CS 461: Machine Learning Principles

Class 16: Oct. 28

Back Propagation Algorithm &

Its Application to Deep Neural Net (DNN)

Instructor: Diana Kim

$$y = \underset{\sim}{\dot{w}}^{t} \cdot \phi(x)$$

Linear Modeling & Finding Global Minimum

So far, we have studied the functions for the task of regression and classification.
They were all linear functions $\underline{y = w^t x}$.
And algorithms to learn the functions from data.
(MMSE, Logistic MLE, and SVM dual)

$\cdot\ y = \underline{w^t} \cdot x$

$y = w_K^t \cdot x$

- Linear Regression: $y = w^t x$

g.t

$$W* = \arg\min_w J(W) = \arg\min_w ||y - \Phi w||^2$$

$$= \arg\min_w (y^t - w^t\Phi^t)(y - \Phi w)$$

$\sim \text{cov}(\tilde{y})$

$$= \arg\min_w \underbrace{||y||^2 - 2y^t\Phi w + w^t\boxed{\Phi^t\Phi} w}$$

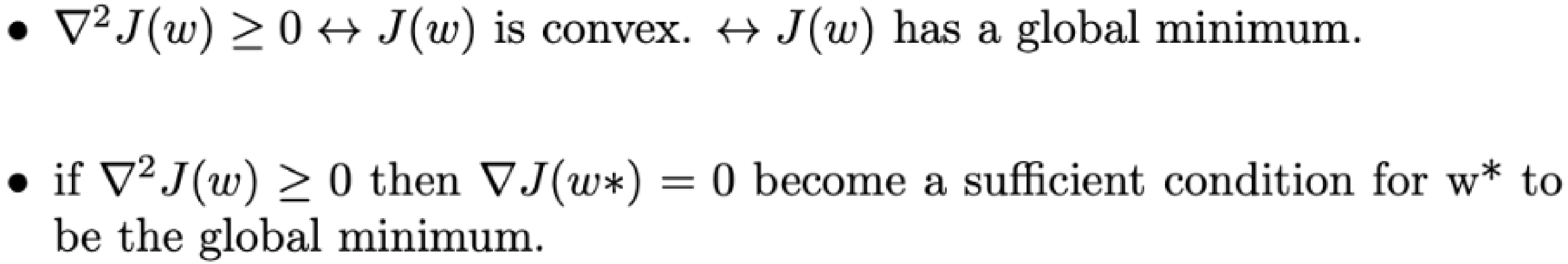
J(w) based data

$\cdot\ \nabla^2 \cdot J(w) \geq 0$

$\cdot\ \nabla J(\overset{*}{w}) = 0$

$\Phi^t \cdot \Phi \cdot w = \Phi^t \cdot y$

Hessian of Objective Function (Loss) & Convexity



- $\nabla^2 J(w) \geq 0 \leftrightarrow J(w)$ is convex. $\leftrightarrow J(w)$ has a global minimum.

- if $\nabla^2 J(w) \geq 0$ then $\nabla J(w*) = 0$ become a sufficient condition for w* to be the global minimum.

So far, we have studied the functions for the task of regression and classification.

They were all linear functions $y = w^t x$.

And algorithms to learn the functions from data.

(MMSE, Logistic MLE, and SVM dual)

$$\nabla^2 J(w) = \begin{bmatrix} \frac{\partial f(w_1, w_2)}{\partial w_1 \partial w_2} & \frac{\partial f(w}{\partial w_1 \partial w} \\ '' & \frac{\partial^2 f(w_1, w_2)}{\partial w_2^2} \end{bmatrix}$$

- Logistic Regression MLE:

$$P(t|w) = \prod_{n=1} \sigma(w^t x_n)^{t_n} (1 - \sigma(w^t x_n))^{1-t_n}$$

argmin $$J(\vec{w}) = -\ln P(t|w) = \sum_{n=1}^{N} -t_n \boxed{\ln \sigma(w^t x_n)} - (1 - t_n) \boxed{\ln (1 - \sigma(w^t x_n))}$$

They are convex.

[convexity]

$$-\frac{\partial \ln \sigma(w^t x)}{\partial w_i} = -\frac{\sigma(w^t x)(1 - \sigma(w^t x))}{\sigma(w^t x)} x_i$$

$$-\frac{\partial^2 \ln \sigma(w^t x)}{\partial w_i \partial w_j} = \underbrace{\overset{P}{\sigma(w^t x)}(1 - \sigma(w^t x))}_{} x_i x_j \quad \geq 0$$

At a data point x and current w

$$\nabla f(x^*) = 0$$

So far, we have studied the functions for the task of regression and classification.
They were all linear functions $y = w^t x$.
And algorithms to learn the functions from data.
(MMSE, Logistic MLE, and SVM dual)

- SVM dual:

$$\begin{bmatrix} t_1 & 0 & \dots & 0 \\ 0 & t_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & t_N \end{bmatrix} \cdot \Phi \cdot \Phi^t \cdot \begin{bmatrix} t_1 & 0 & \dots & 0 \\ 0 & t_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & t_N \end{bmatrix}$$

$$\lambda *_{n=1}^{N} = \arg\max_{\lambda *} \sum_{n=1}^{N} \lambda *_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n^* \lambda_m^* \cdot \boxed{t_n \cdot t_m \cdot \kappa(x_n, x_m)}$$

Semipositive Definite!

$$\text{subject to} \quad 0 \le \lambda_n^* \le C$$

$$\sum_{n=1}^{N} \lambda_n^* \cdot t_n = 0$$

$\underline{Df(C\,x^*)} =$

$-\frac{1}{2}[x \cdots][\;][\vdots]$

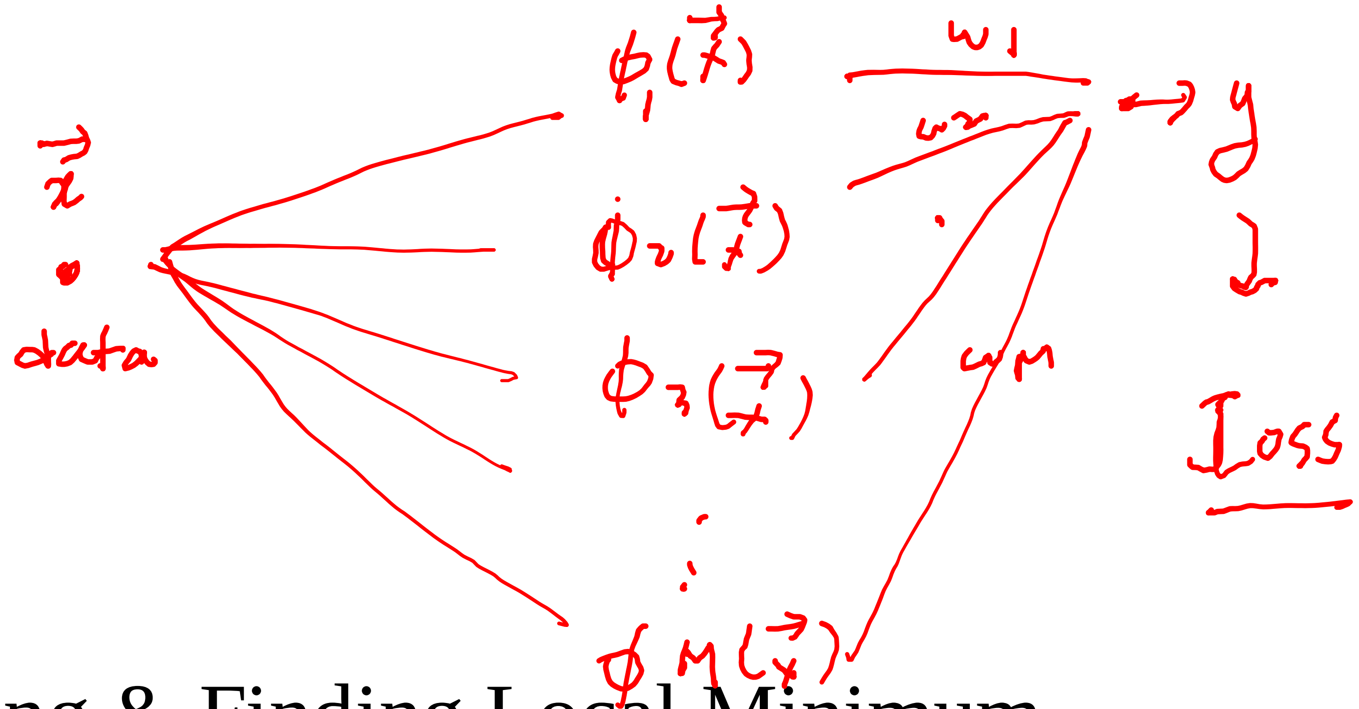
For all linear regression, logistic regression, SVM,

$\nabla J(w*) = 0$ is a sufficient condition for the global optimum $w *$.

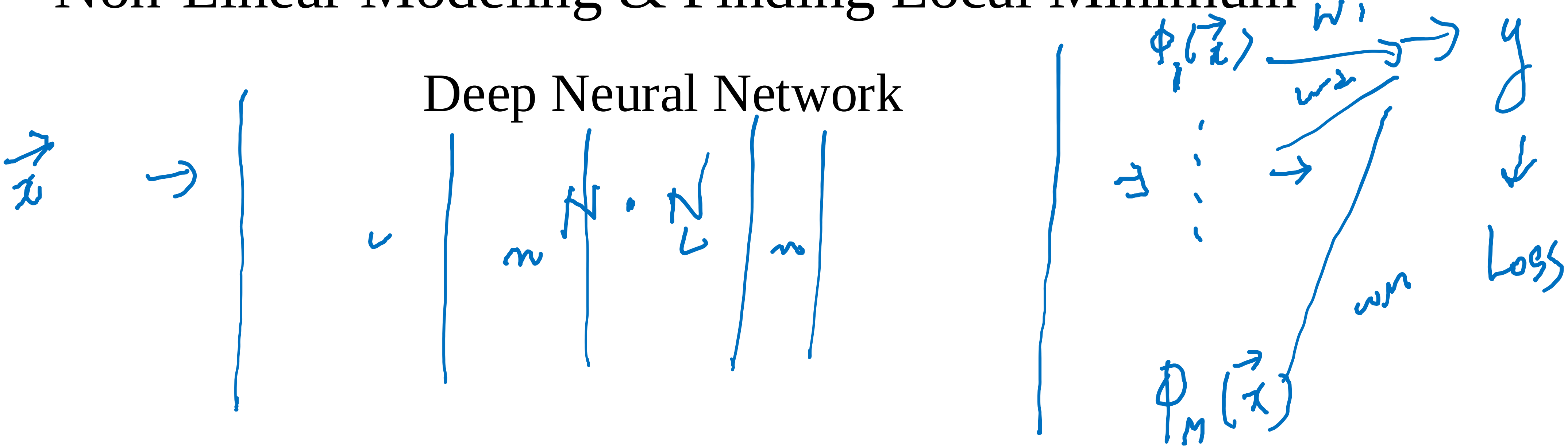
However, why the methods to find were different?

- Normal Equation,
- Stochastic Gradient Descent,
- SMO (Sequential Minimum Optimization)

Non-Linear Modeling & Finding Local Minimum

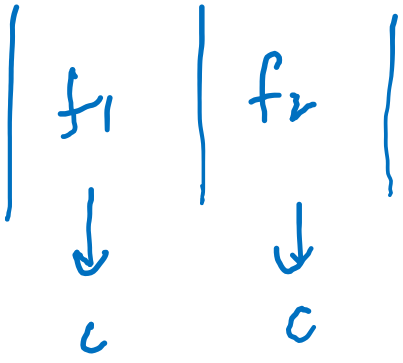


Deep Neural Network



Deep Neural Networks (DNNs) also learn the heuristic functions $y = f(x)$
for tasks such as classification and regression.
Common loss functions like Mean Squared Error (MSE), Maximum Likelihood Estimation (MLE) for Logistic Regression, and Softmax for multiclass classification are compatible with DNNs. However, the learned function is no longer linear; instead, the network automatically learns a complex feature space through its layers, up to the last hidden layer. The function is a non-linear.

