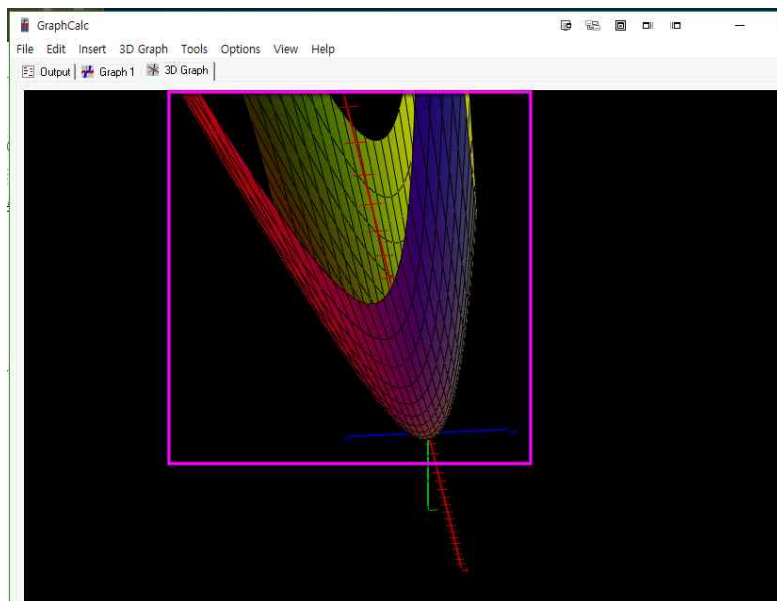
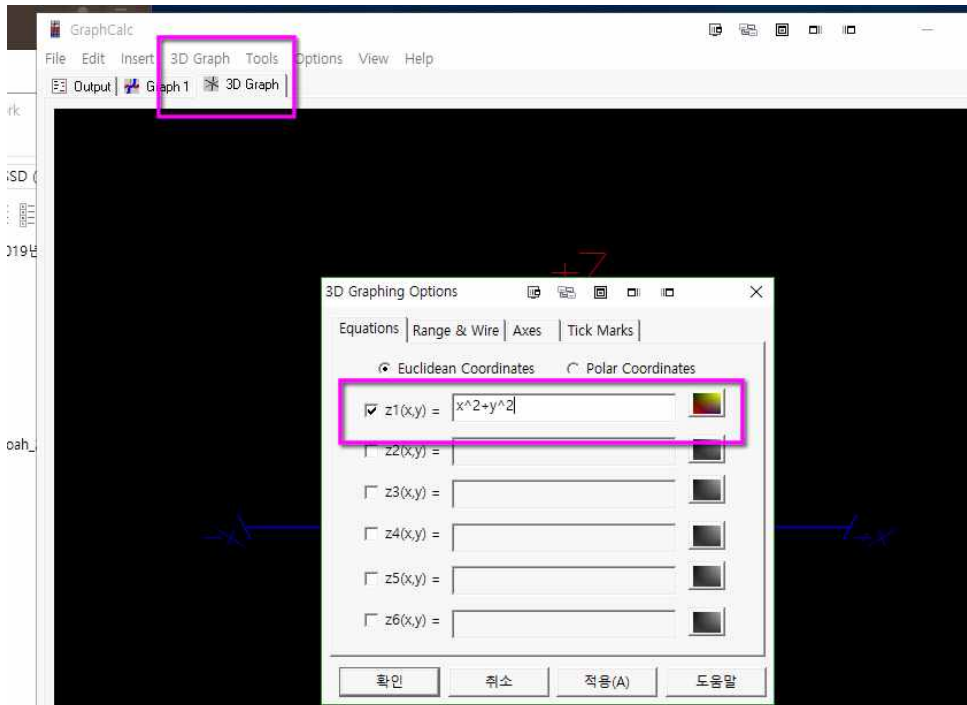


인공 신경망: Back-propagation

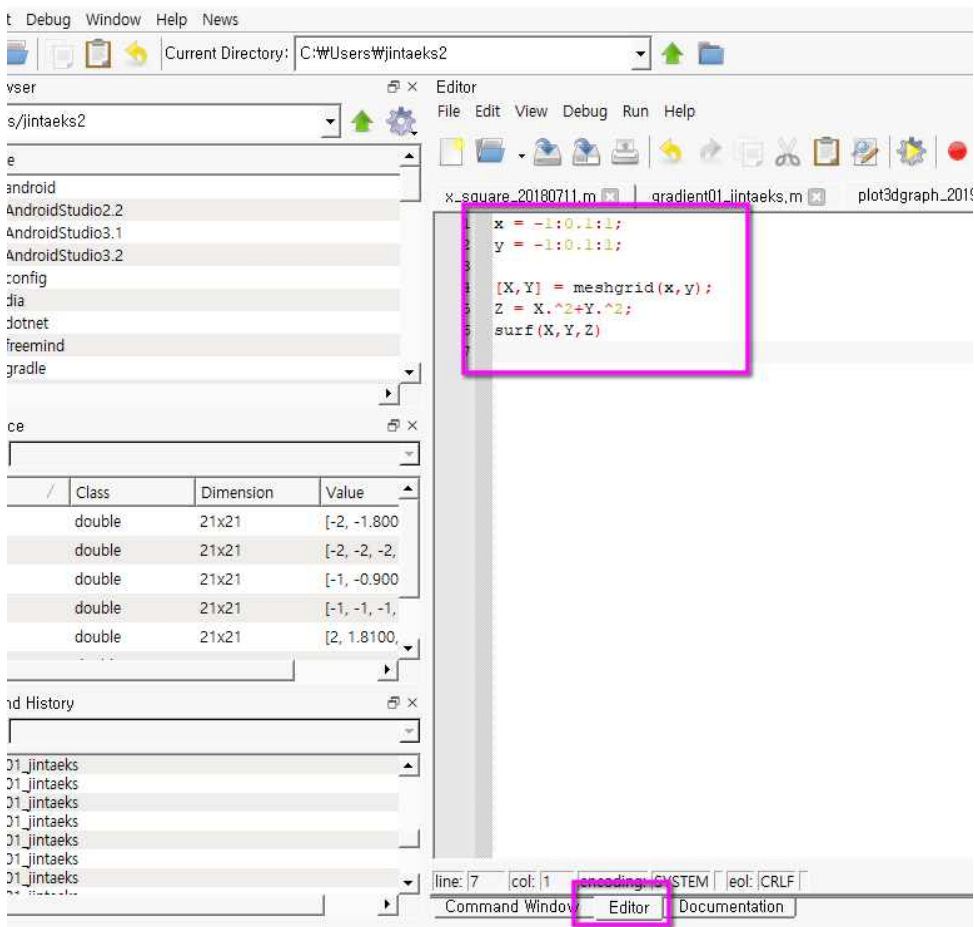
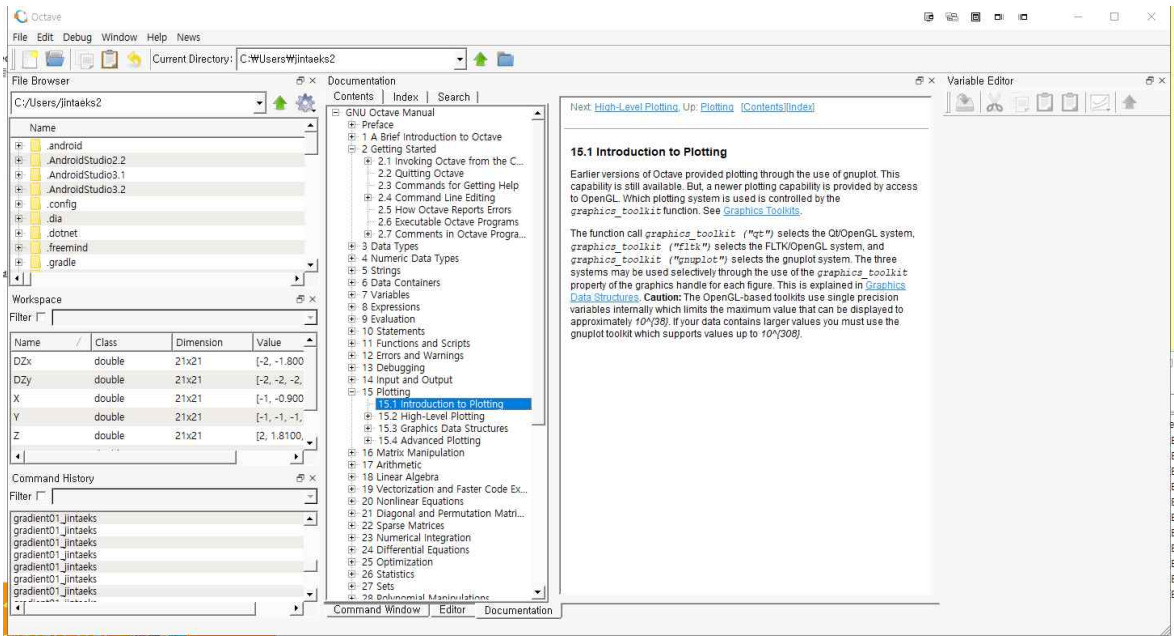
- > 2019년1월31일, 서진택
- > 2019년2월8일, 수정, 서진택

