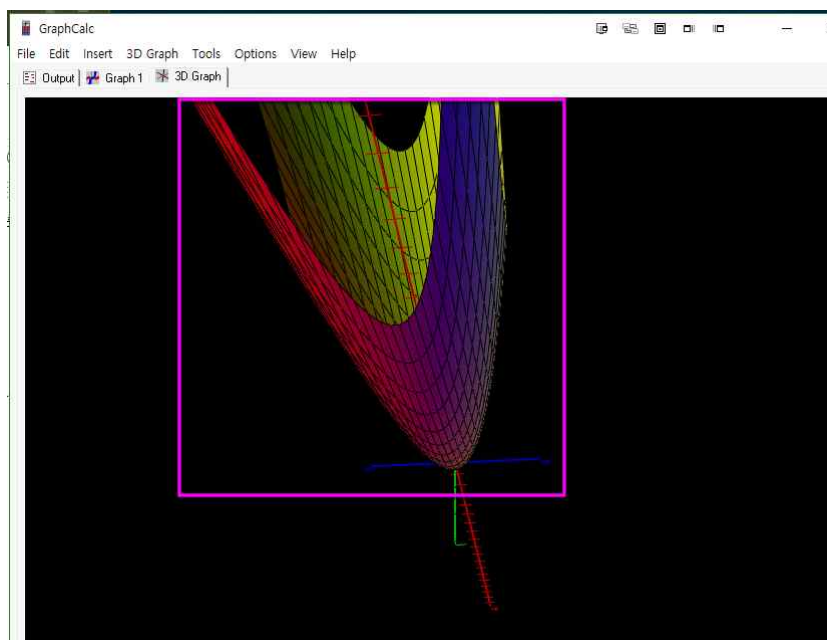
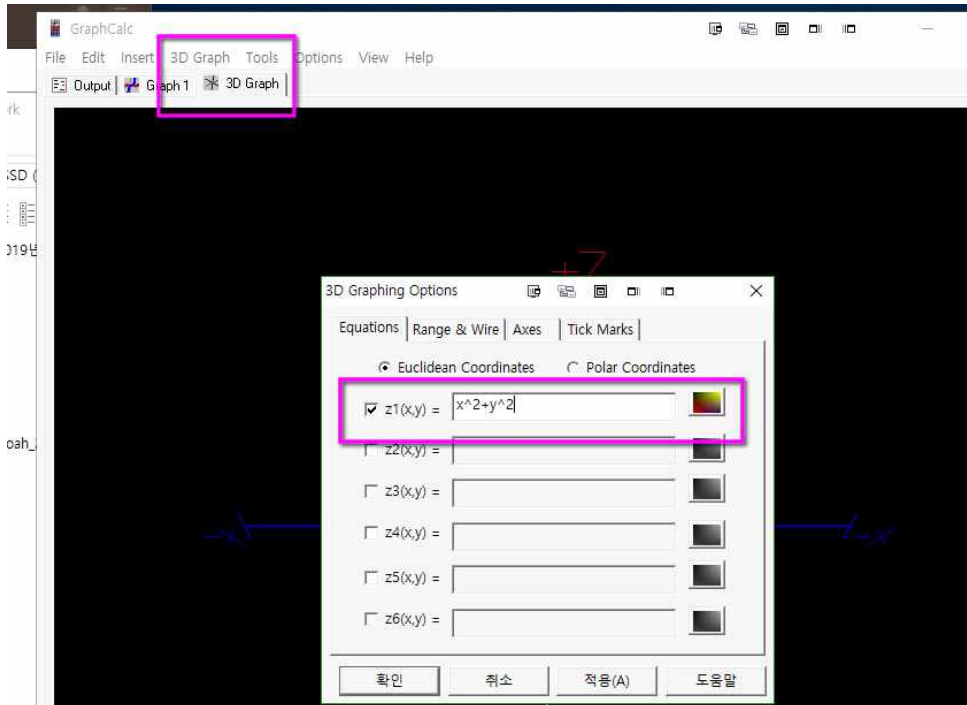


인공 신경망: Back-propagation

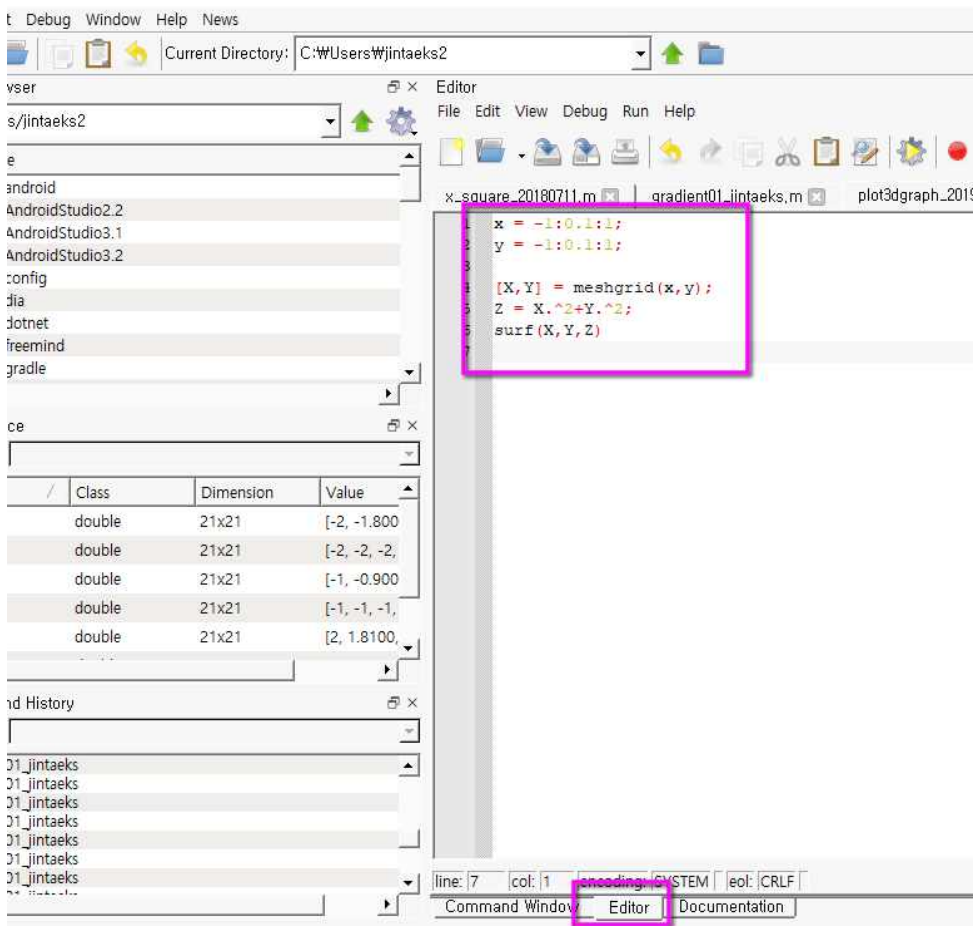
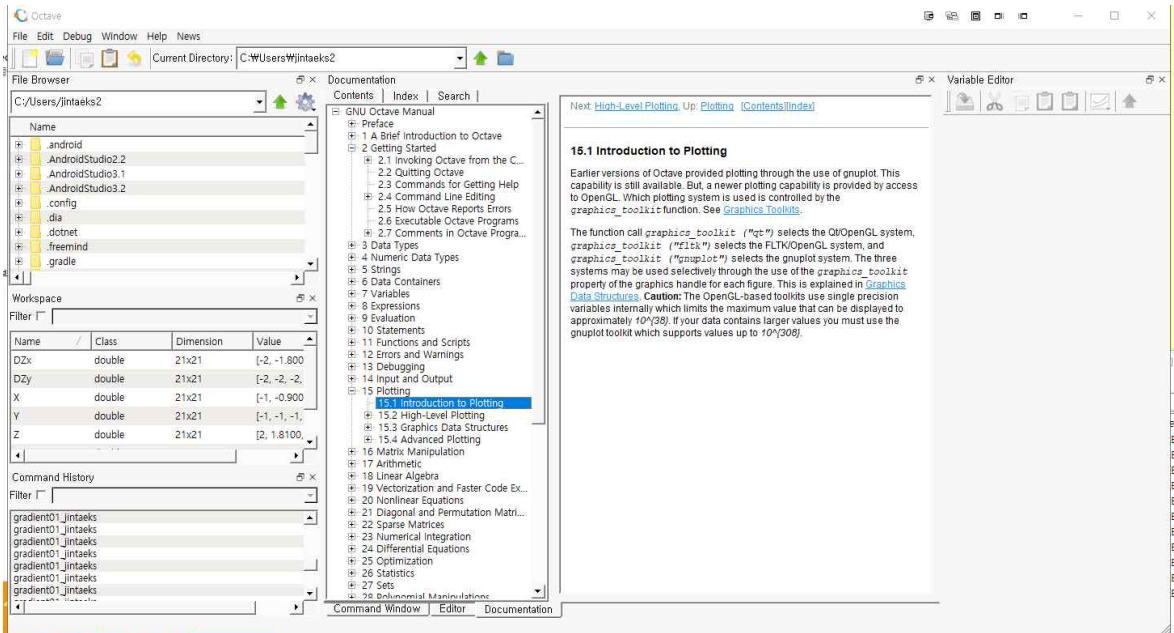
> 2019년1월31일, 서진택

