Introduction to Deep Learning

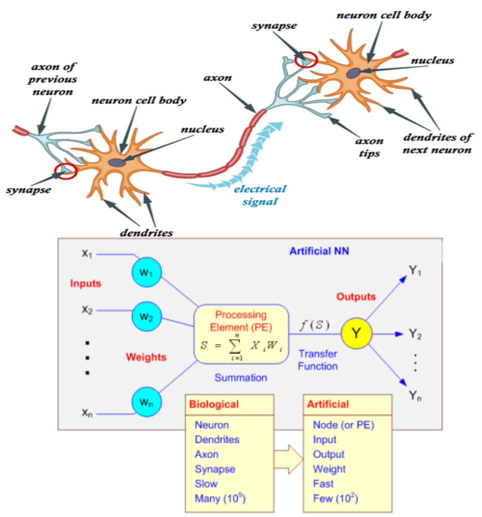
2020 – 2021

Ando Ki, Ph.D.
adki@future-ds.com

Table of contents

- Modeling a neuron
- Perceptron
- How perceptron classifies hyperplane
- Perceptron: Boolean
- Perceptron: Boolean AND training
- Multi-layered perceptron
- Layer-wise organization
- Categories of ANN
- Brief history of neural network
- Popular frameworks
- Artificial neuron: Perceptron
- Artificial neuron: activation functions
- Artificial neural network: ANN
- Fully connected feed-forward network: FC-FFN
- Optional output layer: Softmax
- How to find a good or the best network: Loss/Cost
- How to find a good or the best network: Total Lost
- How to minimize total loss by changing [W] and [b]
- Optimization algorithm: gradient descent
- How to compute gradient
- Neural network
- Popular types of neural network
- Deep neural net
- NN categories by applications

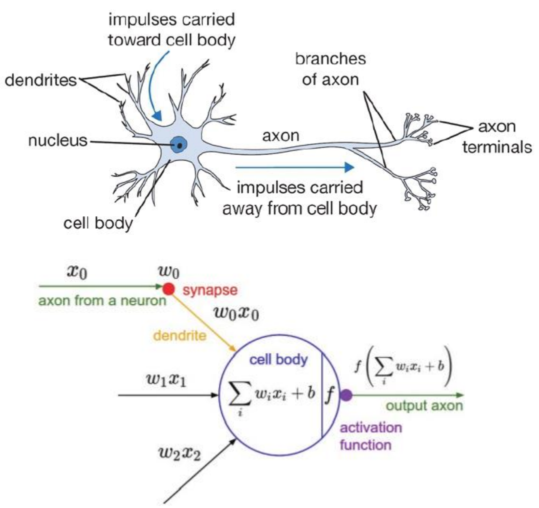
Modeling a neuron



https://www.quora.com/What-is-deep-learning

- Neuron: 신경세포(神經細胞)
  - Dendrite: 수상돌기(樹狀突起)
    - input
  - Axon: 축삭돌기(軸索突起)
    - output
    - Branches of axon
    - Terminals of axon (axon tip)
      - synaptic knob
  - Synapse: 연접
    - junction between two nerve cells
- Human
  - whole brain
    - ~86 billion neurons (Giga, 10^9)
    - ~100 trillion synapses (Tera, 10^{12})
  - cerebral cortex: 대뇌피질
    - 19~23 billion neurons

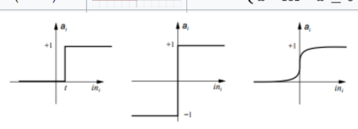
Modeling a neuron



- Activation functions

https://en.wikipedia.org/wiki/Activation_function

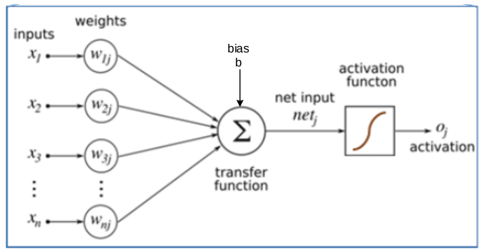
Identity	$f(x) = x$	
Binary step	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$	
Logistic (a.k.a. Soft step)	$f(x) = \frac{1}{1+e^{-x}}$	sigmoid
TanH	$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	sigmoid
ArcTan	$f(x) = \tan^{-1}(x)$	sigmoid
Softsign [7][8]	$f(x) = \frac{x}{1+\lvert x \rvert}$	sigmoid
Rectified linear unit (ReLU)[9]	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	

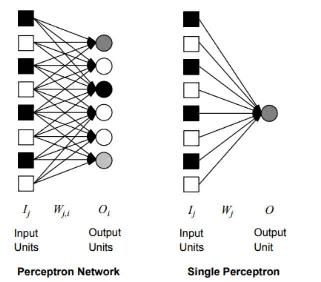


(a) Step function (b) Sign function (c) Sigmoid function

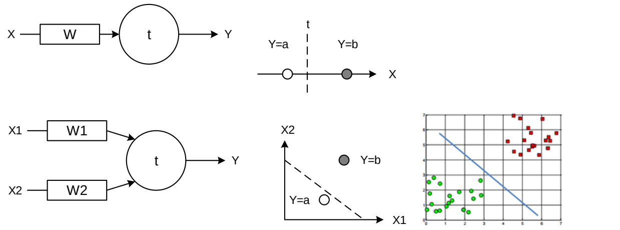
Perceptron: single layer neural network

- Perceptron is a single artificial neuron that computes its weighted input and uses a threshold activation function.
  - It is also called a TLU (threshold logic unit).
  - It effectively separates the input space into two categories by the hyperplane: W*X+b = 0
  - Perceptron is a linear classifier.
    - Cannot deal with non-linear cases
  - Perceptron refers to a particular supervised learning model with backpropagation learning algorithm.
  - Perceptron is an algorithm for supervised learning of binary classifiers.