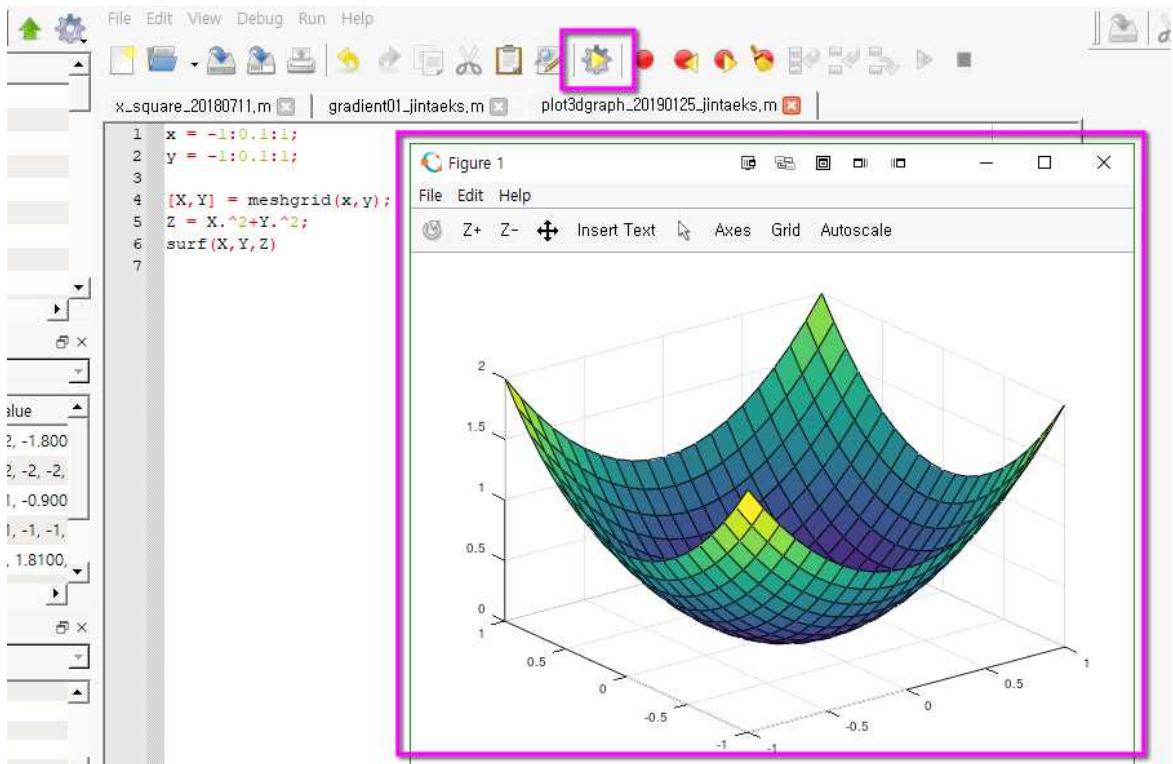
Plotting 3D Graph

GraphCalc



Octave





Differentiation

Product Rule

$$[f \times g]' = f'g + f \times g'$$

$$[(2x+3)^4(x+1)^2]'$$

$$f(x) = (2x+3)^4$$

$$g(x) = (x+1)^2$$

$$f'(x) = 4(2x+3)^3 \times 2 = 8(2x+3)^3$$

$$g'(x) = 2(x+1)^1 \times 1 = 2x+2$$

$$[f \times g]' = f'g + f \times g'$$

$$= 8(2x+3)^3(x+1)^2 + (2x+3)^4(2x+2)$$

Quotient Rule

$$\left[\frac{f}{g} \right]' = \frac{f'g - fg'}{g^2}$$

$$\begin{aligned} & \frac{d}{dx} \left(\frac{3x-5}{x^2+4} \right) \\ &= \frac{3(x^2+4) - (3x-5)(2x)}{(x^2+4)^2} \end{aligned}$$

Chain Rule

$$\frac{d}{dx} (x^2 + 3)^4$$

$$u = (x^2 + 3)$$

$$y = u^4$$

$$\frac{dy}{du} = 4u^3$$

$$\frac{du}{dx} = 2x$$

$$\begin{aligned} y &= (x^2 + 3)^4 \\ \frac{dy}{dx} &= \frac{dy}{du} \times \frac{du}{dx} \\ &= 4u^3 \times 2x \\ &= 4(x^2 + 3)^3 \times 2x \\ &= 8x(x^2 + 3)^3 \end{aligned}$$

Differentiation of Trigonometric Functions

$$\frac{d}{dx} \sin(x) = \cos(x)$$

$$\frac{d}{dx} \cos(x) = -\sin(x)$$

$$\frac{d}{dx} \tan(x) = \left(\frac{\sin(x)}{\cos(x)} \right)' = \frac{\cos^2(x) + \sin^2(x)}{\cos^2(x)}$$

Differentiation of Exponential Functions

$$(e^x)' = e^x$$

$$\begin{aligned} (e^{-x})' &= e^{-x} \times (-1) \\ &= -e^{-x} \end{aligned}$$

Differentiation of Sigmoid Function

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

$$\begin{aligned} \frac{d}{dx} \text{sigmoid}(x) &= ((1 + e^{-x})^{-1})' \\ &= -1 \times (1 + e^{-x})^{-2} \times (-e^{-x}) \\ &= \frac{e^{-x}}{(1 + e^{-x})^2} \\ &= \frac{1}{(1 + e^{-x})} \times \frac{(1 + e^{-x}) - 1}{(1 + e^{-x})} \\ &= \text{sigmoid}(x) \times (1 - \text{sigmoid}(x)) \end{aligned}$$

Gradient

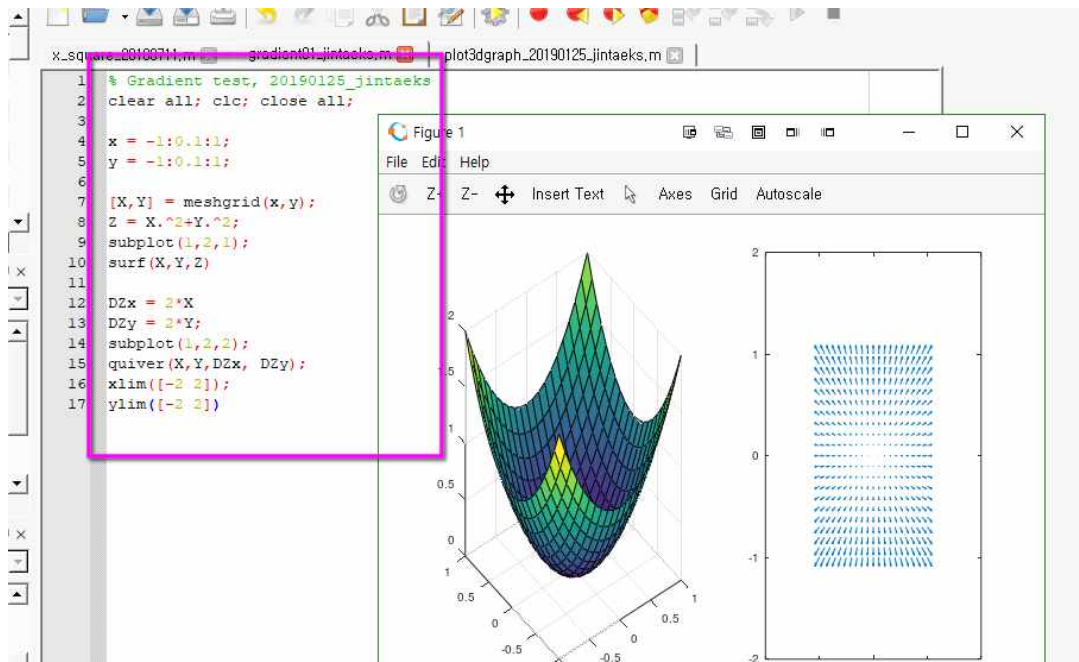
$$f(x,y) = x^2 \sin(y)$$

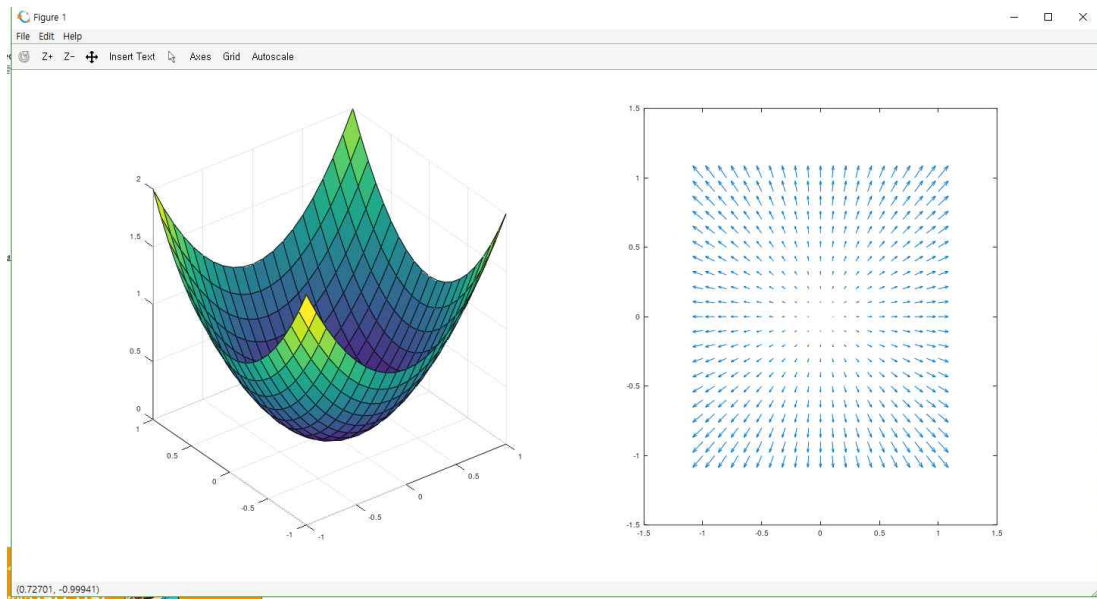
$$\frac{df}{dx} = 2x \sin(y), \frac{df}{dy} = x^2 \cos(y)$$