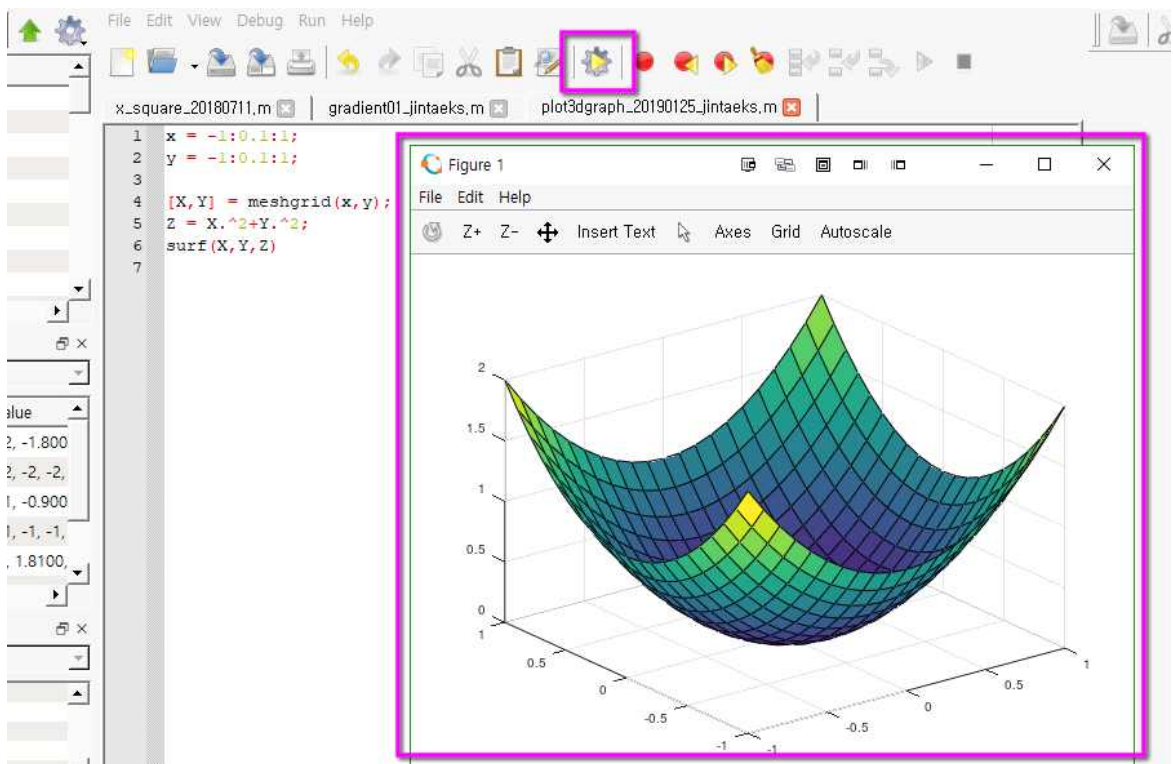
Plotting 3D Graph

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

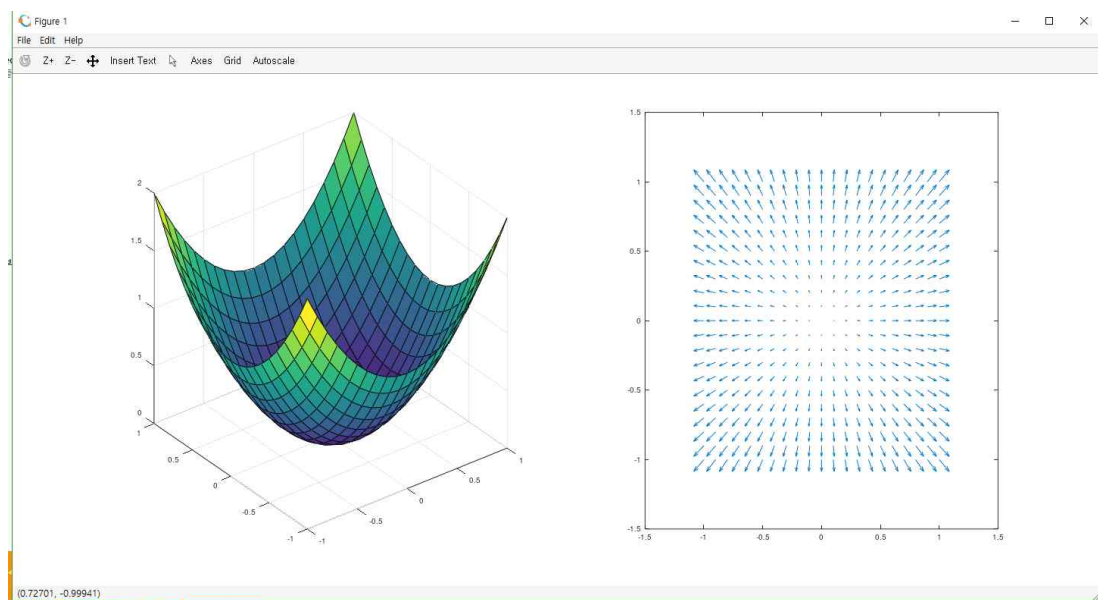
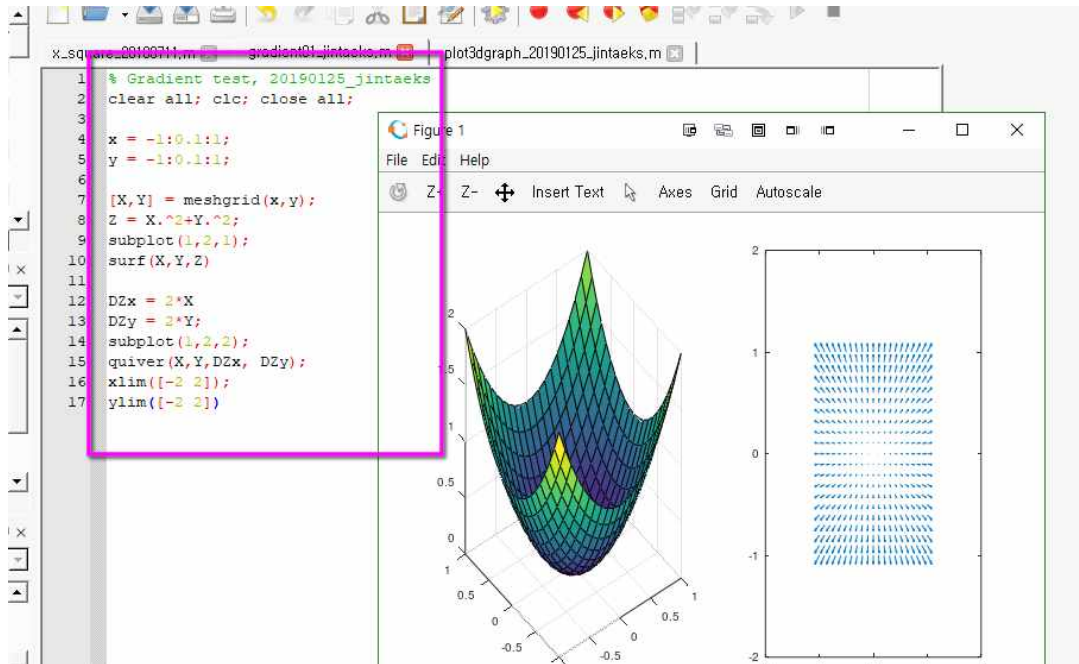
$$f(x, y) = x^2 \sin(y)$$