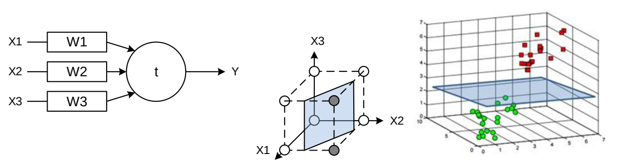




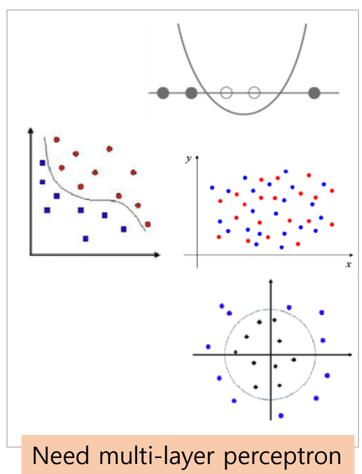
How perceptron classifies hyperplane



W1*x1+W2*x2 + W0 = 0
➔ y = ax + b

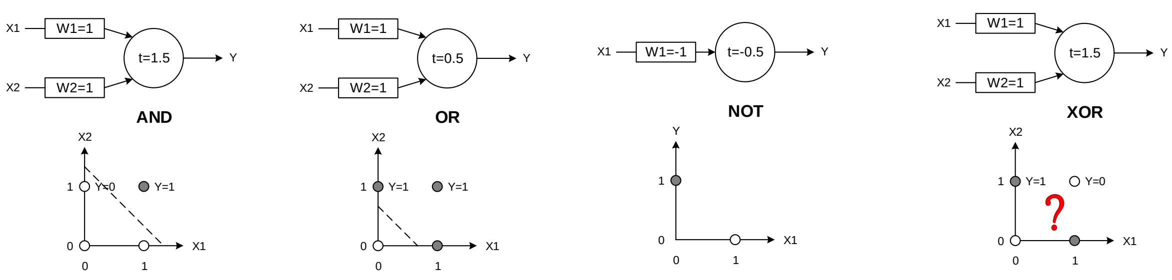


W1*x1+W2*x2 + W3*x3 + W0 = 0
➔ z = ax + by + c



Need multi-layer perceptron

Perceptron: Boolean



Perceptron: Boolean AND training

- Step 1: initialize the weight and the threshold.
  - Weights may be initialized to 0 or to a small random value.
- Step 2: repeat until error is less than a specific value
  - Calculate output (for j-th test set)

$$\begin{aligned} y_j(t) &= f[\mathbf{w}(t) \cdot \mathbf{x}_j] \\ &= f[w_0(t)x_{j,0} + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \cdots + w_n(t)x_{j,n}] \end{aligned}$$

  - Update weights (for i-th path for j-th test set) (d_j is desired or expected value)

$$w_i(t+1) = w_i(t) + (d_j - y_j(t))x_{j,i}\text{, for all features } 0 \le i \le n.$$

  - Calculate error

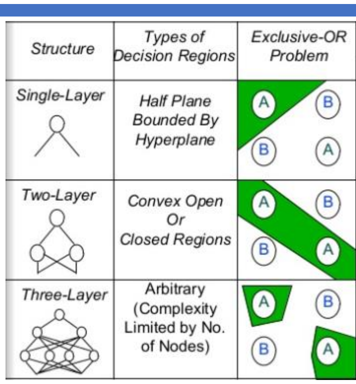
$$\frac{1}{s}\sum_{j=1}^{s}|d_j - y_j(t)|$$

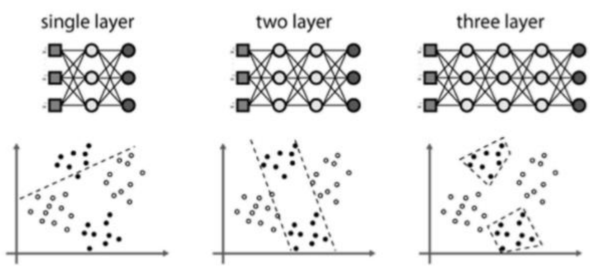
- Training set [{inputs: expected}]
  - T0={0,0:0}, T1={0,1:0}, T2={1,0:0}, T3={1,1:1}
- for T0 and T1 and T2 (assume all weights are 0)
  - y = 0x0+0x0 = 0
  - e = 0-0 = 0 (no error)
  - No update since no error
- for T3
  - y = 1x0+1x0=0
  - e = 1-0 = 1
  - w0 = 0 + (1-0) = 1
  - w1 = 0 + (1-0) = 1
- After updating
  - for T3, T2, and T1
    - y = 1x1+1x1=2 => apply threshold = 1.5
      - e = 1-1 = 0
    - y = 1x1+1x0=1 => apply threshold = 1.5
      - e = 0-0 = 0
    - y = 1x0+1x1=1 => apply threshold = 1.5
      - e = 0-0 = 0
    - y = 1x0+1x0=0 => apply threshold = 1.5
      - e = 0-0 = 0

Perceptron: Boolean OR training

- Training set [{inputs: expected}]
  - T0={0,0:0}, T1={0,1:1}, T2={1,0:1}, T3={1,1:1}
- for T0 (assume all weights are 0)
  - y = 0x0+0x0 = 0
  - e = 0-0 = 0 (no error)
  - No update since no error
- for T1
  - y = 0x0+0x1=0
  - e = 1-1 = 1
  - w0 = 0 + (1-1) = 1
  - w1 = 0 + (1-1) = 1
  - Update w0 and w1
- After updating
  - for T2
    - y = 1x1+1x0=1 => apply threshold = 1
    - e = 1-1 = 0
  - No update since no error
- for T3
  - y = 1x1+1x1=2 ==> apply threshold = 1
  - e = 1-1 = 0
  - No update since no error