For the non linearity and layered structure
the objective function for DNN is not convex any longer.
Why?



For the non linearity and layered structure,
the objective function for DNN is not convex any longer.
Why?

$$\frac{f_2(f_1(x))}{f}$$

+ activation functions often result in non-convex functions.
+ A composite function of convex functions is not always convex.

ex) $f(x) = \frac{1}{\sqrt{x}}$ $g(x) = \frac{1}{x}$ $g(\underbrace{f(x)}_{1/\sqrt{x}}) = ? = \sqrt{x}$

When $J(W)$ is non-convex

$\nabla \mathrm{f}(\mathrm{x}*) = 0$ is a necessary condition for $\mathrm{x}*$ to be a local minimum.

It has multiple local minimum. Hence, hard to find a global minimum, especially when input to the function is multidimensional.

$$\underline{J(w)} \downarrow \quad \boxed{w}$$

$$\downarrow$$

$$\nabla J(w^*) = 0$$

Neural Networks are usually trained
by using iterative and gradient based algorithms.

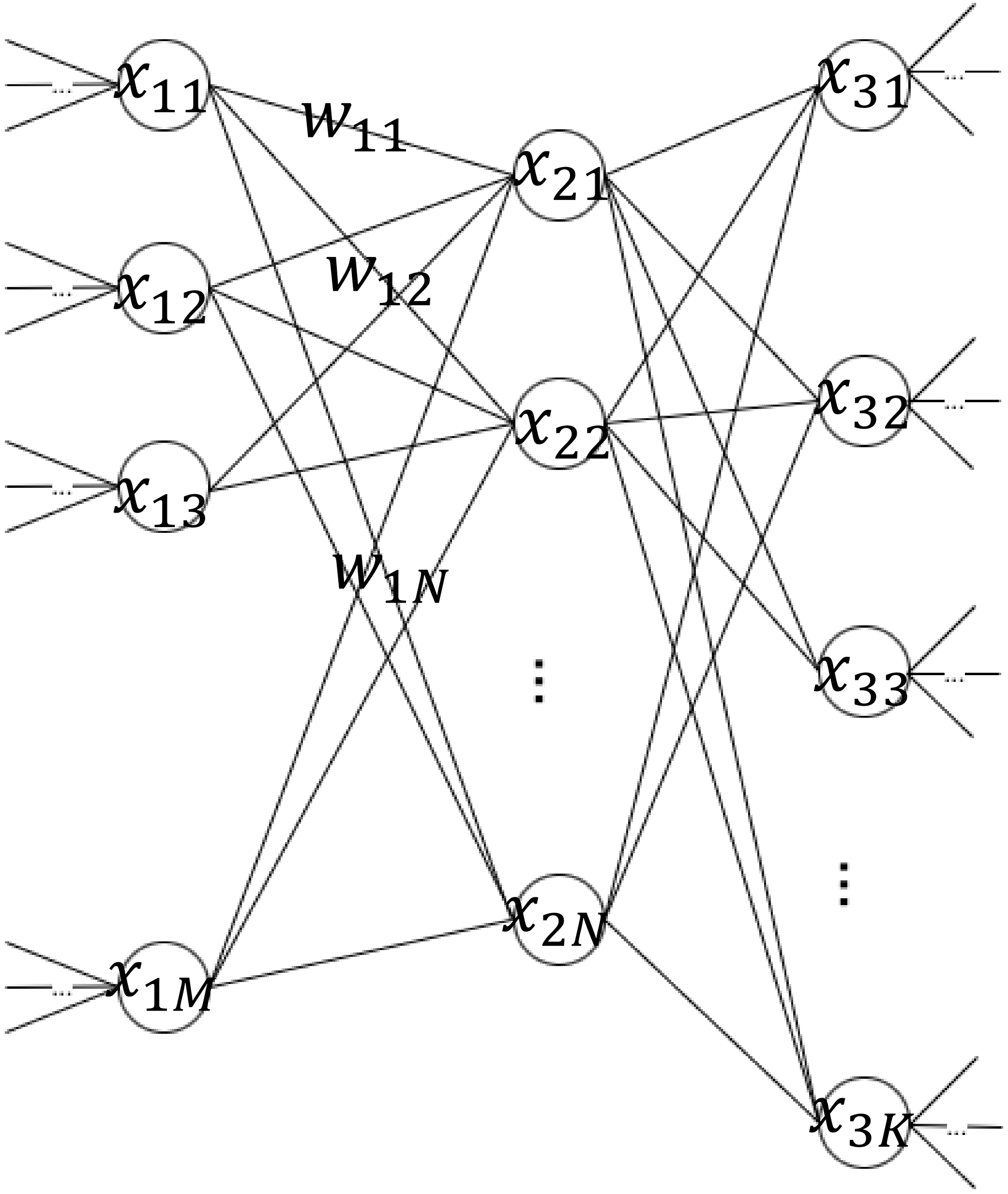
$$\underline{W_{t+1}} = W_t - \eta \boxed{\nabla J(W_t)}$$

"Stochastic Method"

- w_0
- $w_1 = w_0 - \underline{\eta}\, \underline{\nabla J(w_t)}$

How can we compute the gradient $\nabla J(W)$?

$$\nabla_w J(w) = \begin{bmatrix} \frac{\partial J(w)}{\partial w_1} \\ \frac{\partial J(w)}{\partial w_2} \\ \cdots \\ \frac{\partial J(w)}{\partial w_{M-1}} \\ \frac{\partial J(w)}{\partial w_M} \end{bmatrix}$$



# of parameters to learn

- AlexNet : 62.3M
- VGG-16: 138 M
- Chat-GPT2: 1.5 B

Computing Gradient $\nabla J(W)$

Backpropagation Algorithm

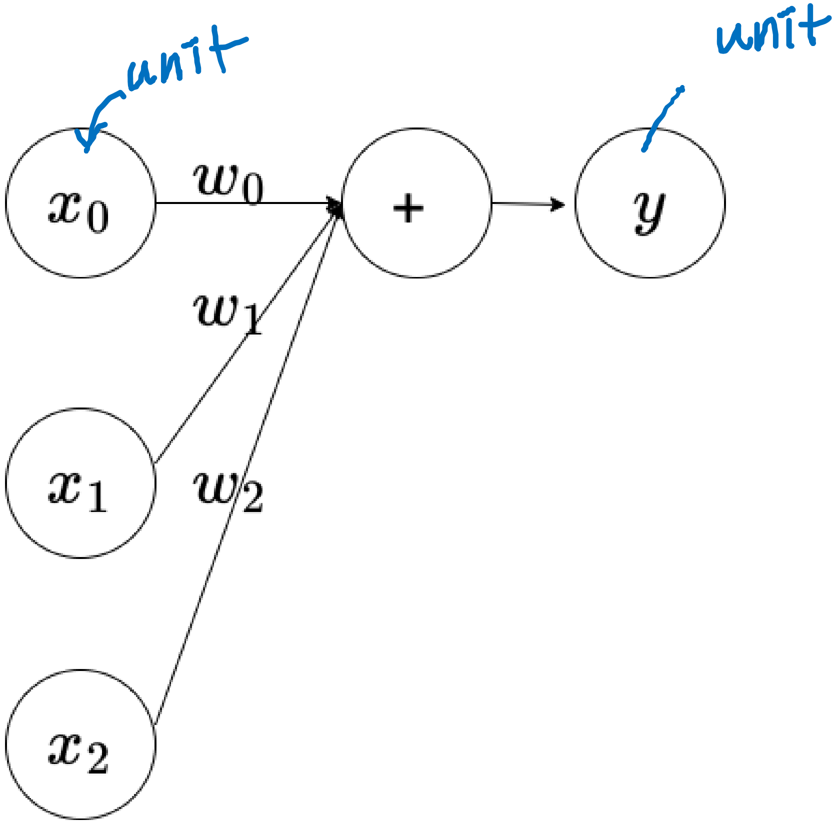
A function $f: \ X \to Y$

A function X from Y where X is the domain and Y is codomain.

$f(x) \ = \ y$, y is the image of x.

Ex) One Elementary Function of DNN

$$y = w^t x$$

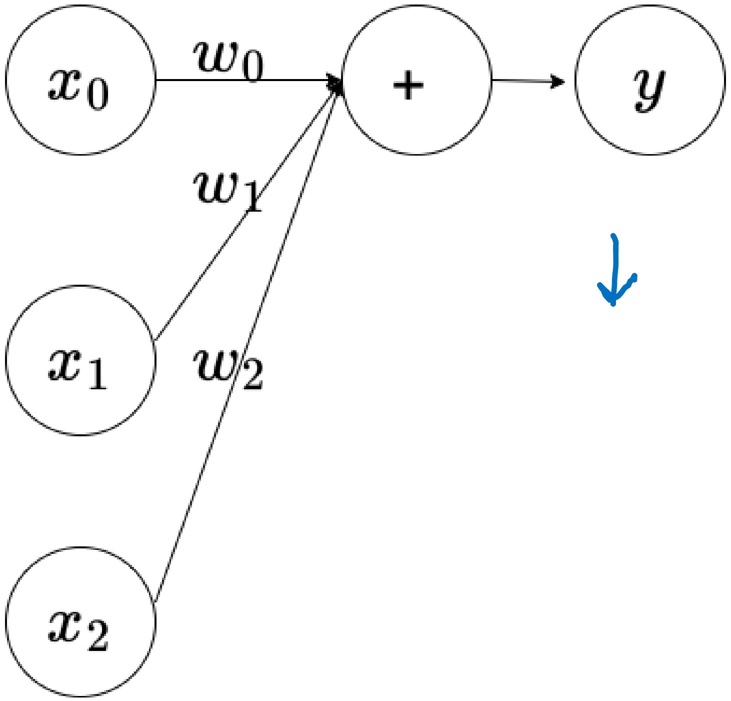


A function is called multivariable
if its input is made up of multiple members.

A function $f\colon\ X \subseteq \mathcal{R}^M \to Y$
A function X from Y, we say X is the domain and Y is codomain.
$f(x) \ = \ y$, y is the image of x
The range (or image) of f is the set of all images of A.

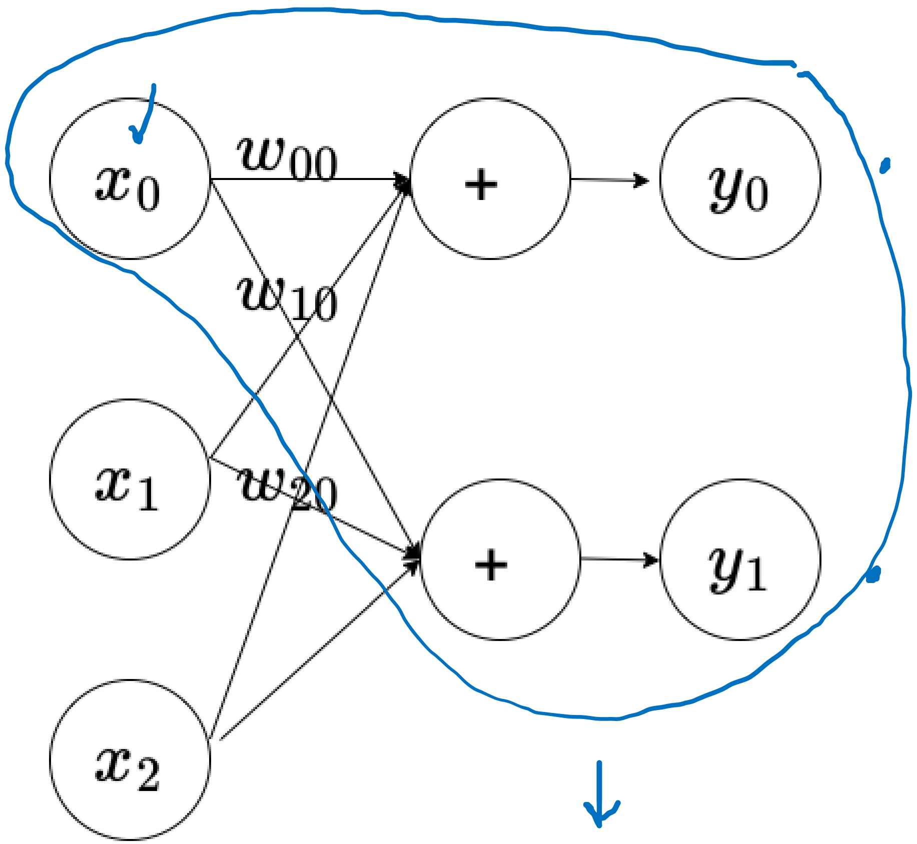
If the output function consists of multiple values,
it is also be called multivariable,
but these one are called vector valued functions.

