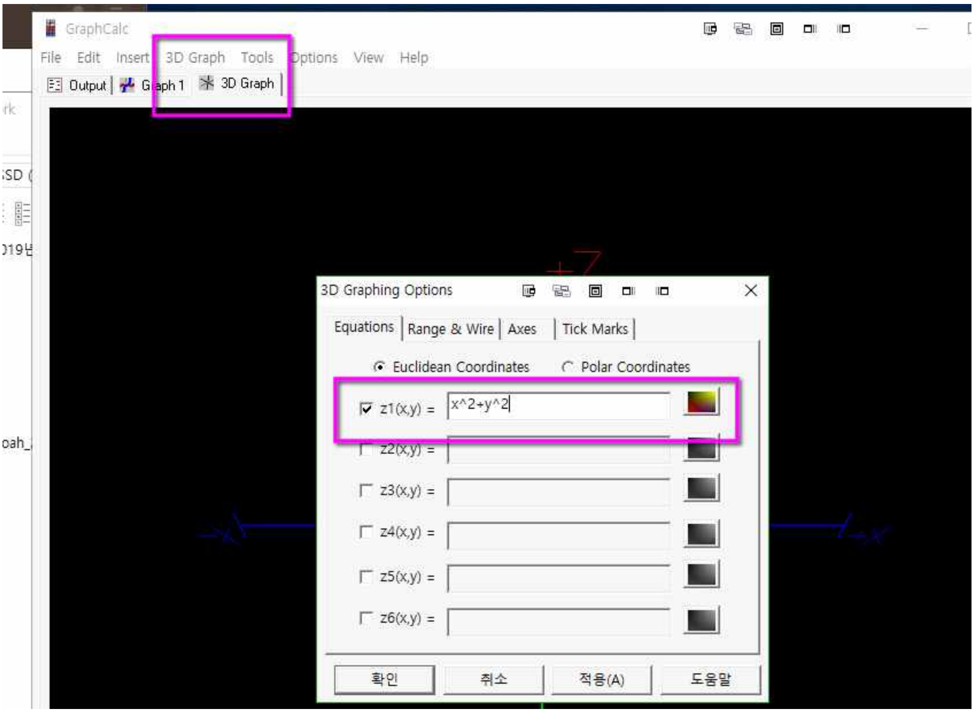
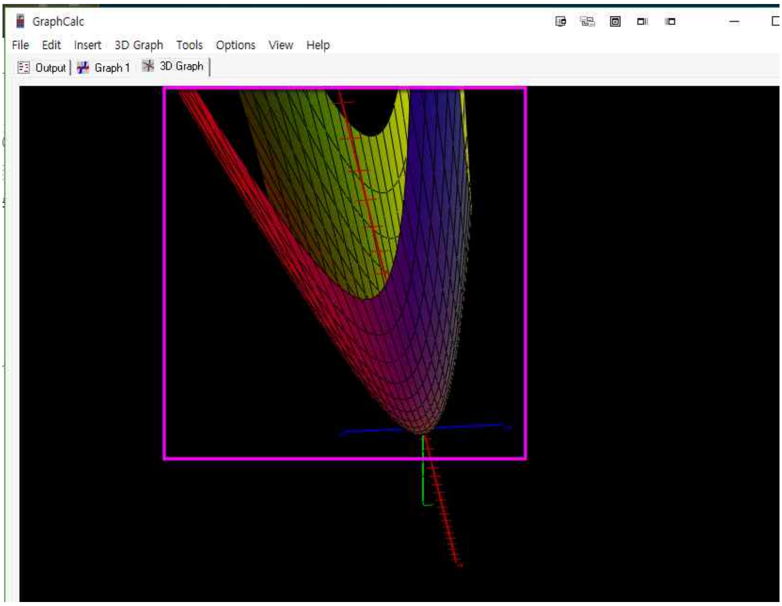
인공 신경망: Back-propagation