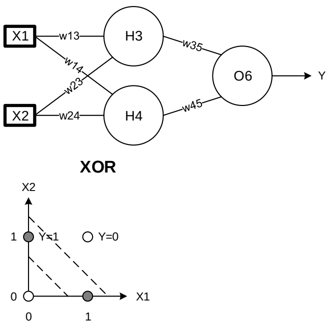
MLP: Multi-layered perceptron (다층 퍼셉트론)

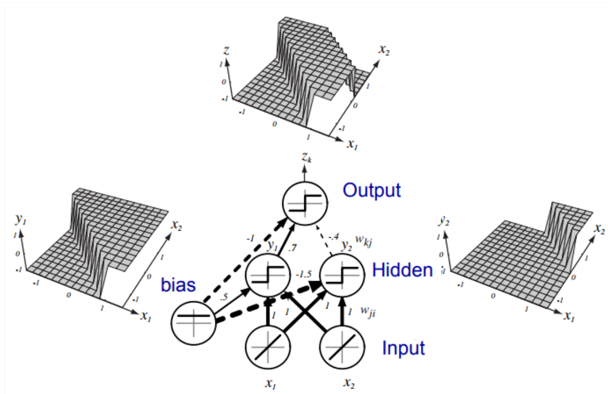




Multi-layered perceptron

- Two-unit network (two layers)

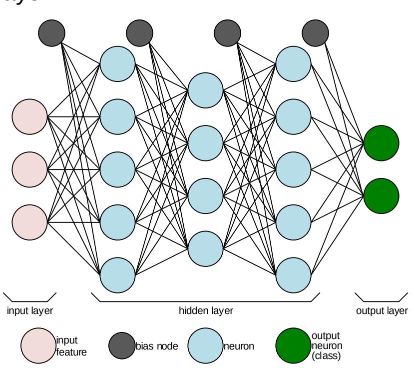




(from Pascal Vincent's slides)

Layer-wise organization

- 3 types of layers
  - Input layer
  - hidden layer
  - output layer



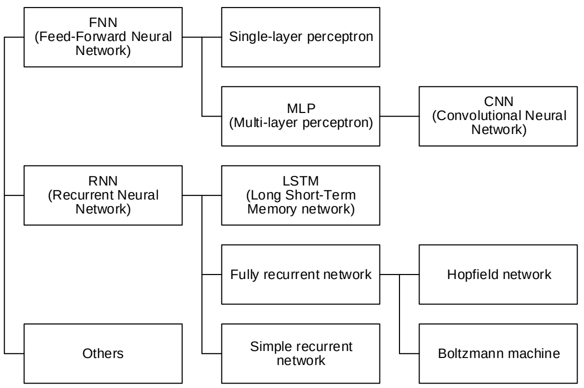
fully-connected multi-layered neural network

- input layer: not counted for the number of layers
- hidden layer
- output layer

- For the picture on the left
  - assume fully connected
  - 4-layered including 3-hidden layers
  - 16 neurons: 5+4+5+2
  - 65 weights: 3x5+5x4+4x5+5x2
    - not including bias
  - 16 biases: 5+4+5+2
  - 82 learnable parameters: 65+16

- Modern neural network
  - 10~20 layers, ~100 million parameters
  - How about 125 layers?

Categories of ANN (Artificial Neural network)



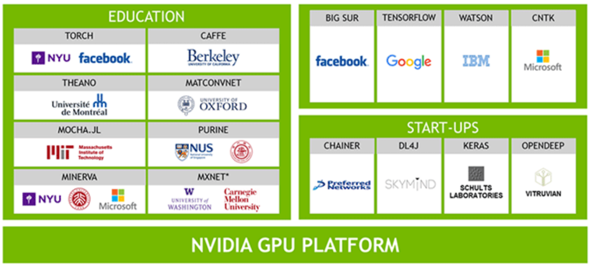
- Fully-Connected NN
  - feed forward
  - Multi-Layer Perceptron (MLP)
- Convolutional NN (CNN)
  - feed forward, sparsely-connected
  - Image recognition
  - AlphaGo
- Recurrent NN (RNN)
  - feedback
- Long Short-Term Memory (LSTM)
  - feedback + storage
  - Microsoft speech recognition
  - Google neural machine translation (GNMT)

See neural network topology: http://www.asimovinstitute.org/neural-network-zoo/

Popular Frameworks

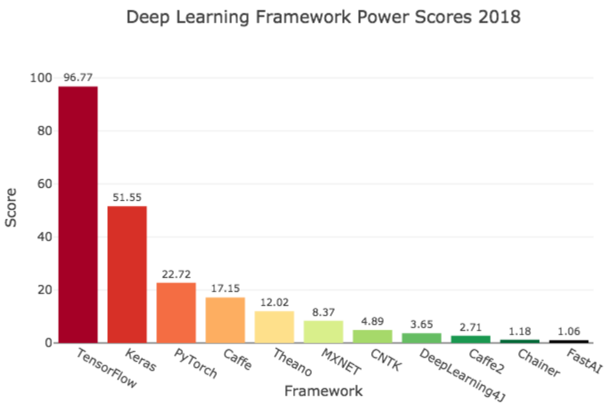
- Popular Frameworks with supported interfaces
  - Caffe
    - Berkeley / BVLC (Berkeley Artificial Intelligence Research)
    - C, C++, Python, Matlab
  - TensorFlow
    - Google Brain
    - C++, Python
  - PyTorch
  - theano
    - U. Montreal
    - Python
  - torch
    - Facebook / NUU
    - C, C++, Lua
  - CNTK
    - Microsoft
  - MXNet
    - Carnegie Mellon University / DMLC (Distributed Machine Learning Community)

https://developer.nvidia.com/deep-learning-frameworks



https://blogs.nvidia.com/blog/2016/01/12/accelerating-ai-artificial-intelligence-gpus/