A function f: $X \subseteq \mathcal{R} \rightarrow Y \subseteq \mathcal{R}^N$
A function X from Y, we say X is the domain and Y is codomain.
$f(x) \; = \; y$, y is the image of x
The range (or image) of f is the set of all images of A.



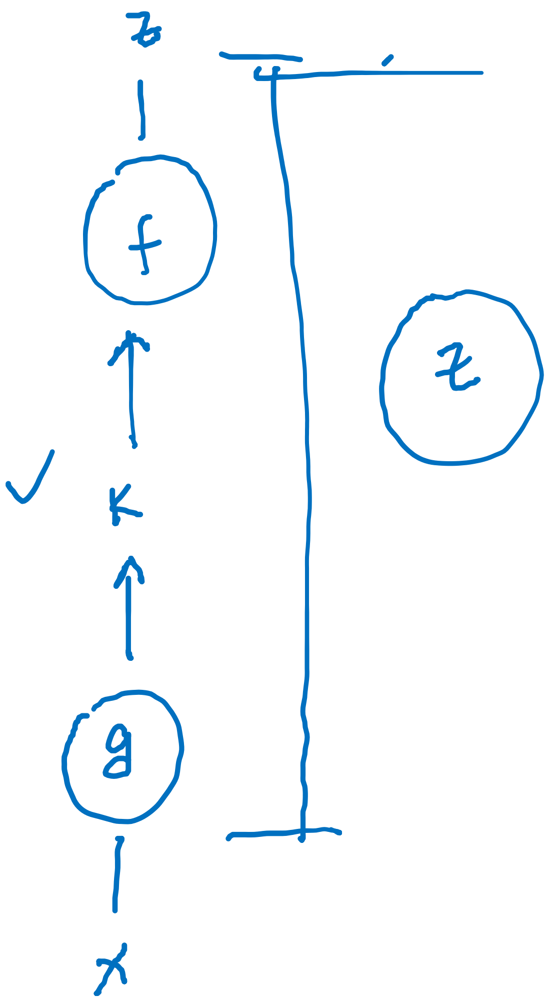
Multivariable Function

→



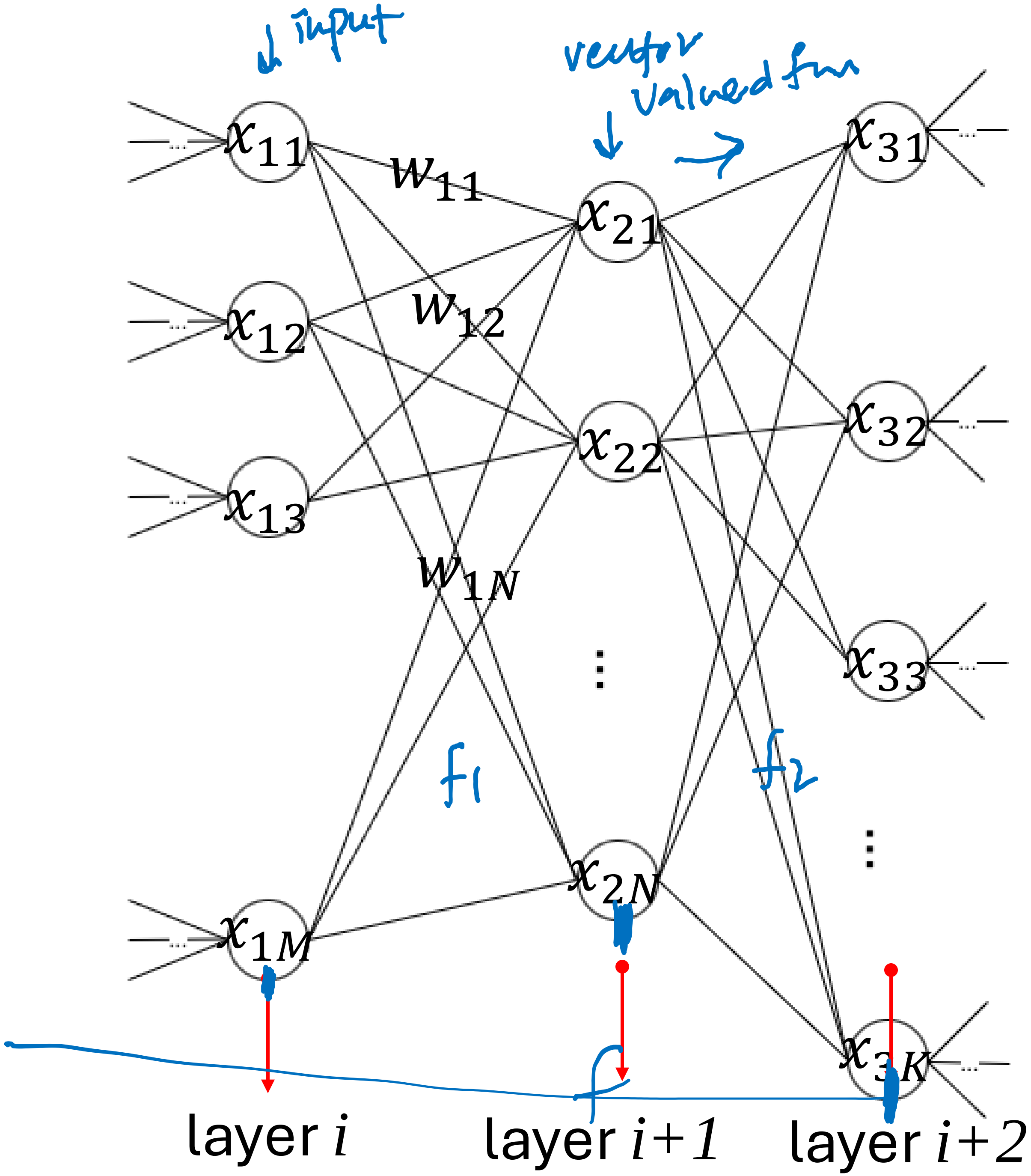
Multivariable and Vector Valued Function

A composite function $\overset{z}{\boxed{f \circ g}}: X \to Y$:
where $g\colon X \to Z$ and $f\colon Z \to Y$



$$f = f_2((f_1(x)))$$

- DNN is a large composite function consisting of many elementary Vector Valued Functions.
- Each vector-valued function becomes one layer.
- The chain structure gives the depth.



Derivative & Partial Derivative

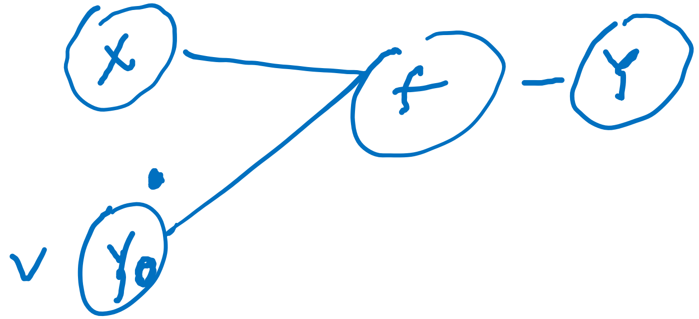
The derivative at point x_0 of a scalar function f :

$$f'(x_0) = \frac{df(x_0)}{dx} = \lim_{h \to 0} \frac{\overset{\Delta y}{f(x_0 + h) - f(x_0)}}{h \quad \underline{\Delta x}}$$

The partial derivative of a function with two variables $\overparen{f(x,y)}$ r. t x at point (x_0, y_0).

$$f_x(x_0, y_0) = \frac{\partial f}{\partial x}(x_0, y_0) = \lim_{h \to 0} \frac{f(x_0 + h, y_0) - f(x_0, y_0)}{h}$$

$f'(x)$



✓

$$\nabla_w J(w) = \begin{bmatrix} \frac{\partial J(w)}{\partial w_1} \\ \frac{\partial J(w)}{\partial w_2} \\ \cdots \\ \frac{\partial J(w)}{\partial w_{M-1}} \\ \frac{\partial J(w)}{\partial w_M} \end{bmatrix}$$ ✓

DNN's $J(w)$ represents a multivariate function, so we need to compute the partial derivatives.

The second or higher order derivatives can be defined.
Some gradient descent algorithm use the second order derivative information, but today we are going to focus on computing first order derivatives.