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

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

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

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

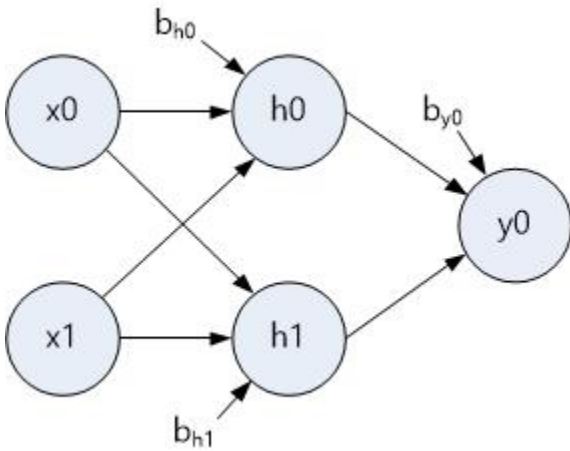


Neural Network

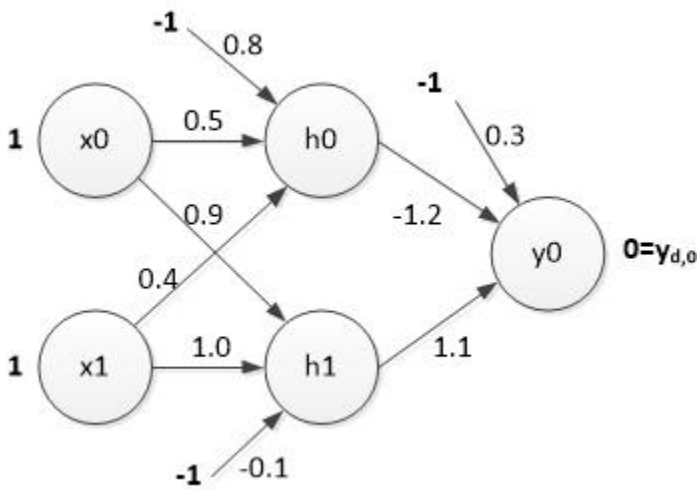
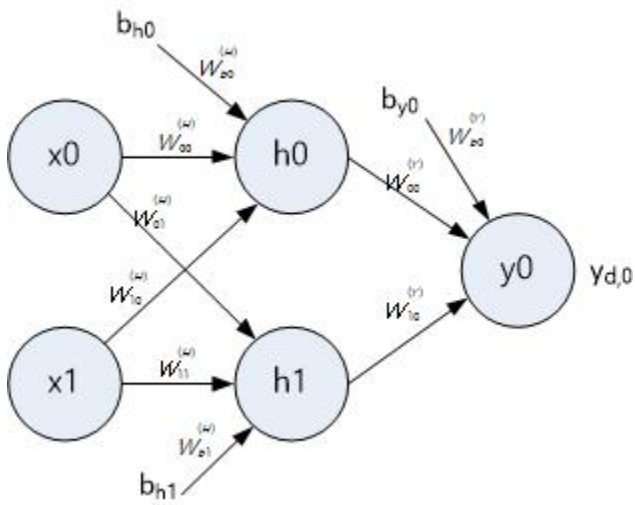
Differentiation of Sigmoid Function

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

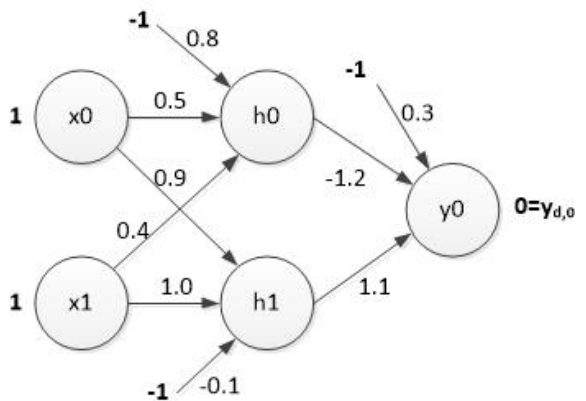
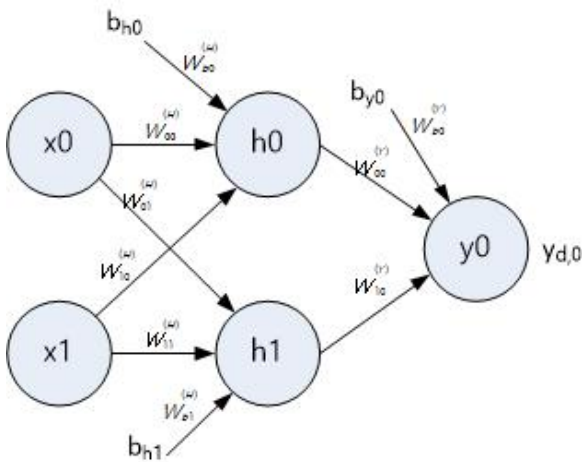
$$\begin{aligned}\frac{d}{dx} \text{sigmoid}(x) &= ((1 + e^{-x})^{-1})' \\ &= -1 \times (1 + e^{-x})^{-2} \times (-e^{-x}) \\ &= \frac{e^{-x}}{(1 + e^{-x})^2} \\ &= \frac{1}{(1 + e^{-x})} \times \frac{(1 + e^{-x}) - 1}{(1 + e^{-x})} \\ &= \text{sigmoid}(x) \times (1 - \text{sigmoid}(x))\end{aligned}$$



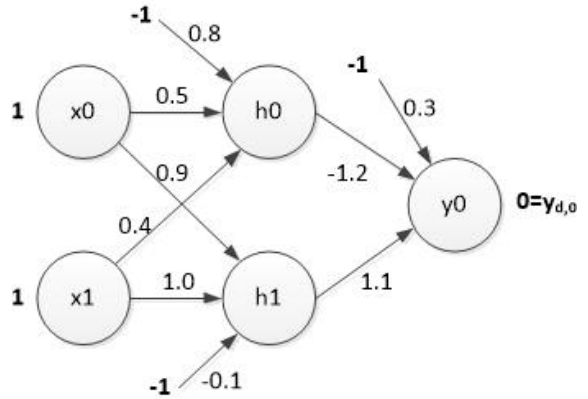
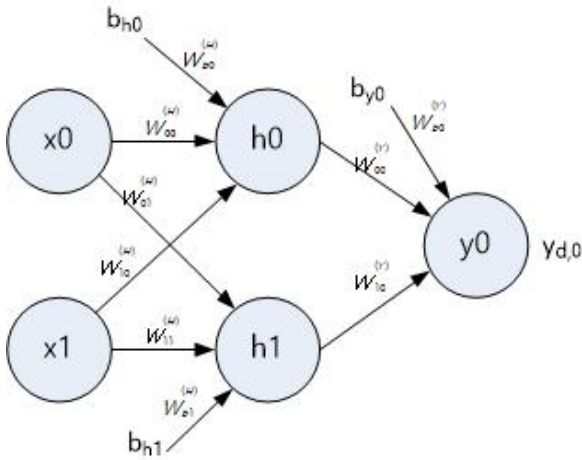
Feed Forward



