Fakultas Informatika
School of Computing
Telkom University

Machine Learning

Backpropagation

ADF

Outline

- Gradient Descent
- Backpropagation
- Neural Network Training
- Advanced Techniques

Aside:
Mathematical Notation

Recall the Notation

$i, j, x, y, z, \ldots$	Scalar, single value	Plain italic letters
$\mathbf{x}, \mathbf{y}, \mathbf{v}, \ldots$	Vector, list	Bold letters
$X, Y, Z, \ldots$	Matrix, tensor	Capital letters
$\mathbb{Z}$	The set of integers	
$\mathbb{R}$	The set of real numbers	
$\mathbf{x} \in \mathbb{R}^n$	$\mathbf{x}$ is a set of n-dimensional vector, of real numbers	
$X \in \mathbb{R}^{a \times b}$	X is a matrix of real numbers with a rows and b columns	

Implementation Note

- In most programming language that supports matrix operation like, a vector is as either column-vector or row-vector

$$\mathbf{a} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \mathbb{R}^d \Rightarrow \mathbf{b} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \mathbb{R}^{d\times 1} \quad \text{or} \quad \mathbb{R}^{1\times d} \; \mathbf{c} = [x_1 \; x_2 \; \cdots x_d]$$

- Which is technically a matrix
- So you can transpose row-vector into column-vector and vice-versa

$$\mathbf{b}^T = [x_1 \; x_2 \; \cdots x_d] \quad \mathbb{R}^{1\times d} \qquad == \mathbf{c} = [x_1 \; x_2 \; \cdots x_d] \quad \mathbb{R}^{1\times d}$$

Implementation Note

- However, in most popular machine learning library and programming language like Python and Torch (Lua), a vector is defined as a 1-dimensional array. It is visualized as row-vector

$$\boxed{\mathbf{a} = [x_1\ x_2\ \cdots x_d]}\ \mathbb{R}^d$$

- And transpose is defined as flipping the dimension.

- Thus transposing a vector (1D/row-vector) is still a vector (1D/row-vector)

$$\boxed{\mathbf{a}^T = [x_1\ x_2\ \cdots x_d]}\ \mathbb{R}^d$$

- This also affects several other matrix/vector operations

Implementation Note

- Therefore, for this slide,
  we will refer row-vectors and column-vectors as 2D matrices

- Note the dimension and illustrations

$$\mathbf{a} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \quad \mathbb{R}^d$$

  - In math,
    default vector is
    a column-vector

$$\mathbf{b} = [x_1 \; x_2 \; \cdots \; x_d] \quad \mathbb{R}^{1\times d}$$

$$\mathbf{c} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \quad \mathbb{R}^{d\times 1}$$

$$E = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix} \quad \mathbb{R}^{n\times d}$$

Aside:
Linear Regression
in Matrix Form

Linear Regression in Matrix

- Linear form

$$\boxed{\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_d x_d}$$

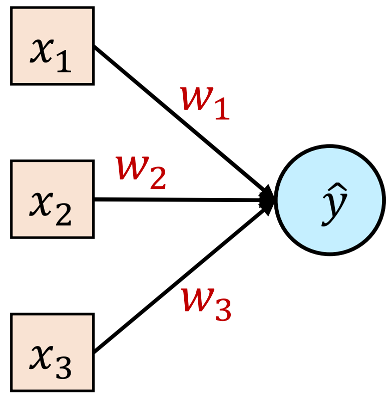
$$\boxed{\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b}$$

- Matrix multiplication form

$$\boxed{\hat{y} = \mathbf{w} \cdot \mathbf{x} + b} \quad \boxed{\hat{y} = \mathbf{w} \cdot \mathbf{x}} \quad \boxed{\hat{y} = \mathbf{w}^T \mathbf{x} + b}$$

$$\boxed{\hat{y} = XW + b} \quad \boxed{\hat{y} = W\mathbf{x}^T + b} \quad \boxed{\hat{y} = WX^T + b}$$

Linear Regression in Matrix

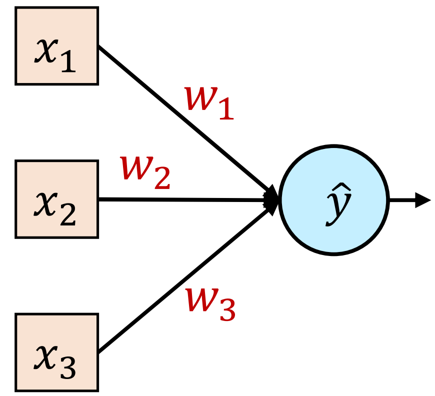


$$\boxed{\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b}$$

$$\underbrace{\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}}_{\mathbb{R}^d} \quad \underbrace{\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}_{\mathbb{R}^d} \quad \underbrace{b}_{\mathbb{R}} \quad \Rightarrow \quad \underbrace{\boxed{\hat{y} = \mathbf{w} \cdot \mathbf{x} + b}}_{\mathbb{R}}$$

$$\boxed{\hat{y} = \mathbf{x} \cdot \mathbf{w} + b}$$

Linear Regression in Matrix



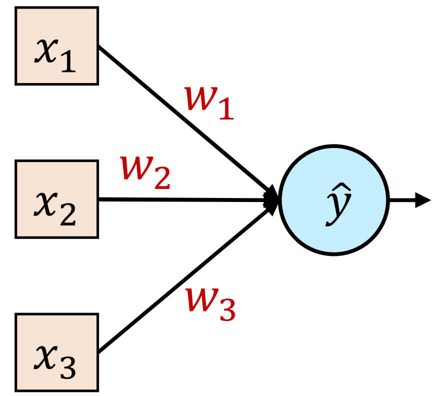
$$\boxed{\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b}$$

$$\underbrace{\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ b \end{bmatrix}}_{\mathbb{R}^{d+1}} \qquad \underbrace{\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ 1 \end{bmatrix}}_{\mathbb{R}^{d+1}}$$

$$\Rightarrow \quad \underbrace{\boxed{\hat{y} = \mathbf{w} \cdot \mathbf{x}}}_{\mathbb{R}} \; !$$

$$\boxed{\hat{y} = \mathbf{x} \cdot \mathbf{w}}$$

Linear Regression in Matrix



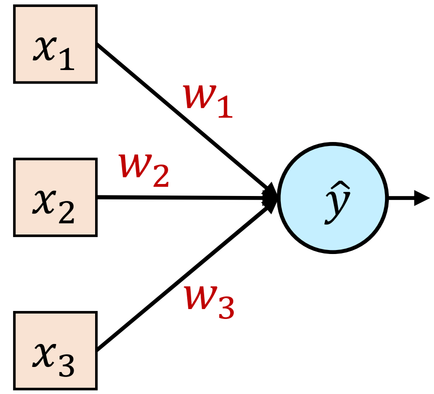
$$\boxed{\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b}$$

- In math, default vector is a column-vector

$$\overset{\mathbb{R}^{d\times 1}}{\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}} \quad \overset{\mathbb{R}^{d\times 1}}{\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}} \quad \overset{\mathbb{R}}{b} \Rightarrow \overset{\mathbb{R}^{1\times 1}}{\boxed{\hat{y} = \mathbf{w}^T \mathbf{x} + b}}$$

!

Linear Regression in Matrix



$$\boxed{\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b}$$

i	x_1	x_2	…	x_d	y
1	0.5	0.2	…	0.7	1
2	0.1	0.3	…	0.2	0
…	…	…	…	…	…
n	0.2	0.5	…	0.2	0

$$X = \begin{bmatrix} 0.5 & 0.2 & \cdots & 0.7 \\ 0.1 & 0.3 & \cdots & 0.2 \\ \vdots & \vdots & \ddots & \vdots \\ 0.2 & 0.5 & \cdots & 0.2 \end{bmatrix} \quad \mathbb{R}^{n \times d}$$

$$\mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \mathbb{Z}^{n}$$

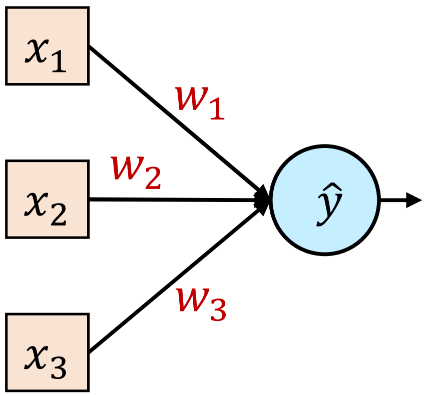
$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \quad \mathbb{R}^{d}$$

$$b \quad \Rightarrow \quad \boxed{\hat{\mathbf{y}} = X\mathbf{w} + b} \quad \mathbb{R}^{n}$$

!

$$\boxed{\hat{\mathbf{y}} = \mathbf{w}X^{T} + b}$$

Default Formula



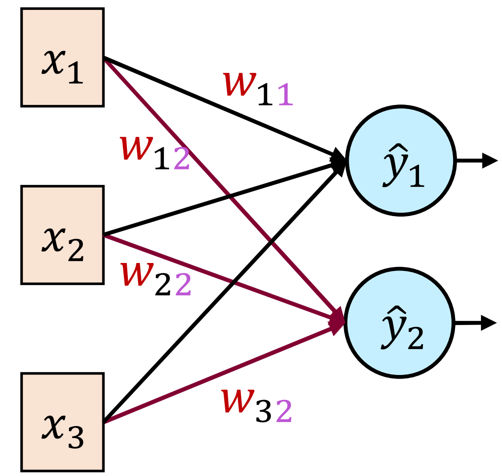
$$\boxed{\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^d \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \in \mathbb{R}^d \qquad b \in \mathbb{R} \quad \Rightarrow \quad \boxed{\hat{y} = X\mathbf{w} + b} \in \mathbb{R}$$