Popularity



1. TensorFlow
2. Keras
3. PyTorch
4. Caffe
5. theano
6. Apache mxnet
7. Microsoft CNTK
8. DL4J
9. Caffe2
10. Chainer
11. fast.ai

Deep Learning Framework Deep Learning Framework Power Scores (by Jeff Hale) http://bit.ly/2GBa3tU

https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Table of contents

- Artificial neuron: Perceptron
- Artificial neuron: activation functions
- Artificial neural network: ANN
- Fully connected feed-forward network: FC-FFN
- Optional output layer: Softmax
- How to find a good or the best network: Loss/Cost
- How to find a good or the best network: Total Lost
- How to minimize total loss by changing [W] and [b]
- Optimization algorithm: gradient descent
- How to compute gradient
- Neural network
- Popular types of neural network
- Deep neural net
- NN categories by applications
- Popular DNNs and Frameworks

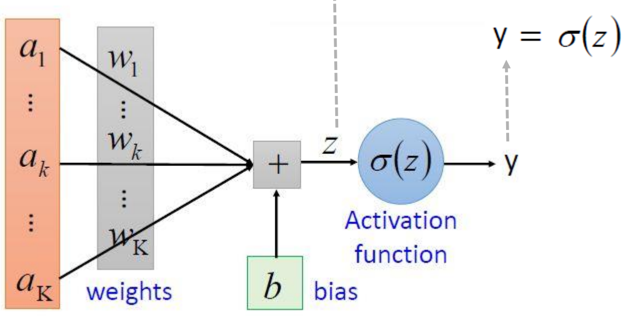
Artificial neuron: Perceptron

- Artificial Neuron: Perceptron
  - inputs
  - output
  - weights
  - bias
  - activation function

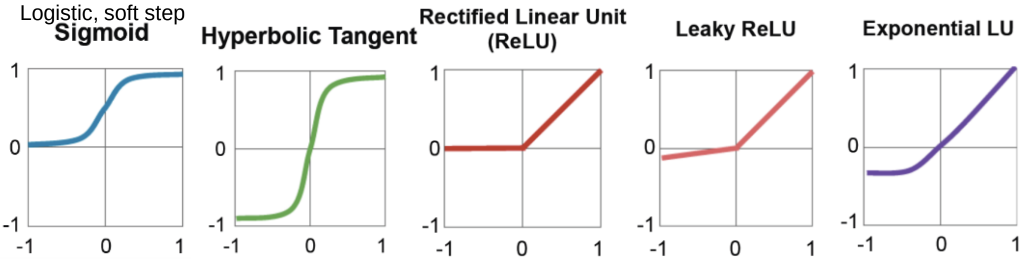
$$(a_1, a_2, \cdots, a_K) \times \begin{pmatrix} W_1 \\ W_2 \\ \vdots \\ W_K \end{pmatrix} + b = z$$

$$z = a_1 w_1 + \cdots + a_k w_k + \cdots + a_K w_K + b$$

$$y = \sigma(z)$$



Artificial neuron: activation functions



$y = \frac{1}{1+e^{-z}}$

$y = \frac{2}{1+e^{-2z}} - 1$

$y = 0 \; for \; z < 0$
$y = z \; for \; z \geq 0$
y=max(x,0)

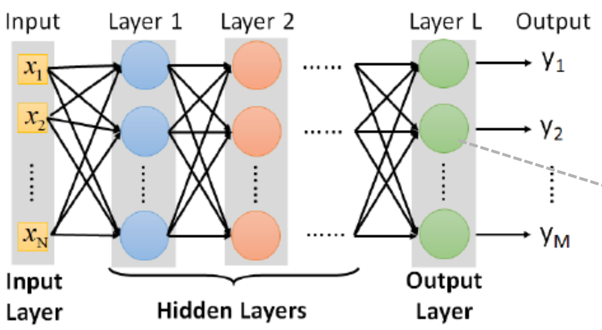
$y = 0.01z \; for \; z < 0$
$y = z \; for \; z \geq 0$

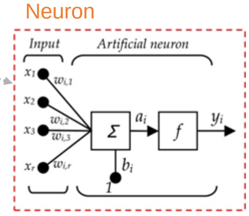
$y = \alpha(e^z - 1) \; for \; z < 0$
$y = z \; for \; z \geq 0$

$$\frac{dy(z)}{dz} = \frac{d}{dz}\left[\frac{1}{1+e^{-z}}\right] = \frac{d}{dz}(1+e^{-z})^{-1} = -(1+e^{-z})^{-2}(-e^{-z}) = y(z)\cdot(1-y(z))$$

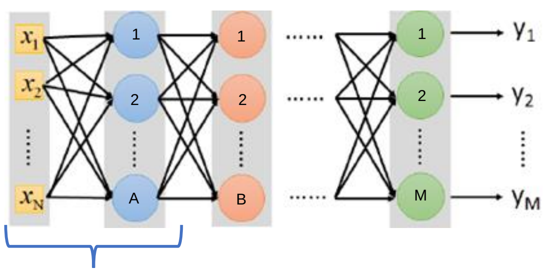
Artificial Neural Network: ANN

- Artificial Neural Network: ANN
  - Network structure by different connections
  - Each neuron can has different values of weights and bias
  - Weights and biases are <u>network parameter</u> θ
  .