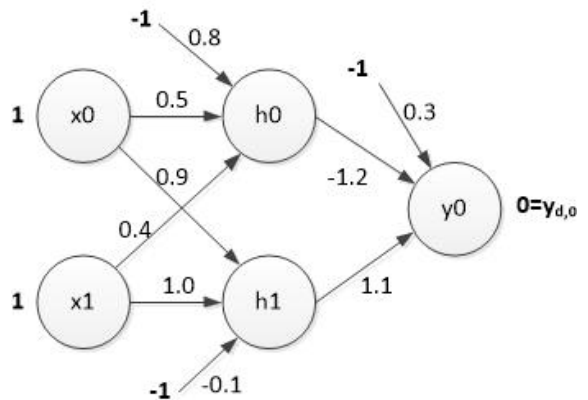
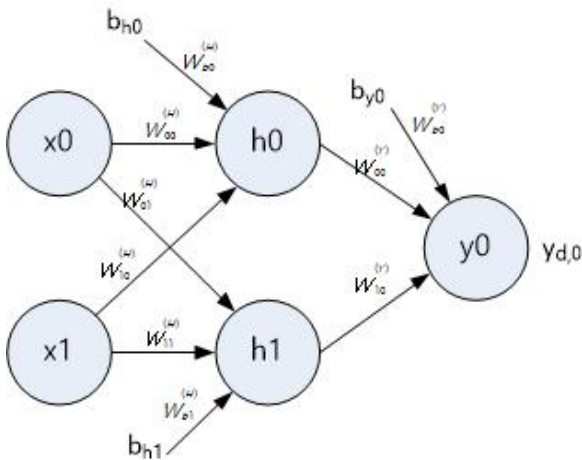
Steps



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



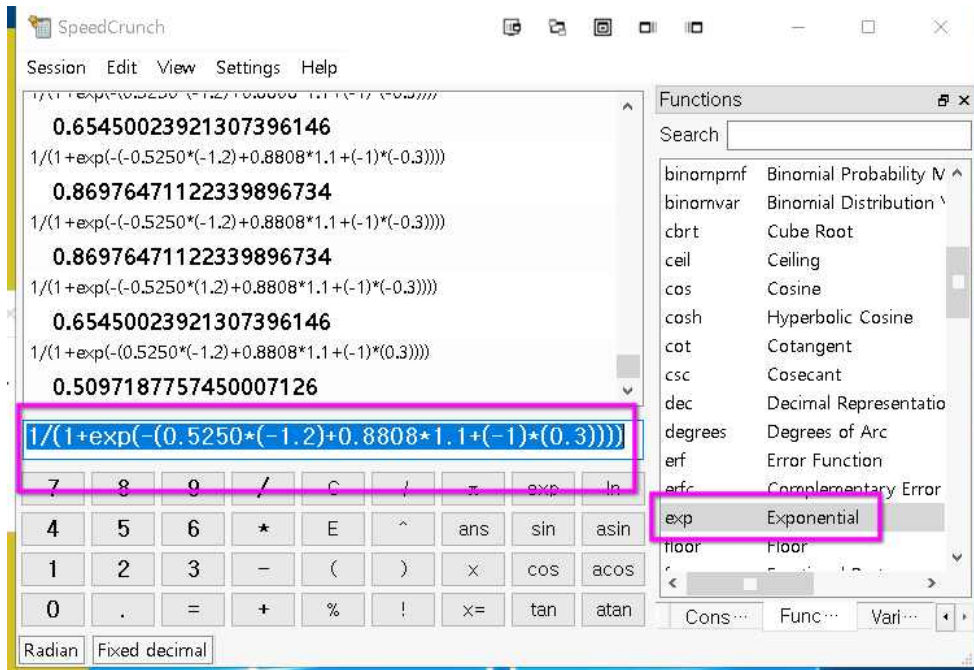
$$\begin{aligned}
h1 &= \text{sigmoid}(x_0 w_{01}^{(H)} + x_1 w_{11}^{(H)} + b_{h1} w_{b1}^{(H)}) \\
&= 1 / (1 + e^{-(x_0 w_{01}^{(H)} + x_1 w_{11}^{(H)} + b_{h1} w_{b1}^{(H)})}) \\
&= 1 / (1 + e^{-(1 \times 0.9 + 1 \times 1.0 + (-1) \times (-0.1))}) \\
&= 0.8808
\end{aligned}$$



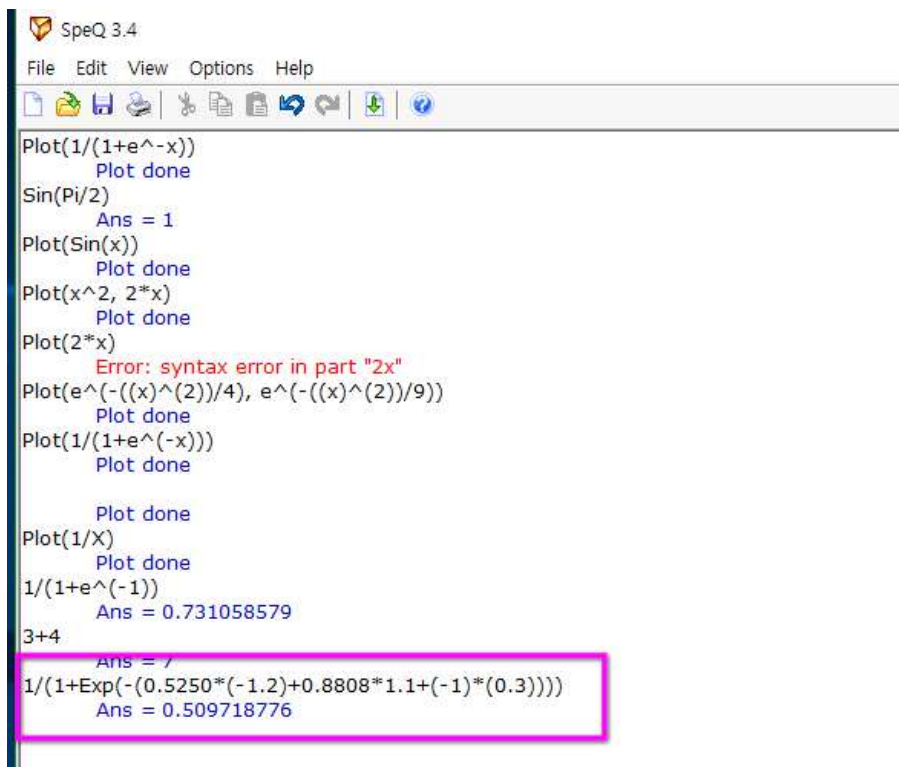
$$\begin{aligned}
y0 &= \text{sigmoid}(h_0 w_{00}^{(Y)} + h_1 w_{10}^{(Y)} + b_{y0} w_{b0}^{(Y)}) \\
&= 1 / (1 + e^{-(0.5250 \times (-1.2) + 0.8808 \times 1.1 + (-1) \times 0.3)}) \\
&= 0.5097
\end{aligned}$$