Linear Regression in Matrix

$$\begin{aligned}
\hat{y}_1 &= w_{11}x_1 + w_{21}x_2 + \cdots + w_{d1}x_d + b_1 \\
\hat{y}_2 &= w_{12}x_1 + w_{22}x_2 + \cdots + w_{d2}x_d + b_2 \\
&\ldots \\
\hat{y}_c &= w_{1c}x_1 + w_{2c}x_2 + \cdots + w_{dc}x_d + b_c
\end{aligned}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_c \end{bmatrix} \quad \mathbb{R}^c$$



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \quad \mathbb{R}^d$$

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1c} \\ w_{21} & w_{22} & \cdots & w_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{dc} & \cdots & w_{dc} \end{bmatrix} \quad \mathbb{R}^{d\times c}$$

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_c \end{bmatrix} \quad \mathbb{R}^c$$

$$\Rightarrow \boxed{\hat{\mathbf{y}} = \mathbf{x}W + b} \quad \mathbb{R}^c$$

!

$$\boxed{\hat{\mathbf{y}} = W\mathbf{x}^T + b}$$

Linear Regression in Matrix

i	x_1	x_2	...	x_d	y_1	y_2	y_3
1	0.5	0.2	...	0.7	1	0	0
2	0.1	0.3	...	0.2	0	1	0
...	...	...	...	...	...	...	...
n	0.2	0.5	...	0.2	0	0	1

$$X = \begin{bmatrix} 0.5 & 0.2 & \cdots & 0.7 \\ 0.1 & 0.3 & \cdots & 0.2 \\ \vdots & \vdots & \ddots & \vdots \\ 0.2 & 0.5 & \cdots & 0.2 \end{bmatrix} \quad \mathbb{R}^{n \times d}$$

$$\mathbf{y} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbb{Z}^{n \times c}$$

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1c} \\ w_{21} & w_{22} & \cdots & w_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{dc} & \cdots & w_{dc} \end{bmatrix} \quad \mathbb{R}^{d \times c}$$

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_c \end{bmatrix} \quad \mathbb{R}^{c}$$

$$\Rightarrow \boxed{\hat{\mathbf{y}} = XW + b} \quad \mathbb{R}^{n \times c}$$

Gradient Descent and Backpropagation

So Far…

- Logistic Regression
  - Standard Neuron with sigmoid activation

$$\boxed{f(\mathbf{x}, W) = \sigma(W\mathbf{x} + b)}$$

$$\boxed{\sigma(v) = \frac{1}{1 + \exp(-v)}}$$

- Intuition:
  - Class score $\hat{y}$ = weighted sum of the attributes (x)
  - Use a transformation of the values of linear function

Logistic Regression

- Intuition:
  - Class score $\hat{y}$ = weighted sum of the attributes (x)
  - Use a transformation of the values of linear function
  - Find weights that minimize the Cost Function
- Problem: $\boxed{\hat{w} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}}$
  - Expensive calculation with the increasing of dimension in x,
  - Need to come up with another technique
- Solution:
  - Gradient Descent Optimization

Loss Score

- Given a binary class dataset of examples

$$\boxed{X \in \mathbb{R}^{n \times d}} \quad \boxed{\mathrm{y} \in \mathbb{Z}^{n}}$$

- Loss over the dataset is a sum of loss over examples

$$\boxed{L = \sum_{i=1}^{N} L_i} \qquad \boxed{L_i = \|y_i - f(\mathbf{x}, W)\|_2^2}$$

L2 Loss (squared error)

Loss Score

- Common choice for loss function of a data is L2 Loss or Squared-error

$$\boxed{L_i = \|f(\mathrm{x}, W) - y_i\|_2^2}$$

$$\boxed{L_i = (\hat{y}_i - y_i)^2}$$

- For that, we get the loss function as **SSE** or sum-of-squared-error

$$\boxed{L = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2}$$

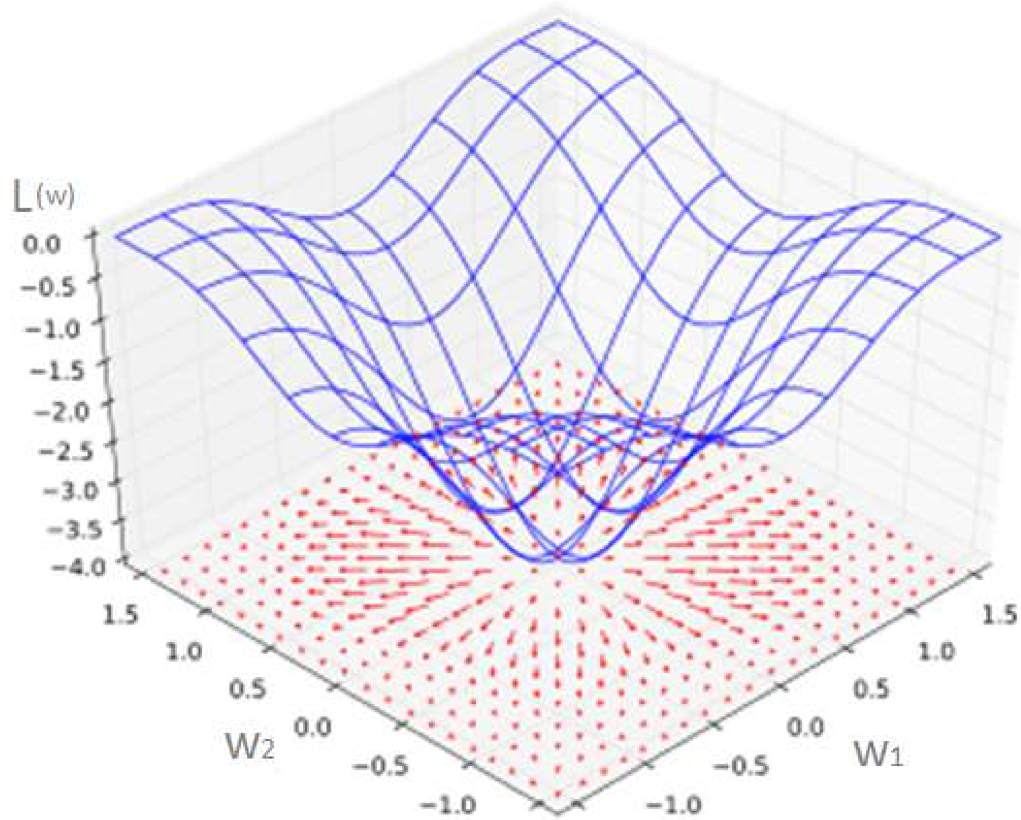
- One can also use **MSE** or mean-of-squared-error

Optimization Problem

$$\hat{y}_i = \sigma(w_1 x_1 + w_2 x_2 + b)$$

$$L = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

- Loss is a function of weights
- Find weights that minimize the loss
- Use calculus to get the gradient



Gradient Descent

- Problem:
  - generalization problem to minimize error for all data
- Gradient:
  - a slope of which way the parameters must go to minimize the error (based on current data)
- Gradient Descent
  - a first-order iterative optimization algorithm for finding the minimum of a function

Gradient Descent
in a Neuron (Logistic Regression)

Gradient Descent in a Neuron

$$\hat{y} = \sigma(\mathbf{x}\mathrm{w} + b)$$

$$\hat{y} = \sigma(v)$$

$$v = \mathbf{x}\mathrm{w} + b$$

$$\sigma(v) = \frac{1}{1 + \exp(-v)}$$

$$L = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

$$L = \frac{1}{2} \sum_{i=1}^{N} \left(\frac{1}{1 + \exp(-(\mathbf{x}\mathrm{w} + b))} - y_i \right)^2$$

- Find w that minimize L
- Calculate gradient from partial derivative

$$\frac{\partial L}{\partial \mathrm{w}} \qquad \frac{\partial L}{\partial b}$$

Gradient Descent in a Neuron

$$v = \mathrm{x}\mathrm{w} + b$$

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

Chain rule

$$\frac{\partial L}{\partial \mathrm{w}} = \frac{\partial\, {}^{1}\!/_{2}\, \Sigma(\hat{y}_i - y_i)^2}{\partial \mathrm{w}} \quad (1)$$

$$= \frac{\partial\, {}^{1}\!/_{2}\, \Sigma(\hat{y}_i - y_i)^2}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial \mathrm{w}} \quad (2)$$

$$= \frac{1}{2}2(\hat{y}_i - y_i)\frac{\partial \hat{y}_i}{\partial \mathrm{w}} \quad (3)$$

$$= (\hat{y}_i - y_i)\frac{\hat{y}_i}{\partial v}\frac{\partial v}{\partial \mathrm{w}} \quad (4)$$

$$= (\hat{y}_i - y_i)(\sigma(v) - \sigma(v)^2)\frac{\partial v}{\partial \mathrm{w}} \quad (5)$$

$$\frac{\partial L}{\partial \mathrm{w}} = (\hat{y}_i - y_i)(\sigma(v) - \sigma(v)^2)\,\mathbf{x} \quad (6)$$

$$= \frac{\partial L}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial v}\frac{\partial v}{\partial \mathrm{w}}$$

Gradient Descent Example

$$v = \mathrm{x}\mathrm{w} + b$$

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

$$\frac{\partial L}{\partial b} = \frac{\partial\, {}^{1}\!/_{2} \sum (\hat{y}_i - y_i)^2}{\partial b} \quad (1)$$