$$\nabla f(x,y) = \begin{bmatrix} 2x \sin(y) \\ x^2 \cos(y) \end{bmatrix}$$

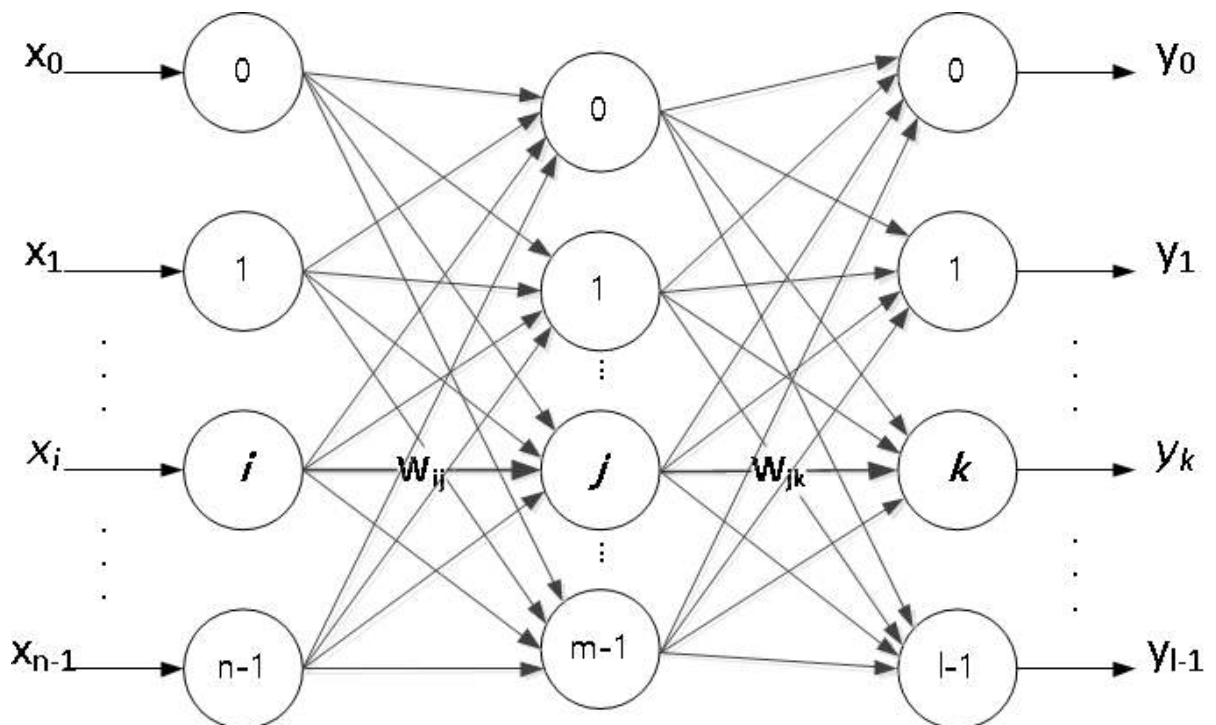
example: $f(x,y) = x^2 + y^2$

$$\nabla f(x,y) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

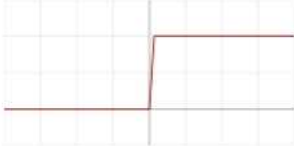







Neural Network



Activation Function

| | | |
|--|---|--|
| Binary step |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Logistic (a.k.a. Sigmoid or Soft step) |  | $f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \text{ [1]}$ |
| TanH |  | $f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$ |
| Rectified linear unit (ReLU) ^[15] |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |

It has been demonstrated for the first time in 2011 to enable better training of deeper networks, compared to the widely-used activation functions prior to 2011, e.g., the logistic sigmoid

Differentiation of Sigmoid Function

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

$$\begin{aligned}
 \frac{d}{dx} \text{sigmoid}(x) &= ((1 + e^{-x})^{-1})' \\
 &= -1 \times (1 + e^{-x})^{-2} \times (-e^{-x}) \\
 &= \frac{e^{-x}}{(1 + e^{-x})^2} \\
 &= \frac{1}{(1 + e^{-x})} \times \frac{(1 + e^{-x}) - 1}{(1 + e^{-x})} \\
 &= \text{sigmoid}(x) \times (1 - \text{sigmoid}(x))
 \end{aligned}$$

Back Propagation Algorithm

$$w_{ij}$$

$$w_{jk}$$

Update Weight w_{jk}

$$e_k(p) = y_{d,k}(p) - y_k(p)$$

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

$$\Delta w_{jk}(p) = \alpha \times y_j \times \delta_k(p) \quad : \delta_k(p) \text{ Error Gradient}$$

$$\delta_k(p) = \frac{\partial y_k(p)}{\partial X_k(p)} \times e_k(p) \quad : y_k(p) = \frac{1}{1 + e^{-X_k(p)}}$$

$$\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times e_k(p)$$

Update Weight w_{ij}

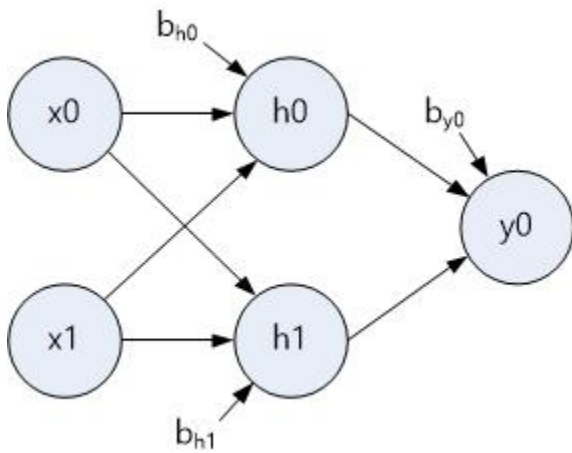
$$\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_i(p)$$

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^l \delta_k(p) w_{jk}(p)$$

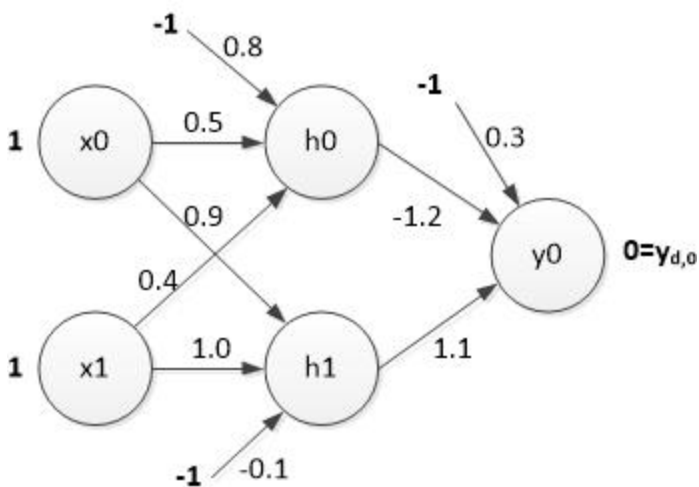
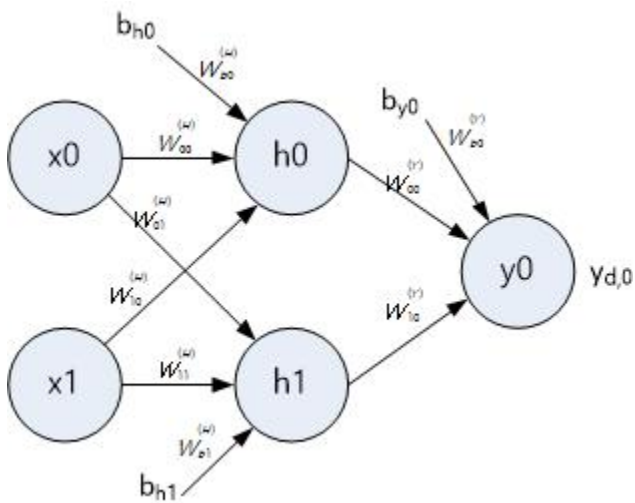
$$y_j(p) = \frac{1}{1 + e^{-X_j(p)}}$$

$$X_j(p) = \sum_{i=1}^n x_i(p) \times w_{ij}(p) + bias_j$$

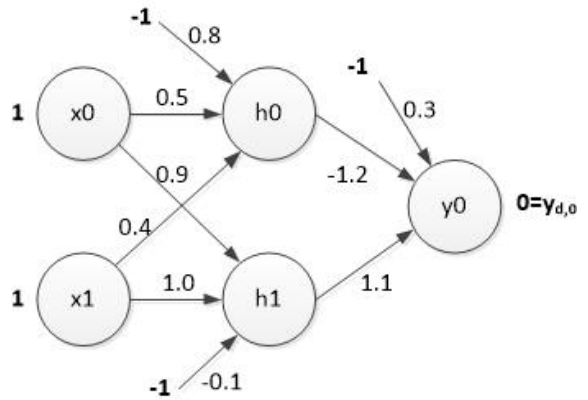
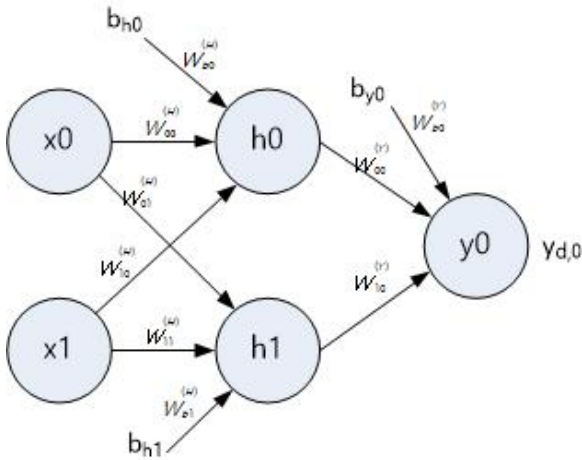
Example



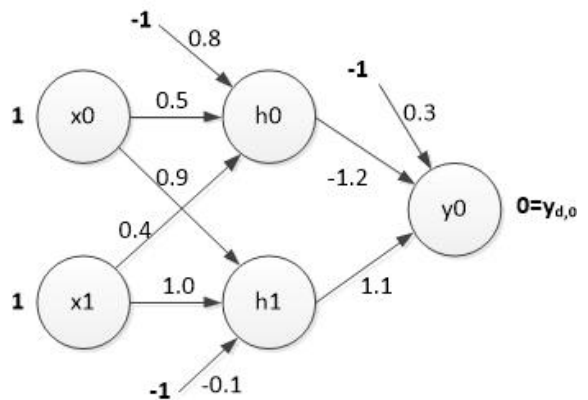
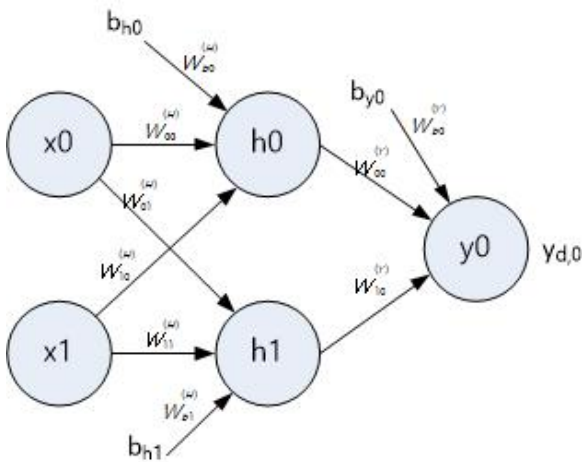
Forward Propagation