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

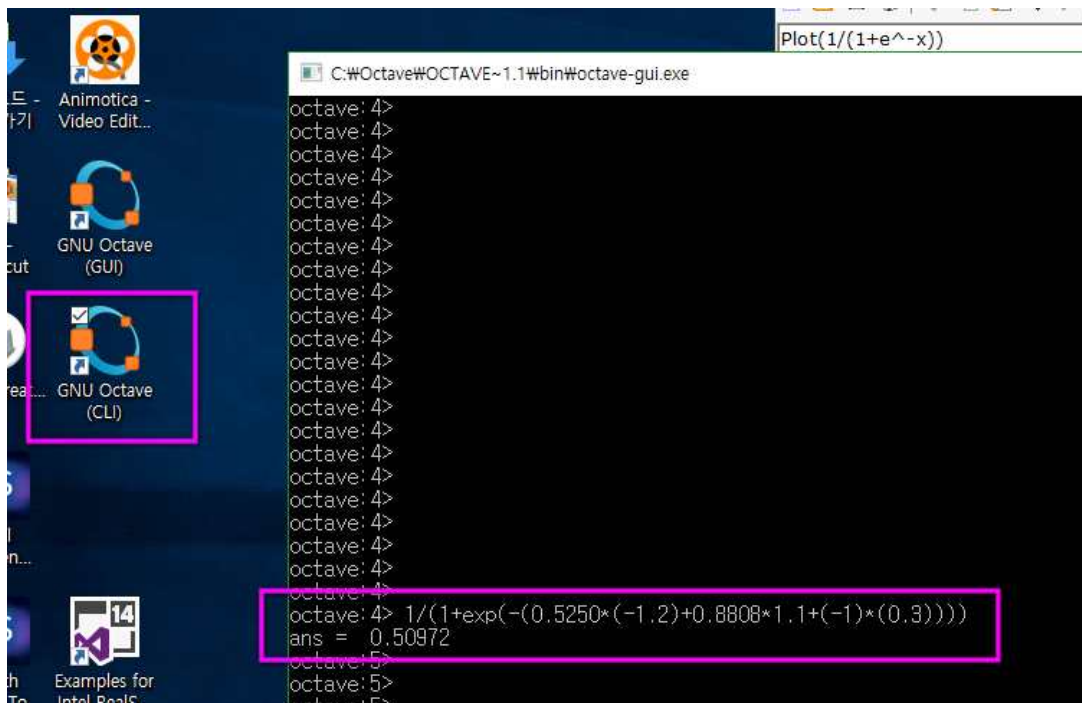
SpeedCrunch



SpeQ



Octave CLI



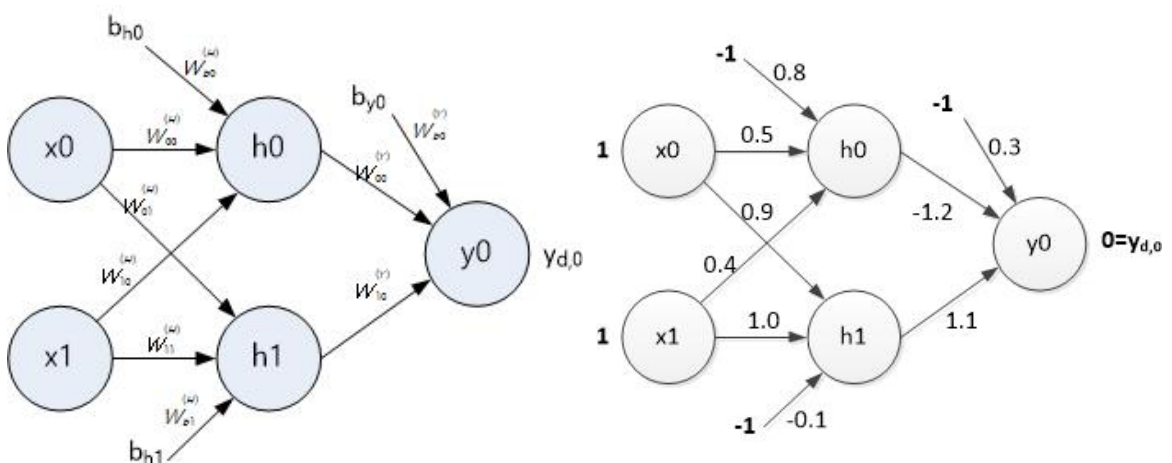
The screenshot shows a Windows desktop with several icons. Two icons for GNU Octave are highlighted with a pink box: 'GNU Octave (GUI)' and 'GNU Octave (CLI)'. A terminal window titled 'C:\Octave\OCTAVE~1.1\bin\octave-gui.exe' is open, displaying a series of 'octave:4>' prompts. A pink box highlights the following command and its output:

```
octave:4> 1/(1+exp(-(0.5250*(-1.2)+0.8808*1.1+(-1)*(0.3))))  
ans = 0.50972
```

Backward Propagation

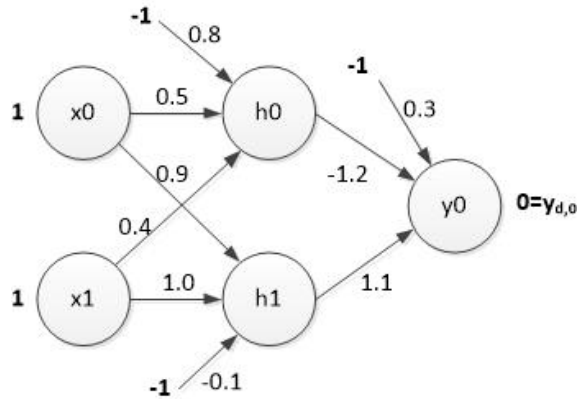
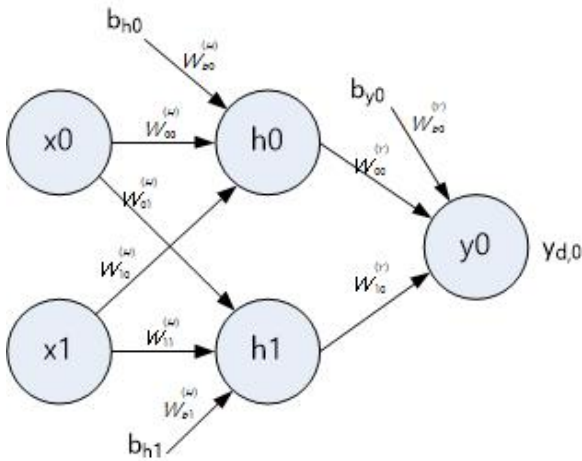
Concept

Output Layer



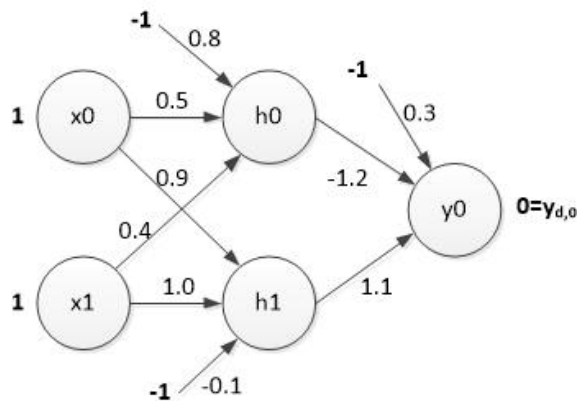
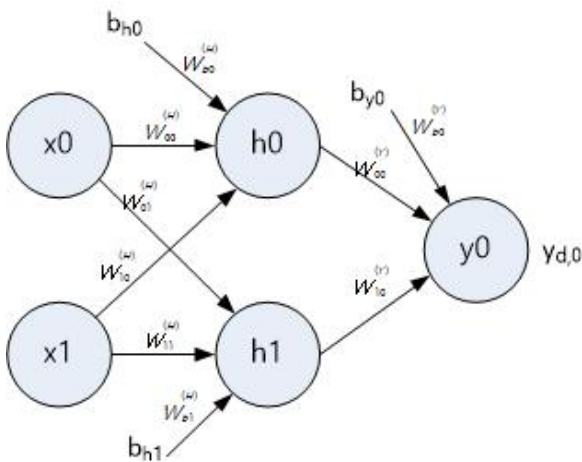
error gradient

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

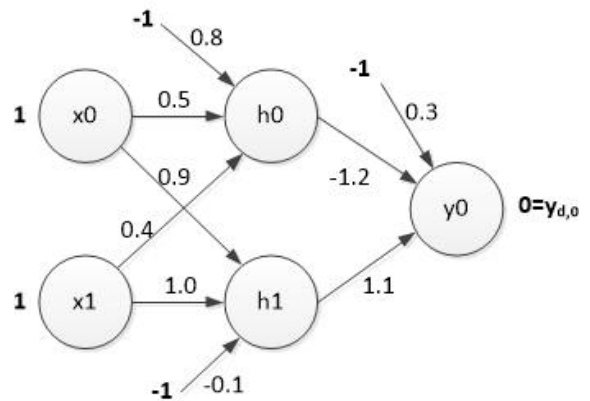
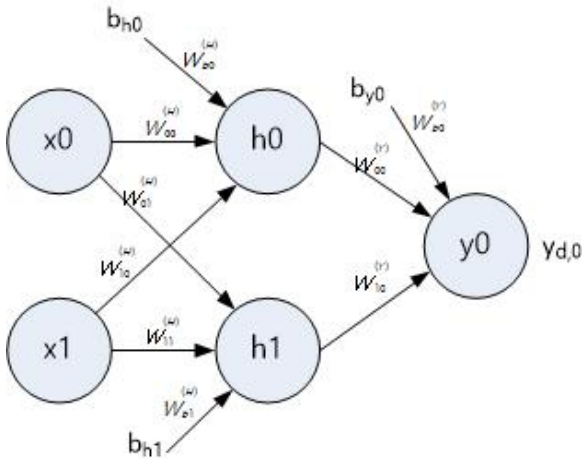