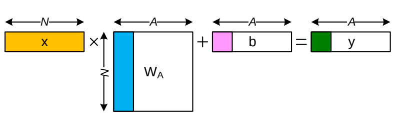




Artificial Neural Network: ANN



N: number of inputs
A: number of hidden layers

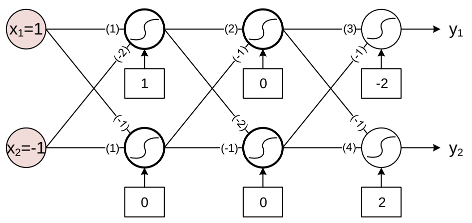


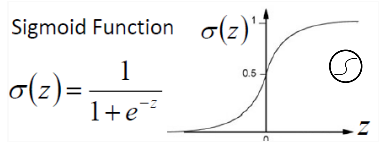
$$\begin{bmatrix} x_1 & x_2 & \circ\circ\circ & x_N \end{bmatrix} \times \begin{bmatrix} w_{1,1} & w_{2,1} & \circ\circ\circ & w_{A,1} \\ w_{1,2} & w_{2,2} & \circ\circ\circ & w_{A,N} \\ \vdots & & & \vdots \\ w_{1,N} & w_{2,N} & \circ\circ\circ & w_{A,N} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & \circ\circ\circ & b_A \end{bmatrix} = \begin{bmatrix} y_1 & y_2 & \circ\circ\circ & y_A \end{bmatrix}$$

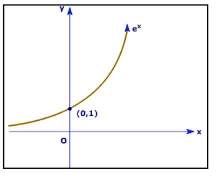
$$\begin{bmatrix} w_{1,1} & w_{2,1} & \circ\circ\circ & w_{A,1} \\ w_{1,2} & w_{2,2} & \circ\circ\circ & w_{A,N} \\ \vdots & & & \vdots \\ w_{1,N} & w_{2,N} & \circ\circ\circ & w_{A,N} \end{bmatrix}^T \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_A \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_A \end{bmatrix}$$

Fully connected feed-forward network: FC-FFN

- Activation function: E.g., Sigmoid – S-shaped function







$$[1, -1] \times \begin{bmatrix} 1, -1 \\ -2, 1 \end{bmatrix} + [1, 0] = [4, -2] \xrightarrow{\text{sigmoid}} [0.98, 0.12]$$

$$[0.98, 0.12] \times \begin{bmatrix} 2, -2 \\ -1, -1 \end{bmatrix} + [0, 0] = [1.84, -2.08] \xrightarrow{\text{sigmoid}} [0.86, 0.11]$$

$$[0.86, 0.11] \times \begin{bmatrix} 3, -1 \\ -1, 4 \end{bmatrix} + [-2, 2] = [\ ??,\ \ ??] \xrightarrow{\text{sigmoid}} [\ ??,\ ??]$$

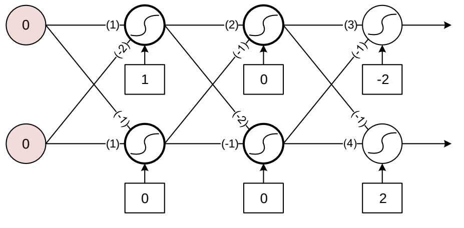
x_1 x_2 y_1 y_2

f ([1, -1]) = [0.62, -0.83]

f ([0, 0]) = [051, 0.85]

Do it yourself

- Calculate the output



Optional output layer: Softmax

- Outputs of artificial neural network will be any values from very small to very large including negative.

  f ([1, -1]) = [0.62, -0.83]
  f ([0, 0]) = [051, 0.85]

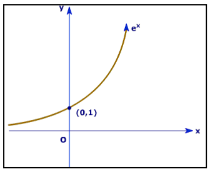
  The output can be any value.
  → Hard to interpret.

- Softmax for output layer
  - Softmax is a function to transform a number of values to a range of value to between 0 ~ 1.
    - ➲ Score (-inf, inf) ==> probabilities [0,1]
  - Multinomial logistic or normalized exponential function

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

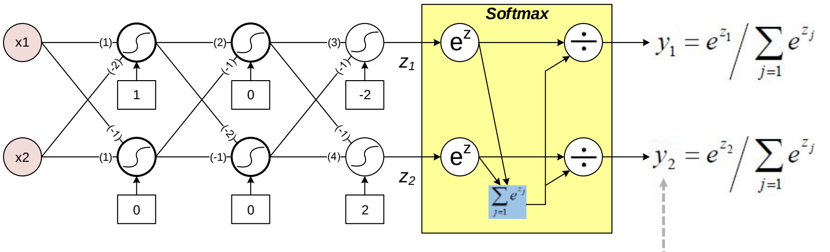
$$\sigma(z)_i = \frac{e^{z_i - m}}{\sum_{k=1}^{K} e^{z_k - m}}$$

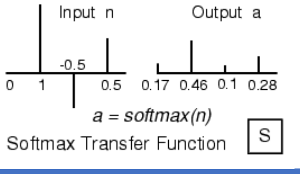
Where 'm' is max{z1,...,zk}



Optional output layer: Softmax

- Softmax converts score to probability: Score (-inf, inf) ==> probabilities [0,1]
  - un-normalized probabilities (summation will not give 1): e^{z_j} for result j.
  - normalized probabilities (summation will give 1): -- see below --