Chain rule

$$= \frac{\partial\, {}^{1}\!/_{2} \sum (\hat{y}_i - y_i)^2}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial b} \quad (2)$$

$$= \frac{1}{2} 2(\hat{y}_i - y_i) \frac{\partial \hat{y}_i}{\partial b} \quad (3)$$

$$= (\hat{y}_i - y_i) \frac{\hat{y}_i}{\partial v} \frac{\partial v}{\partial b} \quad (4)$$

$$= (\hat{y}_i - y_i)(\sigma(v) - \sigma(v)^2) \frac{\partial v}{\partial b} \quad (5)$$

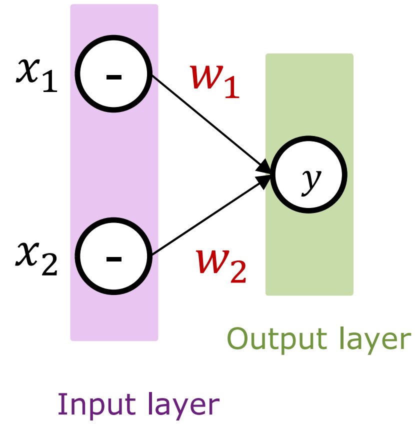
$$\frac{\partial L}{\partial b} = (\hat{y}_i - y_i)(\sigma(v) - \sigma(v)^2) \quad (6)$$

$$= \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial v} \frac{\partial v}{\partial \mathrm{b}}$$