learning ratio $\alpha = 0.1$

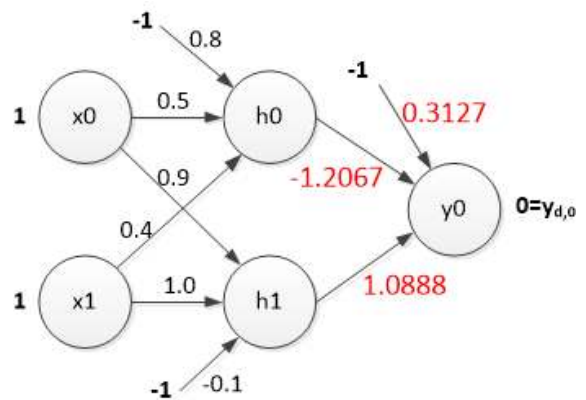
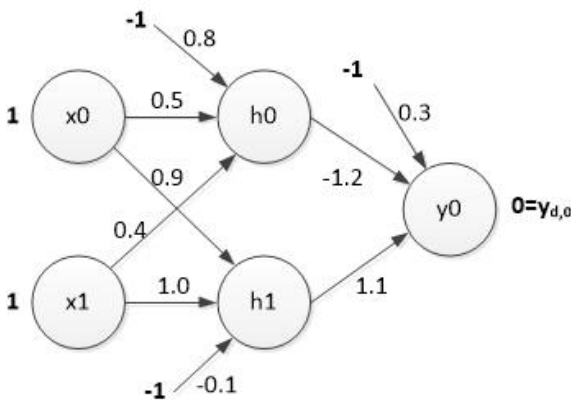
$$\begin{aligned}
 \Delta w_{00}^Y &= \alpha \times h_0 \times \delta_0 \\
 &= 0.1 \times 0.5250 \times (-0.1274) \\
 &= -0.0067
 \end{aligned}$$



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



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



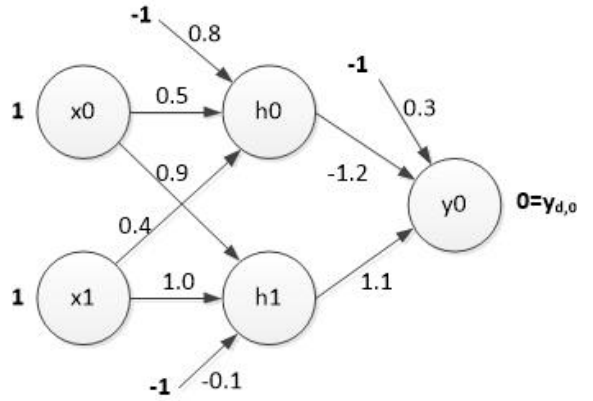
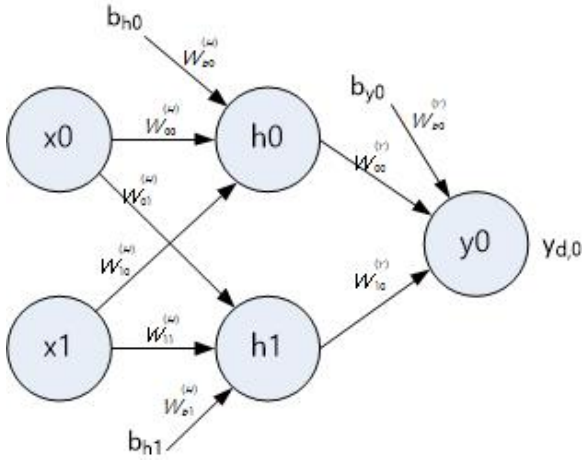
Update Output Weights

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

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

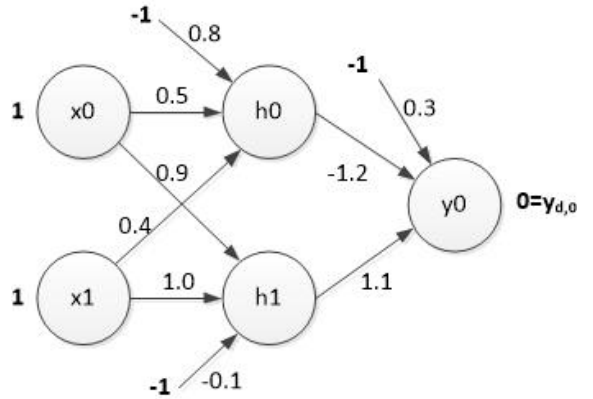
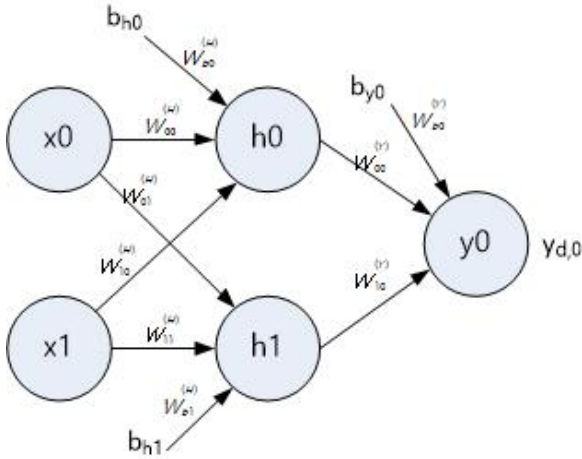
$$w_{b0}^Y = w_{b0}^Y + \Delta w_{b0}^Y = 0.3 + (0.0127) = 0.3127$$

Hidden Layer



$$\begin{aligned}\delta_0^H &= h_0(1 - h_0) \times \delta_0^Y \times w_{00}^Y \\ &= 0.5250 \times (1 - 0.5250) \times (-0.1274) \times (-1.2) \\ &= 0.0381\end{aligned}$$

$$\begin{aligned}\delta_1^H &= h_1(1 - h_1) \times \delta_0^Y \times w_{10}^Y \\ &= 0.8808 \times (1 - 0.8808) \times (-0.1274) \times (1.1) \\ &= -0.0147\end{aligned}$$



$$\Delta w_{00}^H = \alpha \times x_0 \times \delta_0^H = 0.1 \times 1 \times 0.0381 = 0.0038$$

$$\Delta w_{10}^H = \alpha \times x_1 \times \delta_0^H = 0.1 \times 1 \times 0.0381 = 0.0038$$

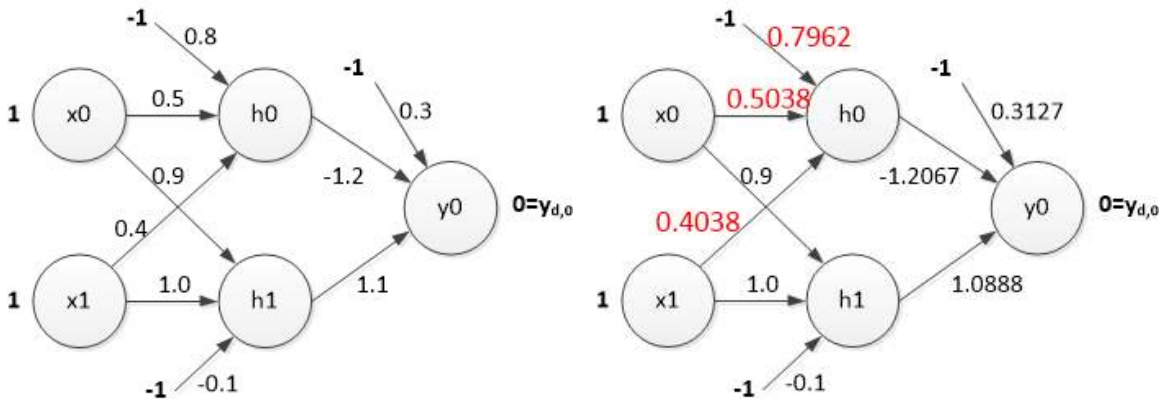
$$\Delta w_{b0}^H = \alpha \times b_{h0} \times \delta_0^H = 0.1 \times (-1) \times 0.0381 = -0.0038$$

$$\Delta w_{01}^H = \alpha \times x_0 \times \delta_1^H = 0.1 \times 1 \times (-0.0147) = -0.0015$$

$$\Delta w_{11}^H = \alpha \times x_1 \times \delta_1^H = 0.1 \times 1 \times (-0.0147) = -0.0015$$

$$\Delta w_{b1}^H = \alpha \times b_{h1} \times \delta_1^H = 0.1 \times (-1) \times (-0.0147) = 0.0015$$