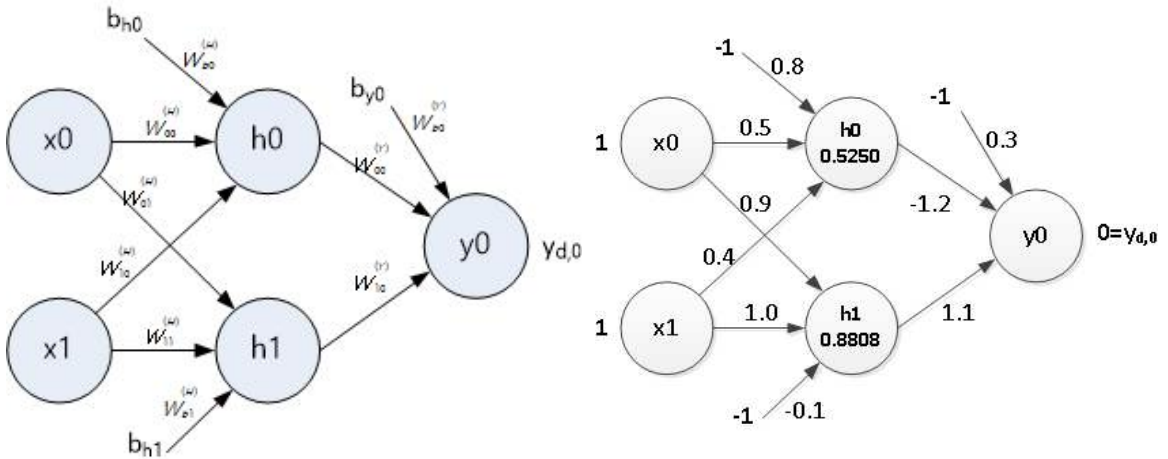
Steps



$$\begin{aligned}
 h0 &= \text{sigmoid}(x_0 w_{00}^{(H)} + x_1 w_{10}^{(H)} + b_{h0} w_{b0}^{(H)}) \\
 &= 1 / (1 + e^{-(x_0 w_{00}^{(H)} + x_1 w_{10}^{(H)} + b_{h0} w_{b0}^{(H)})}) \\
 &= 1 / (1 + e^{-(1 \times 0.5 + 1 \times 0.4 + (-1) \times 0.8)}) \\
 &= 0.5250
 \end{aligned}$$



$$\begin{aligned}
 h1 &= \text{sigmoid}(x_0 w_{01}^{(H)} + x_1 w_{11}^{(H)} + b_{h1} w_{b1}^{(H)}) \\
 &= 1 / (1 + e^{-(x_0 w_{01}^{(H)} + x_1 w_{11}^{(H)} + b_{h1} w_{b1}^{(H)})}) \\
 &= 1 / (1 + e^{-(1 \times 0.9 + 1 \times 1.0 + (-1) \times (-0.1))}) \\
 &= 0.8808
 \end{aligned}$$



$$\begin{aligned}
 y_0 &= \text{sigmoid}(h_0 w_{00}^{(Y)} + h_1 w_{10}^{(Y)} + b_{y0} w_{b0}^{(Y)}) \\
 &= 1 / (1 + e^{-(0.5250 \times (-1.2) + 0.8808 \times 1.1 + (-1) \times 0.3)}) \\
 &= 0.5097
 \end{aligned}$$

$$e = y_{d,0} - y_0 = 0 - 0.5097 = -0.5097$$

Calculating Sigmoid

1. Python

import math

print (1/(1+math.exp(-1*(0.5250*-1.2+0.8808*1.1+-1*0.3))))

The screenshot shows the OnlineGDB web interface. The code being executed is as follows:

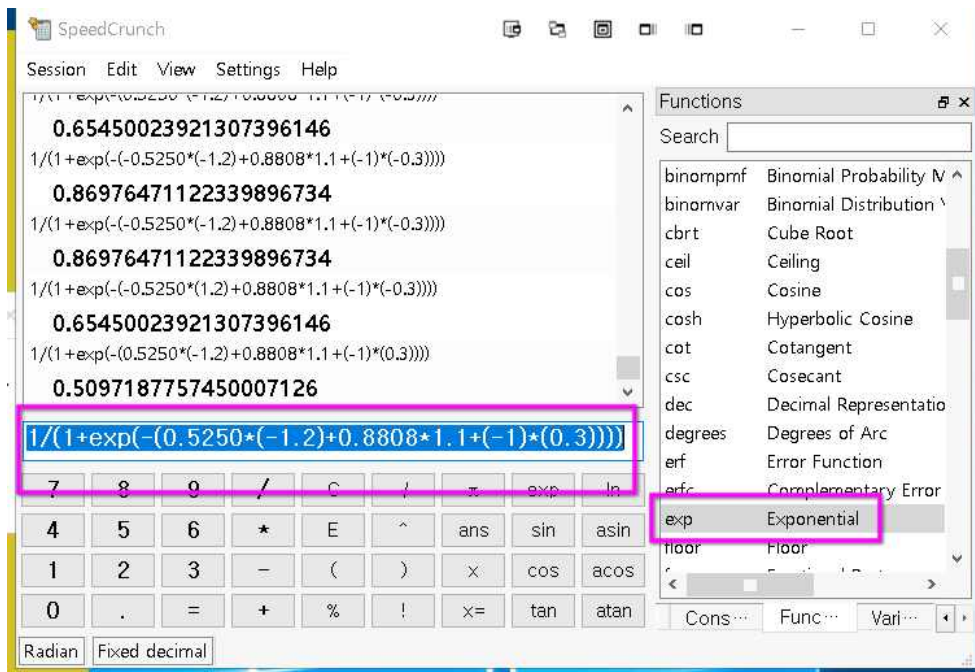
```

1 import math
2 print (1/(1+math.exp(-1*(0.5250*-1.2+0.8808*1.1+-1*0.3))))
3 print(0.8808*(1-0.8808)*(-0.1274)*1.0888)
4 print(0.5250*(1-0.5250)*(-0.1274)*(-1.2067))

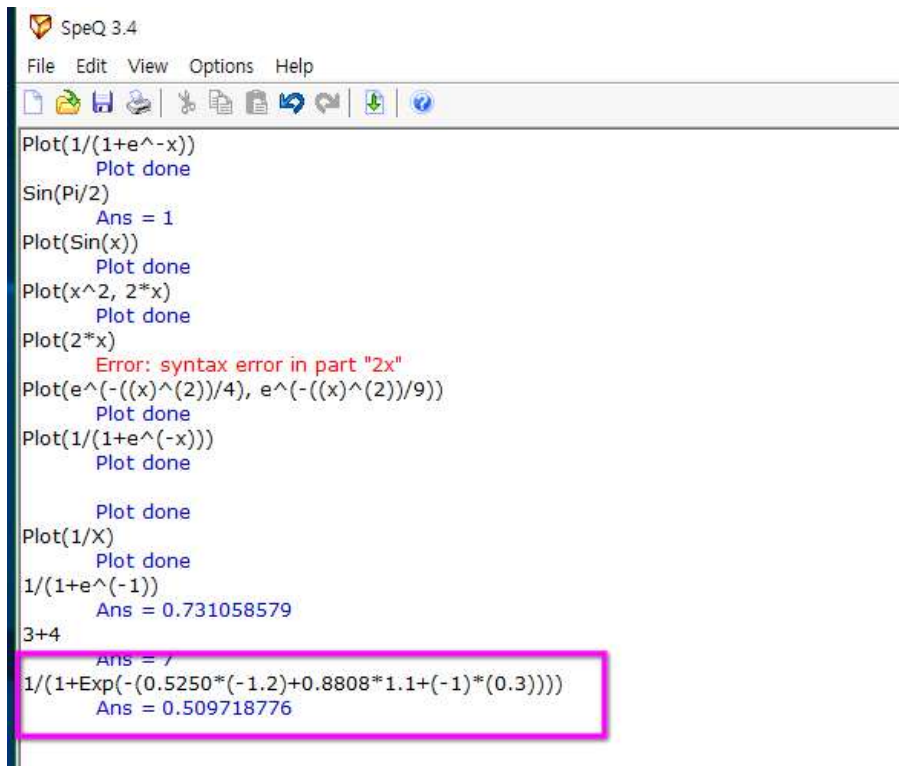
```

The output of the code is displayed on the right side of the interface.