Gradient Descent
Example for single data

Gradient Descent Example

i	x_1	x_2	y
1	-0.1	0.2	0
2	0.1	0.8	0
3	0.9	-0.2	0
4	0.8	0.8	1



$$f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-v)}$$

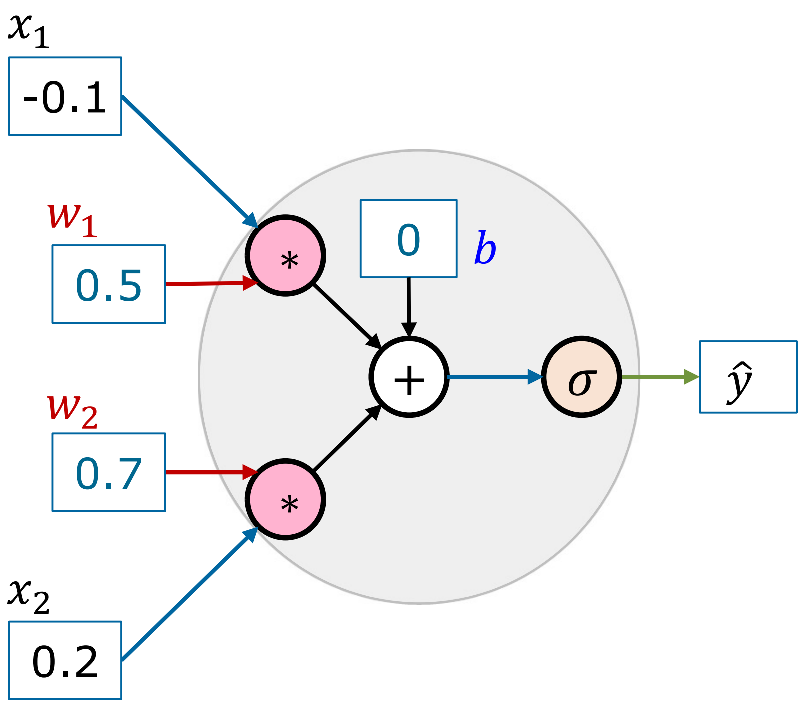
$$v = w_1 x_1 + w_2 x_2 + b$$

- Initialize weights with random and bias with zero

w_1	0.5
w_2	0.7

b	0

Neuron Gate Structure



$$f(\mathrm{x}, \mathrm{w}) = \frac{1}{1 + \exp(-v)}$$

$$v = w_1 x_1 + w_2 x_2 + b$$

w_1	0.5
w_2	0.7

b	0

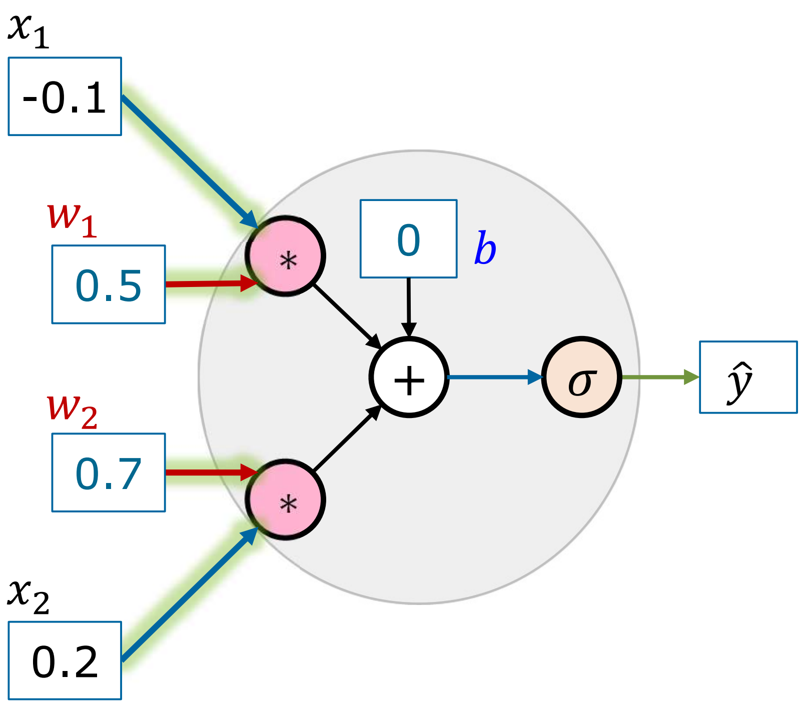
x_1	x_2	y
-0.1	0.2	0

$$m = w_1 x_1$$

$$n = w_2 x_2$$

$$v = m + n + b$$

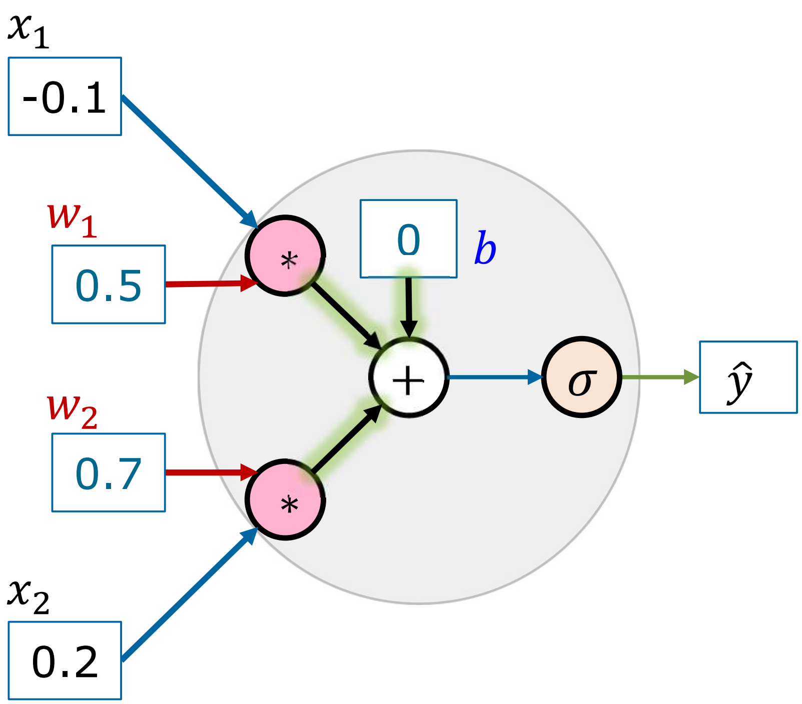
Forward Pass



$$m = w_1 x_1 = -0.05$$

$$n = w_2 x_2 = 0.14$$

Forward Pass

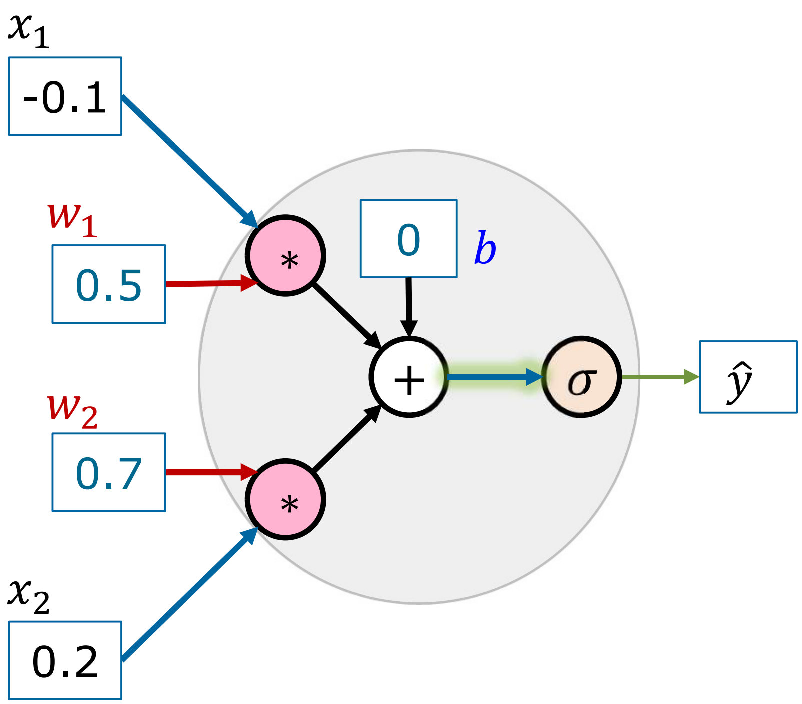


$$m = w_1 x_1 = -0.05$$

$$n = w_2 x_2 = 0.14$$

$$v = m + n + b = -0.09$$

Forward Pass



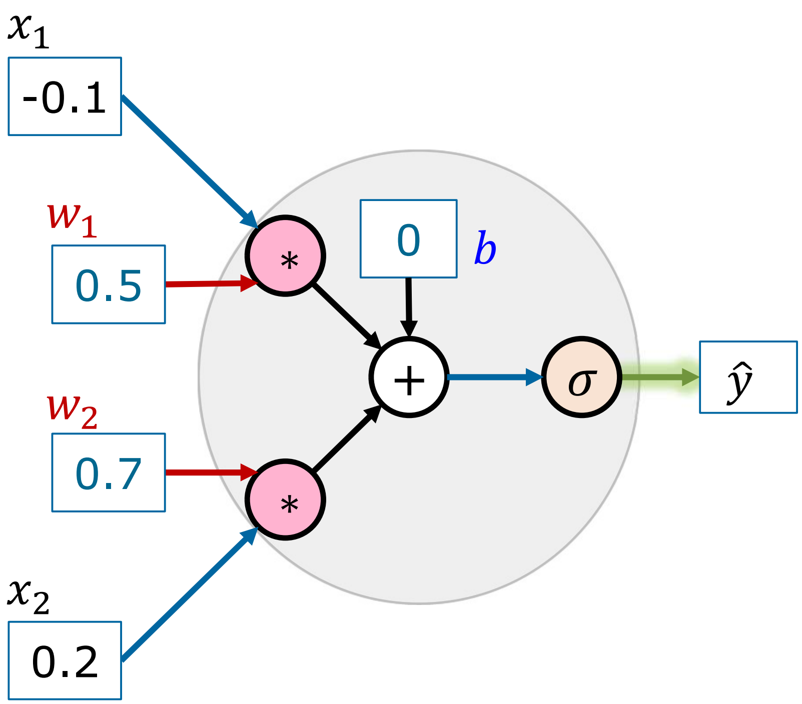
$$m = w_1 x_1 = -0.05$$

$$n = w_2 x_2 = 0.14$$

$$v = m + n + b = -0.09$$

$$\hat{y} = \frac{1}{1 + \exp(-v)} = 0.522$$

Loss Gradient



$$m = w_1 x_1 = -0.05$$

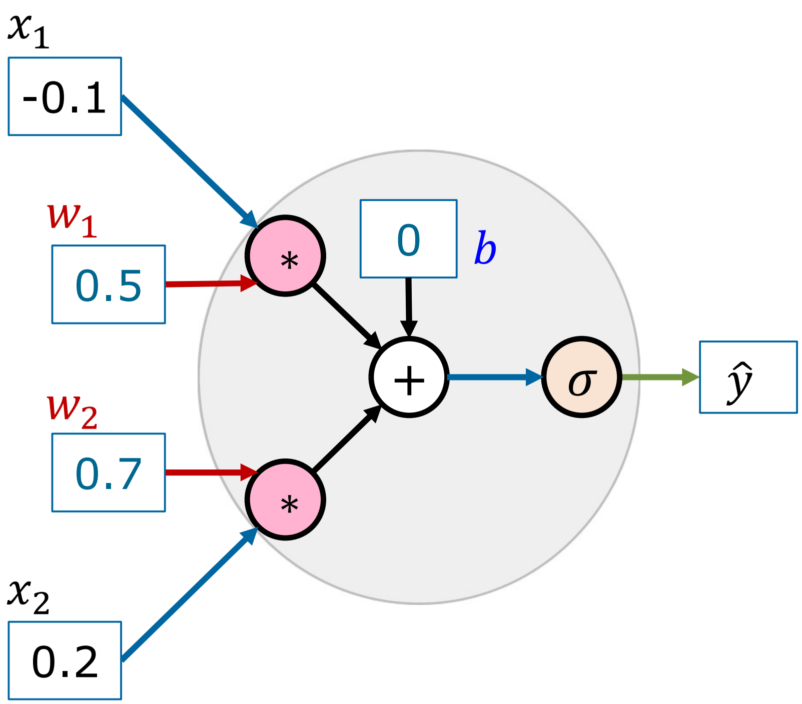
$$n = w_2 x_2 = 0.14$$

$$v = m + n + b = -0.09$$

$$\hat{y} = \frac{1}{1 + \exp(-v)} = 0.522$$

$$\frac{\partial L}{\partial \hat{y}} = (\hat{y}_i - y_i) = 0.522$$

Loss Gradient



Gradient

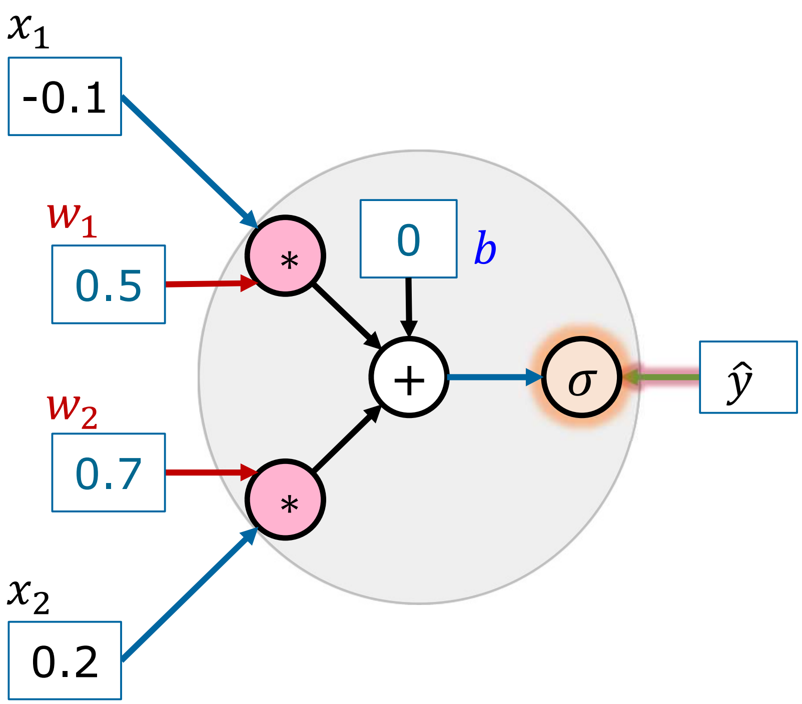
$$\frac{\partial L}{\partial \hat{y}} = (\hat{y}_i - y_i) = 0.522$$

$$\frac{\partial L}{\partial w_1} = ?$$

$$\frac{\partial L}{\partial w_2} = ?$$

$$\frac{\partial L}{\partial b} = ?$$

Backward Pass



Gradient

$$\frac{\partial L}{\partial \hat{y}} = (\hat{y}_i - y_i) = 0.522$$

$$\hat{y} = \frac{1}{1 + \exp(-v)} = 0.522$$

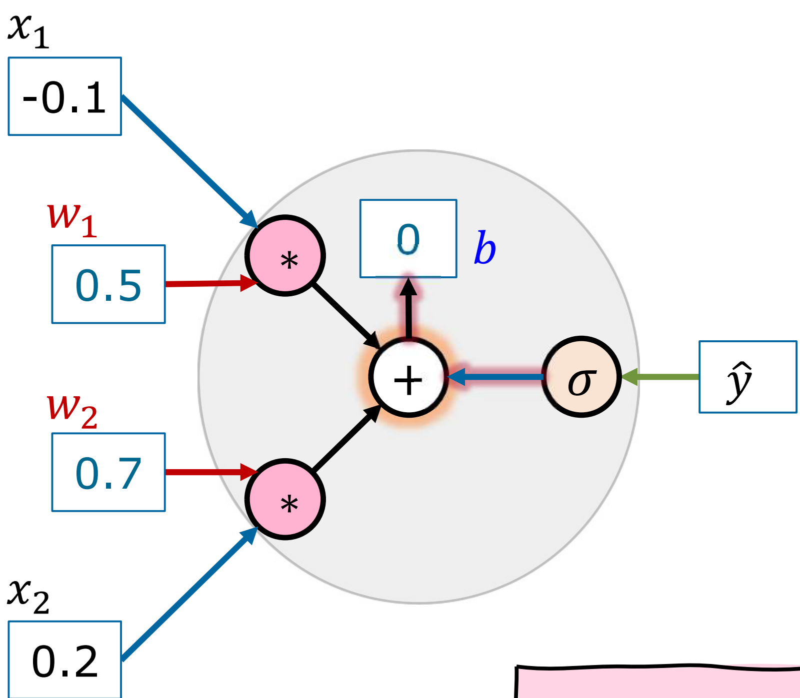
$$\frac{\partial \hat{y}}{\partial v} = \hat{y} - \hat{y}^2 = 0.249$$

Local Gradient

Chain Rule

$$\frac{\partial L}{\partial v} = \frac{\partial \hat{y}}{\partial v}\frac{\partial L}{\partial \hat{y}} = 0.13$$