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

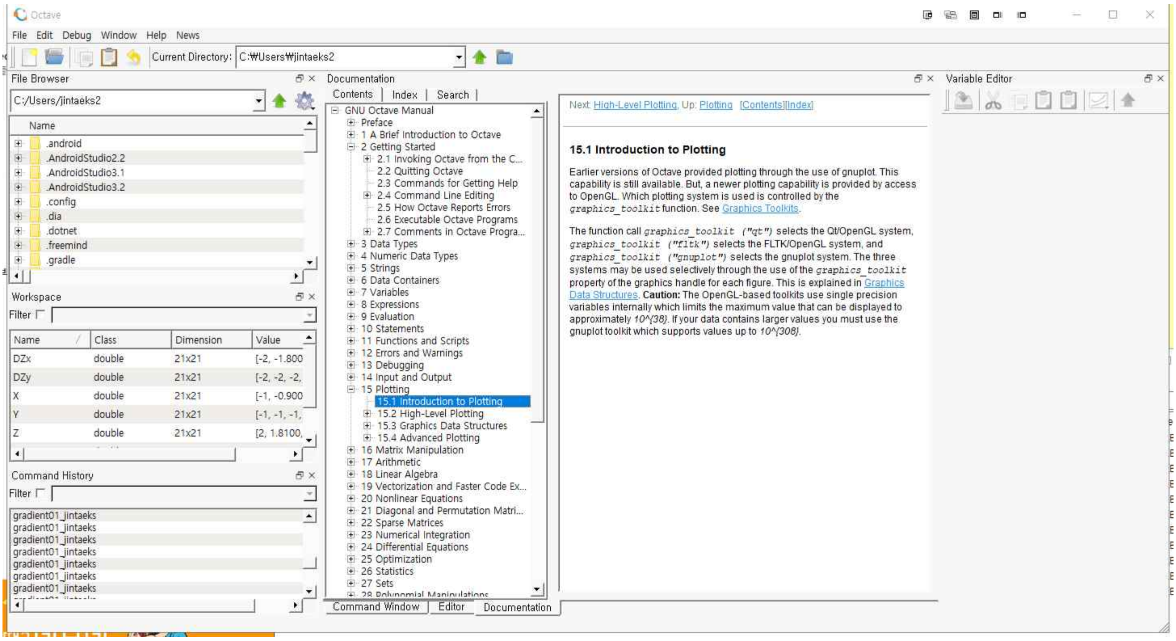
Plotting 3D Graph

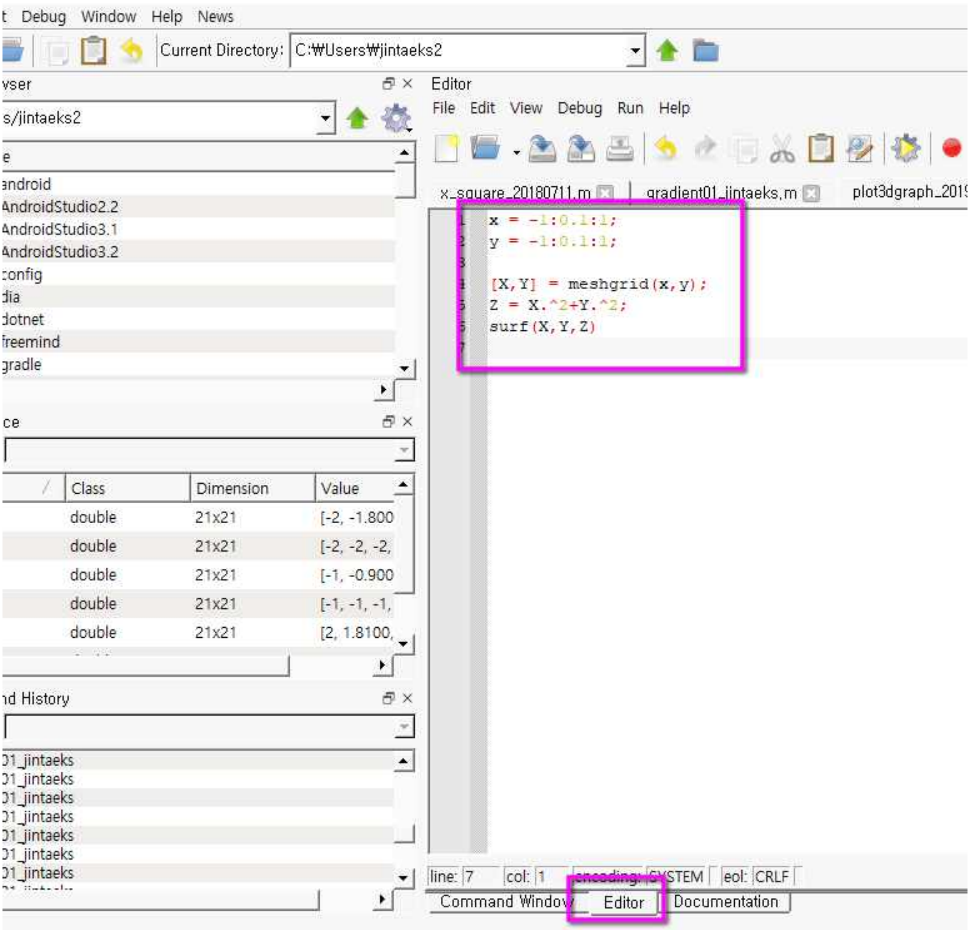
GraphCalc

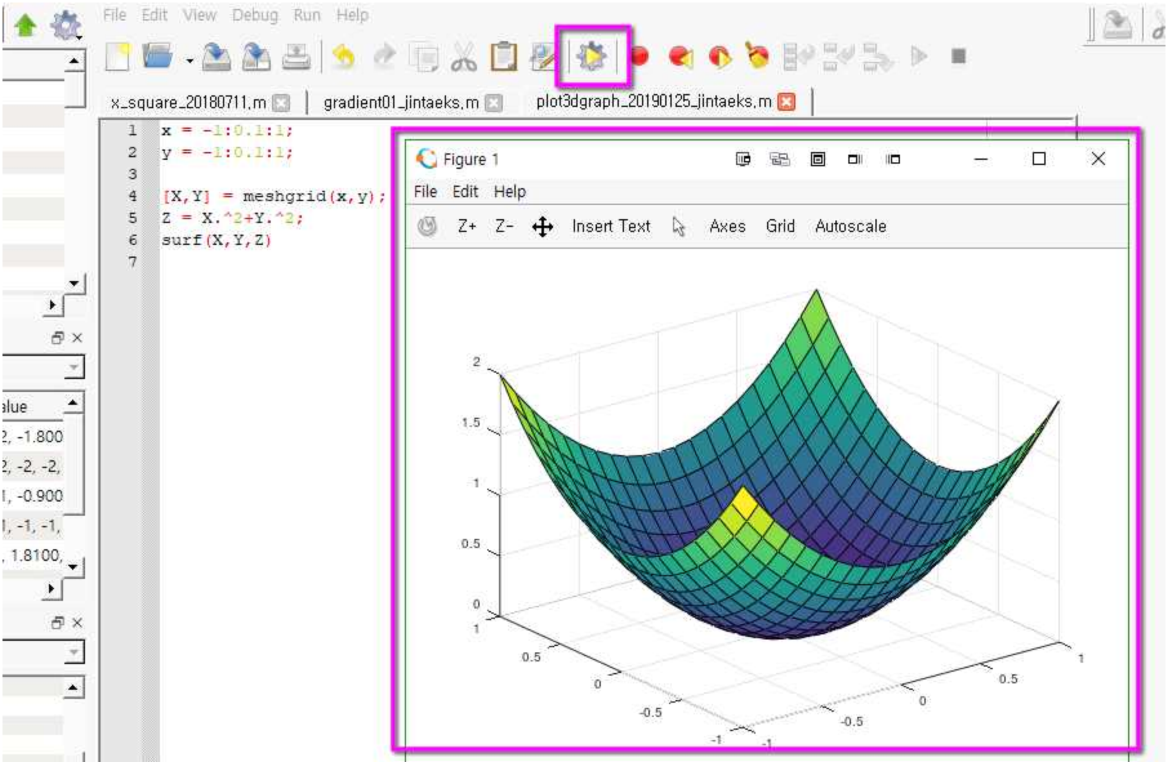




Octave







Differentiation

Product Rule

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

$$\left[(2x+3)^4(x+1)^2\right]'$$
$$f(x) = (2x+3)^4$$
$$g(x) = (x+1)^2$$

$$f'(x) = 4(2x+3)^3 \times 2 = 8(2x+3)^3$$
$$g'(x) = 2(x+1)^1 \times 1 = 2x+2$$

$$[f \times g]' = f'g + f \times g'$$
$$= 8(2x+3)^3(x+1)^2 + (2x+3)^4(2x+2)$$

Quotient Rule

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

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

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

$$\left(e^x\right)' = e^x$$

$$\left(e^{-x}\right)'$$
$$= e^{-x} \times (-1)$$
$$= -e^{-x}$$

Differentiation of Sigmoid Function

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

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

Gradient

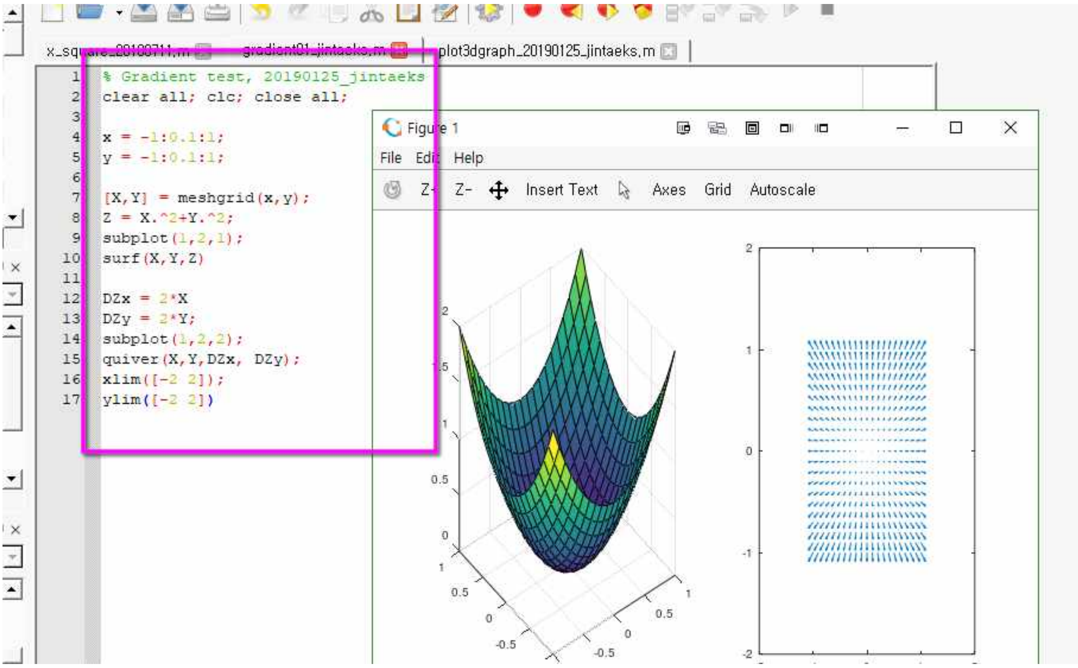
$f(x,y) = x^2 \sin(y)$

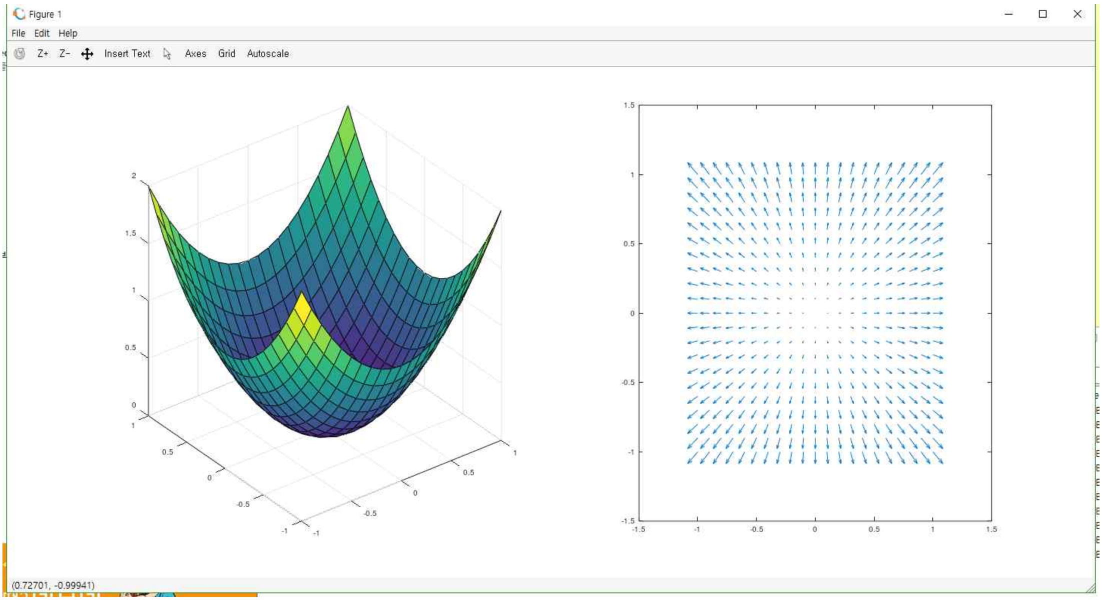
$$\frac{df}{dx} = 2x\sin(y), \frac{df}{dy} = x^2\cos(y)$$

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

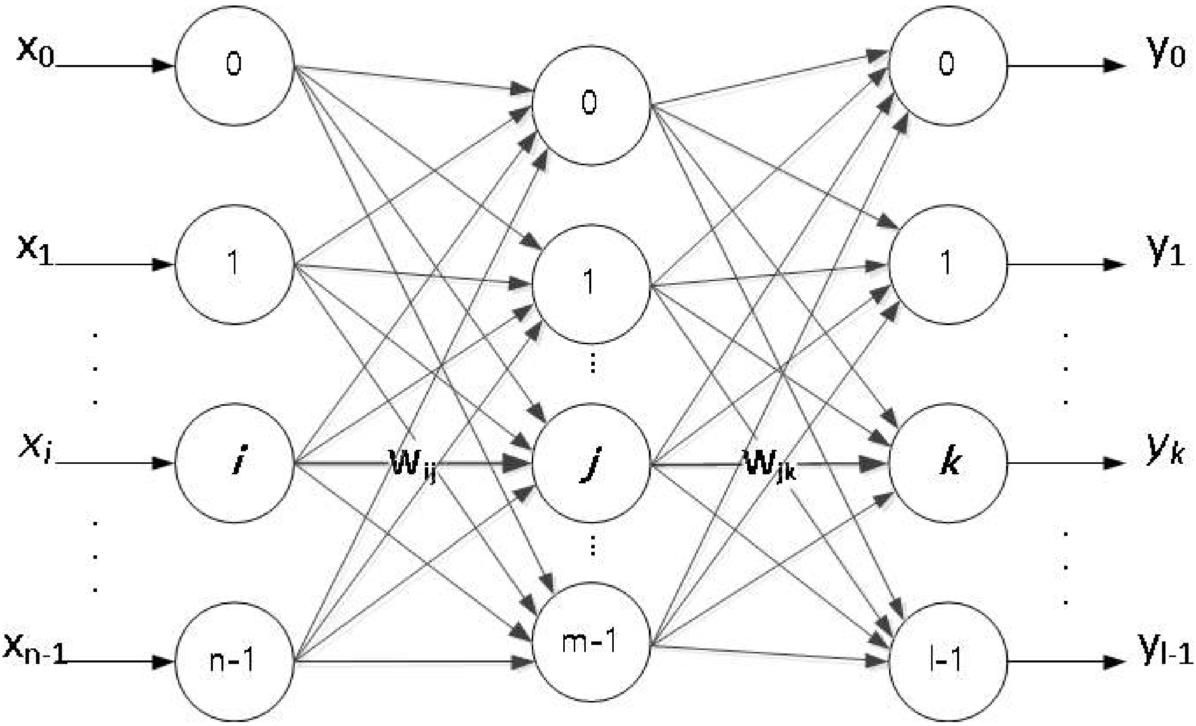
example: $f(x,y) = x^2 + y^2$

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

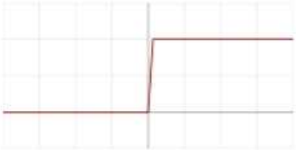
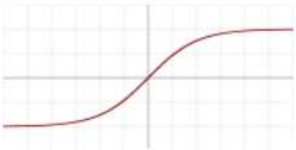




Neural Network



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)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$ [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