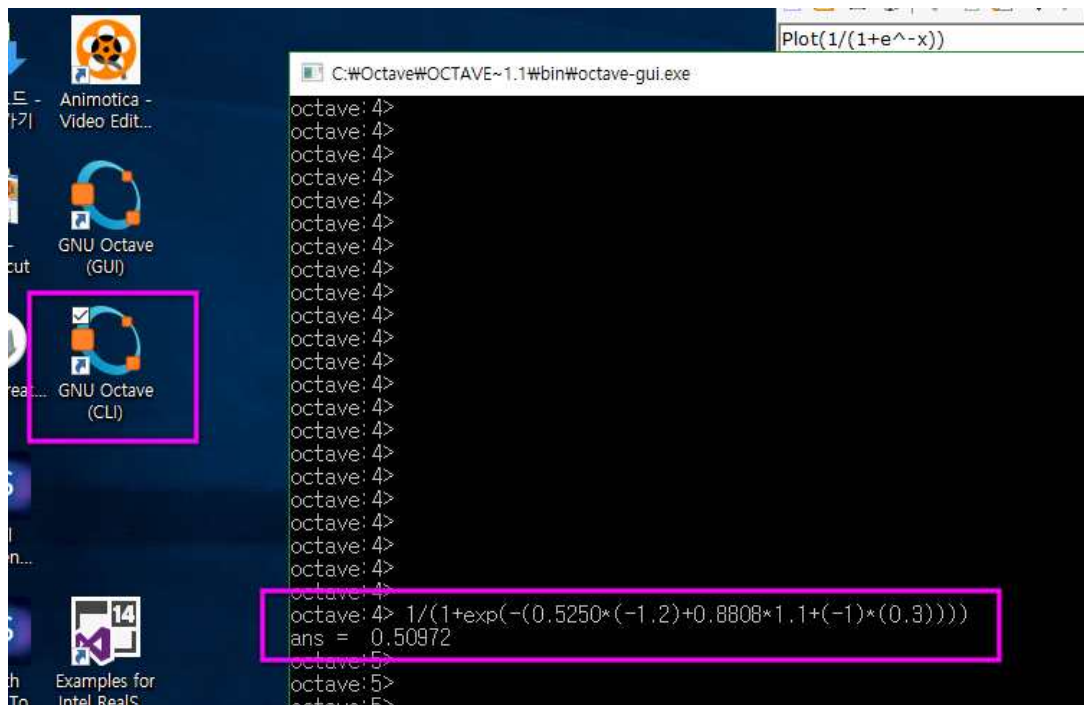
2. SpeedCrunch



3. SpeQ

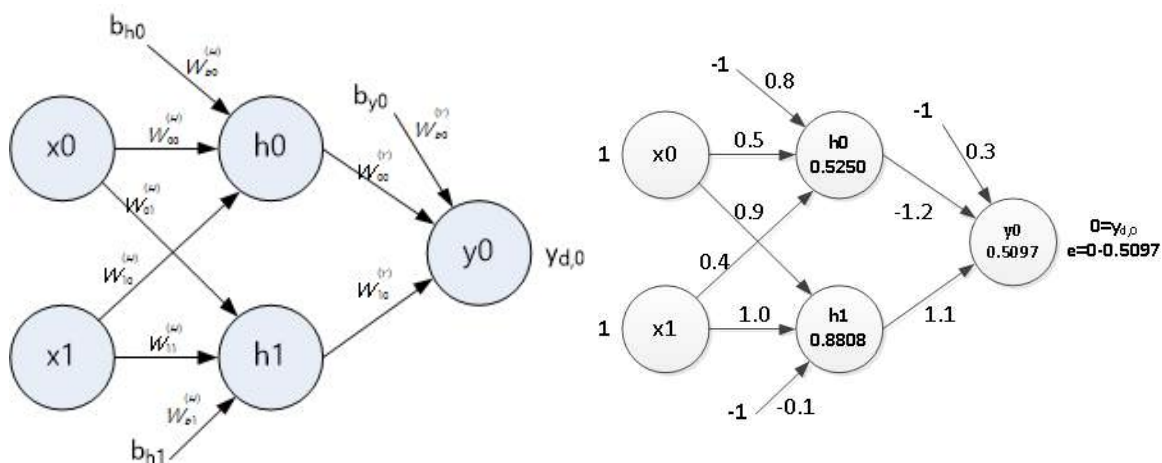


4. Octave CLI



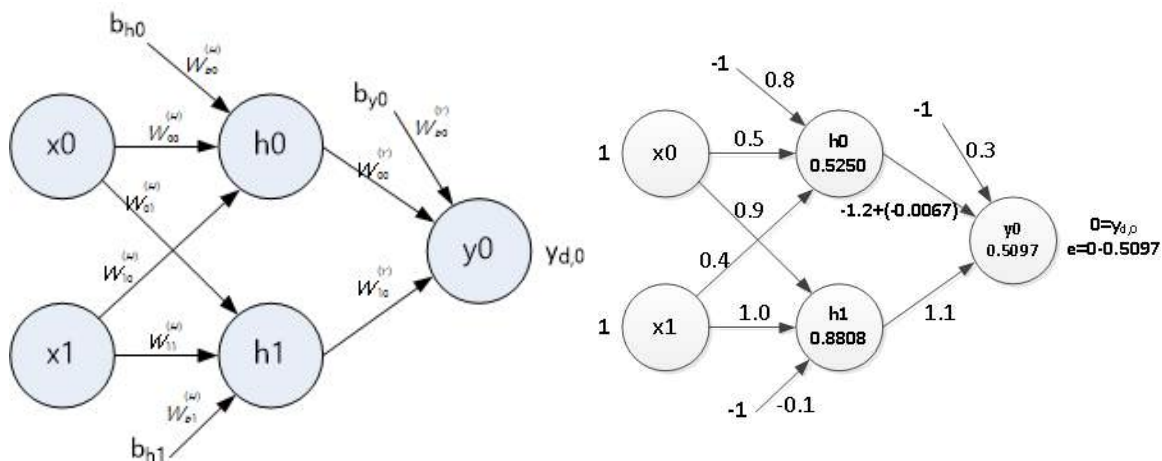
Backward Propagation

Output Layer



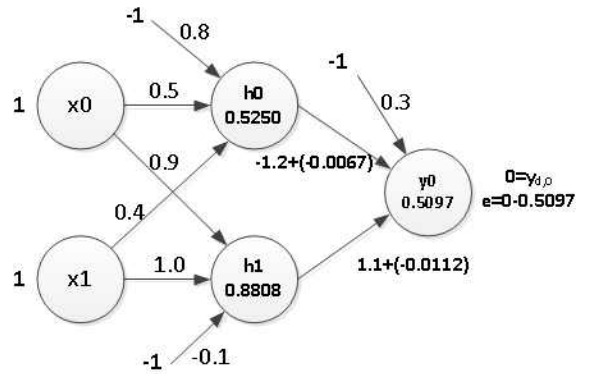
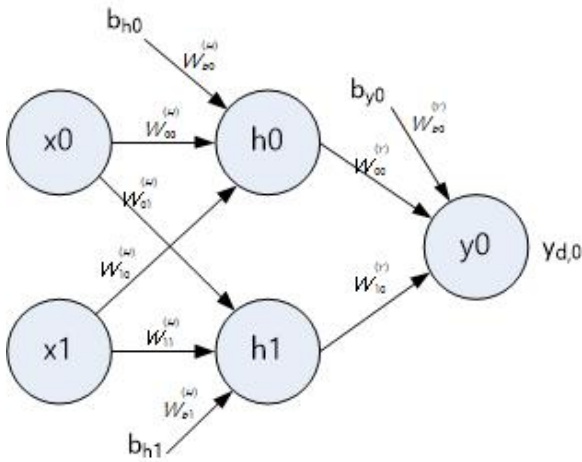
error gradient

$$\begin{aligned}\delta_0^Y &= y_0(1 - y_0)e \\ &= 0.5097 \times (1 - 0.5097) \times (-0.5097) \\ &= -0.1274\end{aligned}$$

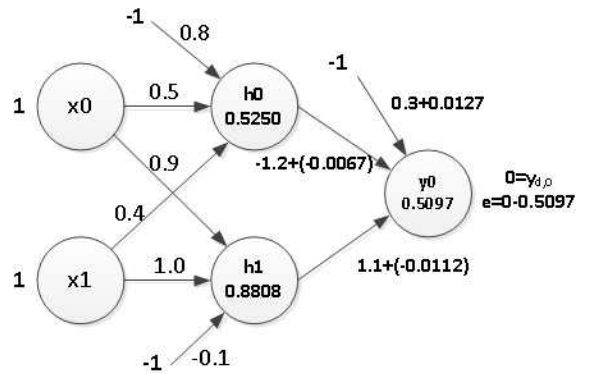
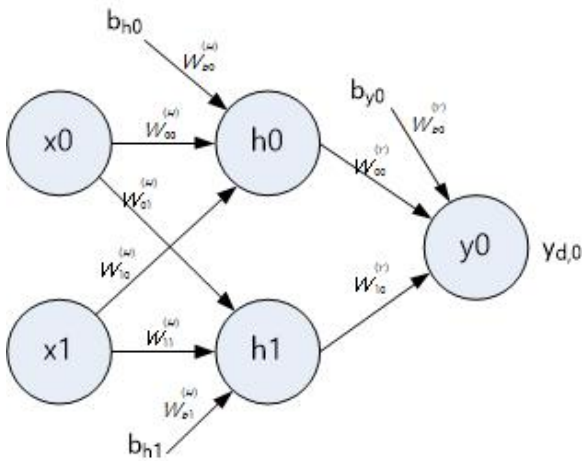


learning ratio $\alpha = 0.1$

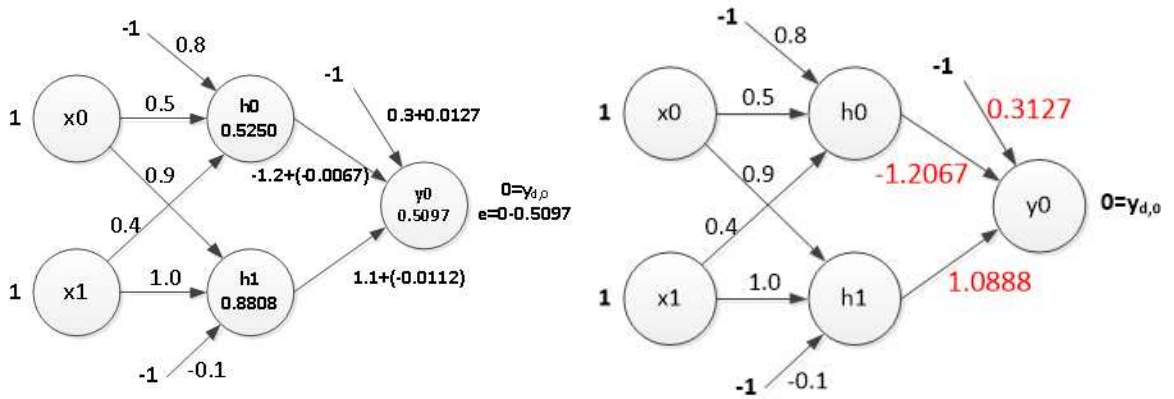
$$\begin{aligned}\Delta w_{00}^Y &= \alpha \times h_0 \times \delta_0 \\ &= 0.1 \times 0.5250 \times (-0.1274) \\ &= -0.0067\end{aligned}$$



$$\begin{aligned}\Delta w_{10}^Y &= \alpha \times h_1 \times \delta_0 \\ &= 0.1 \times 0.8808 \times (-0.1274) \\ &= -0.0112\end{aligned}$$



$$\begin{aligned}\Delta w_{b0}^Y &= \alpha \times b_{y0} \times \delta_0 \\ &= 0.1 \times (-1) \times (-0.1274) \\ &= 0.0127\end{aligned}$$