Backward Pass



Gradient

$$\frac{\partial L}{\partial v} = \frac{\partial \hat{y}}{\partial v}\frac{\partial L}{\partial \hat{y}} = 0.13$$

$$v = m + n + b$$

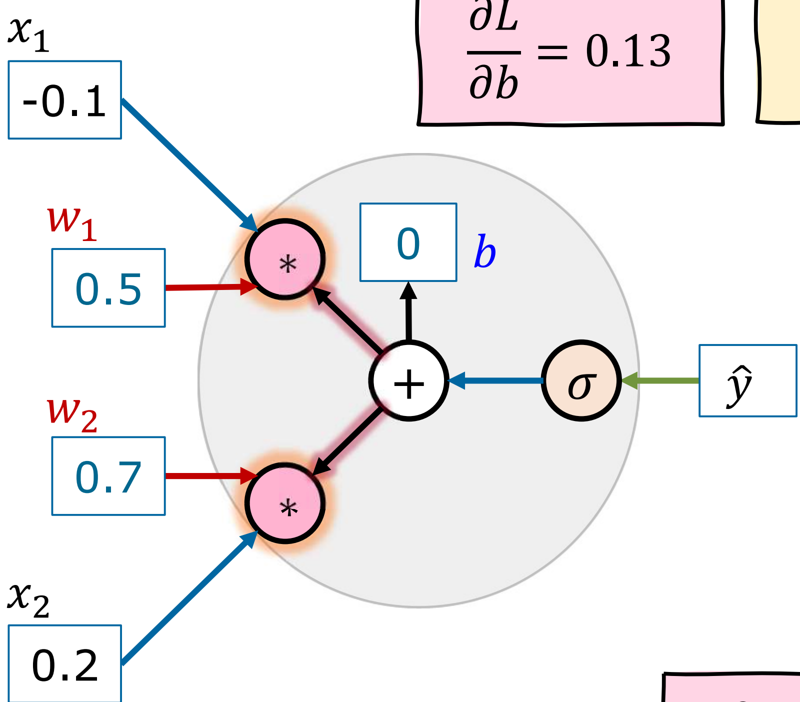
Local Gradient

$$\frac{\partial v}{\partial m} = 1 \qquad \frac{\partial v}{\partial n} = 1 \qquad \frac{\partial v}{\partial b} = 1$$

Chain Rule

$$\frac{\partial L}{\partial b} = 0.13 \qquad \frac{\partial L}{\partial n} = 0.13 \qquad \frac{\partial L}{\partial m} = \frac{\partial v}{\partial m}\frac{\partial \hat{y}}{\partial v}\frac{\partial L}{\partial \hat{y}} = 0.13$$

Backward Pass



Gradient

$$\frac{\partial L}{\partial b} = 0.13 \qquad \frac{\partial L}{\partial n} = 0.13 \qquad \frac{\partial L}{\partial m} = \frac{\partial v}{\partial m}\frac{\partial \hat{y}}{\partial v}\frac{\partial L}{\partial \hat{y}} = 0.13$$

$$m = w_1 x_1 \qquad n = w_2 x_2$$

$$\frac{\partial m}{\partial w_1} = x_1 \qquad \frac{\partial n}{\partial w_2} = x_2$$

Local Gradient

Chain Rule

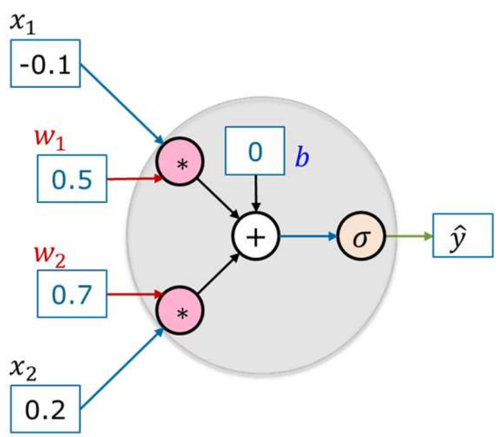
$$\frac{\partial L}{\partial w_2} = 0.026 \qquad \frac{\partial L}{\partial w_1} = \frac{\partial m}{\partial w_1}\frac{\partial v}{\partial m}\frac{\partial \hat{y}}{\partial v}\frac{\partial L}{\partial \hat{y}} = -0.013$$