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

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

Back Propagation Algorithm

w_{ij}

w_{jk}

Update Weight w_{jk}

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

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

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

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

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

Update Weight w_{ij}

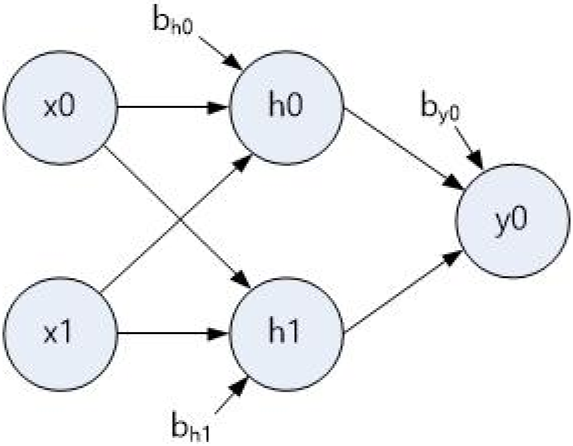
$\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_i(p)$

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

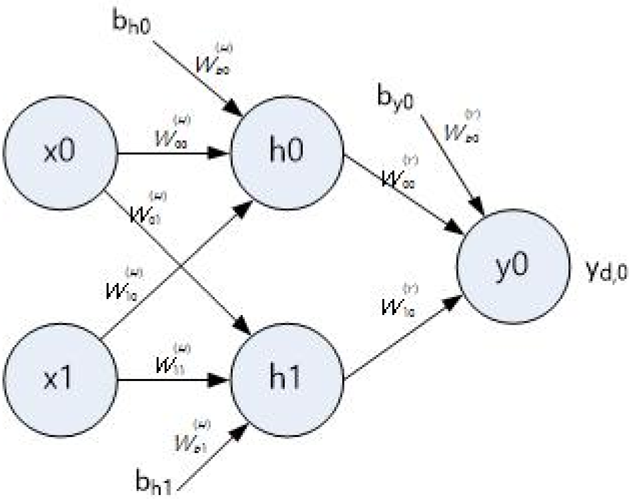
$$y_j(p) = \frac{1}{1+e^{X_j(p)}}$$

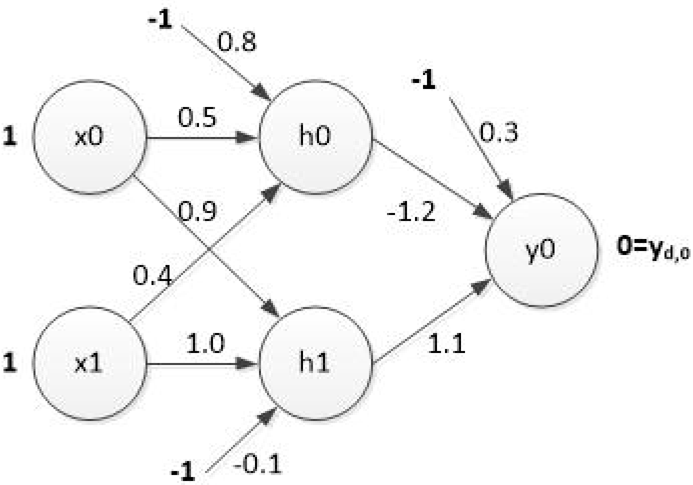
$$X_j(p) = \sum_{i=1}^{n} x_i(p) \times w_{ij}(p) + bias_j$$

Example

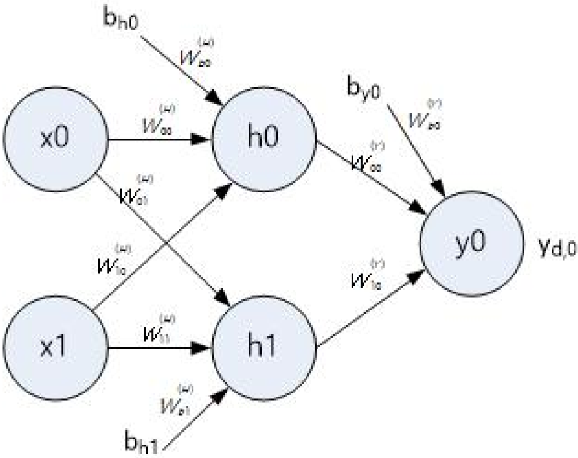


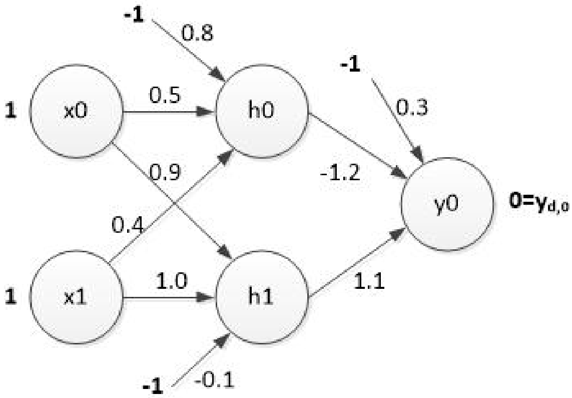
Forward Propagation



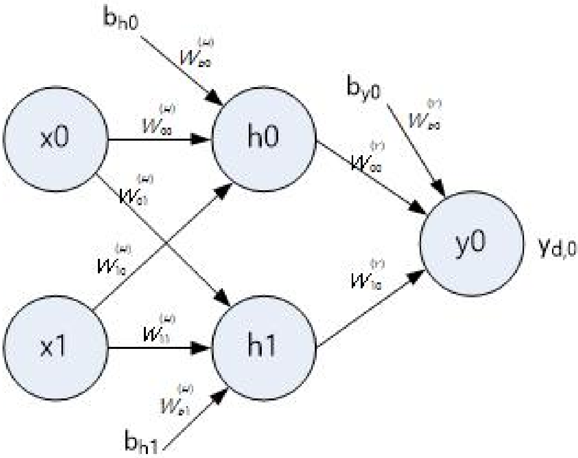


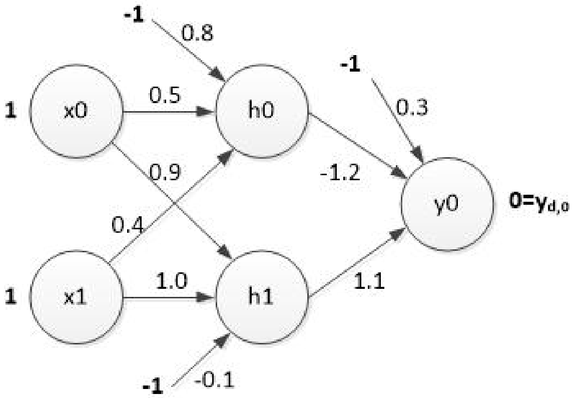
Steps



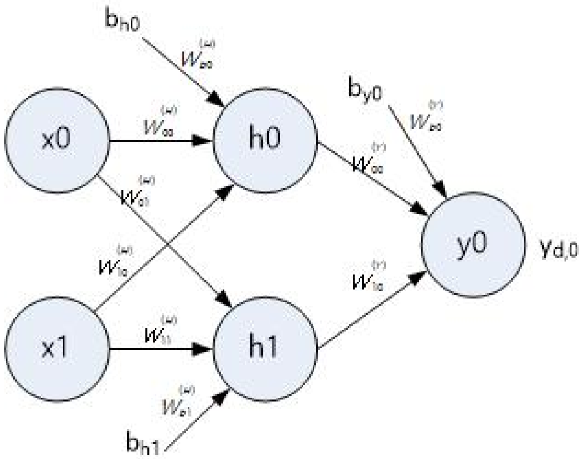


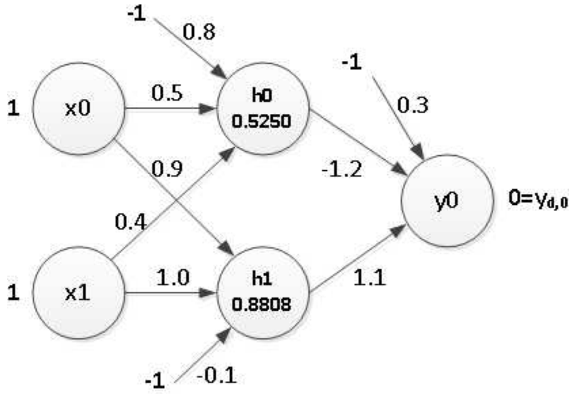
$$
\begin{aligned}
h0 &= sigmoid(x_0 w_{00}^{(H)} + x_1 w_{10}^{(H)} + b_{h0} w_{b0}^{(H)}) \\
&= 1/(1+e^{-(x_0 w_{00}^{(H)} + x_1 w_{10}^{(H)} + b_{h0} w_{b0}^{(H)})}) \\
&= 1/(1+e^{-(1\times 0.5 + 1\times 0.4 + (-1)\times 0.8)}) \\
&= 0.5250
\end{aligned}
$$





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





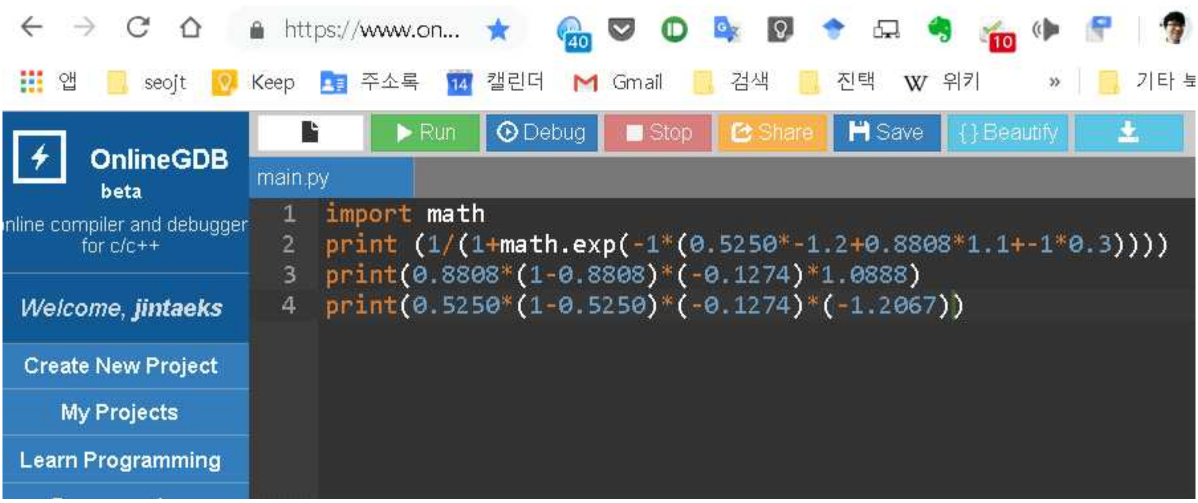
$$
\begin{aligned}
y0 &= sigmoid(h_0 w_{00}^{(Y)} + h_1 w_{10}^{(Y)} + b_{y0} w_{b0}^{(Y)}) \\
&= 1/(1+e^{-(0.5250\times(-1.2)+0.8808\times 1.1+(-1)\times 0.3)}) \\
&= 0.5097
\end{aligned}
$$

