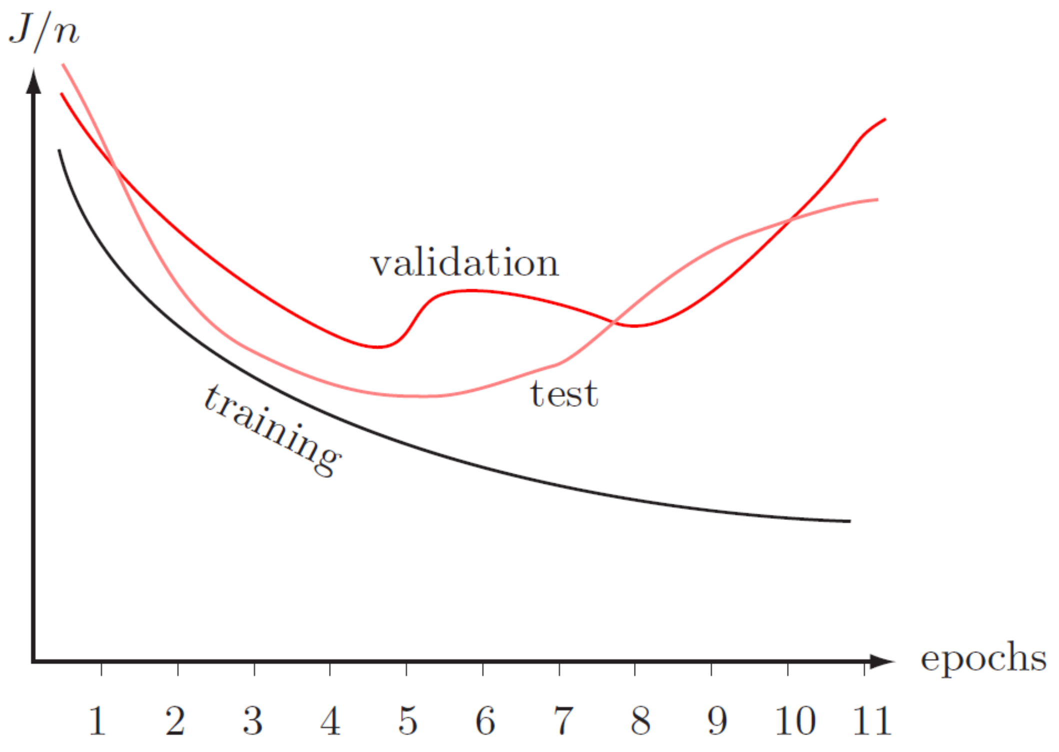
Neural Networks: Learning

- 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

Neural Networks: Learning

- 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.



AGENDA

R Exercise

- Dataset: Wisconsin Diagnostic Breast Cancer (WDBC)
  - ✓ Predicting whether a patient is malignant or benign
  - ✓ The real-valued features for the following information are computed for each cell nucleus:
    - a) radius (mean of distances from center to points on the perimeter)
    - b) texture (standard deviation of gray-scale values)
    - c) perimeter
    - d) area
    - e) smoothness (local variation in radius lengths)
    - f) compactness (perimeter^2 / area - 1.0)
    - g) concavity (severity of concave portions of the contour)
    - h) concave points (number of concave portions of the contour)
    - i) symmetry
    - j) fractal dimension ("coastline approximation" - 1)
  - ✓ Mean, standard error, and worst values are used as input variables

R Exercise

- Install and load "nnet" package

```r
# Artificial Neural Network -------------------------------------------------
# nnet package install
install.packages("nnet", dependencies = TRUE)
library(nnet)
```

- Performance evaluation function

```r
# Performance Evaluation Function -----------------------------------------
perf_eval <- function(cm){

  # True positive rate: TPR
  TPR = cm[2,2]/sum(cm[2,])
  # True negative rate: TNR
  TNR = cm[1,1]/sum(cm[1,])
  # Simple Accuracy
  ACC = (cm[1,1]+cm[2,2])/sum(cm)
  # Balanced Correction Rate
  BCR = sqrt(TPR*TNR)

  return(c(TPR, TNR, ACC, BCR))
}
```