Weight Update

w_1	0.5
w_2	0.7

b	0

α or η
1



$$W_{t+1} = W_t - \alpha \nabla f(W_t)$$

$$\nabla w_1 = -0.013$$

$$w_1 = w_1 - \alpha\ \nabla w_1 = 0.513$$

$$\nabla w_2 = 0.026$$

$$w_2 = w_2 - \alpha\ \nabla w_2 = 0.674$$

$$\nabla b = 0.13$$

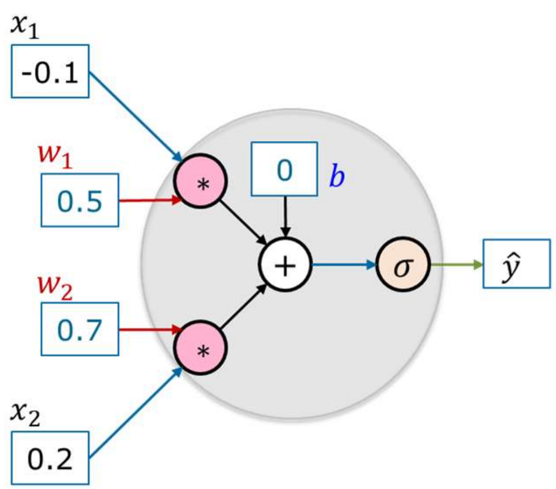
$$b = b - \alpha\ \nabla b = -0.37$$

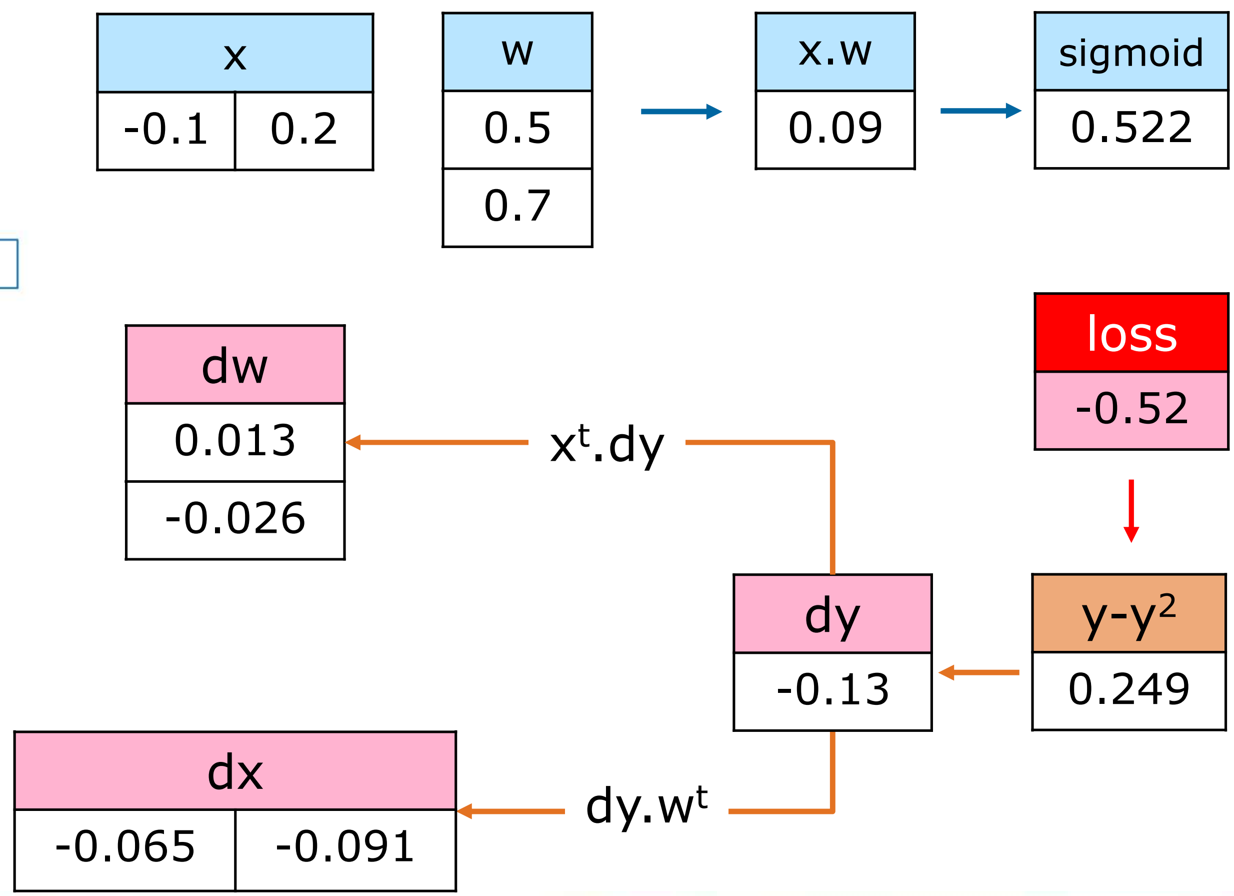
w_1	0.513
w_2	0.674

b	-0.37

Gradient Descent Example Vectorized

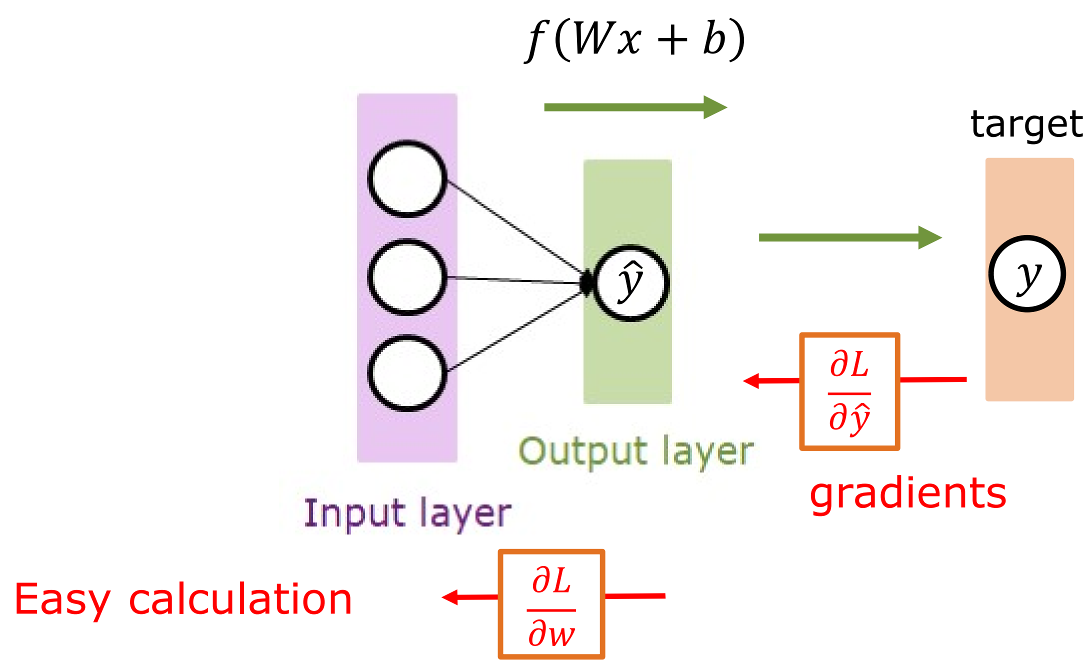
Chain Rule Vectorized





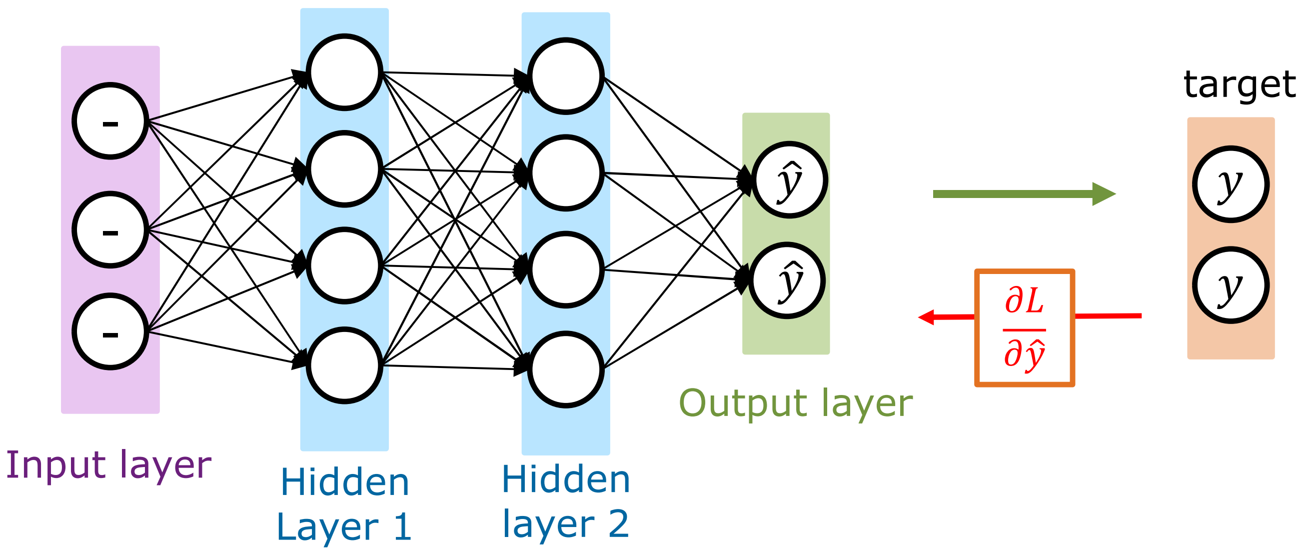
Backpropagation in Neural Network

Single Layer Derivative, Easy



Multi Layer Derivative?

$$f(W_3\ f(W_2\ f(W_1 x))) = \frac{1}{1+e^{-\left(W_3 \frac{1}{1+e^{-\left(W_2 \frac{1}{1+e^{-(W_1 x)}}\right)}}\right)}}$$



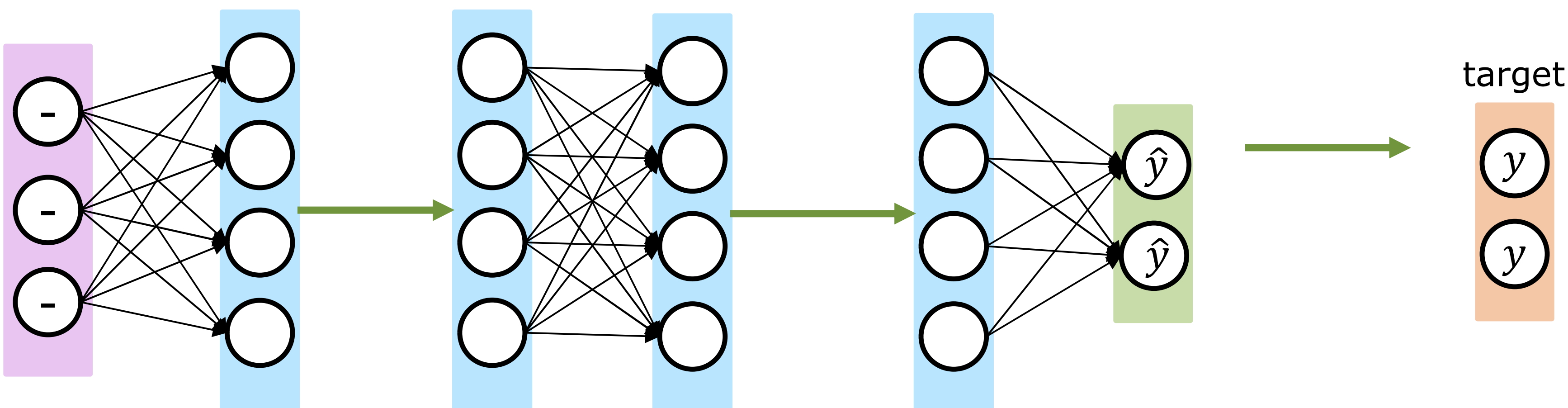
$\boxed{\frac{\partial L}{\partial W} = ???}$

Complicated to calculate as a whole,
Modifying any function require re-derive from scratch

MLPs = Stacks of Linear Functions

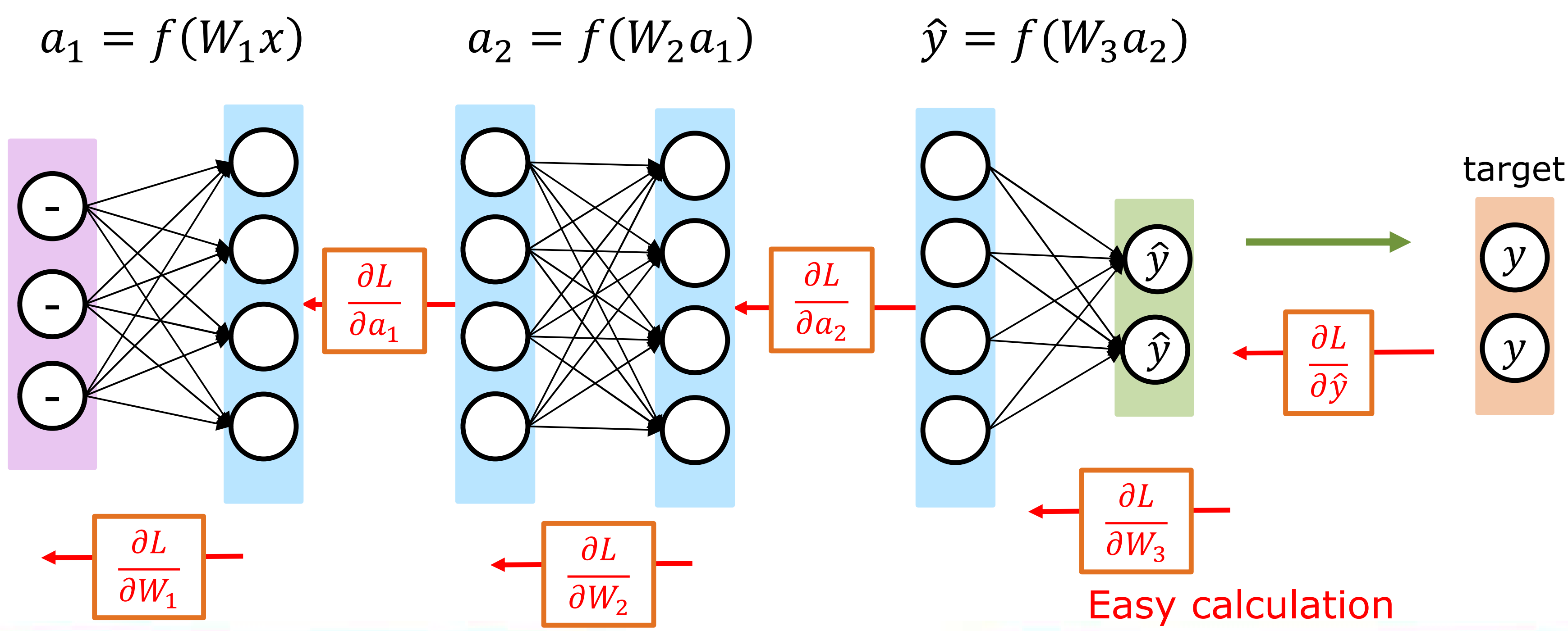
- Each layer use the same function
  - Remember that when calculating gradient, we get ∂w and ∂x