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

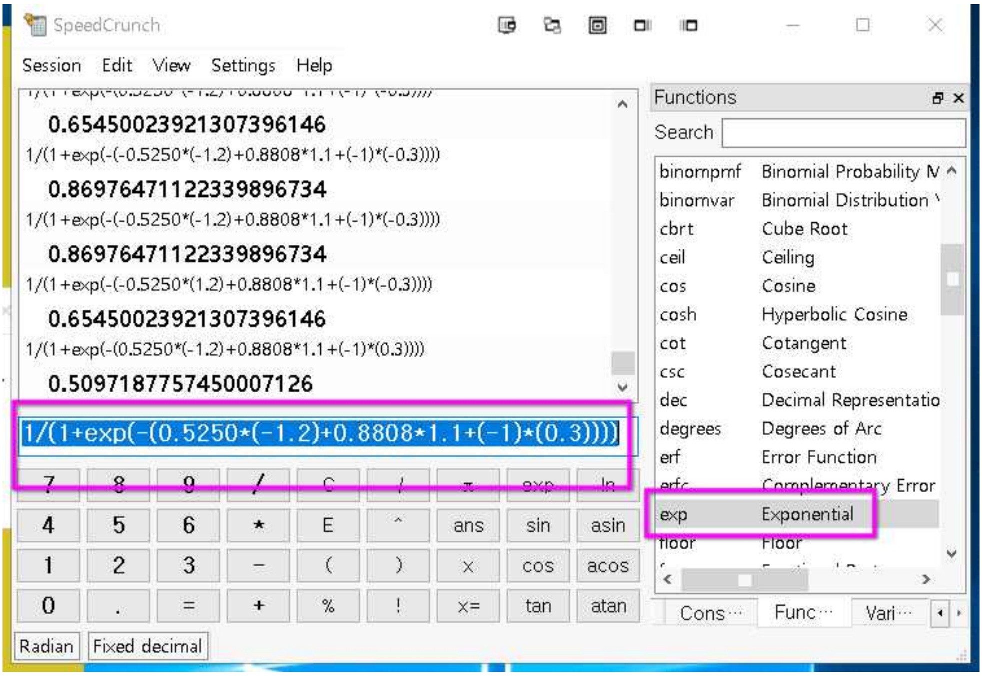
Calculating Sigmoid

1. Python

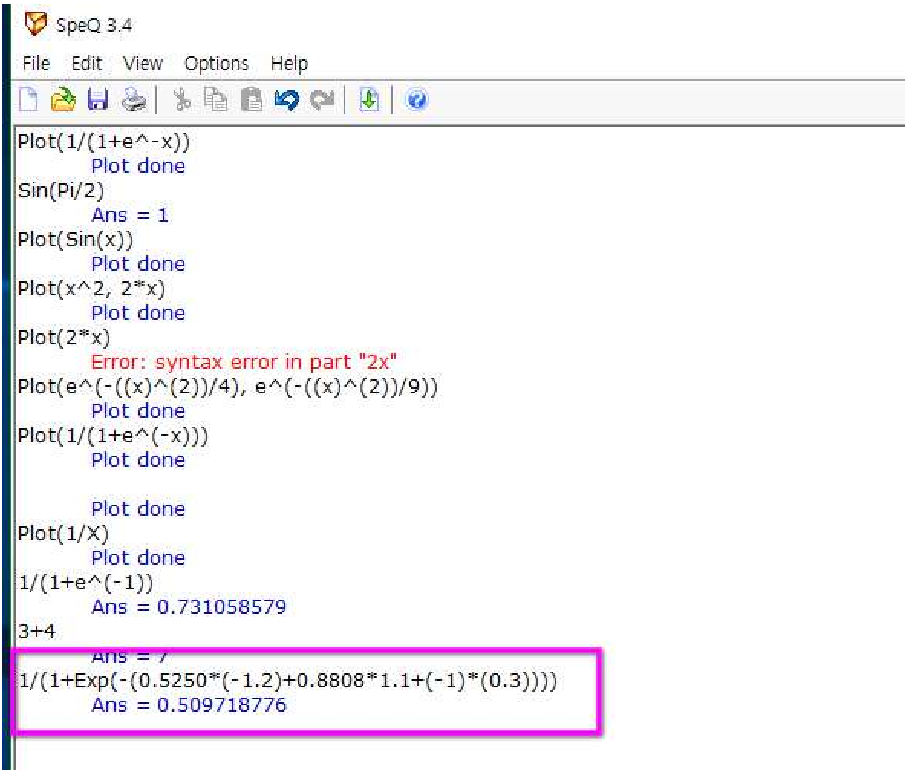
```
import math
print (1/(1+math.exp(-1*(0.5250*-1.2+0.8808*1.1+-1*0.3))))
```



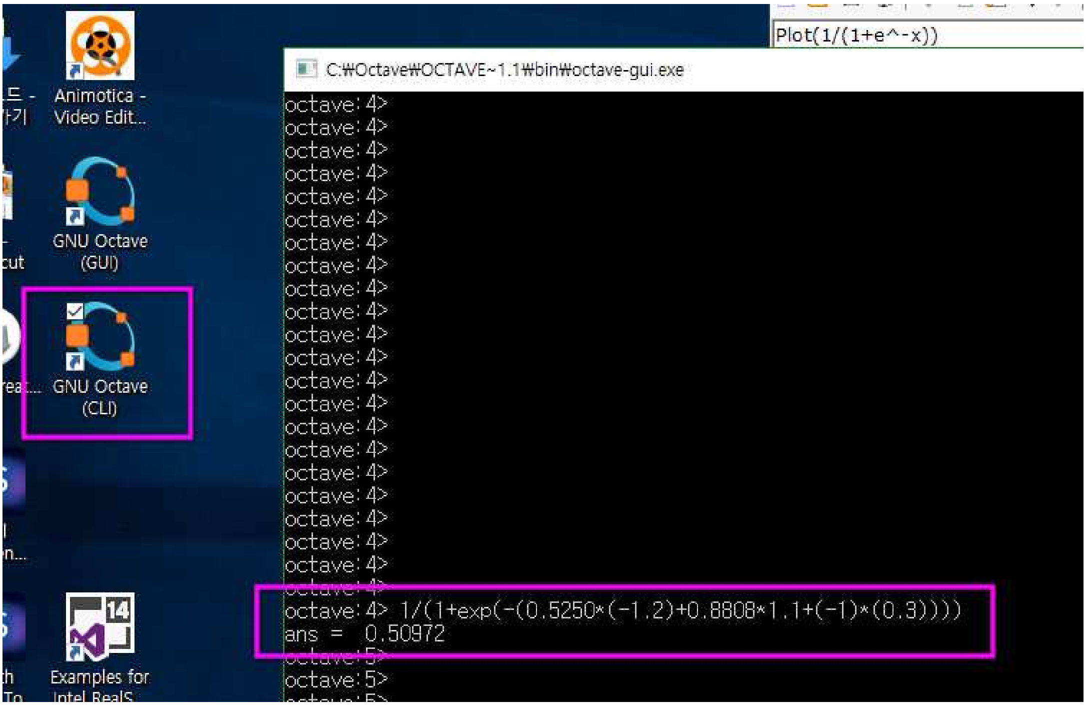
2. SpeedCrunch



3. SpeQ

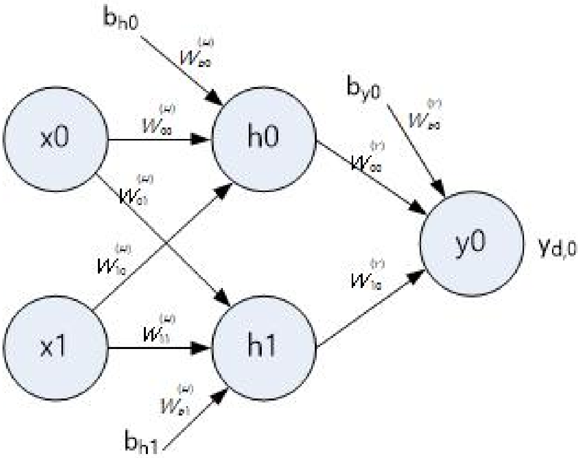


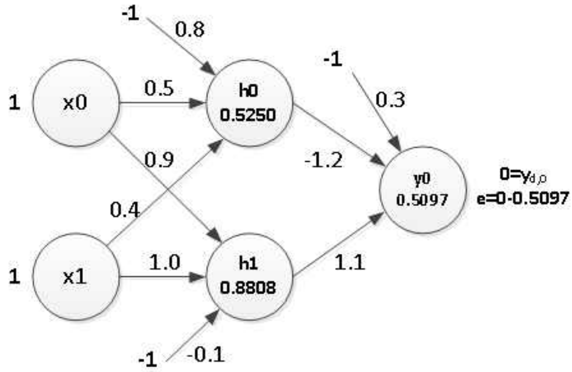
4. Octave CLI


Plot(1/(1+e^-x))
C:₩Octave₩OCTAVE~1.1₩bin₩octave-gui.exe
Animotica - Video Edit...
GNU Octave (GUI)
GNU Octave (CLI)
Examples for Intel RealS
octave:4> 1/(1+exp(-(0.5250*(-1.2)+0.8808*1.1+(-1)*(0.3))))
ans = 0.50972
octave:5>