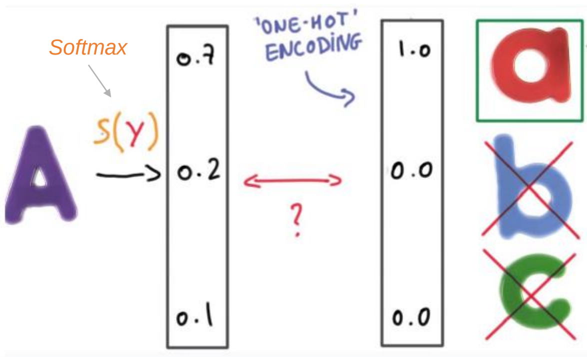




$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

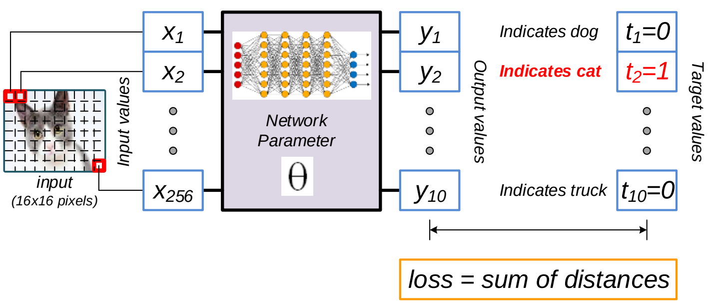
Optional output layer: one-hot encoding

- One-hot encoding by encoding class labels
- Select one only among many.



How to find a good or the best network: Loss/Cost

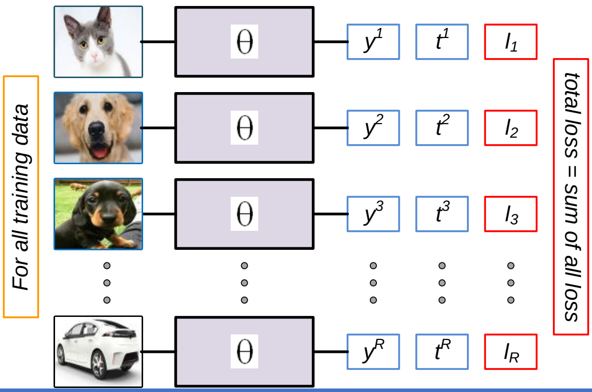
- **Loss function** is the distance between the network output and the target
  - cost function or error function
  - It indicates how good the result is.
  - There can be different loss functions.
    - The simplest one will be a summation of | t – y |.
      - Perfect match will give 0.



- Training error: error by training data set
- Generalization error (test error): error by test data set in order to evaluate the training model.

How to find a good or the best network: Total Loss

- Total loss (L) is a sum of losses (l_r)
  - Make it as small as possible
- Training means to find the network parameter θ that minimize total loss L.
  - This means we should modify the network parameter according to the total loss.



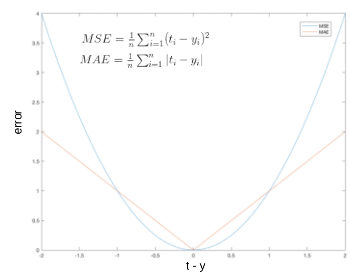
$$L = \sum_{r=1} l_r$$

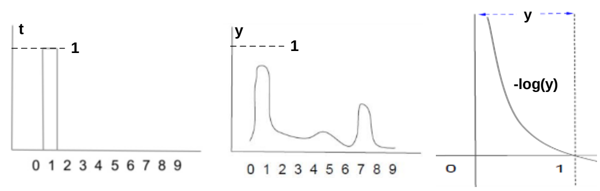
Sum of losses for R test images

Cost functions (error function)

- Absolute error
  - Sum of absolute errors
    - sum(|t – y|)
  - Mean absolute errors (MAE)
    - sum(|t-y|)/n
- Squared error loss
  - Sum of squared errors
    - sum((t - y)**2)
  - Mean squared errors (MSE)
    - sum((t-y)**2)/n
  - Root mean square errors (RMSE)
    - (MSE)**(1/2)
- Cross-entropy loss
  - For classification after Softmax
  - Sum of cross-entropy loss
    - -sum(t*log(y))
      - all except t=1 does not contribute
    - or - sum[t*log(y) + (1-t)*log(1-y)]
      - add cost when t is not 1.

- y: inference value or calculated value
- t: target value

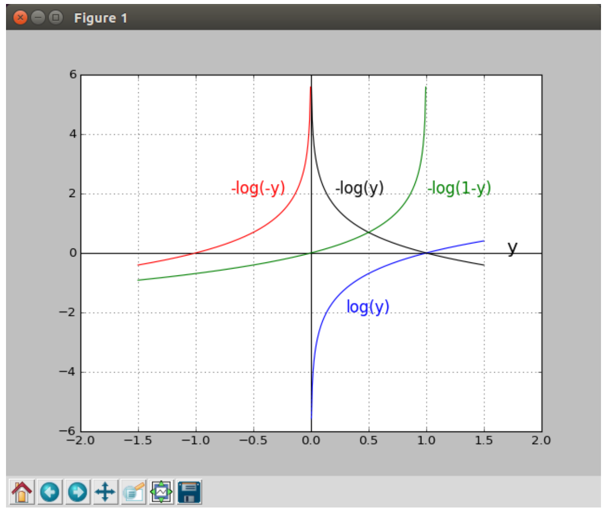




-log(y) emphasizes error (y) when softmax result (y) is small.
y<1 means error, y=1 means correct.