$$f''(x_0) = \frac{\mathrm{d}^2 f(x_0)}{\mathrm{d}x^2}$$

Scalar function

$$f_{xx}(x_0, y_0) = \frac{\partial^2 f(x_0, y_0)}{\partial x^2}$$

$$f_{xy}(x_0, y_0) = \frac{\partial^2 f(x_0, y_0)}{\partial x \partial y}$$

The function multiple variable

$g(x) = x$

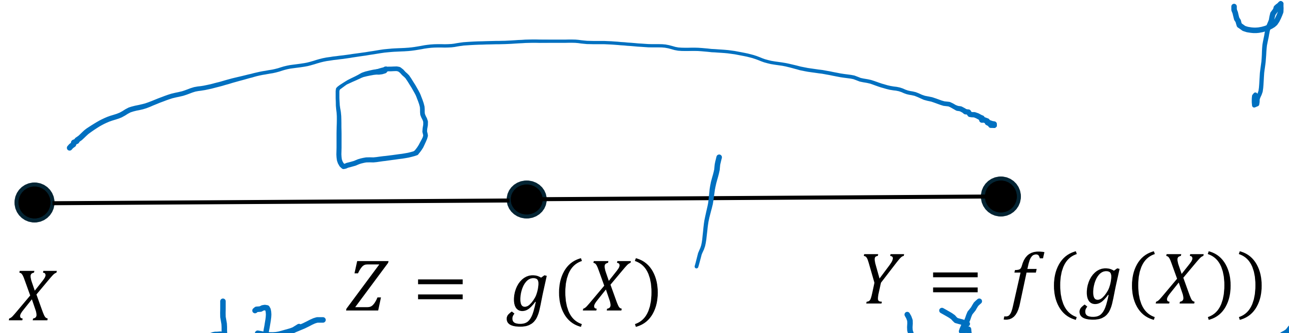
$f(z) = z^2$

$f(g(x)) = \boxed{x^2}$

Chain Rule

Computing the Derivative of Composite Functions

$Y = \underline{w}(x)$



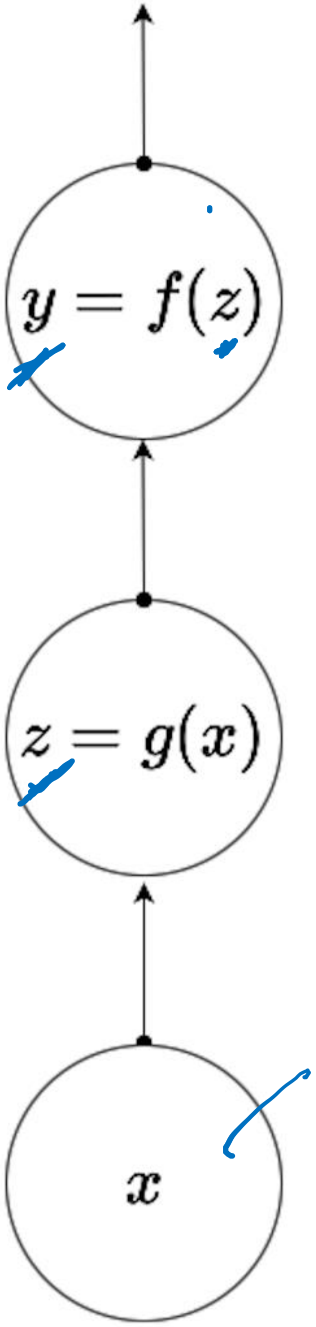
X $\quad Z = g(X) \quad Y = f(g(X))$

$\frac{d\cancel{Z}}{dX} \cdot \frac{dY}{d\cancel{Z}} = \frac{dY}{dX}$

Chain Rule for Scalar Functions

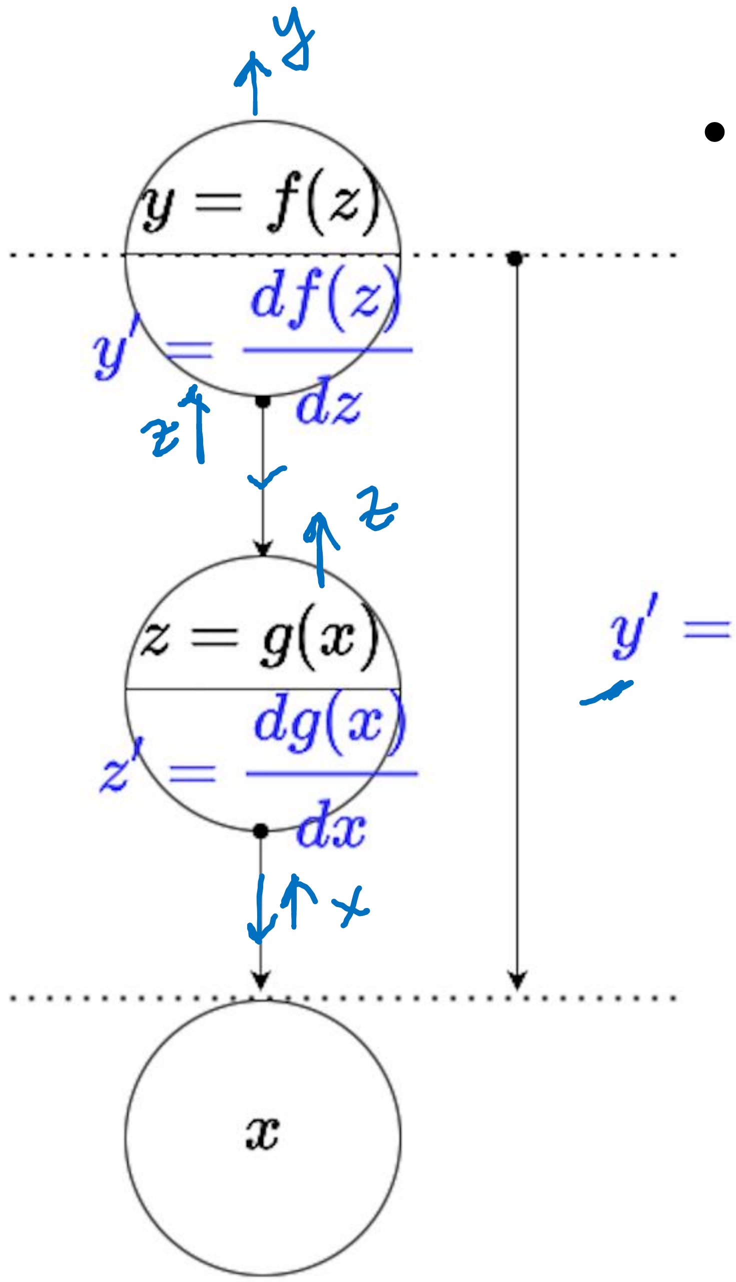
Let $z = g(\overset{\checkmark}{x})$ and $y = f(\overset{\checkmark}{z})$,

Then $$\frac{d(f\circ g\ (x))}{dx} = \boxed{\overset{\cdot\ F}{\frac{dy}{dx}}} = \overset{\cdot}{\underset{\checkmark}{\frac{dz}{dx}}} \cdot \overset{\cdot}{\underset{\checkmark}{\frac{dy}{dz}}}$$



- The feedforward direction
  to compute a composite function $f \circ g$

unit .



- The backward direction to compute the derivate of composite function $\frac{df\circ g\,(x)}{dx}$

def backward(·)

inpu

out = $\boxed{\text{input}}\ \boxed{\ }$

mydev

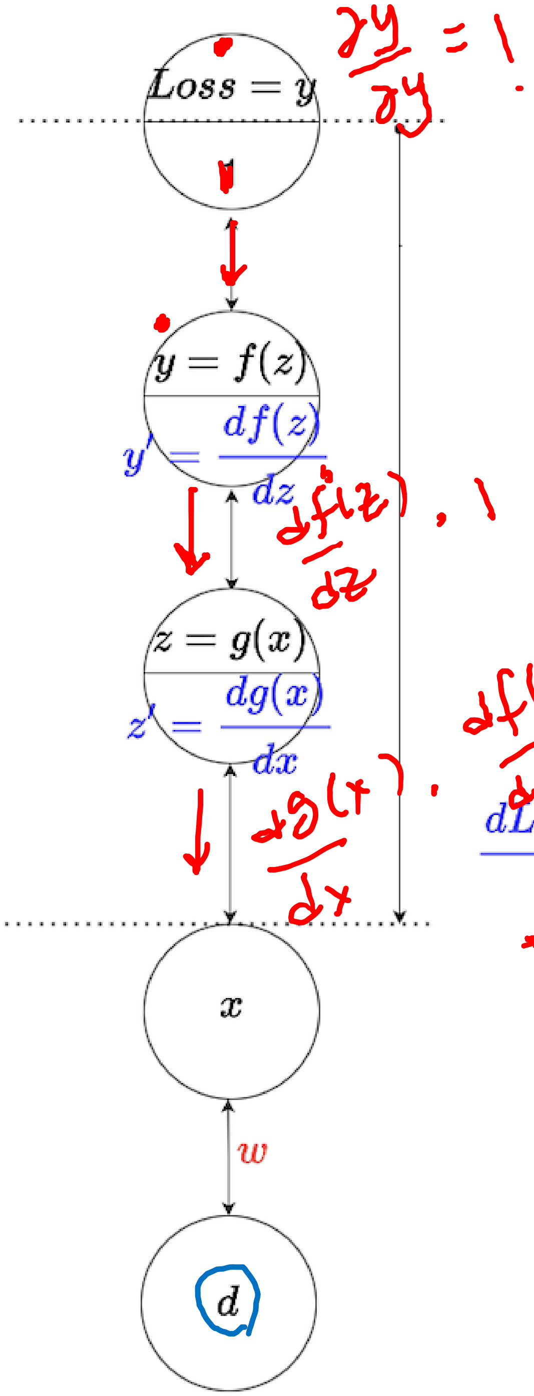
$$y' = \frac{df(g(x))}{dx} = \underbrace{\underbrace{\frac{df(z)}{dz}}_{\text{input}} \cdot \frac{dg(x)}{dx}}_{\text{ouput}}$$

$$\frac{d\,Loss}{dw} = \frac{d\,Loss}{dx} \cdot \left(\frac{dx}{dw}\right)$$

d

- Computing Gradients in a Neural Net:

Q: How can we compute the $\frac{dLoss}{dw}$ by using $\frac{dLoss}{dx}$?



$\frac{df(z)}{dz} \cdot 1$

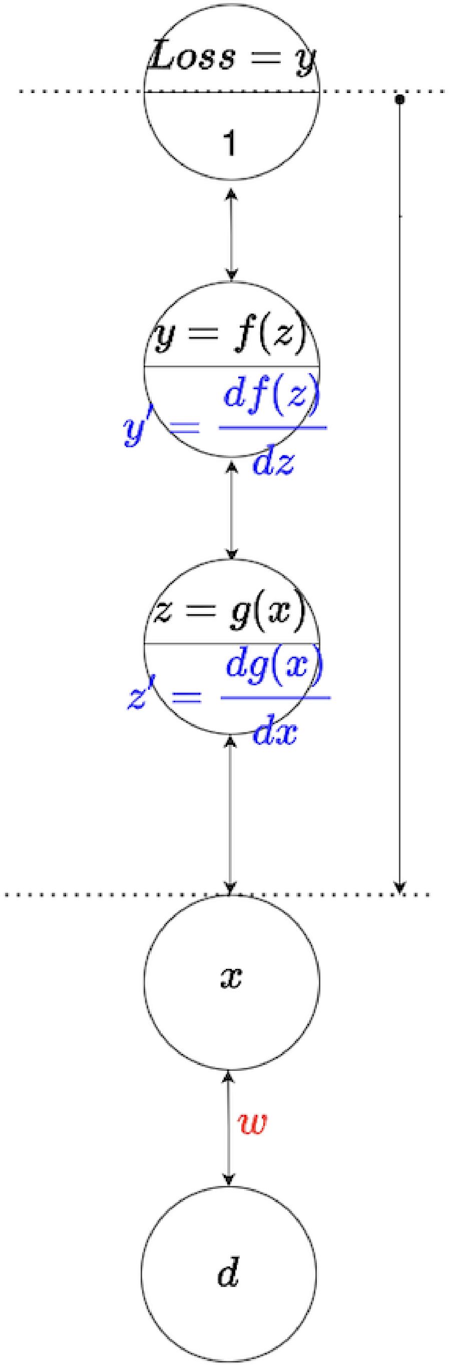
$$\frac{dLoss(x)}{dx} = 1 \cdot \frac{df(z)}{dz} \cdot \frac{dg(x)}{dx}$$

$x = w \cdot d$

$\frac{dx}{dw} = d$

$$\frac{d\,Loss}{d\cancel{x}} = \frac{d\,Loss}{dW} \cdot \boxed{\frac{d\cancel{w}}{dx}}$$

dw



- Computing Gradients in a Neural Net:

$$\frac{dLoss}{dw} = \frac{dLoss}{dx} \cdot \frac{dx}{dw} = \frac{dLoss}{dx} \cdot d$$

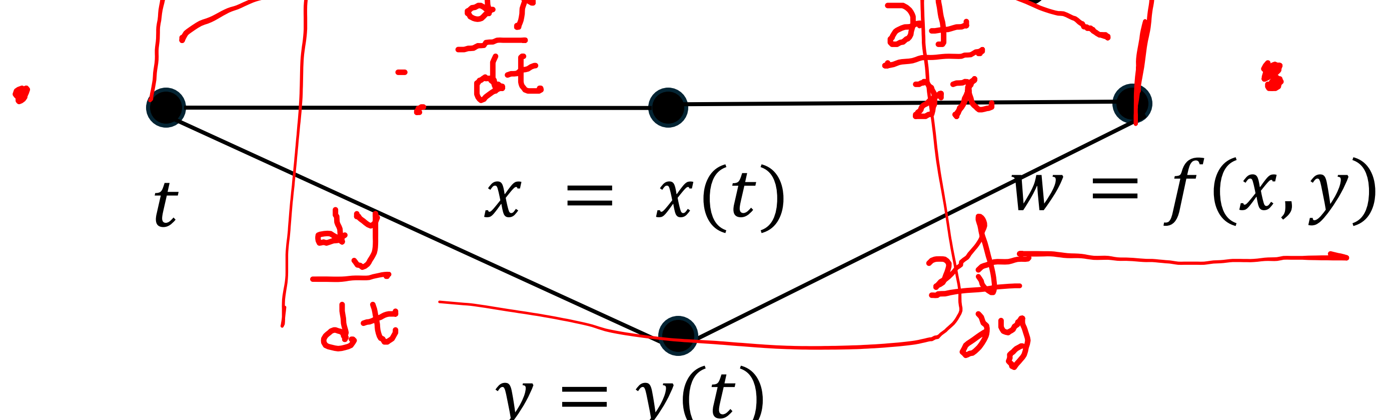
$$\frac{dLoss(x)}{dx} = 1 \cdot \frac{df(z)}{dz} \cdot \frac{dg(x)}{dx}$$ ✓