Backward Propagation

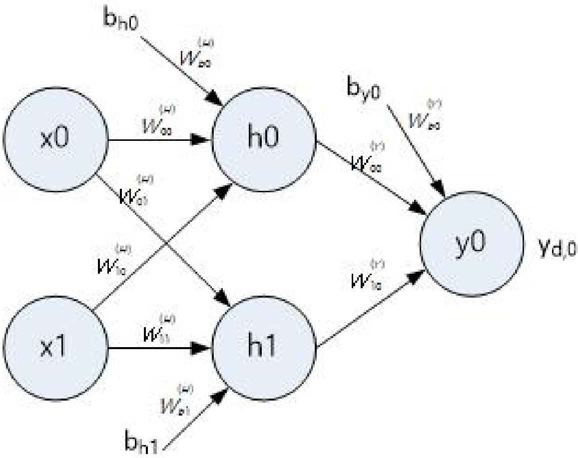
Output Layer

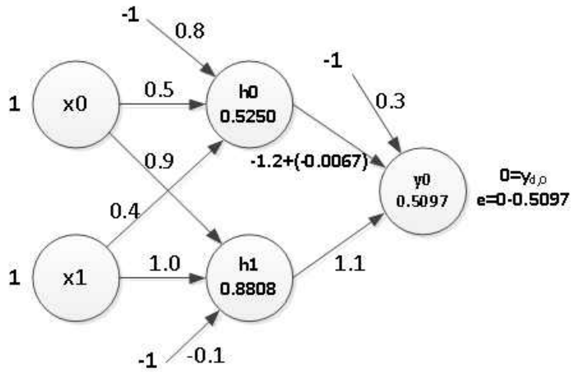




error gradient

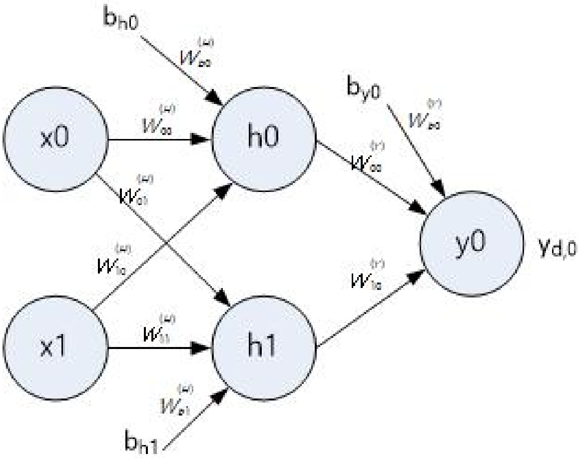
$$
\begin{aligned}
\delta_0^Y &= y_0(1-y_0)e \\
&= 0.5097 \times (1-0.5097) \times (-0.5097) \\
&= -0.1274
\end{aligned}
$$

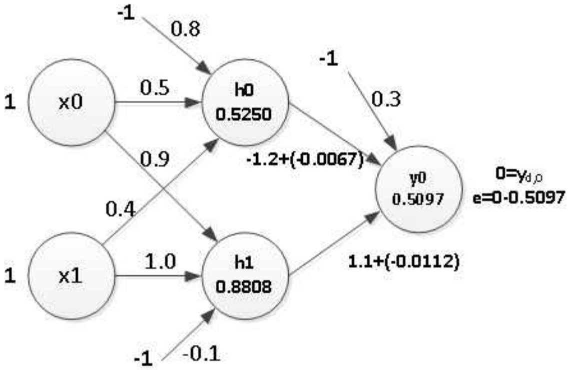




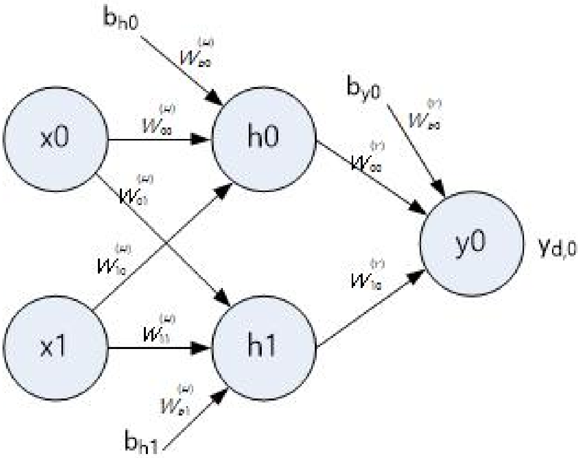
learning ratio $\alpha = 0.1$

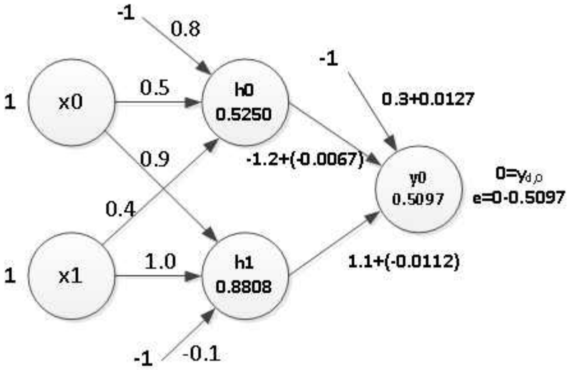
$$
\begin{aligned}
\Delta w_{00}^Y &= \alpha \times h_0 \times \delta_0 \\
&= 0.1 \times 0.5250 \times (-0.1274) \\
&= -0.0067
\end{aligned}
$$



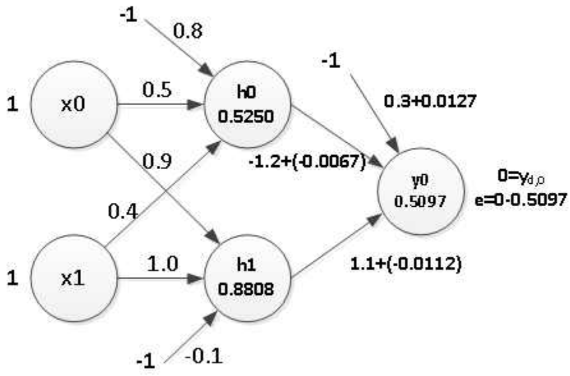


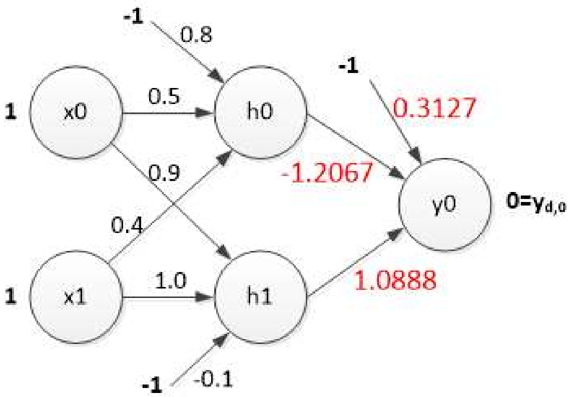
$$\Delta w_{10}^{Y} = \alpha \times h_1 \times \delta_0$$
$$= 0.1 \times 0.8808 \times (-0.1274)$$
$$= -0.0112$$





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





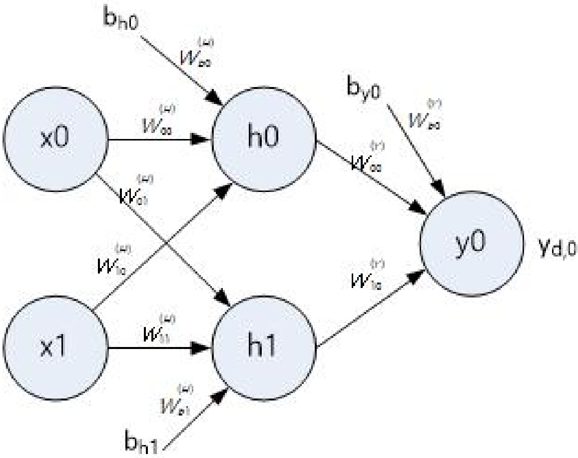
Update Output Weights

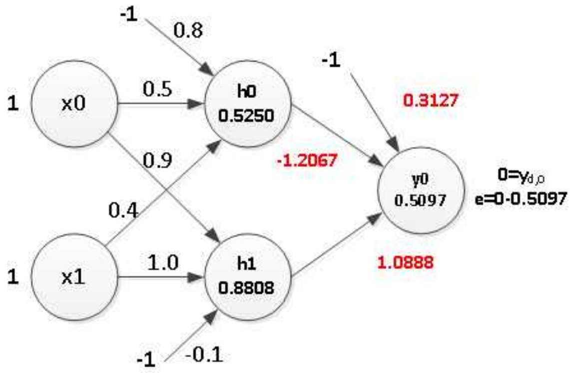
$$w_{00}^{Y} = w_{00}^{Y} + \Delta w_{00}^{Y} = -1.2 + (-0.0067) = -1.2067$$

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

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

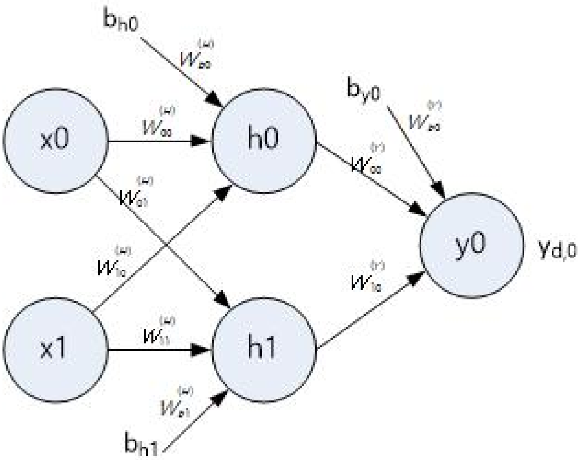
Hidden Layer

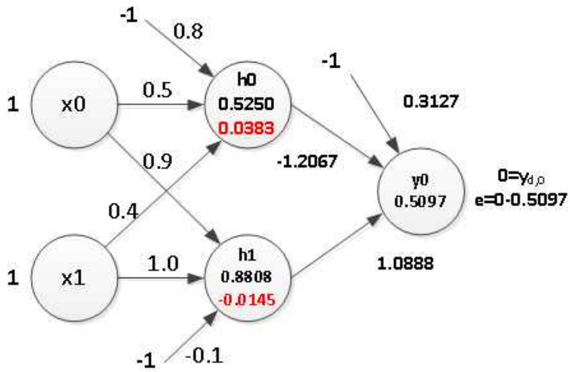




$$
\begin{aligned}
\delta_0^H &= h_0(1-h_0)\times\delta_0^Y\times w_{00}^Y\\
&= 0.5250\times(1-0.5250)\times(-0.1274)\times(-1.2067)\\
&= 0.0383
\end{aligned}
$$