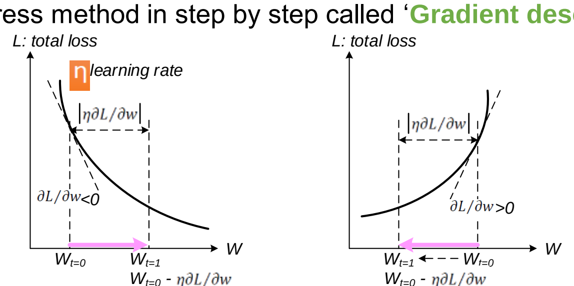
Log plots

```
import numpy as np
from matplotlib import pyplot as plt

y = np.linspace(-1.5, 1.5, 400)

plt.plot(y, np.log(y), color='blue')
plt.text(0.3, -2, 'log(y)', fontsize=15, color='blue')

plt.plot(y, -np.log(y), color='black')
plt.text(0.2, 2, '-log(y)', fontsize=15, color='black')

plt.plot(y, -np.log(-y), color='red')
plt.text(-0.7, 2, "-log(-y)", fontsize=15, color='red')

plt.plot(y, -np.log(1-y), color='green')
plt.text(1.0, 2, "-log(1-y)", fontsize=15, color='green')

plt.grid()
plt.show()
```



How to minimize total loss by changing [W] and [b]

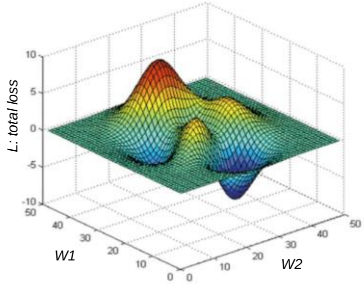
- If we can find how the network parameters affect the total loss, it may be possible to figure out how to minimize the total loss.
- However, the number of parameters is too larger to figure out.
  - AlexNet: 650K neurons, 8 layers, 60 Million parameters
- So we apply gradual iterative progress method in step by step called '**Gradient descent**'. It is called *optimization algorithm.*

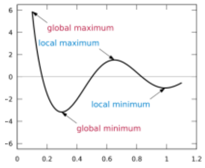


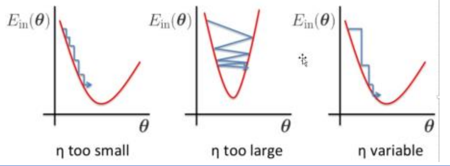
  - *Negative slope ➔ increase W by some function of learning rate*
  - *Positive slope ➔ decrease W*
  - *Steep slope ➔ large change of W for the next time*
  - *go on until the slope is small enough, i.e., inflection point*

Optimization algorithm: gradient descent

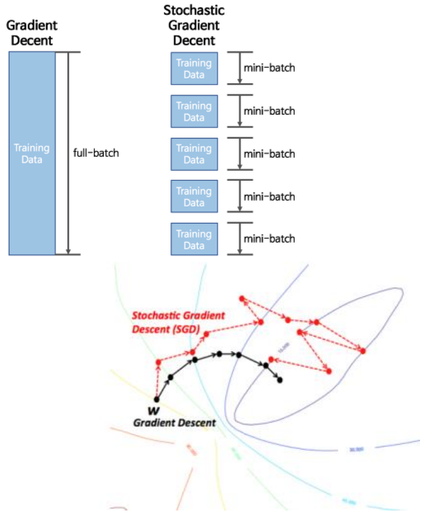
- *Initial value problem*
  - *different initial point leads to different minima*
- *Local minimum problem (get stuck in local minima)*
  - *never guarantee global minima*
- *Learning rate problem*
  - *large learning rate could cause oscillation*
  - *small learning rate results in slow learning*
- *Vanishing gradient problem*
  - If a change in the parameter's value causes very small change in the network's output - the network just can't learn the parameter effectively, which is a problem.
- Gradient Exploding

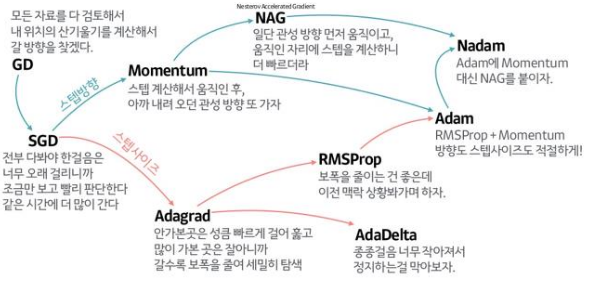






Optimization





ref: 하용호: https://www.slideshare.net/yongho/ss-79607172

How to compute gradient

- Backpropagation
  - 1. Feed-forward computation
  - 2. Back-propagation to the output layer
  - 3. Back-propagation to the hidden layers
  - 4. Weight updates

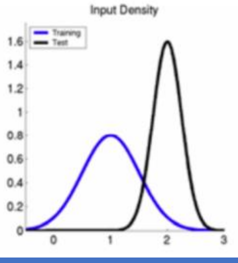
$\partial L / \partial w$

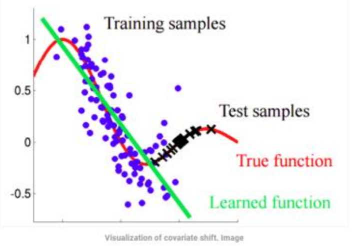
Do not panic. You do not need to worry about it because the program will do it.

Covariate shift

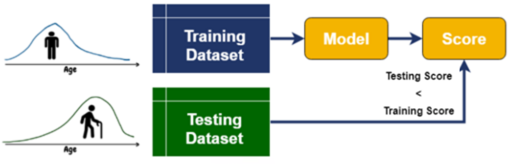
- Covariate shift as one of dataset shift (dataset drift) ➔ need normalization
  - occurs when the distribution of input variables is different between training and testing dataset.
  - Internal covariate shift ➔ need (batch) normalization
    - each hidden (internal) layer has the same problem.