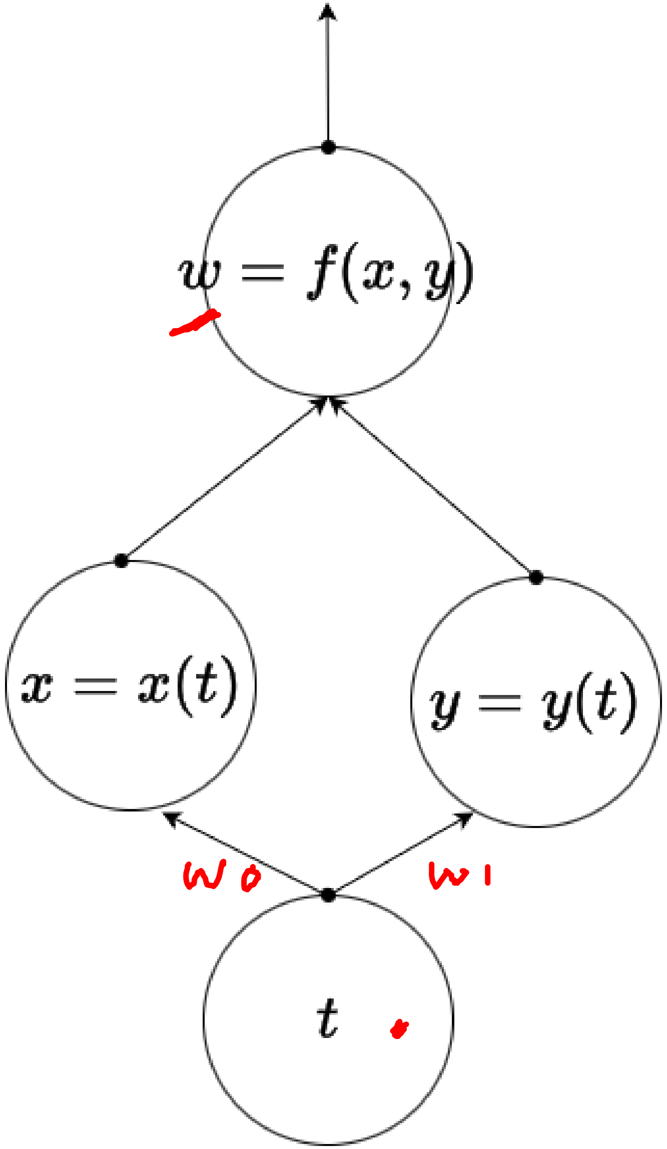
Chain Rule for Multivariable Functions

$$\frac{dw}{dt} = \frac{dx}{dt}\cdot\frac{\partial f}{\partial x} + \frac{dy}{dt}\cdot\frac{\partial f}{\partial y}$$

Let $w = f(x,y)$, $where\ x = x(t)\ and\ y = y(t)$

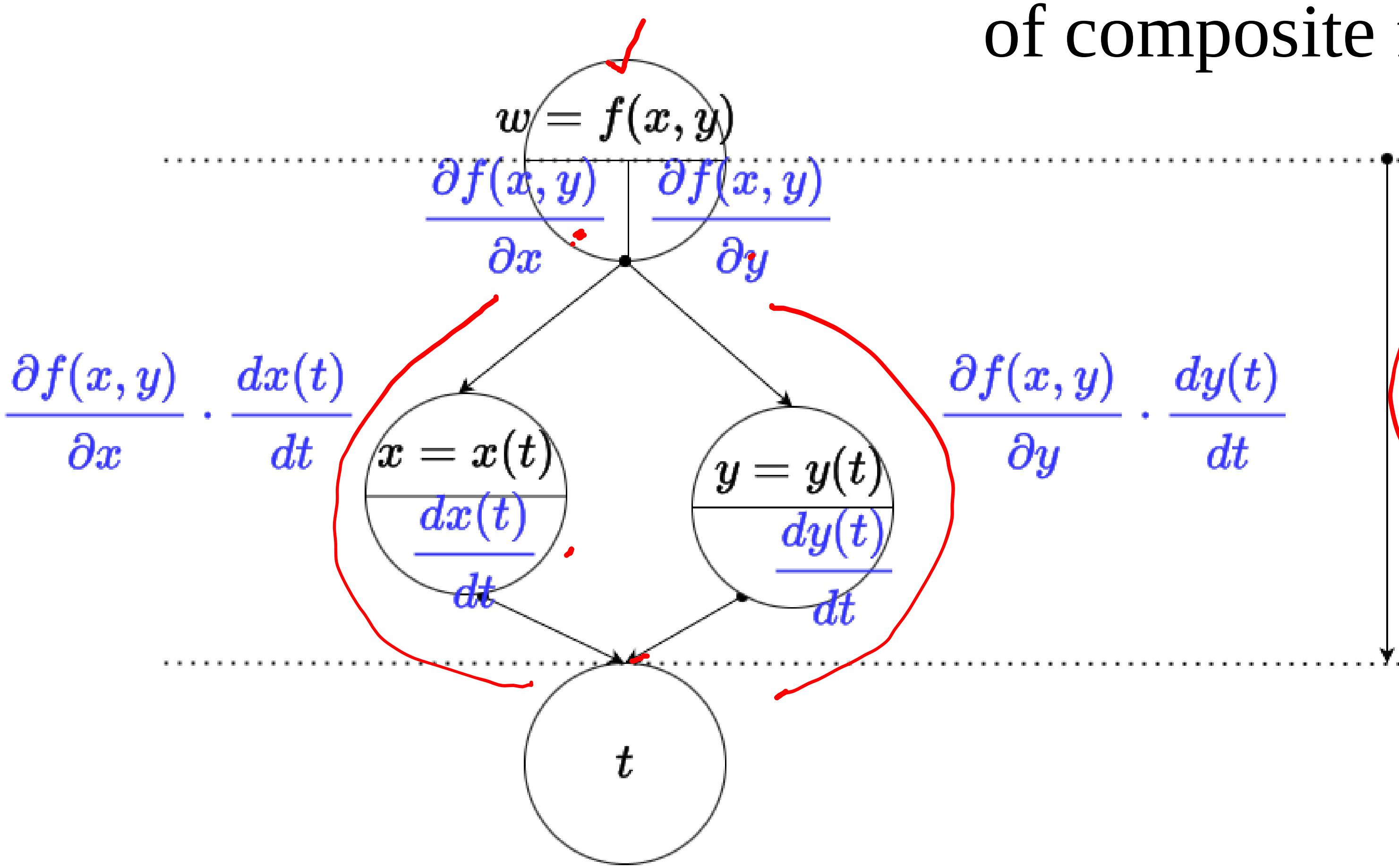
Then $$\frac{df(x,y)}{dt} = \frac{dw}{dt} = \frac{\partial w}{\partial x}\cdot\frac{dx}{dt} + \frac{\partial w}{\partial y}\cdot\frac{dy}{dt}$$



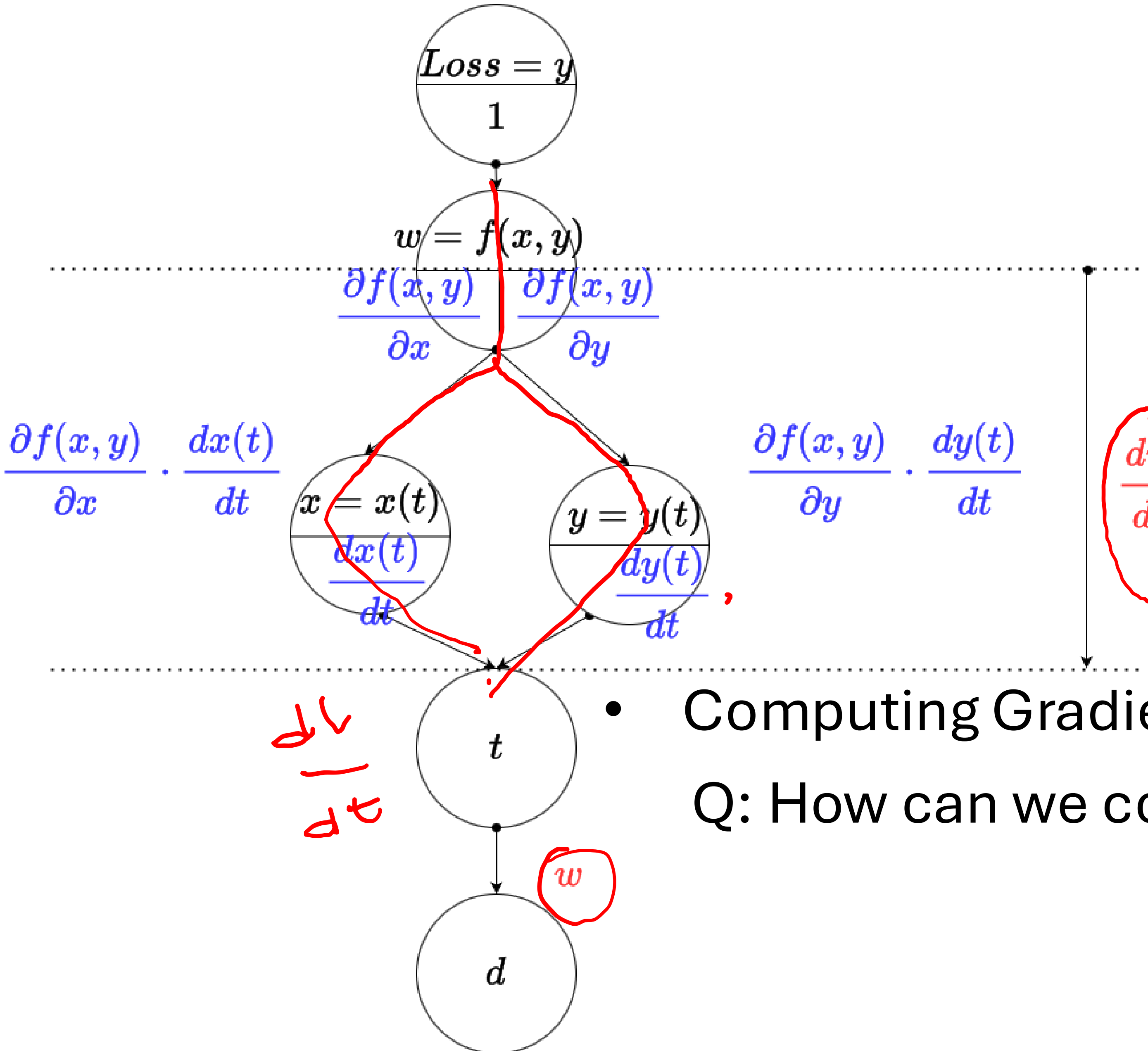


- The feedforward direction to compute a composite function $w(t) = f(x(t), y(t))$

- The backward direction to compute the derivate of composite function $\frac{dw}{dt}$


$w = f(x,y)$
$\frac{\partial f(x,y)}{\partial x}$
$\frac{\partial f(x,y)}{\partial y}$
$\frac{\partial f(x,y)}{\partial x} \cdot \frac{dx(t)}{dt}$
$x = x(t)$
$\frac{dx(t)}{dt}$
$y = y(t)$
$\frac{dy(t)}{dt}$
$\frac{\partial f(x,y)}{\partial y} \cdot \frac{dy(t)}{dt}$
t

$$\frac{dw}{dt} = \frac{\partial f(x,y)}{\partial x} \cdot \frac{dx(t)}{dt} + \frac{\partial f(x,y)}{\partial y} \cdot \frac{dy(t)}{dt}$$


$Loss = y$
1
$w = f(x,y)$
$\frac{\partial f(x,y)}{\partial x}$
$\frac{\partial f(x,y)}{\partial y}$
$\frac{\partial f(x,y)}{\partial x} \cdot \frac{dx(t)}{dt}$
$x = x(t)$
$\frac{dx(t)}{dt}$
$y = y(t)$
$\frac{dy(t)}{dt}$
$\frac{\partial f(x,y)}{\partial y} \cdot \frac{dy(t)}{dt}$
t
$\frac{dL}{dt}$
w
d