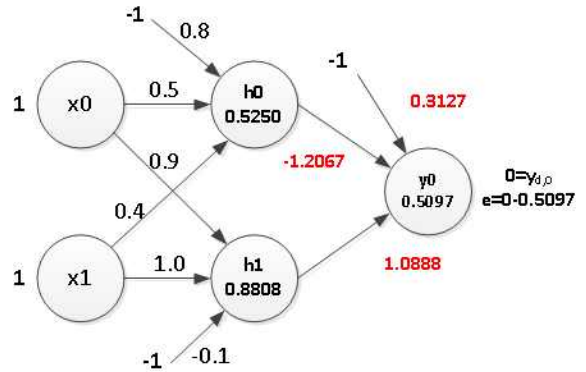
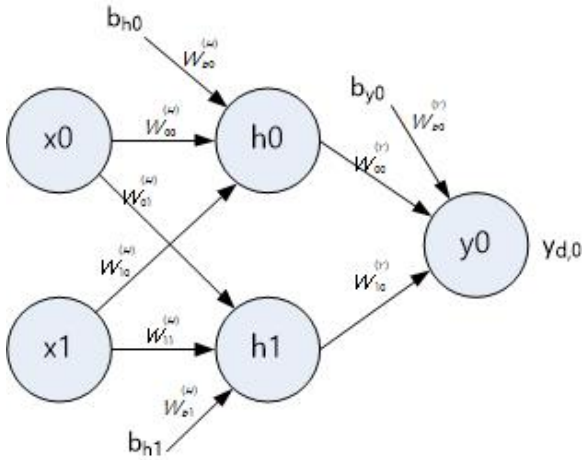
Update Output Weights

$$w_{00}^Y = w_{00}^Y + \Delta w_{00}^Y = -1.2 + (-0.0067) = -1.2067$$

$$w_{10}^Y = w_{10}^Y + \Delta w_{10}^Y = 1.1 + (-0.0112) = 1.0888$$

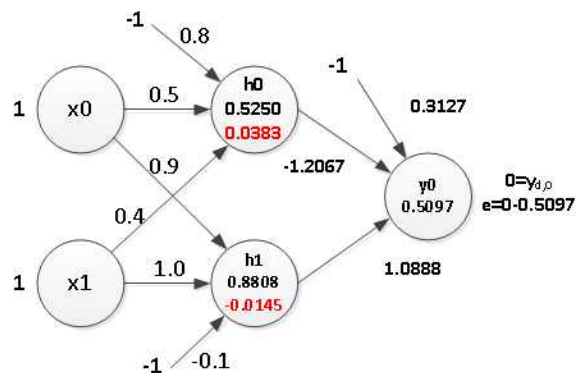
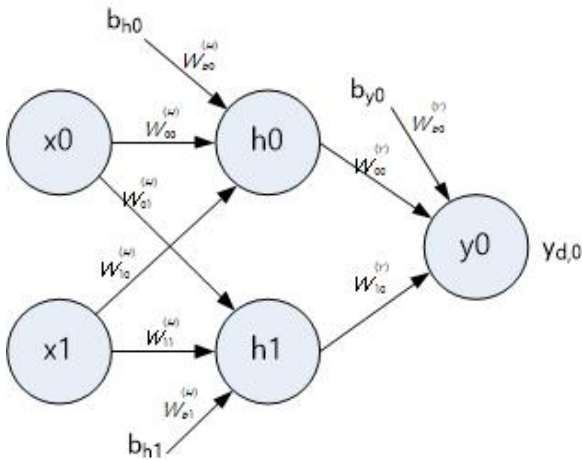
$$w_{b0}^Y = w_{b0}^Y + \Delta w_{b0}^Y = 0.3 + (0.0127) = 0.3127$$

Hidden Layer



$$\begin{aligned}\delta_0^H &= h_0(1 - h_0) \times \delta_0^Y \times w_{00}^Y \\ &= 0.5250 \times (1 - 0.5250) \times (-0.1274) \times (-1.2067) \\ &= 0.0383\end{aligned}$$

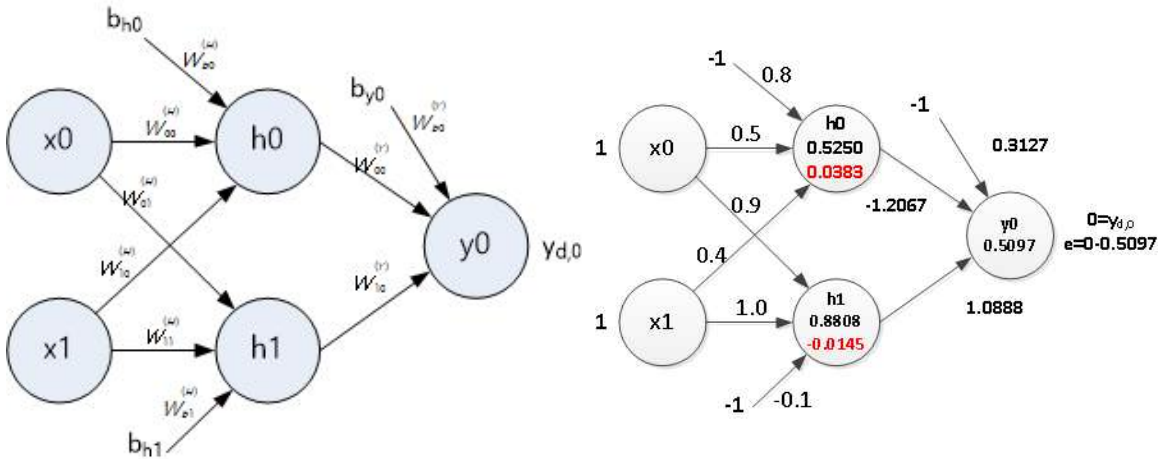
$$\begin{aligned}\delta_1^H &= h_1(1 - h_1) \times \delta_0^Y \times w_{10}^Y \\ &= 0.8808 \times (1 - 0.8808) \times (-0.1274) \times (1.0888) \\ &= -0.0145\end{aligned}$$



$$\Delta w_{00}^H = \alpha \times x_0 \times \delta_0^H = 0.1 \times 1 \times 0.0383 = 0.0038$$

$$\Delta w_{10}^H = \alpha \times x_1 \times \delta_0^H = 0.1 \times 1 \times 0.0383 = 0.0038$$

$$\Delta w_{b0}^H = \alpha \times b_{h0} \times \delta_0^H = 0.1 \times (-1) \times 0.0383 = -0.0038$$

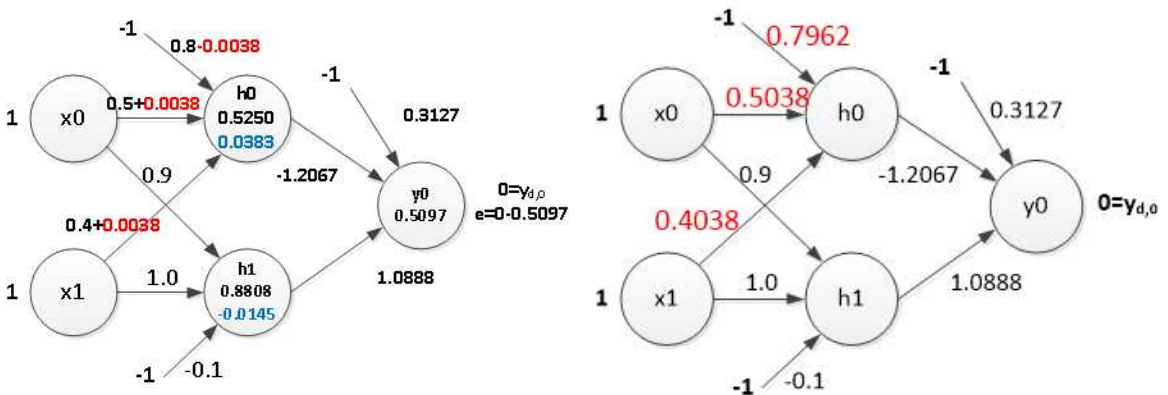


$$\Delta w_{01}^H = \alpha \times x_0 \times \delta_1^H = 0.1 \times 1 \times (-0.0145) = -0.0015$$

$$\Delta w_{11}^H = \alpha \times x_1 \times \delta_1^H = 0.1 \times 1 \times (-0.0145) = -0.0015$$

$$\Delta w_{b1}^H = \alpha \times b_{h1} \times \delta_1^H = 0.1 \times (-1) \times (-0.0145) = 0.0015$$

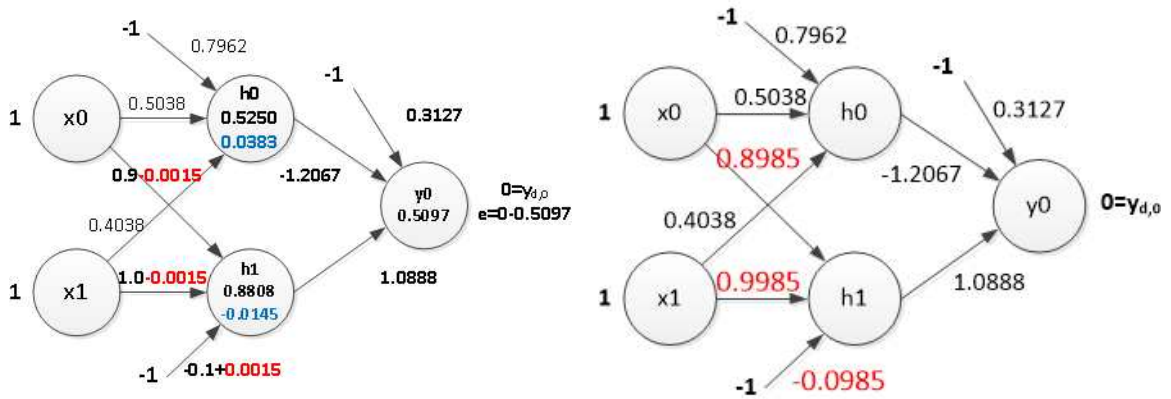
Update Hidden Weights



$$w_{00}^H = w_{00}^H + \Delta w_{00}^H = 0.5 + (0.0038) = 0.5038$$

$$w_{10}^H = w_{10}^H + \Delta w_{10}^H = 0.4 + (0.0038) = 0.4038$$

$$w_{b0}^H = w_{b0}^H + \Delta w_{b0}^H = 0.8 + (-0.0038) = 0.7962$$



$$w_{01}^H = w_{01}^H + \Delta w_{01}^H = 0.9 + (-0.0015) = 0.8985$$

$$w_{11}^H = w_{11}^H + \Delta w_{11}^H = 1.0 + (-0.0015) = 0.9985$$

$$w_{b1}^H = w_{b1}^H + \Delta w_{b1}^H = (-0.1) + 0.0015 = -0.0985$$

Sum of Square Errors

$$SSE = \sum_{i=1}^n (x_i - \bar{x})^2$$

$$D_i = \sum_{i=0}^n (y_i - y_{d,i})^2$$

$$SSE = \sum_{i=0}^n (y_i - \bar{y})^2 + \sum_{i=0}^n (y_{d,i} - \bar{y})^2$$

$$= D_i / 2 = \left\{ \sum_{i=0}^n (y_i - y_{d,i})^2 \right\} / 2$$


Naive C Implementation

<https://www.gnu.org/software/gneuralnetwork/>

seojt Keep 주소록 캘린더 Gmail 검색 진택 위키 Wiki 동서대 참고 YouTube All Courses

Skip to main text Set language

English [en]

 **GNU Operating System**
Sponsored by the Free Software Foundation

ABOUT GNU PHILOSOPHY LICENSES EDUCATION SOFTWARE DOCS HELP GNU More

GNU Gneural Network

Introduction

Gneural Network is the GNU package which implements a programmable neural network. The current version, 0.9.1, has th

https://rimstar.org/science_electronics_projects/backpropagation_neural_network_software_3_layer.htm

https://rimstar.org/science_electronics_projects/backpropagation_neural_network_software_3_layer.htm

Keep 주소록 캘린더 Gmail 검색 진택 위키 Wiki 동서대 참고 YouTube All Courses

rimstar.org
an addiction to science,
renewable energy and
building stuff

Backpropagation neural network software (3 layer)

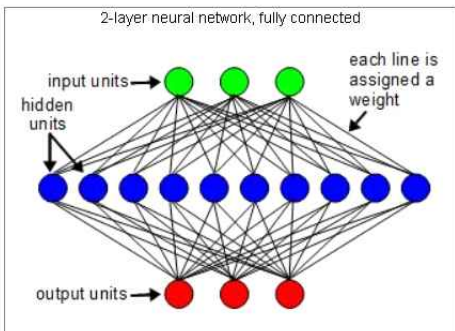
Home
Renewable Energy
Science/Electronics
Projects
Build Van de Graaff
Wimshurst machine
Kelvin water dropper
Electroscope
Corona motors
Ball bearing motor
Crystal radios
AM Radio Transmitter
Fresnel lens
Electrets
Piezoelectric
Electrolytic capacitor
Joule thief
Peltier/Seebeck effect
Franklins bell
Can Stirling
Big Stirling
555 timer music
TEA laser

This page is about a simple and configurable neural network software library I wrote a while ago that uses the backpropagation algorithm to teach it. The software is written in C and is available and detailed below so that anyone can use it.