R Exercise

- Data Import & Normalization

```r
RawData <- read.csv("wdbc.csv", header = FALSE)
head(RawData)

# Normlaize the input data
Class <- RawData[,31]
InputData <- RawData[,1:30]
ScaledInputData <- scale(InputData, center = TRUE, scale = TRUE)
head(ScaledInputData)
```

	V1	V2	V3	V4	V5	V6	V7
1	13.540	14.36	87.46	566.3	0.09779	0.08129	0.0666400
2	13.080	15.71	85.63	520.0	0.10750	0.12700	0.0456800
3	9.504	12.44	60.34	273.9	0.10240	0.06492	0.0295600
4	13.030	18.42	82.61	523.8	0.08983	0.03766	0.0256200
5	8.196	16.84	51.71	201.9	0.08600	0.05943	0.0158800
6	12.050	14.63	78.04	449.3	0.10310	0.09092	0.0659200
7	13.490	22.30	86.91	561.0	0.08752	0.07698	0.0475100
8	11.760	21.60	74.72	427.9	0.08637	0.04966	0.0165700
9	13.640	16.34	87.21	571.8	0.07685	0.06059	0.0185700
10	11.940	18.24	75.71	437.6	0.08261	0.04751	0.0197200

	V1	V2	V3	V4	V5
1	-0.166652555	-1.146154e+00	-0.18556471	-0.251735001	0.101657123
2	-0.297184231	-8.322759e-01	-0.26087651	-0.383301188	0.792066077
3	-1.311926130	-1.592558e+00	-1.30166089	-1.082619515	0.429441395
4	-0.311372457	-2.021951e-01	-0.38516156	-0.372503099	-0.464321793
5	-1.683090114	-5.695485e-01	-1.65681982	-1.287214880	-0.736645819
6	-0.589461680	-1.083378e+00	-0.57323529	-0.584202471	0.479213410
7	-0.180840780	6.999132e-01	-0.20819940	-0.266795493	-0.628569443
8	-0.671753388	5.371617e-01	-0.70986653	-0.645012760	-0.710337754
9	-0.138276103	-6.857996e-01	-0.19585321	-0.236106188	-1.387237161
10	-0.620675776	-2.440455e-01	-0.66912408	-0.617449218	-0.977684579

R Exercise

- Data separation & Input-output mapping

```r
# Divide the dataset into the training (50%) and test (50%) datasets
trn_idx <- sample(1:length(Class), round(0.5*length(Class)))
trnInputs <- ScaledInputData[trn_idx,]
trnTargets <- Class[trn_idx]
tstInputs <- ScaledInputData[-trn_idx,]
tstTargets <- Class[-trn_idx]

trnData <- data.frame(trnInputs, trnTargets)
colnames(trnData)[31] <- "Target"
tstData <- data.frame(tstInputs, tstTargets)
colnames(tstData)[31] <- "Target"
```

R Exercise

- Find the best number of hidden nodes using 5-fold cross validation

```
# 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), dim(ann_trn_input)[1], replace = TRUE, prob = c(0.2,0.2,0.2,0.2,0.2))
val_perf <- matrix(0, length(nH), 5)

ptm <- proc.time()

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 = 300)

    # Evaluate the model withe 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,2:5] <- t(perf_eval(cfm))
}

proc.time() - ptm
```

R Exercise

- Use the best parameter to train the network

```
ordered_val_perf <- val_perf[order(val_perf[,5], decreasing = TRUE),]
colnames(ordered_val_perf) <- c("nH", "TPR", "TNR", "ACC", "BCR")
ordered_val_perf
# Find the best number of hidden node
best_nH <- ordered_val_perf[1,1]

# Test the ANN
ann_tst_input = tstInputs
ann_tst_target = class.ind(tstTargets)

wdbc_nnet <- nnet(ann_trn_input, ann_trn_target, size = best_nH, decay = 5e-4, maxit = 300)
```

R Exercise

- Evaluate the performance

```r
# Performance evaluation
prey <- predict(wdbc_nnet, ann_tst_input)
tst_cm <- table(max.col(ann_tst_target), max.col(prey))

perf_eval(tst_cm)
```

```
> perf_eval(tst_cm)
[1] 0.9583333 0.9316239 0.9473684 0.9448843
```