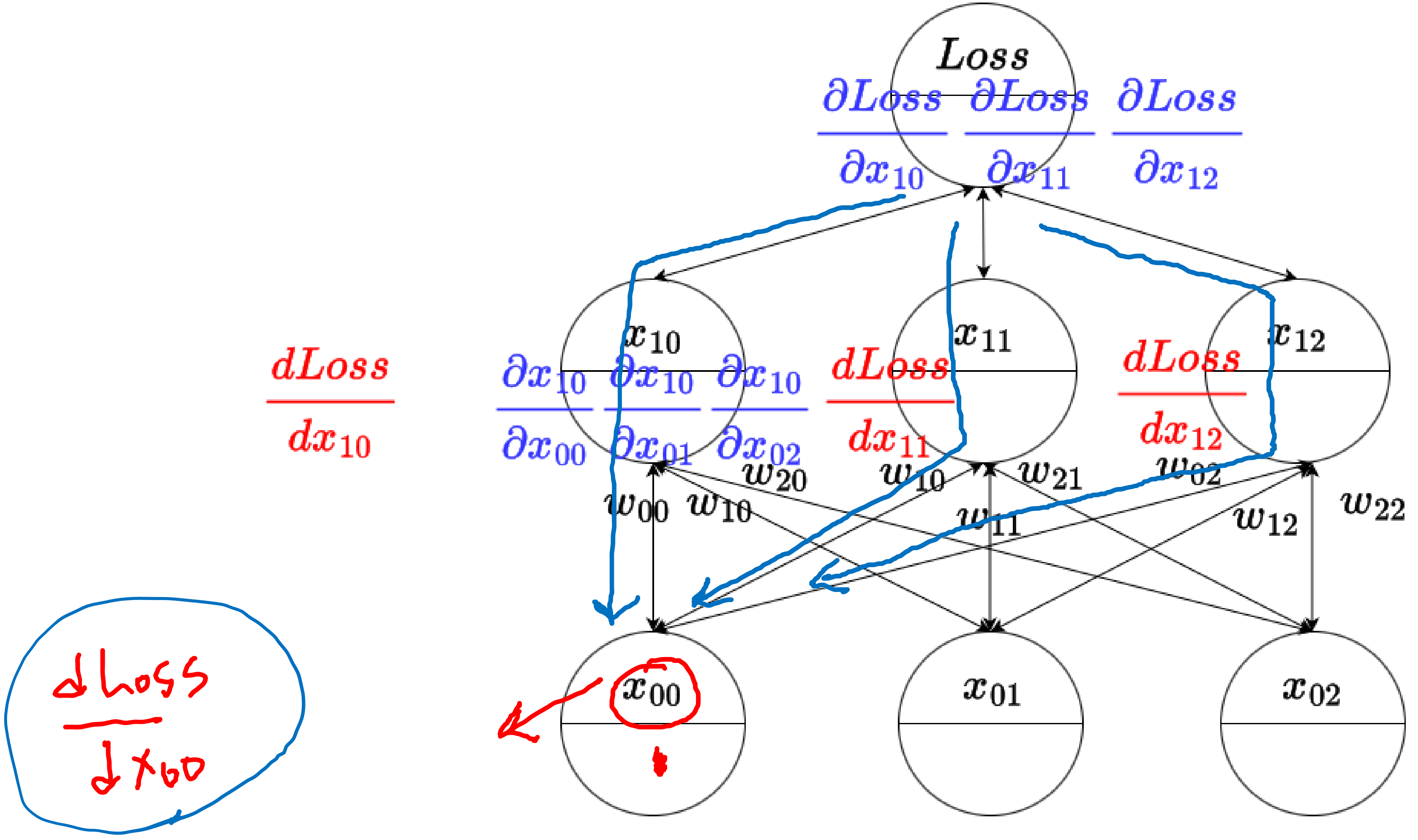
$$\frac{d\overset{Loss}{\cancel{w}}}{dt} = 1 \cdot \underbrace{\frac{\partial f(x,y)}{\partial x} \cdot \frac{dx(t)}{dt}}_{\textcircled{1}} + 1 \cdot \underbrace{\frac{\partial f(x,y)}{\partial y} \cdot \frac{dy(t)}{dt}}_{\textcircled{2}}$$

- Computing Gradients in a Neural Net:

Q: How can we compute the $\frac{dLoss}{dw}$ by using $\frac{dLoss}{dt}$?

At a Certain W, $[W_{t+1} = W_t - \eta \nabla J(W_t)]$

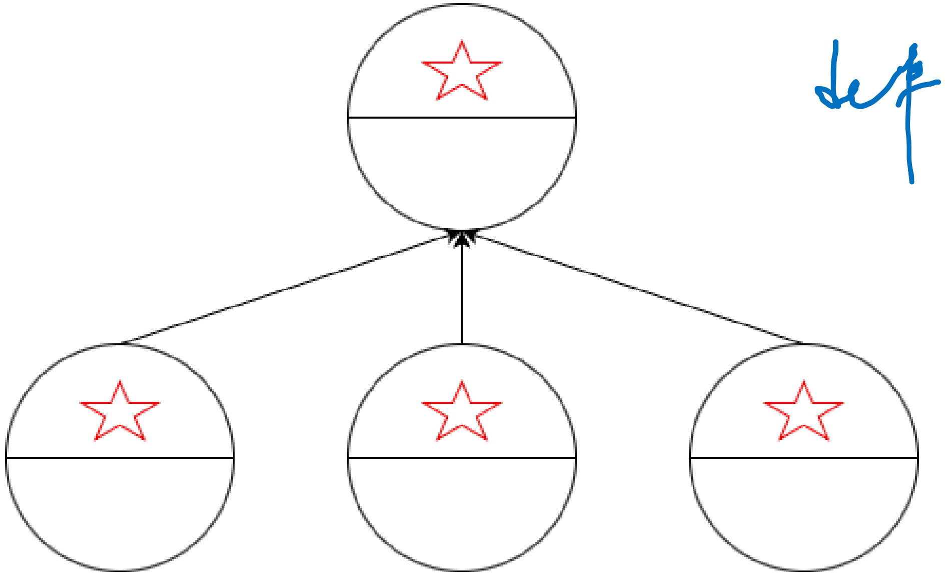
The Derivative of Loss at Each Node is Updated through Backpropagation.



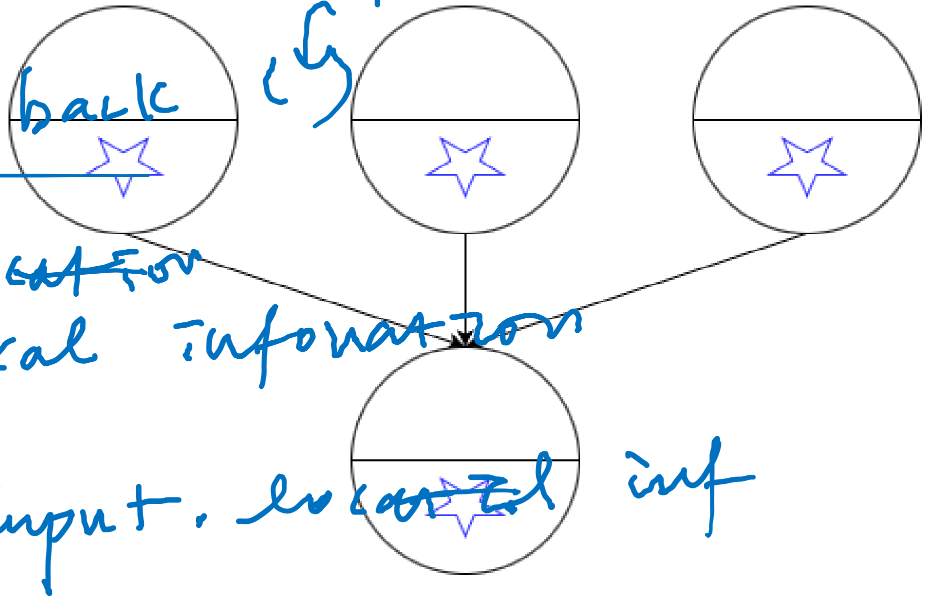
$Loss$
$\frac{\partial Loss}{\partial x_{10}}$ $\frac{\partial Loss}{\partial x_{11}}$ $\frac{\partial Loss}{\partial x_{12}}$
x_{10}
x_{11}
x_{12}
$\frac{dLoss}{dx_{10}}$
$\frac{\partial x_{10}}{\partial x_{00}}$ $\frac{\partial x_{10}}{\partial x_{01}}$ $\frac{\partial x_{10}}{\partial x_{02}}$
$\frac{dLoss}{dx_{11}}$
$\frac{dLoss}{dx_{12}}$
w_{00}
w_{10}
w_{20}
w_{10}
w_{11}
w_{21}
w_{02}
w_{12}
w_{22}
x_{00}
x_{01}
x_{02}
$\frac{dLoss}{dx_{00}}$

$$\frac{dLoss}{dx_{00}} = \frac{dx_{10}}{dx_{00}}\frac{\partial Loss}{\partial x_{10}} + \frac{dx_{11}}{dx_{00}}\frac{\partial Loss}{\partial x_{11}} + \frac{dx_{12}}{dx_{00}}\frac{\partial Loss}{\partial x_{12}}$$

```
class unit ( ) { memory,
  def feedforward ( )
    operation ( input , output )
  def feedback ( )
    • ~~location~~
      cal information
    • input , ~~local~~ inf
        ⇓
    • out
```



Feedforward Update
(For Inference)

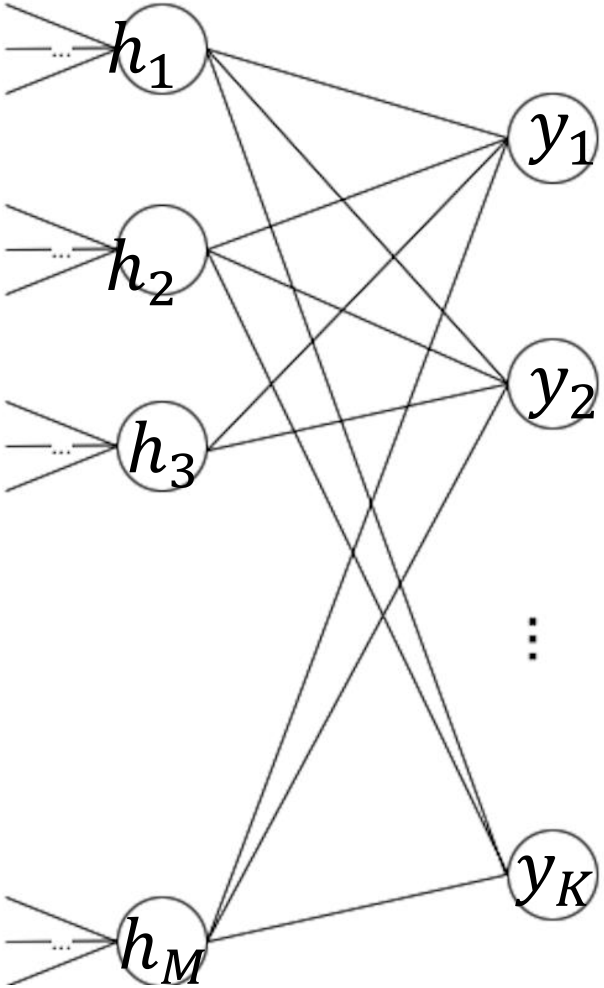


Feedbackward Update
(For Computing Gradient)

Output Layers of DNN and Gradient Descent

[1] MMSE Loss for Regression

[1] MMSE Loss for Regression



In regression modeling,

(1) $\vec{y} = f(\vec{x}) + \varepsilon$ where $\varepsilon \sim \mathcal{N}(0, \Sigma)$

$$then\ y \sim \mathcal{N}(f(x), \Sigma)$$

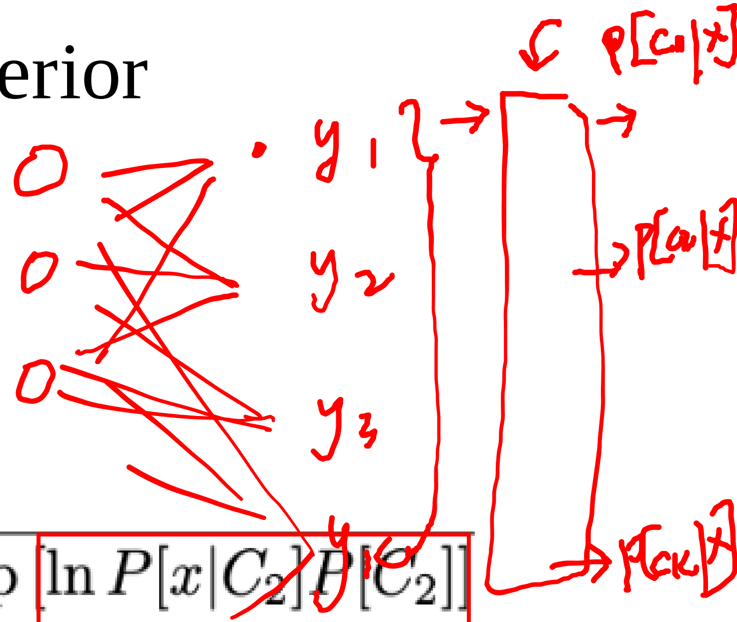
(2) For MLE,

Take a NLL (negative log likelihood)