2-layer neural network, fully connected

input units → hidden units → output units

each line is assigned a weight



C++ conversion

<https://github.com/GP101/UnderstandingUnity/tree/master/NeuralNetwork>

GitHub, Inc. [US] | <https://github.com/GP101/UnderstandingUnity/tree/master/NeuralNetwork>

주소록 14 캘린더 Gmail 검색 진택 위키 Wiki 동서대 참고 YouTube All Courses

Pull requests Issues Marketplace Explore

GP101 / UnderstandingUnity Unwatch 5

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

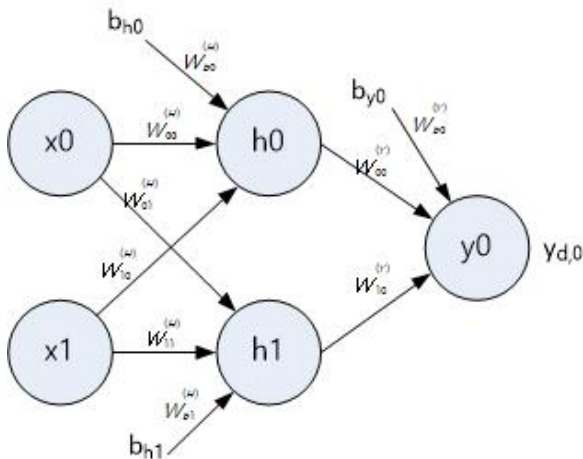
Branch: master UnderstandingUnity / NeuralNetwork / Create new file Upload

jintaeks first commit for Unity Machine Learning related stuff. ... La

..

| | |
|--|--|
| ConsoleApplication_NeuralNetwork | first commit for Unity Machine Learning related stuff. |
| OctaveFiles | first commit for Unity Machine Learning related stuff. |
| Tensorflow | first commit for Unity Machine Learning related stuff. |
| NeuralNetwork Backpropagation Example... | first commit for Unity Machine Learning related stuff. |
| NeuralNetworkSimplePerceptronExample_... | first commit for Unity Machine Learning related stuff. |
| NeuralNetworkSimplePerceptronExample_... | first commit for Unity Machine Learning related stuff. |
| NeuralNetwork_20190131_JintaekSeo.pdf | first commit for Unity Machine Learning related stuff. |

Implementation Issues



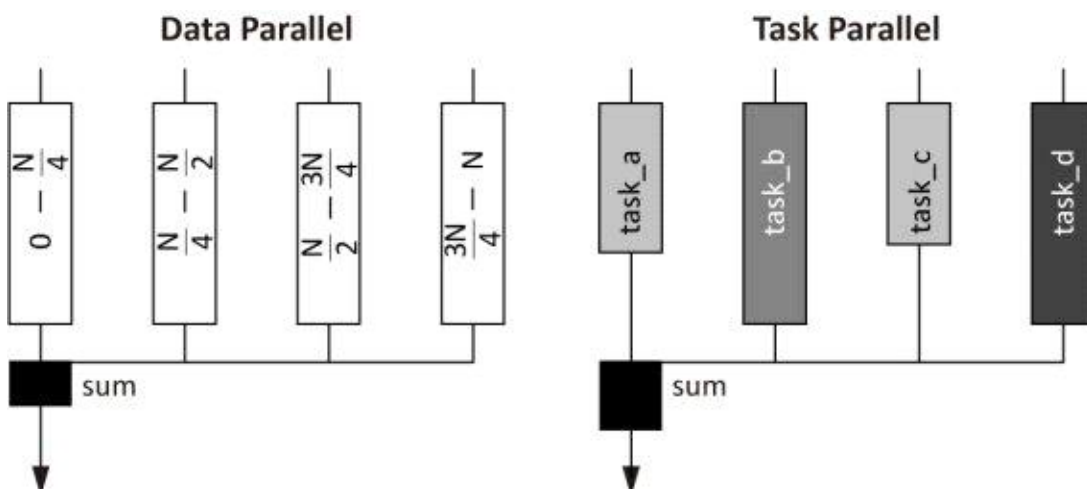
$$h0 = \text{sigmoid}(x_0 w_{00}^{(H)} + x_1 w_{10}^{(H)} + b_{h0} w_{b0}^{(H)})$$

$$h1 = \text{sigmoid}(x_0 w_{01}^{(H)} + x_1 w_{11}^{(H)} + b_{h1} w_{b1}^{(H)})$$

$$\begin{bmatrix} x_0 & x_1 \end{bmatrix} \begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} + \begin{bmatrix} bias_0 \\ bias_1 \end{bmatrix} = \begin{bmatrix} WeightSum_0 \\ WeightSum_1 \end{bmatrix}$$

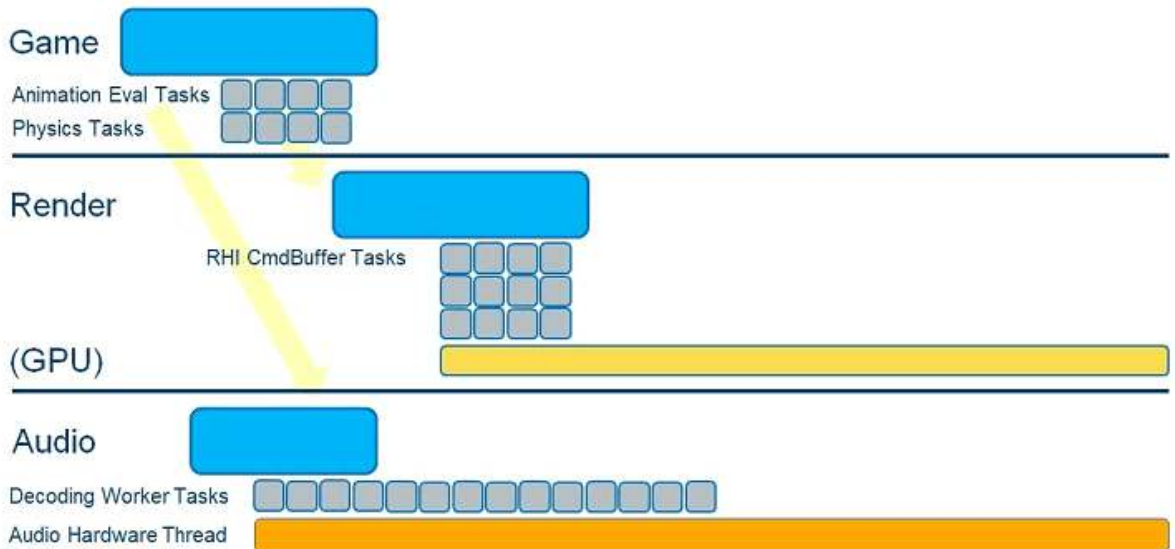
$$\begin{bmatrix} W_{00} & W_{10} \\ W_{01} & W_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} + \begin{bmatrix} bias_0 \\ bias_1 \end{bmatrix} = \begin{bmatrix} WeightSum_0 \\ WeightSum_1 \end{bmatrix}$$

How to implement with multi-thread?



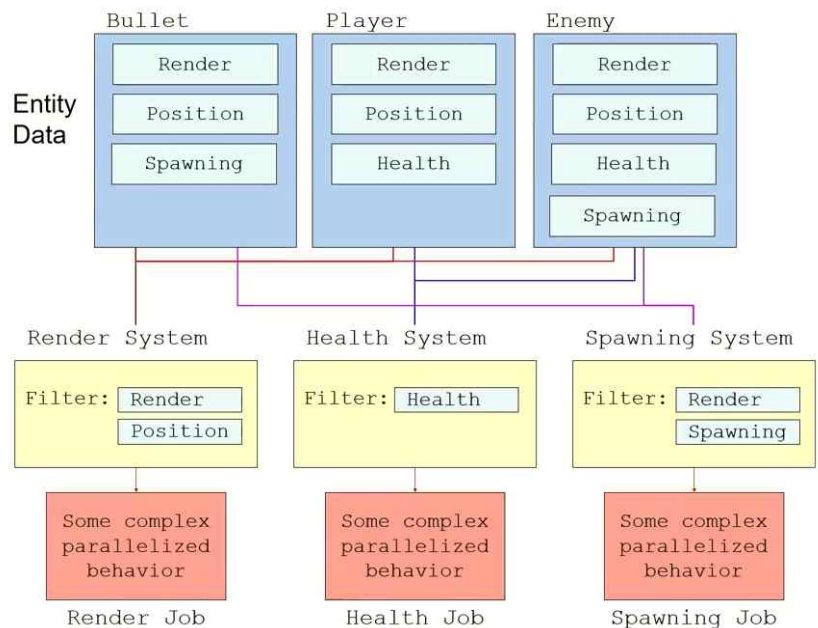
Unreal Engine 4 threading model

<https://software.intel.com/en-us/articles/intel-software-engineers-assist-with-unreal-engine-419-optimizations>