$a_1 = f(W_1 x)$ $a_2 = f(W_2 a_1)$ $\hat{y} = f(W_3 a_2)$

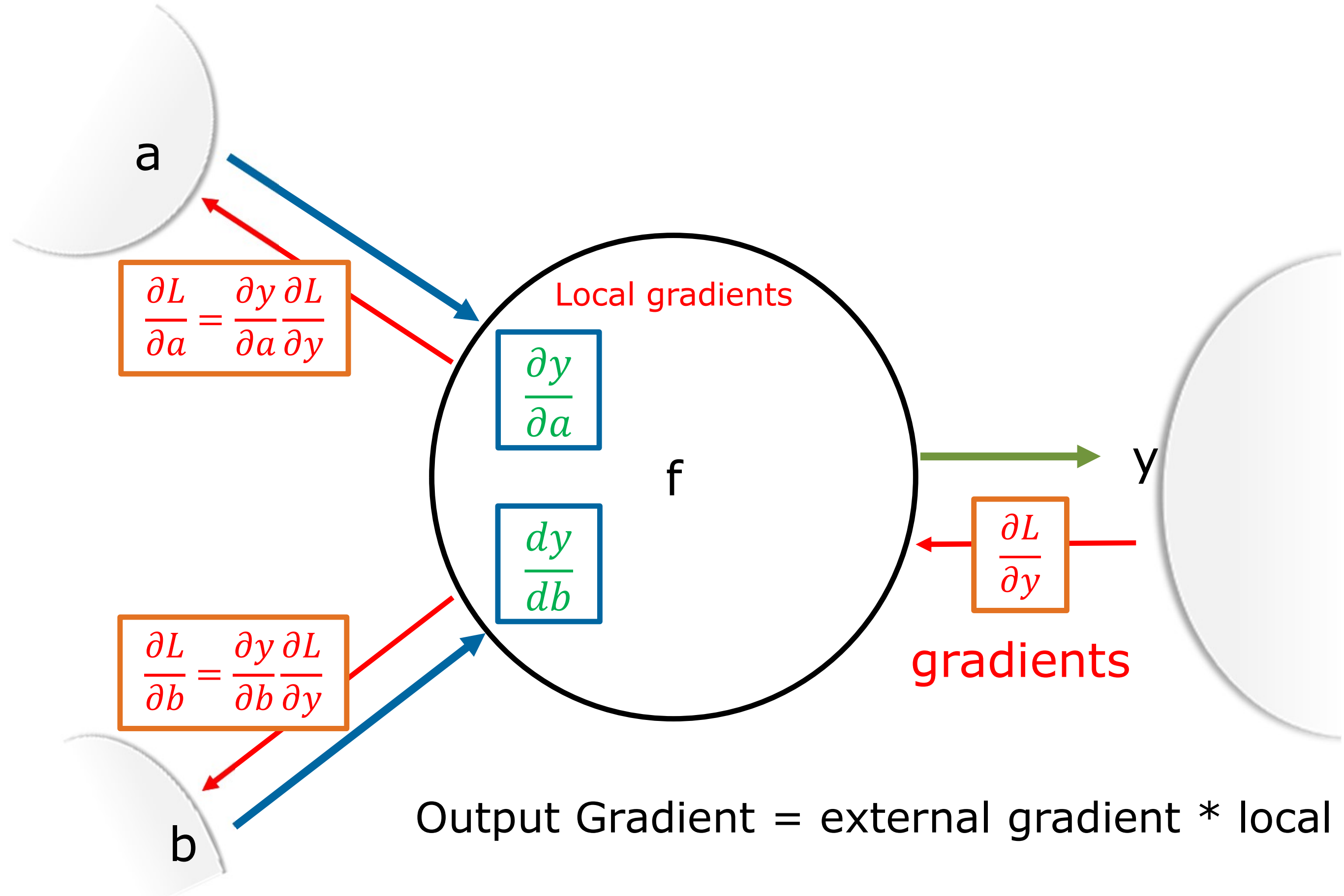


Back Propagate the Gradient

- Calculate gradient one-by one
  - Output Gradient = external gradient * local gradient



This is called: Backpropagation



Output Gradient = external gradient * local gradient

Backpropagation

- Backward propagation of errors
- Calculating gradient of weights in the stacked functions by flowing gradient error that is computed at the output end and distributed backwards throughout the network's layers

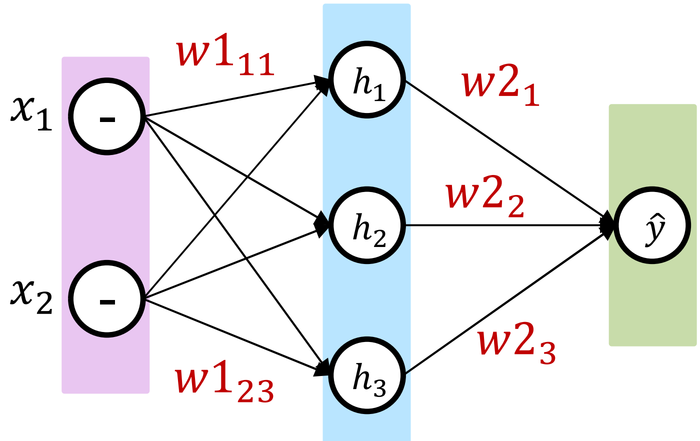
2-steps in Backpropagation

- Forward Pass
  - Calculate forward the input with the weights in each layer toward the output
  - Calculate the error of the output based on the target
- Backward Pass
  - Propagate back the errors to the weights via their gradients in each layer

Backpropagation Example in 2-Layer Neural Net

XOR Gate Example

i	x_1	x_2	y
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0



$$v = X\,W + b$$

$$A = \frac{1}{1 + \exp(-v)}$$

Forward Pass

$$v1 = \mathbf{x}W1 + b1$$ 1

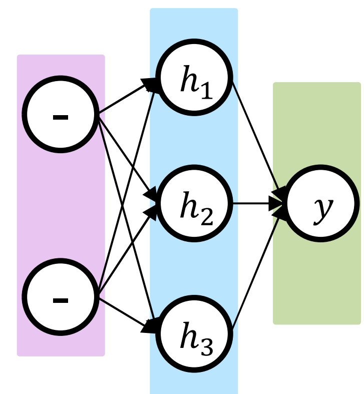
$$A1 = \frac{1}{1 + \exp(-v1)}$$ 2

$$v2 = A1W2 + b2$$ 3

$$\hat{y} = \frac{1}{1 + \exp(-v2)}$$ 4

Calculate Error / Loss

$$E = (y - \hat{y})$$ 5



Backward Pass

$db1 = \sum dv1$

$dW1 = x^T \cdot dv1$ 6

$dv1 = dA1 * (A1 - A1^2)$ 5

$\frac{\partial v1}{\partial b1} = 1$

$\frac{\partial v1}{\partial W1} = x$

$\frac{\partial A1}{\partial v1} = A1 - A1^2$

$dA1 = dv2 \cdot W2^T$ 4

$\frac{\partial v2}{\partial A1} = W2$

$dv2 = E * (\hat{y} - \hat{y}^2)$ 2

$\frac{\partial v2}{\partial b2} = 1$

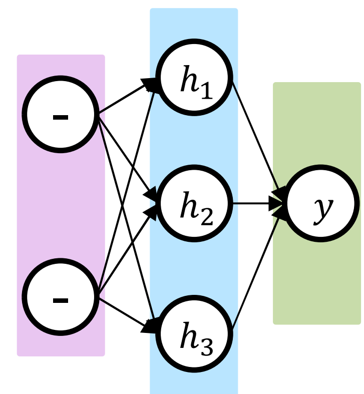
$\frac{\partial v2}{\partial W2} = A1$

$\frac{\partial \hat{y}}{\partial v2} = \hat{y} - \hat{y}^2$

$db2 = \sum dv2$

$dW2 = A1^T \cdot dv2$ 3

$E = (y - \hat{y})$ 1



Complete Step (Iteration)

$v1 = x\,W1 + b1$

$A1 = 1/(1 + \exp(-v1)\,)$

$v2 = A1W2 + b2$

$\hat{y} = 1/(1 + \exp(-v2)\,)$

$E = (y - \hat{y})$

$dv2 = E * (\hat{y} - \hat{y}^2)$

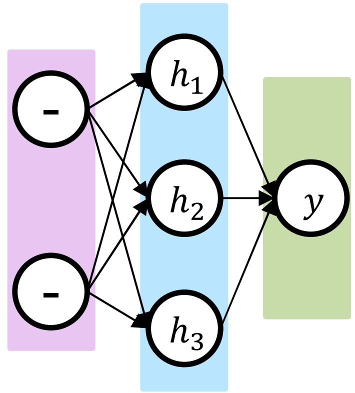
$dW2 = A1^T \cdot dv2$ $\quad db2 = \sum dv2$ $\quad dA1 = dv2 \cdot W2^T$