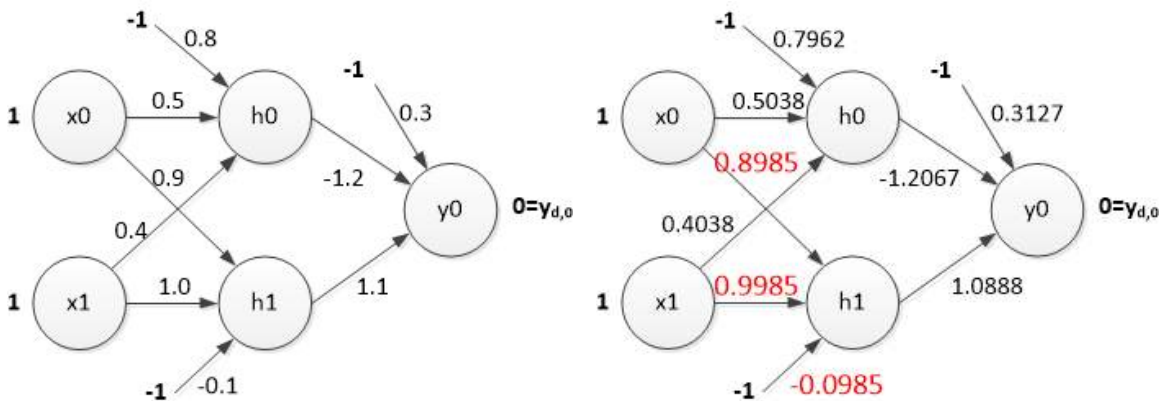
Update Hidden Weights



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

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

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



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

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

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

Sum of Square Errors

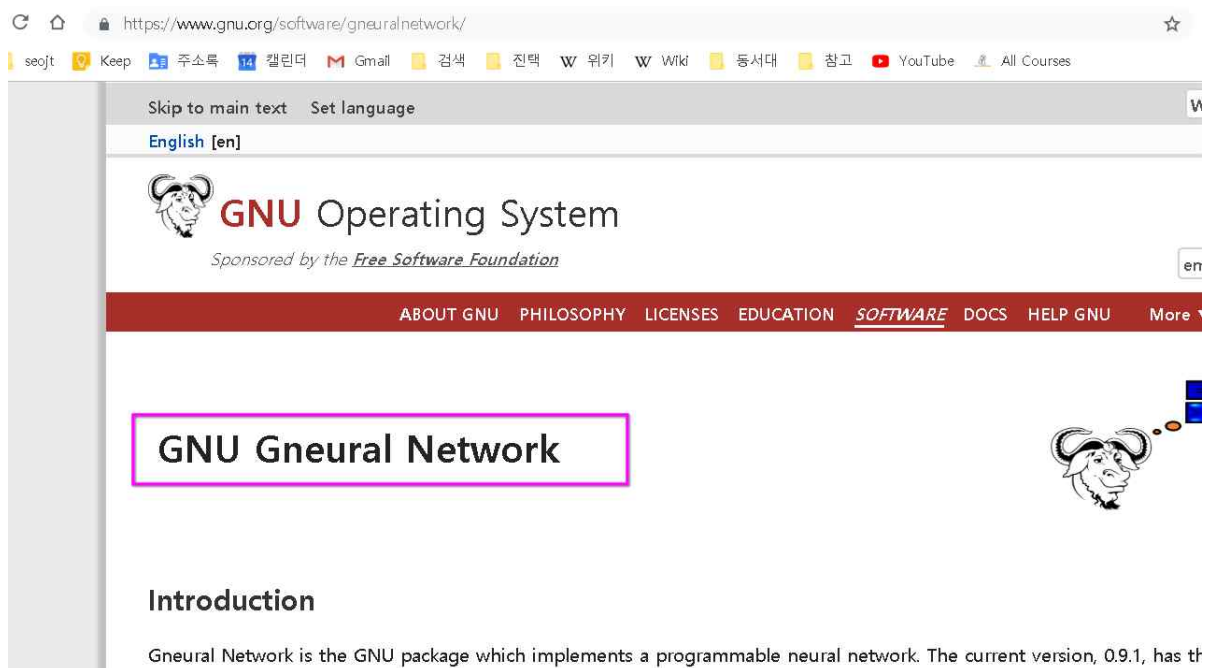
$$SSE = \sum_{i=1}^n (x_i - \bar{x})^2$$

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

$$\begin{aligned} SSE &= \sum_{i=0}^n (y_i - \bar{y})^2 + \sum_{i=0}^n (y_{d,i} - \bar{y})^2 \\ &= D_i/2 = \left\{ \sum_{i=0}^n (y_i - y_{d,i})^2 \right\} / 2 \end{aligned}$$


Naive Implementation

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



Skip to main text Set language

English [en]

 **GNU** Operating System

Sponsored by the *Free Software Foundation*

en

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)

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.

- 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
- Franklin's bell
- Can Stirling
- Big Stirling
- 555 timer music
- TEA laser
- LED diode drive

2-layer neural network, fully connected

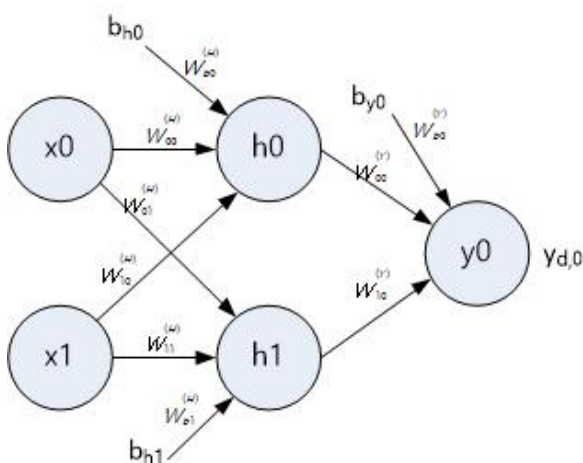
input units → hidden units → output units

each line is assigned a weight

C++ conversion

(github 링크를 명시할 것)

Implementation Issues



$$h_0 = \text{sigmoid}(x_0 w_{00}^{(H)} + x_1 w_{10}^{(H)} + b_{h0} w_{b0}^{(H)})$$

$$h_1 = \text{sigmoid}(x_0 w_{01}^{(H)} + x_1 w_{11}^{(H)} + b_{h1} w_{b1}^{(H)})$$

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

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

Covector

a function for a column vector.

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

$$\begin{bmatrix} 2 & 1 \end{bmatrix} \left(\begin{bmatrix} x \\ y \end{bmatrix} \right) = 2x + 1y$$