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

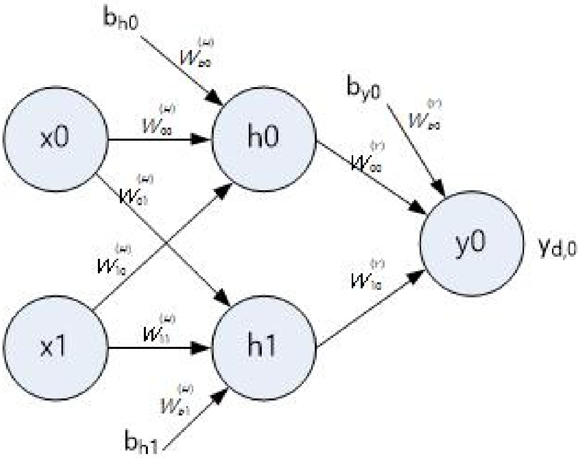


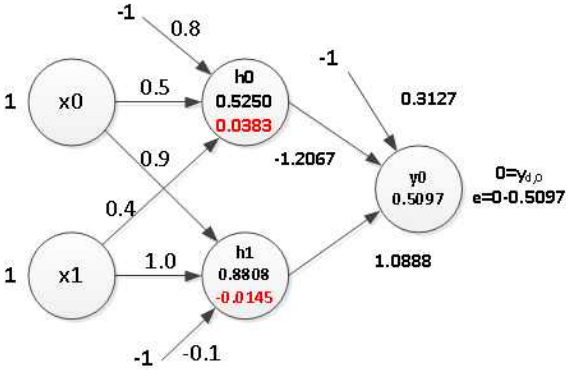


$$\Delta w_{00}^H = \alpha\times x_0\times\delta_0^H = 0.1\times 1\times 0.0383 = 0.0038$$

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

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



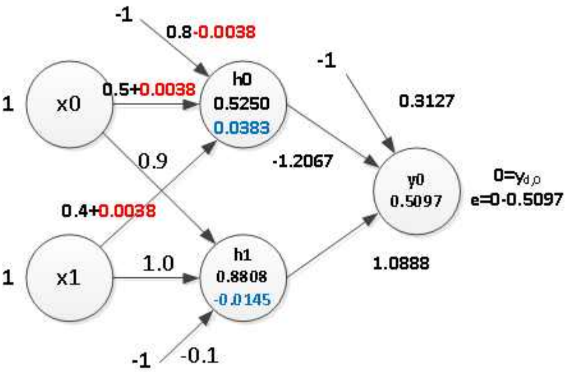


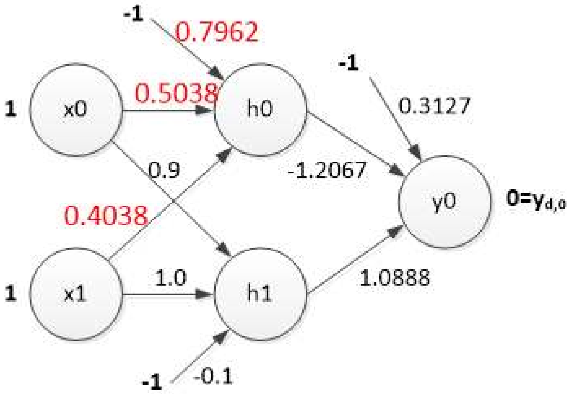
$$\Delta w_{01}^{H} = \alpha \times x_0 \times \delta_1^H = 0.1 \times 1 \times (-0.0145) = -0.0015$$

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

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

Update Hidden Weights

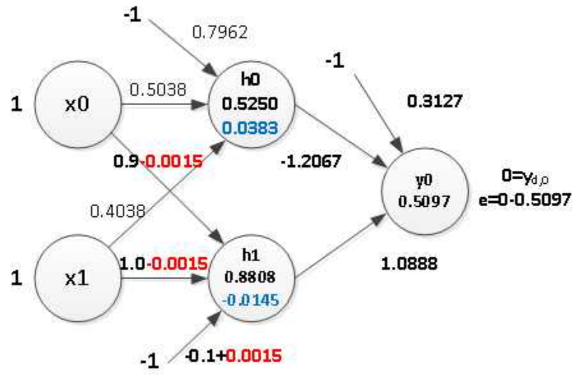


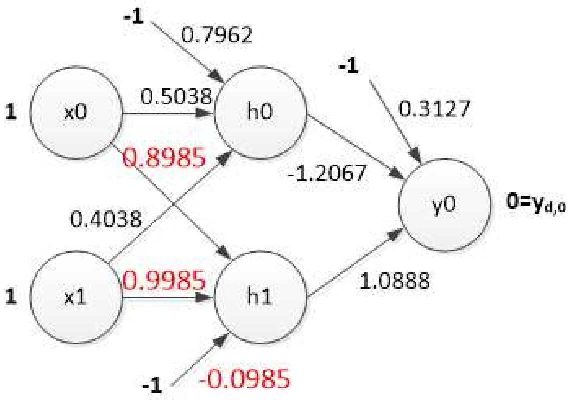


$$w_{00}^{H} = w_{00}^{H} + \Delta w_{00}^{H} = 0.5 + (0.0038) = 0.5038$$

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

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





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

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

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

Sum of Square Errors

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

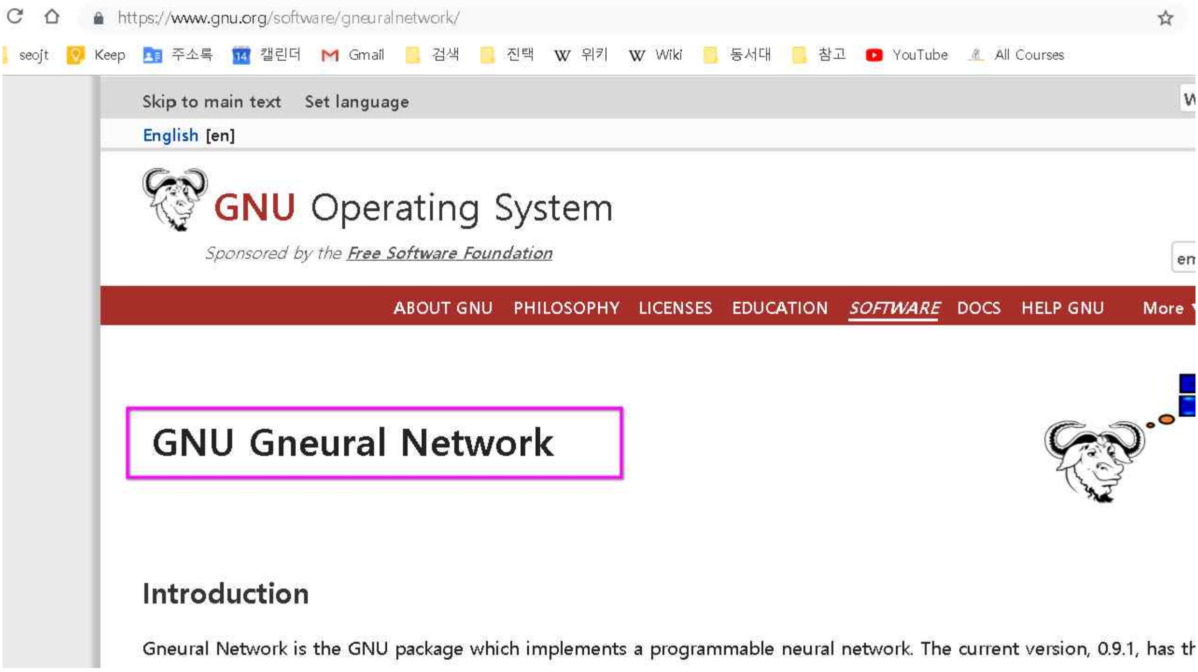
$$D_i = \sum_{i=0}^{n} (y_i - y_{d,i})^2$$

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

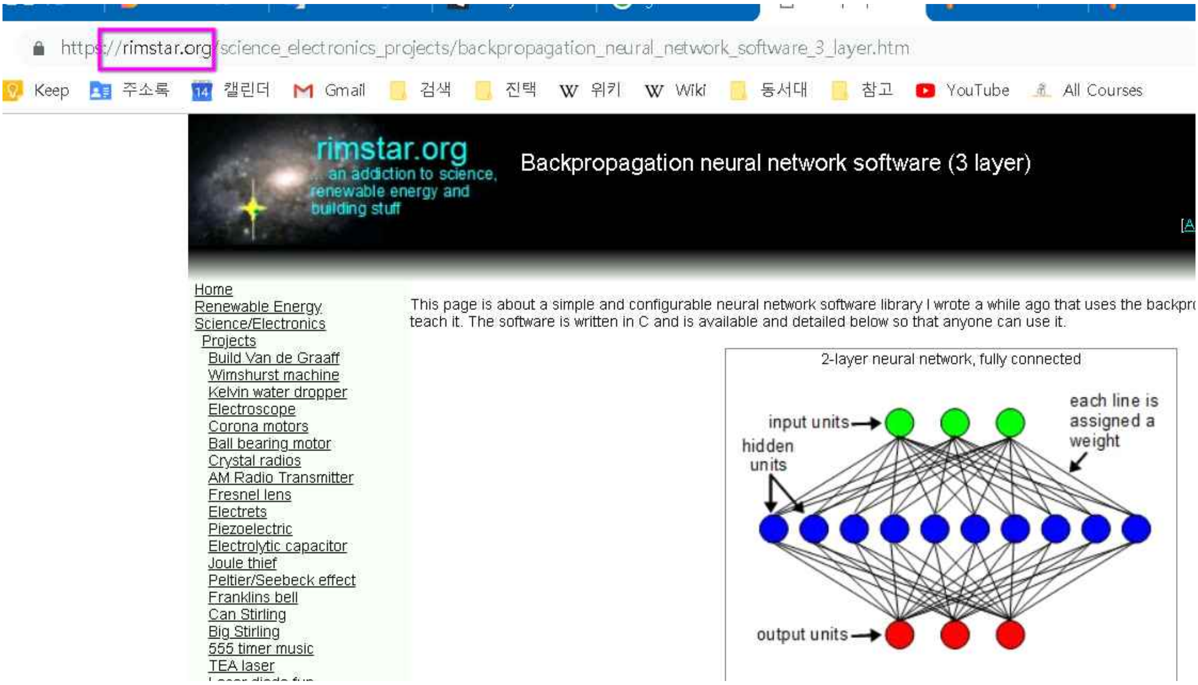
$$= D_i / 2 = \left\{ \sum_{i=0}^{n} (y_i - y_{d,i})^2 \right\} / 2$$