$dv1 = dA1 * (A1 - A1^2)$

$dW1 = x^T \cdot dv1$ $\quad db1 = \sum dv1$

$b1 = b1 + \alpha * db1$
$W1 = W1 + \alpha * dW1$

$b2 = b2 + \alpha * db2$
$W2 = W2 + \alpha * dW2$

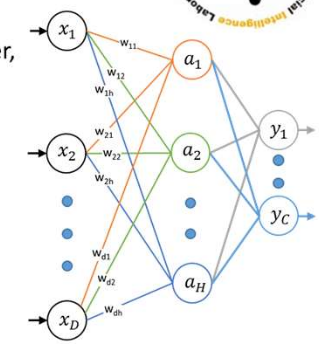


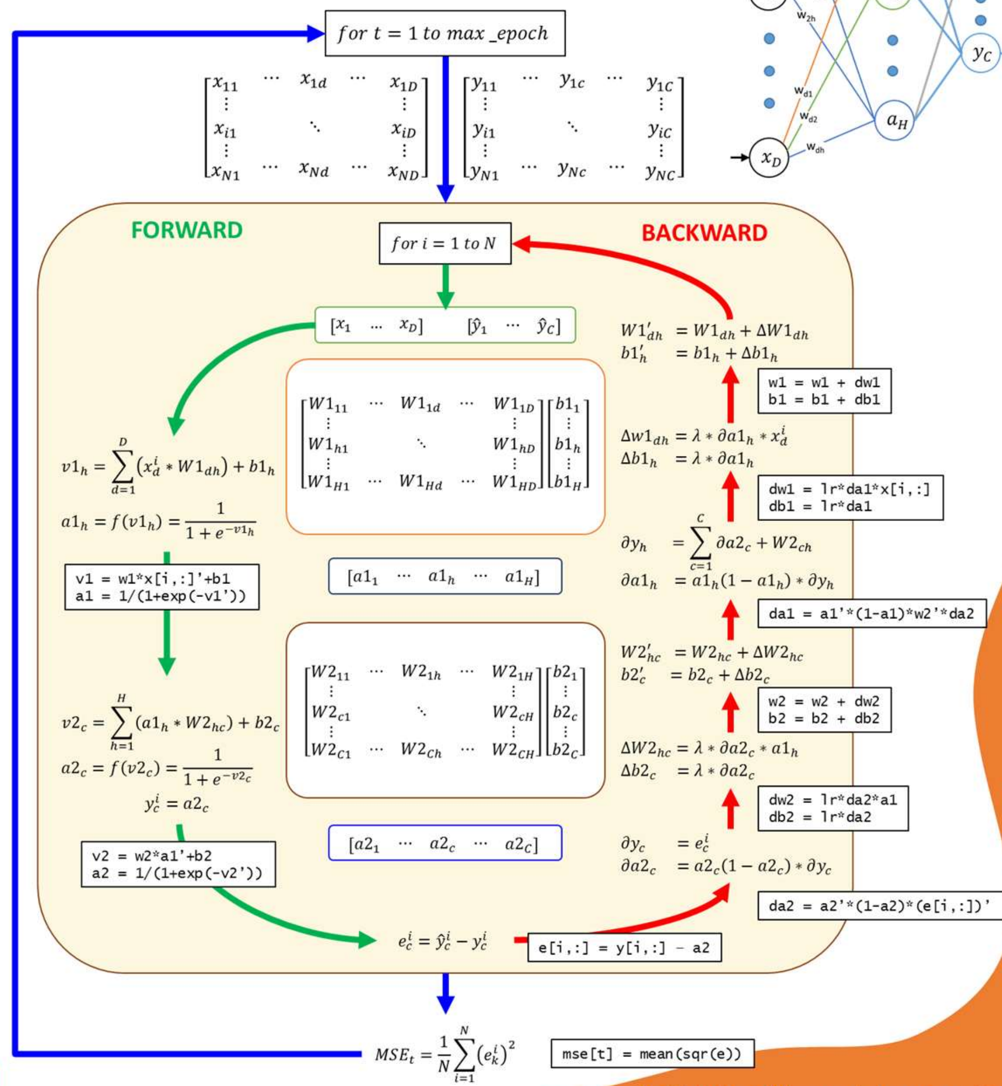
2-layer NN needs ~11 lines

```
import numpy as np

data = np.array([[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1]])
target = np.array([[0, 1, 1, 0]]).T

lr = 0.01
maxep = 60000
nparam = len(data[0])
nhid = 4
noutput = 1

W1 = 2 * np.random.rand(nparam, nhid) - 1
W2 = 2 * np.random.rand(nhid, noutput) - 1
for i in xrange(maxep):
    A1 = 1 / (1 + np.exp(-(np.dot(data, W1))))
    A2 = 1 / (1 + np.exp(-(np.dot(A1, W2))))
    error = target - A2
    D2 = error * (A2 * (1 - A2))
    D1 = D2.dot(W2.T) * (A1 * (1 - A1))
    W2 += A1.T.dot(D2) * lr
    W1 += data.T.dot(D1) * lr
```

Neural Network Simplified

Berikut adalah skema proses pembelajaran Jaringan Syaraf Tiruan dengan 1 *hidden layer* atau bisa disebut sebagai ***2-layer ANN***. Input dataset adalah **X** sebanyak **N** data yang memiliki **D** parameter, di mana setiap data memiliki target **Y** dengan **C** parameter output. Arsitektur yang dipakai *Fully Connected* dengan fungsi aktivasi *sigmoid* dan memiliki sebanyak **H** neuron di dalam *hidden layer*





API Building

Forward/Backward API

*rough pseudo code

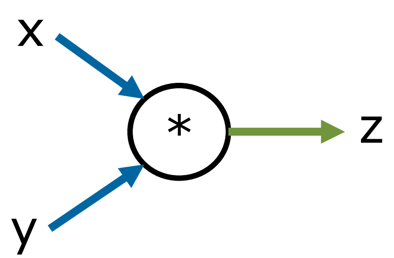
```
class net(object):
    # ...
    def forward(inputs):
        # 1. process inputs to input layer
        # 2. forward computational graph
        # ...
        for layer in layers:
            layer.forward()
        return loss

    def backward():
        for layer in reversed(layers):
            # backprop (chain rule applied)
            layer.backward()
            return gradients
```

Forward/Backward API

```
class MultiplyLayer(object):
    # ...
    def forward(x, y):
        z = x*y
        return z

    def backward(dz):
        dx = ??
        dy = ??
        return dx, dy
```

$\frac{\partial L}{\partial z}$ (loss)

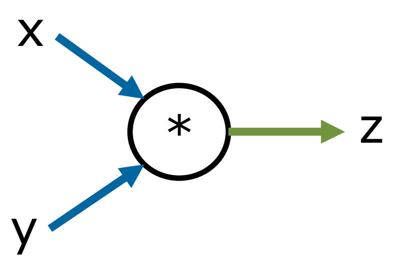
$\frac{\partial L}{\partial x}$ $\frac{\partial L}{\partial y}$ Gradient x and y



x, y, z are scalars

Forward/Backward API

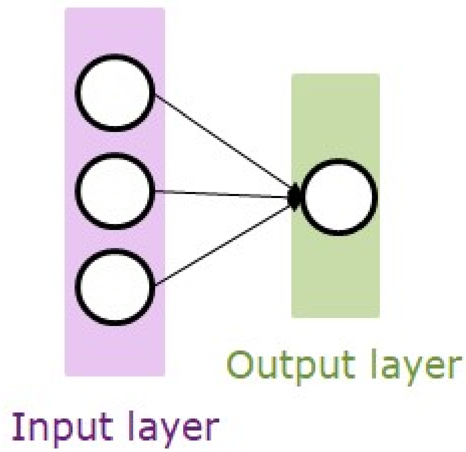
```
class MultiplyLayer(object):
    # ...
    def forward(x, y):
        z = x*y
        self.x = x # need to keep these
        self.y = y
        return z

    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return dx, dy
```



x, y, z are scalars

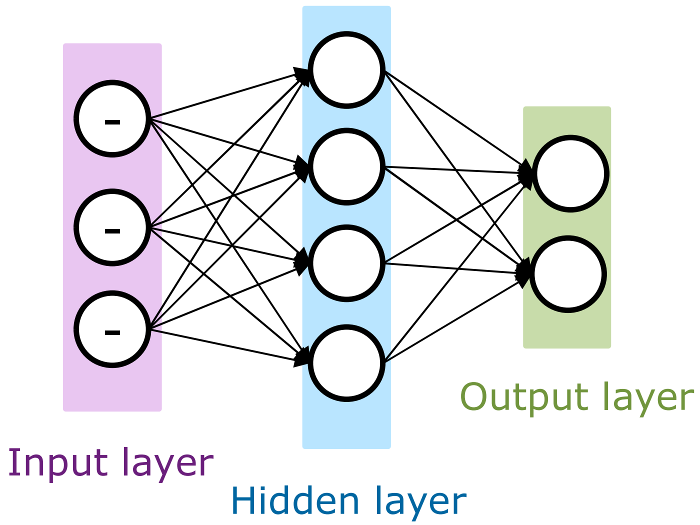
Complete API by storing intermediate gradient that needed in backward pass

Single Layer using API




```
# single layer
v1 = affine_forward(x, w, b)
a1 = sigmoid_forward(v1)

err = y-a1
print('mse =', np.mean(err**2))

dv1 = sigmoid_backward(err, a1)
dx, dw, db = affine_backward(dv1, x, w, b)
```

Multi Layer Perceptron




```
# 2 layers
v1 = affine_forward(x, w1, b1)
a1 = sigmoid_forward(v1)
v2 = affine_forward(a1, w2, b2)
a2 = sigmoid_forward(v2)

err = y-a2
print('mse =', np.mean(err**2))

dv2 = sigmoid_backward(err, a2)
da1, dw2, db2 = affine_backward(dv2, a1, w2, b2)
dv1 = sigmoid_backward(da1, a1)
dx, dw, db = affine_backward(dv1, x, w1, b1)
```

Summary so far

- Neural nets will be very large
  - Impractical to write down gradient formula by hand for all parameters
- Backpropagation
  - Recursive application of the chain rule along a computational graph to compute the gradients of all input/parameters/intermediates
- Implementations maintain a graph structure
  - forward() / backward() API

Question?


Fakultas Informatika
School of Computing
Telkom University

THANK YOU