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

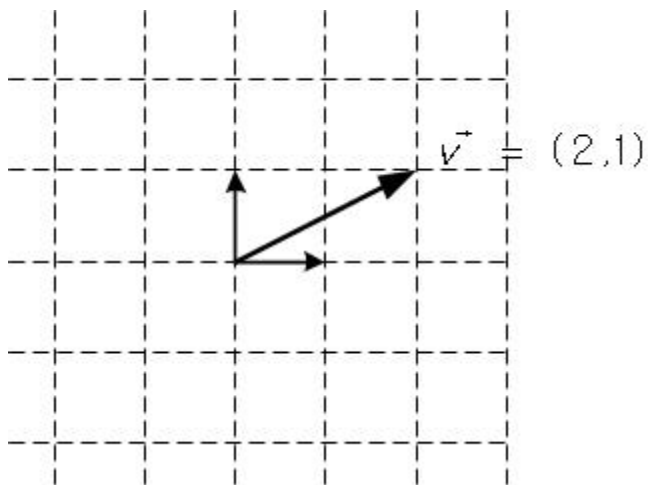
$$2x + 1y = -1$$

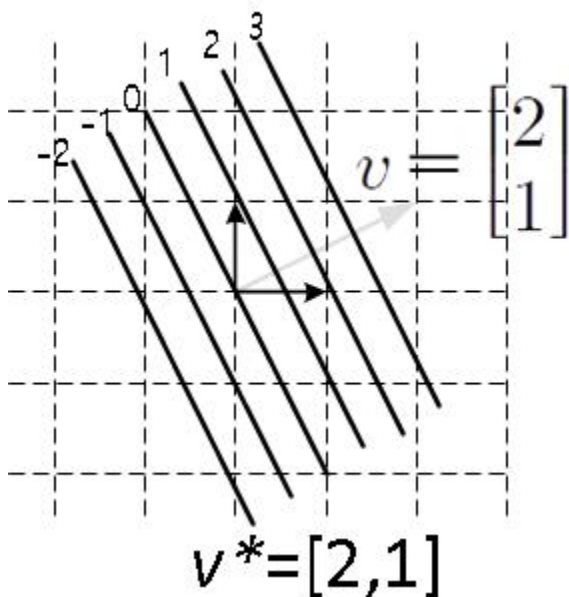
$$2x + 1y = 0$$

$$2x + 1y = 1$$

$$2x + 1y = 2$$

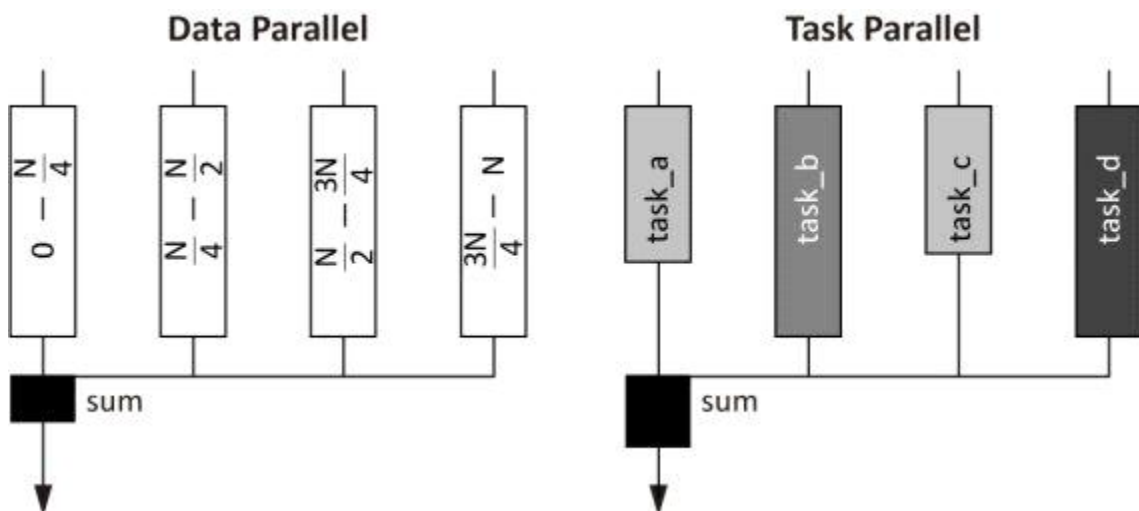
$$v = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$





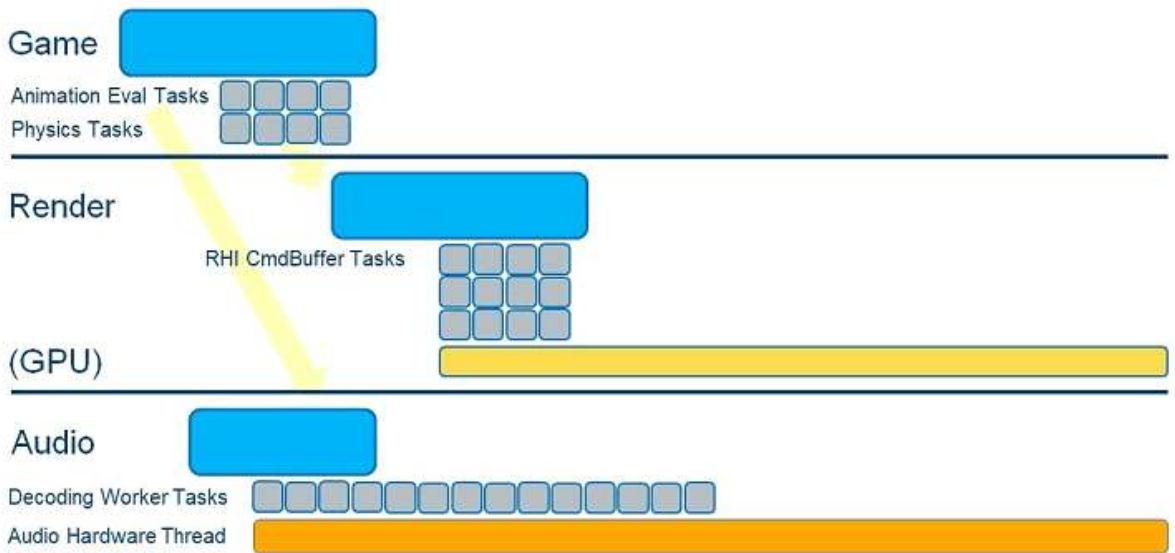
Machine Learning Library

How to implement with multi-thread?



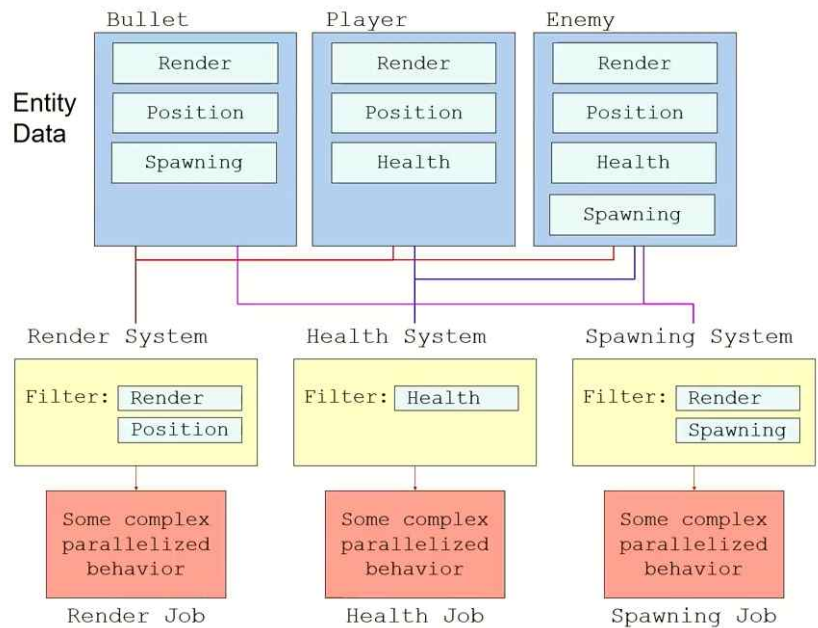
Unreal Engine 4 threading model

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

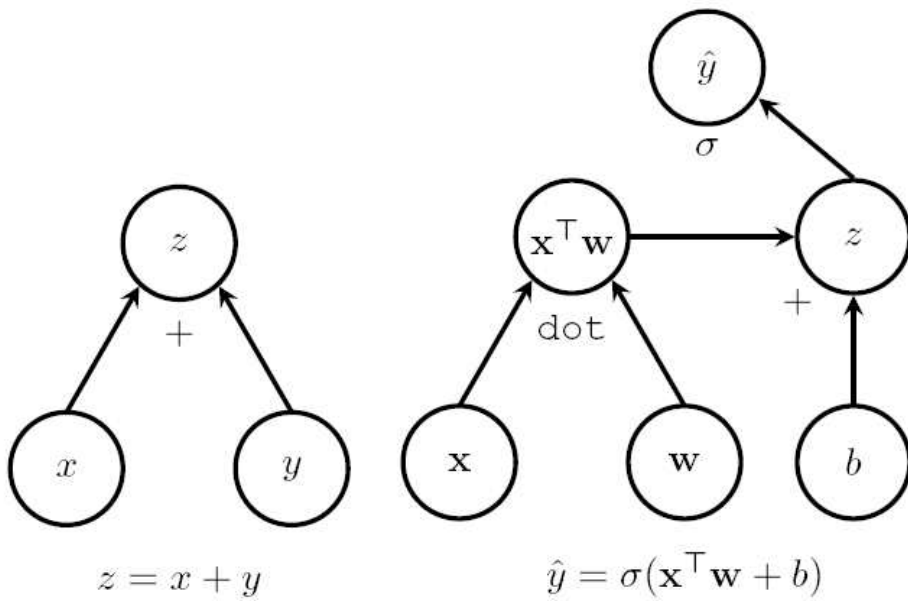


Unity ECS(Entity Component System)

Entity Component System



Computational Graph



Ex)

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

@