Naive C Implementation

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

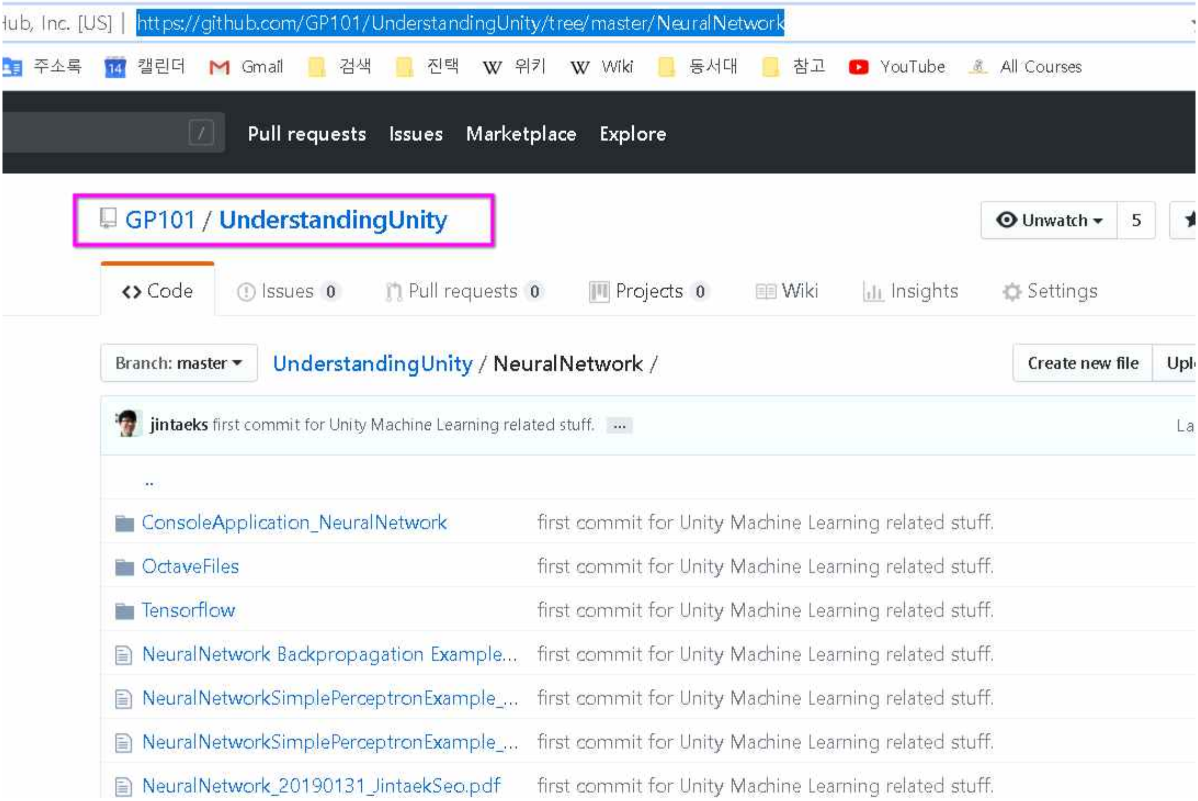


https://rimstar.org/science_electronics_projects/backpropagation_neural_network_software_3_layer.htm

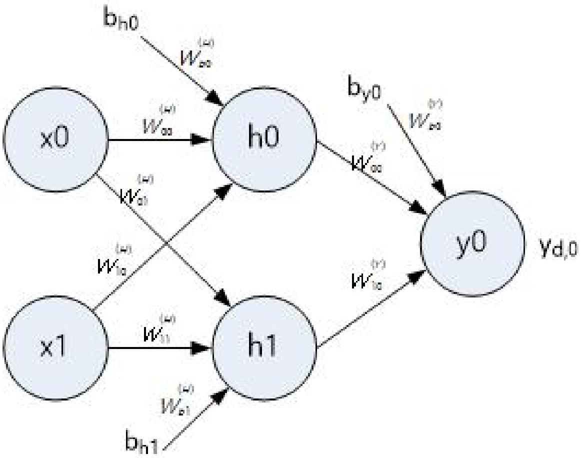


C++ conversion

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



Implementation Issues



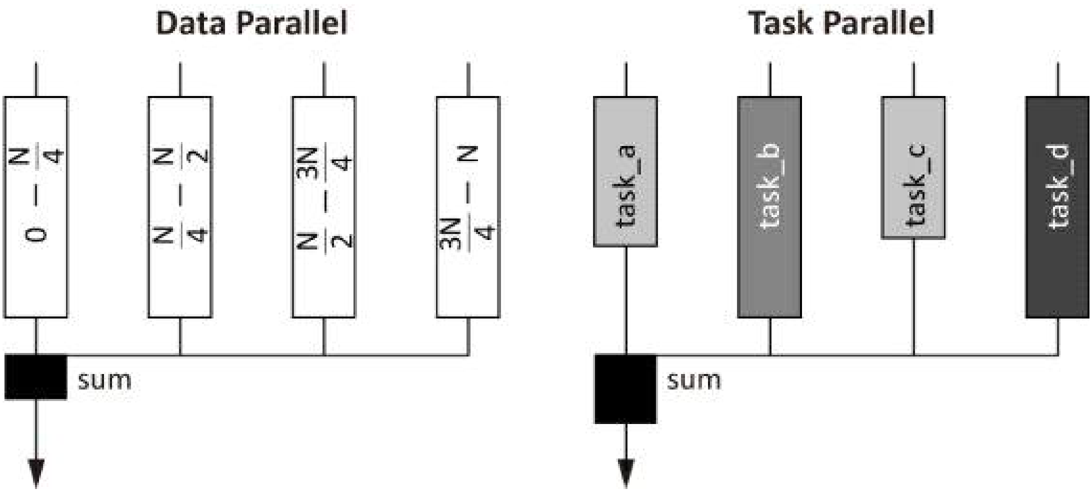
$$h0 = sigmoid(x_0 w_{00}^{(H)} + x_1 w_{10}^{(H)} + b_{h0} w_{b0}^{(H)})$$

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

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

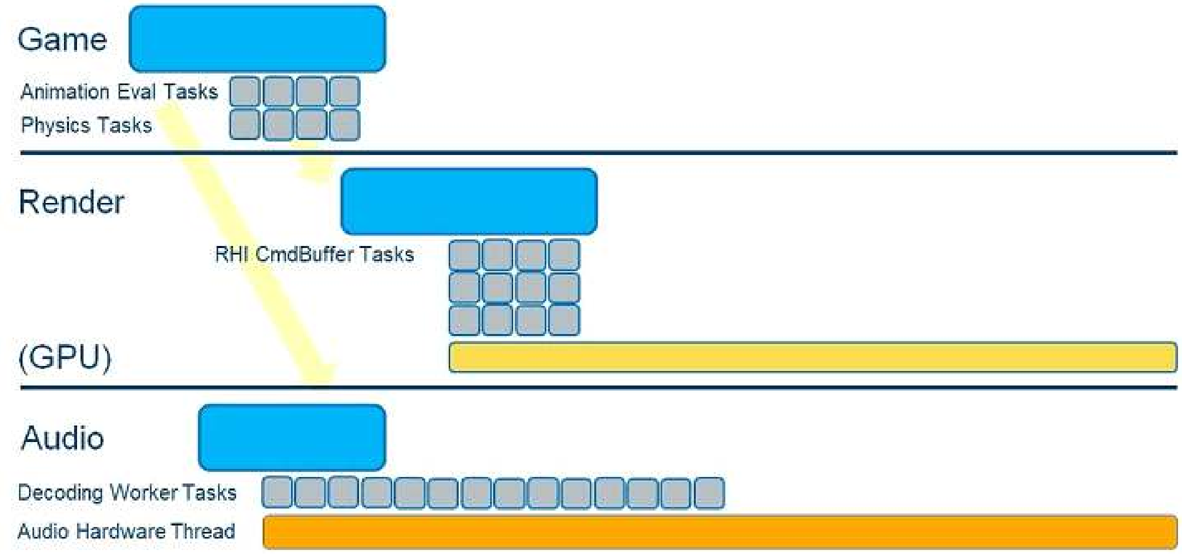
$$\begin{bmatrix} W_{00} & W_{10} \\ W_{01} & W_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} + \begin{bmatrix} bias_0 \\ bias_1 \end{bmatrix} = \begin{bmatrix} WeightSum_0 \\ WeightSum_1 \end{bmatrix}$$

How to implement with multi-thread?