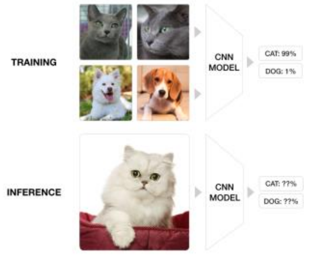
Training and test input follow
different distributions, but functional
relation remains unchanged.





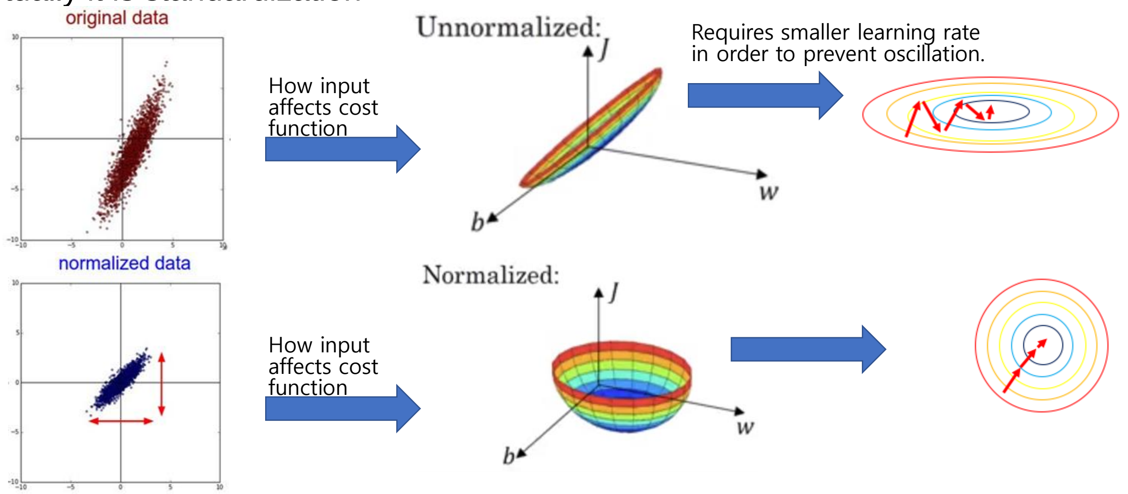
Visualization of covariate shift. Image





Input normalization: normalizing input data

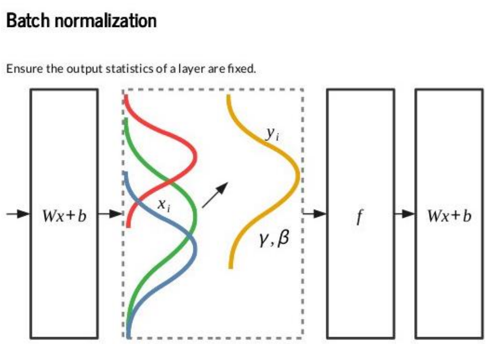
- Actually it is standardization



Batch normalization

- Batch normalization ensures the output statistics of a layer are fixed.
  - calculate mean (mu) and standard deviation (sigma) from before or after activation function for each training (mini-)batch
  - apply standardization (called normalization) on training
  - need to update scaling factor (gamma) and bias (b) during training

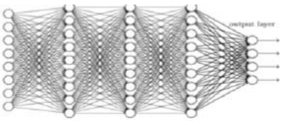
- z = (x – m)/s * g + b
  - x: input
  - m: mean
  - s: standard deviation (sqrt(variance**2))
  - g: gamma (scaling factor))
  - b: bias



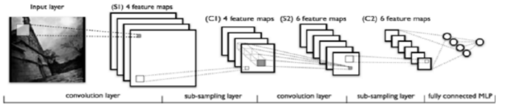
$$y = \frac{x - E[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

Popular types of Neural Network (NN)

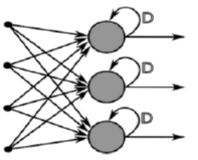
- DNN: Deep NN
  - More general model
  - fully connected
  - feed-forward (i.e., MLP: multilayer perceptron)
  - speech, image processing, natural language processing (NLP)



- CNN: Convolutional NN
  - Common image optimization
  - connected locally (i.e., sparsely-connected)
  - feed-forward
  - object/facial recognition

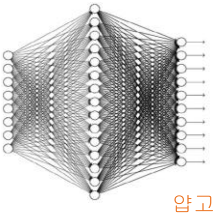


- RNN: Recurrent NN
  - context driven, time-series optimization
  - variable connectivity
  - feed-back in addition to feed-forward
  - NLP and speech recognition
  - Long Short-Term Memory (LSTM)
    - ➲ feed-back + storage



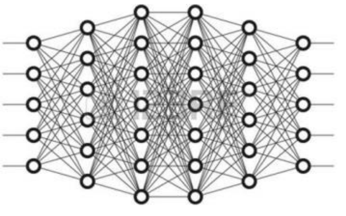
Deep neural net

- Any continuous function can be realized by a network with one hidden layer with sufficient neurons. (Universality theorem, universal approximation theorem)
  - A hidden layer network can represent any continuous function
  - A shallow fat neural net.



얍고 굵다(두껍다)

- Deep thin neural net (deep NN) is better than shallow fat net.
  - Using multiple layers of neurons to represent some functions are much simpler.
    - ➲ Less parameters ➔ less computation



깊고 가늘다(얇다)