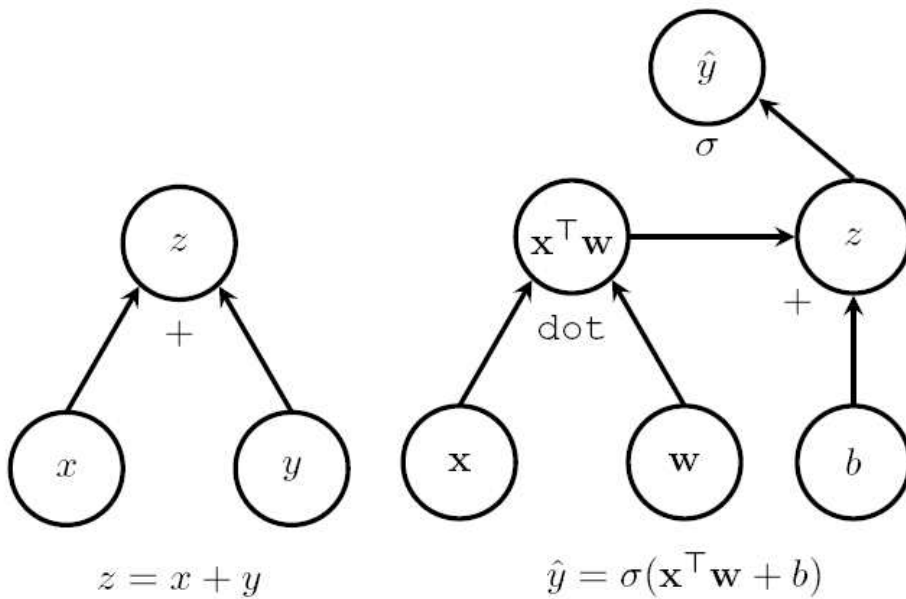


Unity ECS(Entity Component System)

Entity Component System



Computational Graph



Ex)

$$\begin{bmatrix} W_{00} & W_{10} \\ W_{01} & W_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} + \begin{bmatrix} bias_0 \\ bias_1 \end{bmatrix} = \begin{bmatrix} WeightSum_0 \\ WeightSum_1 \end{bmatrix}$$

Tensor

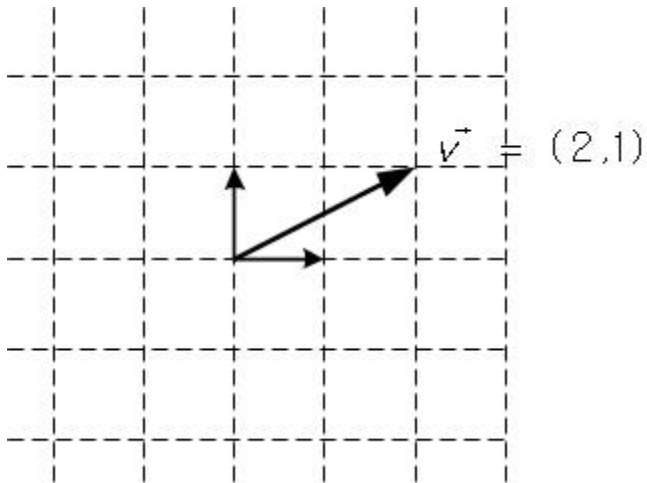
Scalar

mass(real number)

Vector

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

$$v = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$



Covector

a function for a column vector.

$$\begin{bmatrix} 2 & 1 \end{bmatrix} \left(\begin{bmatrix} 4 \\ 5 \end{bmatrix} \right) = 2 \times 4 + 1 \times 5 = 13$$

$$\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = 2x + 1y$$

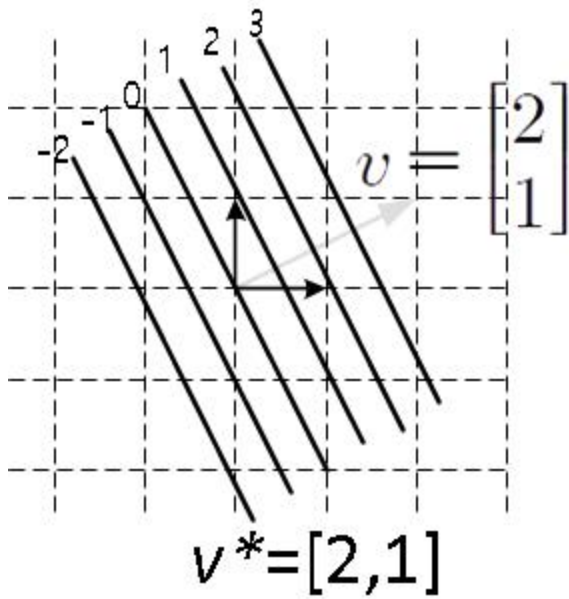
$$2x + 1y = -2$$

$$2x + 1y = -1$$

$$2x + 1y = 0$$

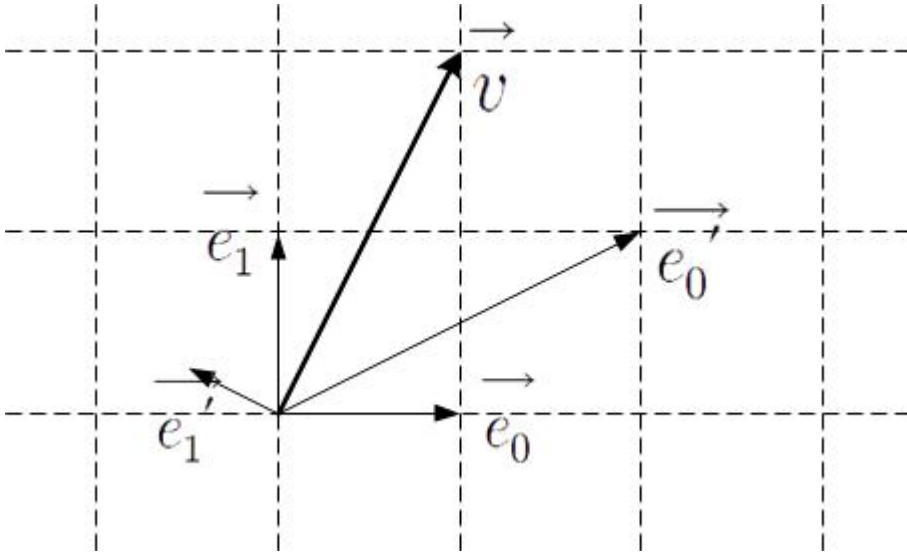
$$2x + 1y = 1$$

$$2x + 1y = 2$$



Dual Space

Metric Tensor



$$\vec{e}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \vec{e}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \vec{v} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\vec{v} = 1 \times \vec{e}_0 + 2 \times \vec{e}_1$$

$$\vec{v} = 1 \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 2 \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\|\vec{v}\|^2 = \vec{v} \cdot \vec{v}$$

$$\begin{aligned} \|\vec{v}\|^2 &= (v^0 \vec{e}_0 + v^1 \vec{e}_1) \cdot (v^0 \vec{e}_0 + v^1 \vec{e}_1) \\ &= (v^0)^2 (\vec{e}_0 \cdot \vec{e}_0) + 2v^0 v^1 (\vec{e}_0 \cdot \vec{e}_1) + (v^1)^2 (\vec{e}_1 \cdot \vec{e}_1) \end{aligned}$$

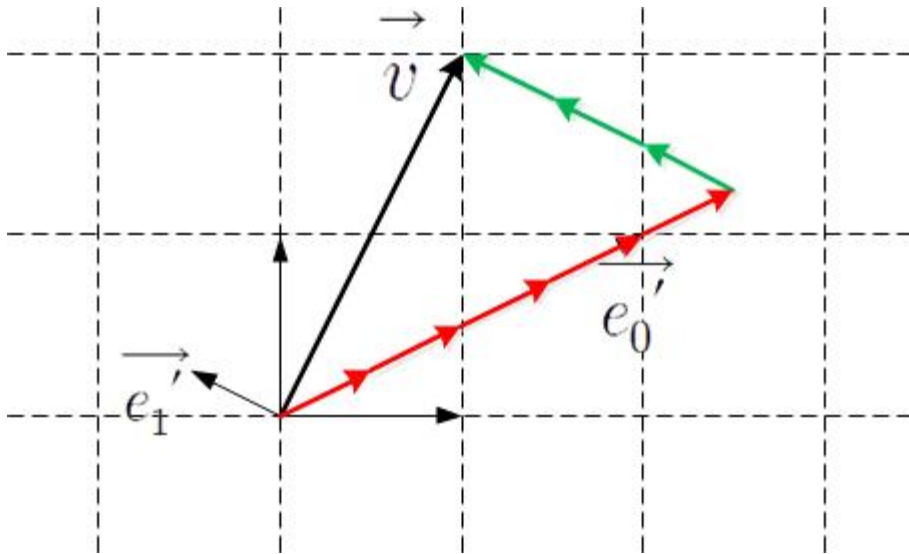
$$\begin{bmatrix} v_0 & v_1 \end{bmatrix} \begin{bmatrix} (\vec{e}_0 \cdot \vec{e}_0) & (\vec{e}_0 \cdot \vec{e}_1) \\ (\vec{e}_1 \cdot \vec{e}_0) & (\vec{e}_1 \cdot \vec{e}_1) \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \end{bmatrix}$$

$$\begin{bmatrix} v_0 & v_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \end{bmatrix}$$

$$= \begin{bmatrix} v_0 & v_1 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \end{bmatrix}$$

$$= (v_0)^2 + (v_1)^2$$

$$1^2 + 2^2 = 5$$



$$\vec{e_0'} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\vec{e_1'} = \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{4} \end{bmatrix}$$

$$\begin{bmatrix} \frac{5}{4} & 3 \end{bmatrix} \begin{bmatrix} (\vec{e_0'} \cdot \vec{e_0'}) & (\vec{e_0'} \cdot \vec{e_1'}) \\ (\vec{e_1'} \cdot \vec{e_0'}) & (\vec{e_1'} \cdot \vec{e_1'}) \end{bmatrix} \begin{bmatrix} \frac{5}{4} \\ 3 \end{bmatrix}$$

$$(\vec{e_0'} \cdot \vec{e_0'}) = 5$$

$$(\vec{e_0'} \cdot \vec{e_1'}) = -\frac{3}{4}$$

$$(\vec{e_1'} \cdot \vec{e_1'}) = \frac{5}{16}$$

$$\begin{bmatrix} \frac{5}{4} & 3 \end{bmatrix} \begin{bmatrix} 5 & -\frac{3}{4} \\ -\frac{3}{4} & \frac{5}{16} \end{bmatrix} \begin{bmatrix} \frac{5}{4} \\ 3 \end{bmatrix} = 5$$

Machine Learning Library: TensorFlow



@