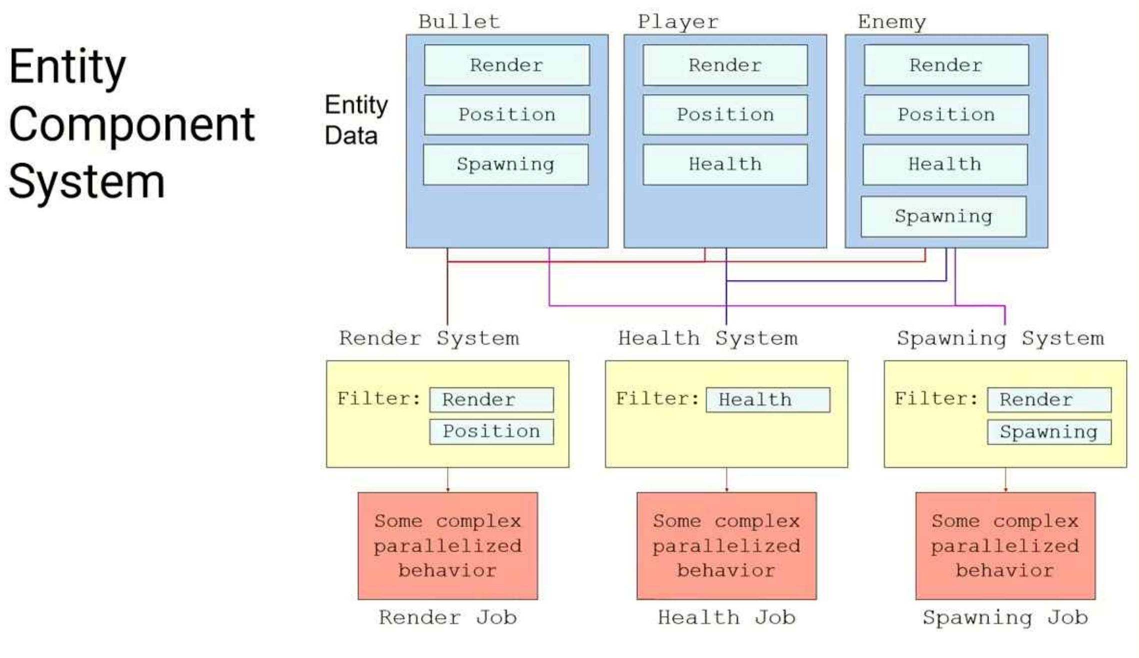


Unreal Engine 4 threading model

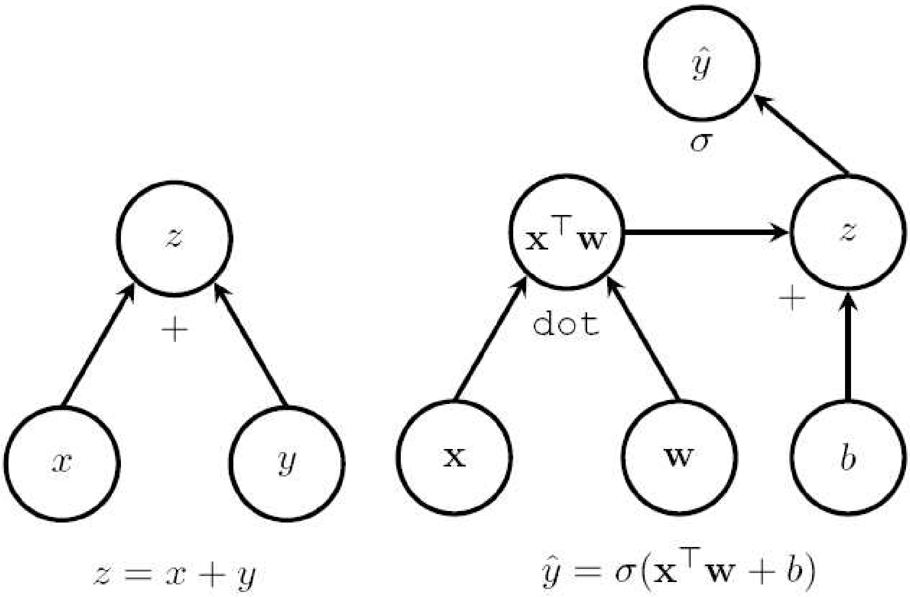
https://software.intel.com/en-us/articles/intel-software-engineers-assist-with-unreal-engine-419-optimizations



Unity ECS(Entity Component System)



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

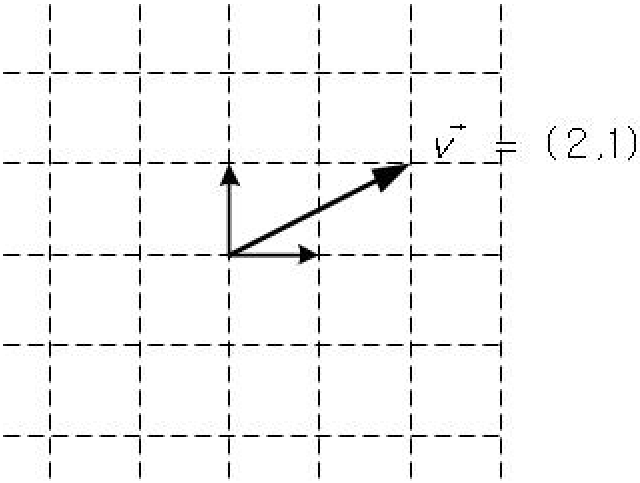
Scalar

mass(real number)

Vector

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

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



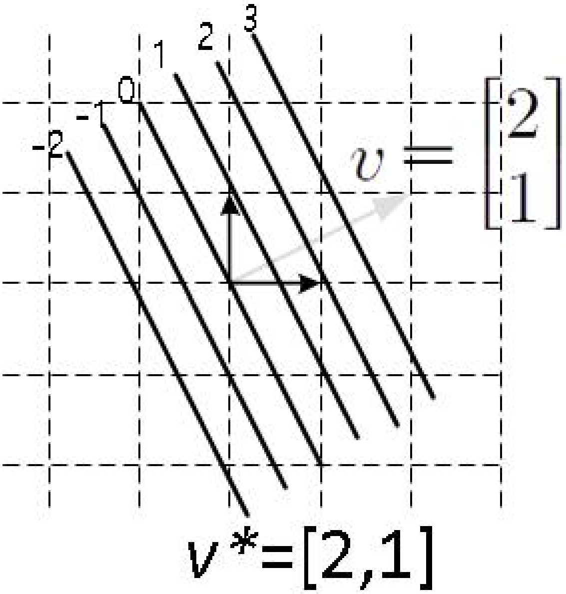
Covector

a function for a column vector.

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

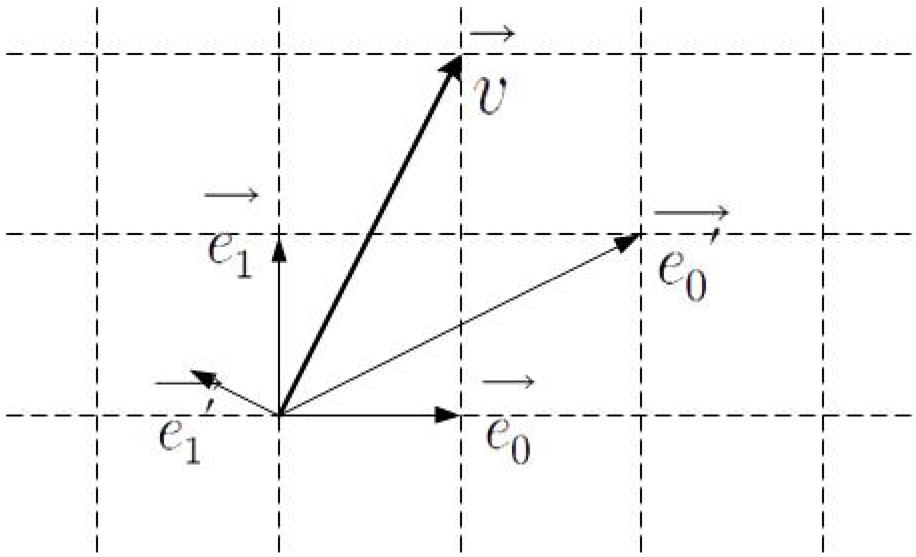
$$[2\ \ 1]\left(\begin{bmatrix}x\\y\end{bmatrix}\right)=2x+1y$$

$$2x+1y=-2$$
$$2x+1y=-1$$
$$2x+1y=0$$
$$2x+1y=1$$
$$2x+1y=2$$



Dual Space

Metric Tensor



$$\overrightarrow{e_0} = \begin{bmatrix}1\\0\end{bmatrix},\quad \overrightarrow{e_1} = \begin{bmatrix}0\\1\end{bmatrix},\quad \vec{v} = \begin{bmatrix}1\\2\end{bmatrix}$$

$$\vec{v} = 1 \times \overrightarrow{e_0} + 2 \times \overrightarrow{e_1}$$

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

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

$$\begin{aligned} \| \vec{v} \|^2 &= \left(v^0 \overrightarrow{e_0} + v^1 \times \overrightarrow{e_1}\right) \bullet \left(v^0 \overrightarrow{e_0} + v^1 \times \overrightarrow{e_1}\right) \\ &= (v^0)^2 (\overrightarrow{e_0} \bullet \overrightarrow{e_0}) + 2 v^0 v^1 (\overrightarrow{e_0} \bullet \overrightarrow{e_1}) + (v^1)^2 (\overrightarrow{e_1} \bullet \overrightarrow{e_1}) \end{aligned}$$

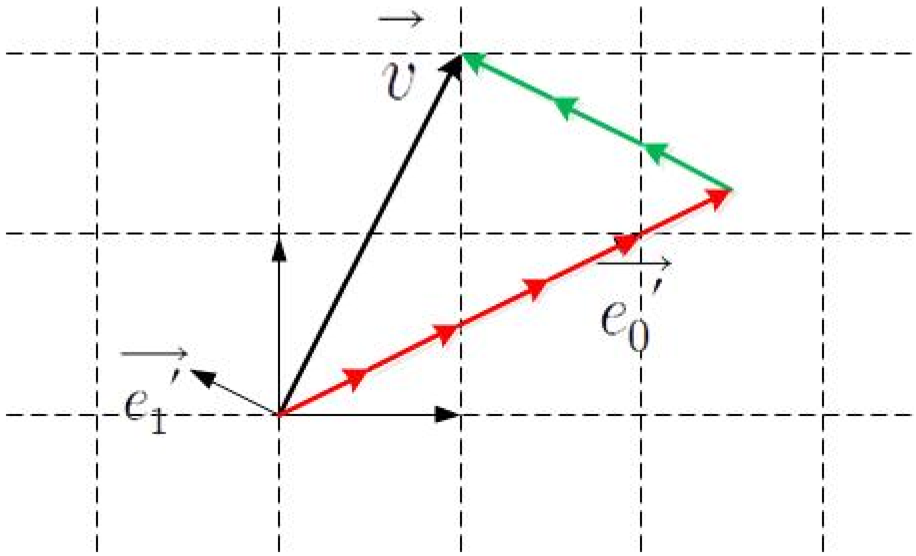
$$[v_0 \ \ v_1] \begin{bmatrix} (\overrightarrow{e_0} \bullet \overrightarrow{e_0}) & (\overrightarrow{e_0} \bullet \overrightarrow{e_1}) \\ (\overrightarrow{e_0} \bullet \overrightarrow{e_1}) & (\overrightarrow{e_1} \bullet \overrightarrow{e_1}) \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \end{bmatrix}$$

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

$$= [v_0 \ \ v_1] \begin{bmatrix} v_0 \\ v_1 \end{bmatrix}$$

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

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



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

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

$$[\frac{5}{4} \quad 3] \begin{bmatrix} (\vec{e_0}' \bullet \vec{e_0}') & (\vec{e_0}' \bullet \vec{e_1}') \\ (\vec{e_0}' \bullet \vec{e_1}') & (\vec{e_1}' \bullet \vec{e_1}') \end{bmatrix} \begin{bmatrix} \dfrac{5}{4} \\ 3 \end{bmatrix}$$

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

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

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

$$[\frac{5}{4} \quad 3] \begin{bmatrix} 5 & -\dfrac{3}{4} \\ -\dfrac{3}{4} & \dfrac{5}{16} \end{bmatrix} \begin{bmatrix} \dfrac{5}{4} \\ 3 \end{bmatrix} = 5$$

Machine Learning Library: TensorFlow

@