[2] Softmax for Multiple Classification

K-class → # K discriminant

SoftMax Representation of Multiclass Posterior



$$P[C_0|x] = \frac{P[x|C_0]P[C_0]}{P[x|C_0]P[C_0] + P[x|C_1]P[C_1] + P[x|C_2]P[C_2]}$$

$$P[C_0|x] = \frac{\exp[\ln P[x|C_0]P[C_0]]}{\exp\boxed{[\ln P[x|C_0]P[C_0]]} + \exp\boxed{[\ln P[x|C_1]P[C_1]]} + \exp\boxed{[\ln P[x|C_2]P[C_2]]}}$$

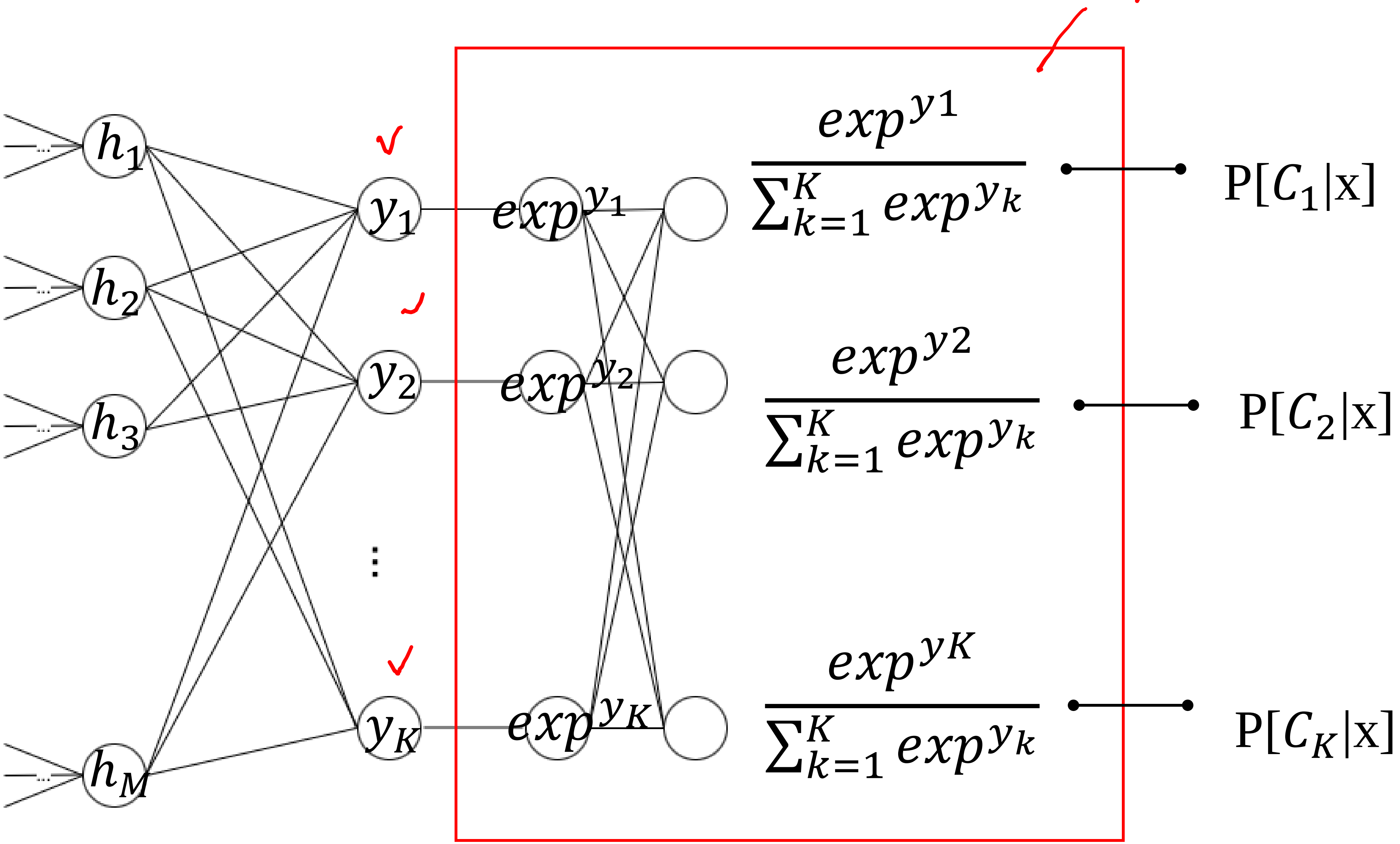
+ K= 3 discriminant functions

The posterior will be designed with softmax & $K = 3$ linear function of x

& $K = 3$ linear function of $\phi(x)$

[2] SoftMax Ouput for Multiclass Classification

Softmax

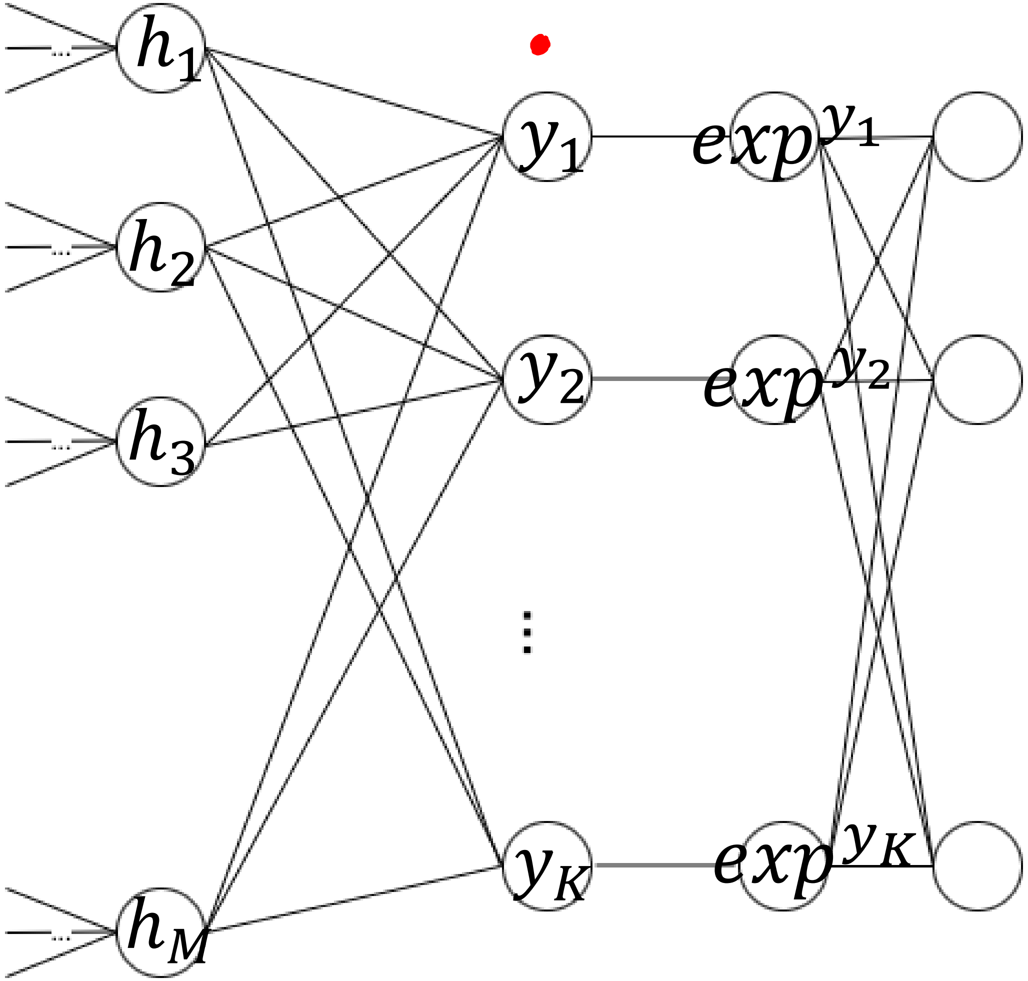


$$\frac{exp^{y1}}{\sum_{k=1}^{K} exp^{y_k}}$$ P$[C_1|$x]

$$\frac{exp^{y2}}{\sum_{k=1}^{K} exp^{y_k}}$$ P$[C_2|$x]

$$\frac{exp^{yK}}{\sum_{k=1}^{K} exp^{y_k}}$$ P$[C_K|$x]

[3] SoftMax Unit and MLE Loss



$$P[T|W,x] = \prod_{n=1}^{N}\prod_{k=1}^{K} P[C_k|W,x_n]^{t_{nk}}$$

$$J(W) = -\ln P[T|W,x] = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\ln P[C_k|W,x_n]$$

Once we know $\frac{dJ}{dy_j}$,

we can compute the gradients of other units by backpropagation.

Sofmax Derivative r.t y_j

$$P[T|W,x] = \prod_{n=1}^{N} \prod_{k=1}^{K} P[C_k|W,x_n]^{t_{nk}}$$

$$J(W) = -\ln P[T|W,x] = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln P[C_k|W,x_n]$$

$$\frac{\partial J}{\partial y_j} = \sum_{n=1}^{N} \{t_{n1} P[C_j|x_n,W] + ... + t_{nj}(P[C_j|x_n,W] - 1) + t_{nK} P[C_j|x_n,W]\}$$

$$\frac{\partial J}{\partial y_j} = \sum_{n=1}^{N} \{P[C_j|x_n,W] - t_{nj}\}$$

SoftMax Derivative [Slide Oct. 7th]

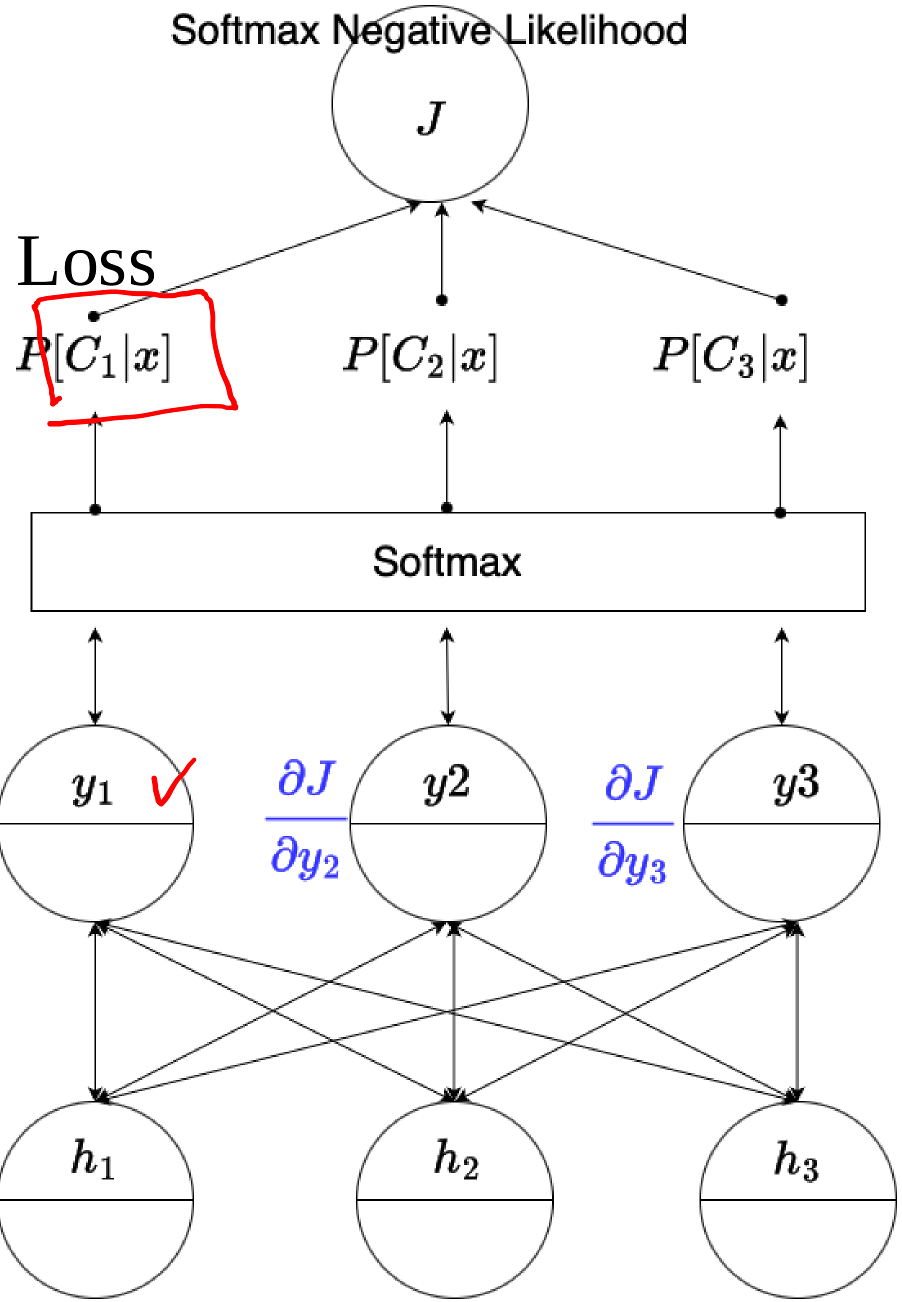
✓

$$\nabla_{W_j} P[C_j|x] = P[C_j|x] \cdot (1 - P[C_j|x])\phi(x)$$
$$\nabla_{W_i} P[C_j|x] = P[C_j|x] \cdot (-P[C_i|x])\phi(x)$$

+ we can compute the P[Cj|x] at current W!

The derivatives of Softmax Negative Likelihood Loss
: they backpropagated throughout the network
to compute the gradients of the network units.
The parameters connected to the units
are updated based on the updated gradients.

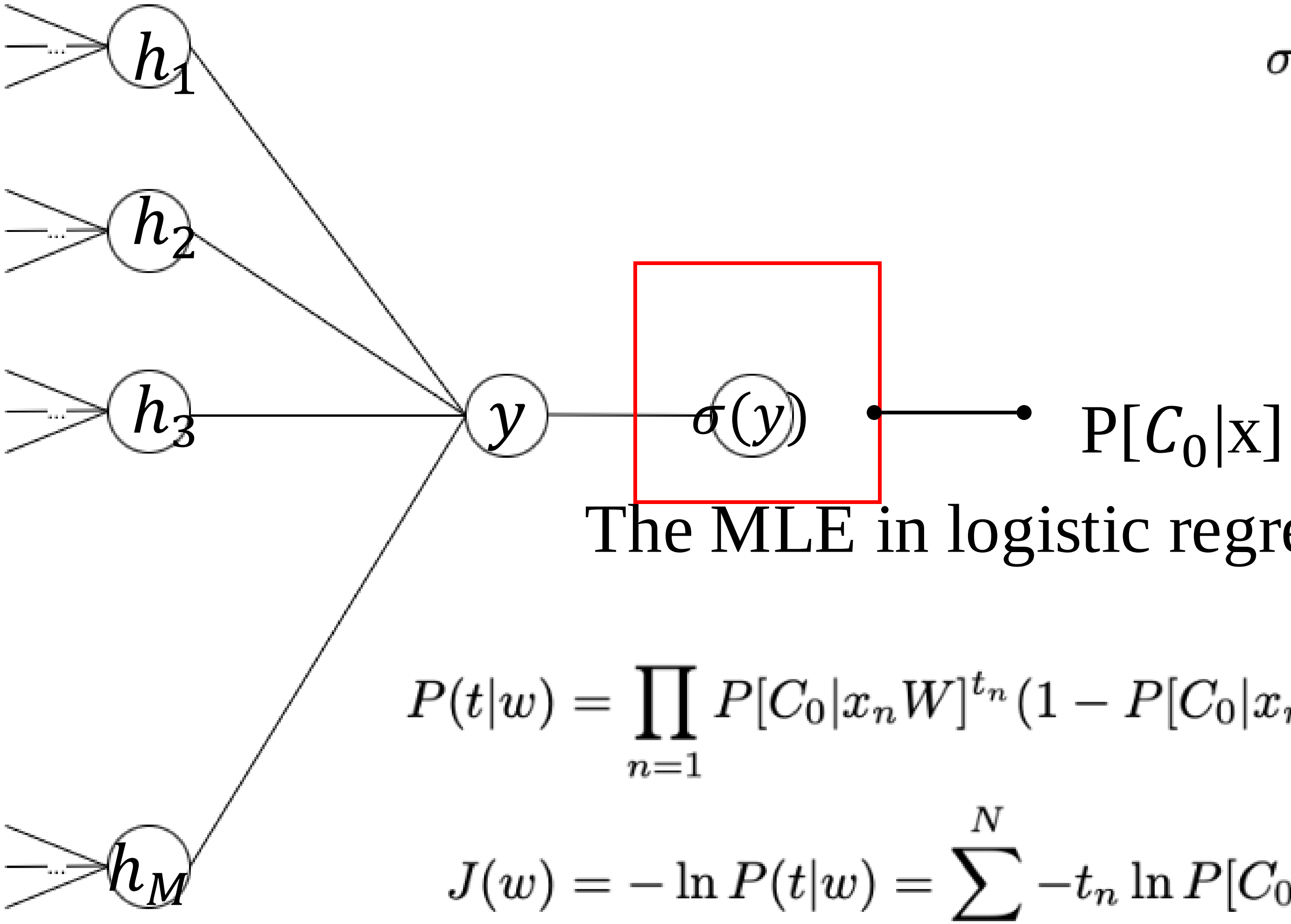
$$\frac{\partial J}{\partial y_1} = P[C_1|x] - t1$$


Softmax Negative Likelihood
J
$P[C_1|x]$
$P[C_2|x]$
$P[C_3|x]$
Softmax
y_1
$\frac{\partial J}{\partial y_2}$
$y2$
$\frac{\partial J}{\partial y_3}$
$y3$
h_1
h_2
h_3

[2] Logistic Regression for Binary Classification

[3] Sigmoid Unit for Binary Classification

$$\sigma(y) = \frac{1}{1+\exp -y} = \frac{1}{1+\exp -w^t h}$$



The MLE in logistic regression modeling,

$$P(t|w) = \prod_{n=1} P[C_0|x_n W]^{t_n}(1 - P[C_0|x_n W])^{1-t_n}$$

$$J(w) = -\ln P(t|w) = \sum_{n=1}^{N} -t_n \ln P[C_0|x_n W] - (1-t_n)\ln(1 - P[C_0|x_n W])$$

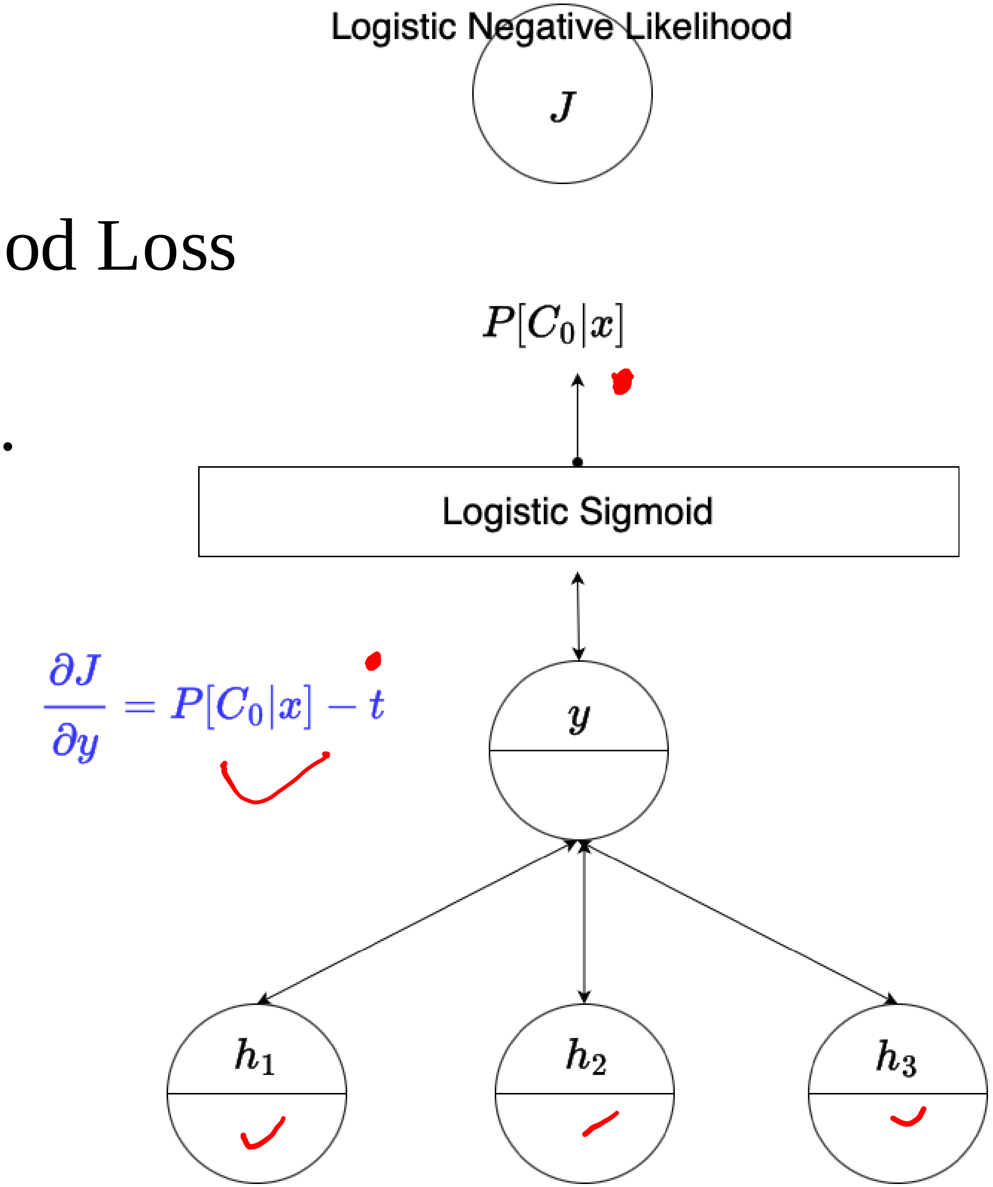
KL divergence!
Why?

Logistic Regression Derivative r.t y

$$J(w) = -\ln P(t|w) = \sum_{n=1}^{N} -t_n \ln P[C_0|x_n W] - (1 - t_n) \ln (1 - P[C_0|x_n W])$$

$$\begin{aligned} \frac{\partial J}{\partial y} &= \sum_{n=1}^{N} -t_n (1 - \sigma(y|x_n)) - (1 - t_n)(-\sigma(y|x_n)) \\ &= \sum_{n=1}^{N} \sigma(y|x_n) - t_n \end{aligned}$$

The derivatives of Sigmoid Negative Likelihood Loss : it is backpropagated throughout the network to compute the gradients of the network units. The parameters connected to the units are updated based on the updated gradients.


Logistic Negative Likelihood
J
$P[C_0|x]$
Logistic Sigmoid
y
$\frac{\partial J}{\partial y} = P[C_0|x] - t$
h_1
h_2
h_3