Neural Networks

The perceptron

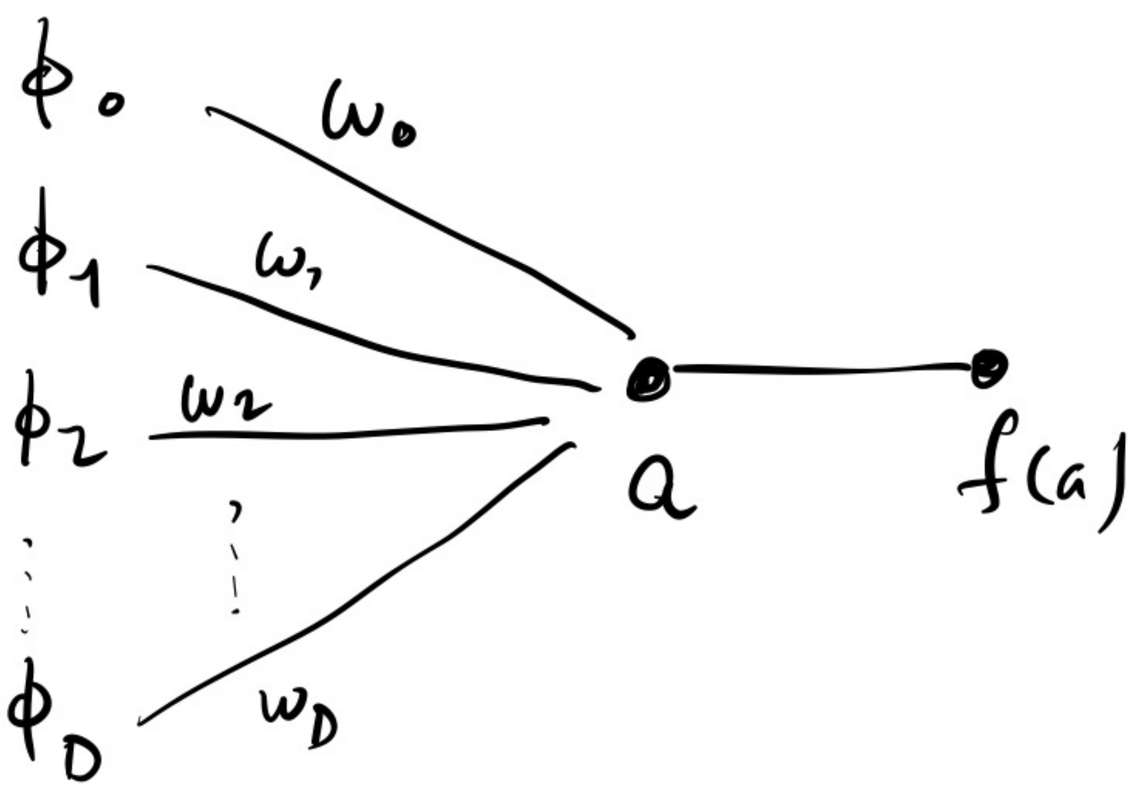
↳ originally → binary classification

$$y(\vec{x}) = f(\vec{w}^T \cdot \vec{\phi}(\vec{x})) = p(C_1 | \vec{x}, \vec{w})$$

↳

$$f(a) = \begin{cases} +1 & a \geq 0 \\ -1 & a < 0 \end{cases}$$

Convenient to label as $t_i = \{-1, 1\}$



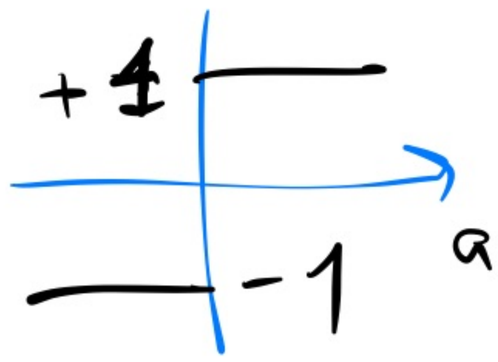
(reminds of a NN (also Logistic Regression)

To train the perceptron model

↳ minimise some cost function

↓ Cross-entropy : problematic

$\Rightarrow \dfrac{df}{da}$ ⇏ discontinuous



Instead :

$$E_p(\vec{w}) = -\sum_{i \in \mathcal{M}} \vec{w}^T \cdot \vec{\phi} \cdot t_i$$

↳ misclassified points only

Note : if $t_i = +1 \Rightarrow$ a correct predictor has $\vec{w}^T \vec{\phi}_i > 0$

$t_i = -1 \Rightarrow$ " " " $\vec{w}^T \vec{\phi}_i < 0$

↳ $\vec{w}^T \vec{\phi} t_i > 0$ ∀ points correctly classified

misclassif. points have $\vec{w}^T \vec{\phi}_i t_i < 0$

$E_p(\vec{w})$ is positive defined

How to minimise $E_p(\vec{w})$?

↳ Stochastic Gradient Descent

Gradient Descent

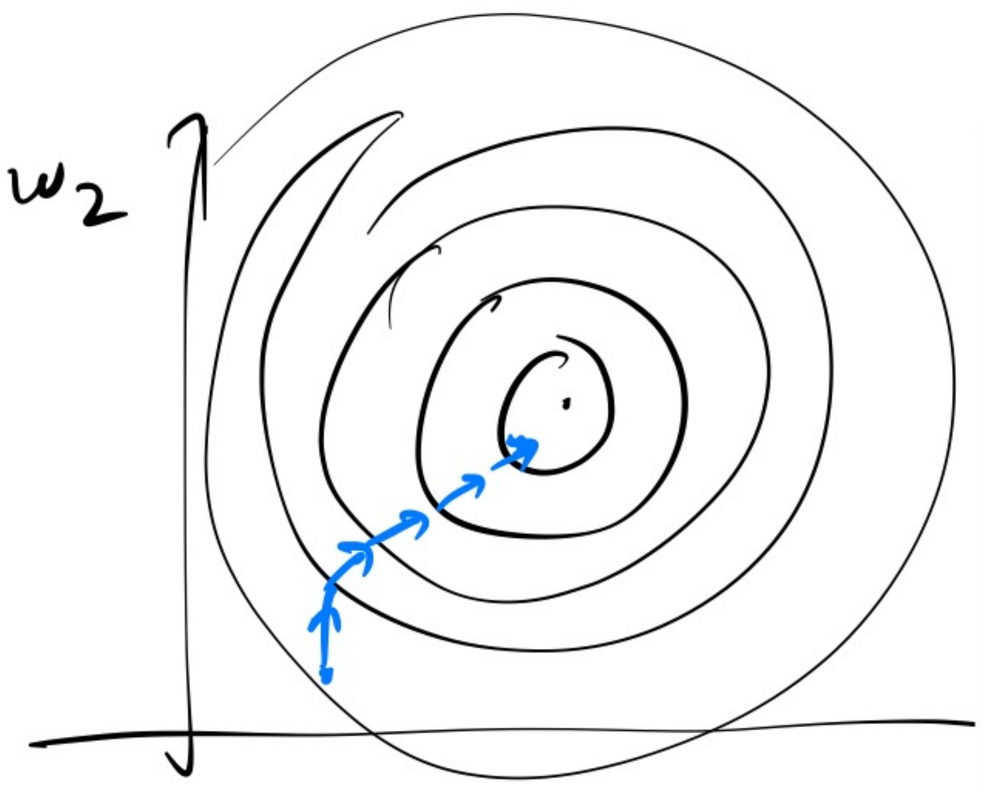
iterative algo. for optimising a function

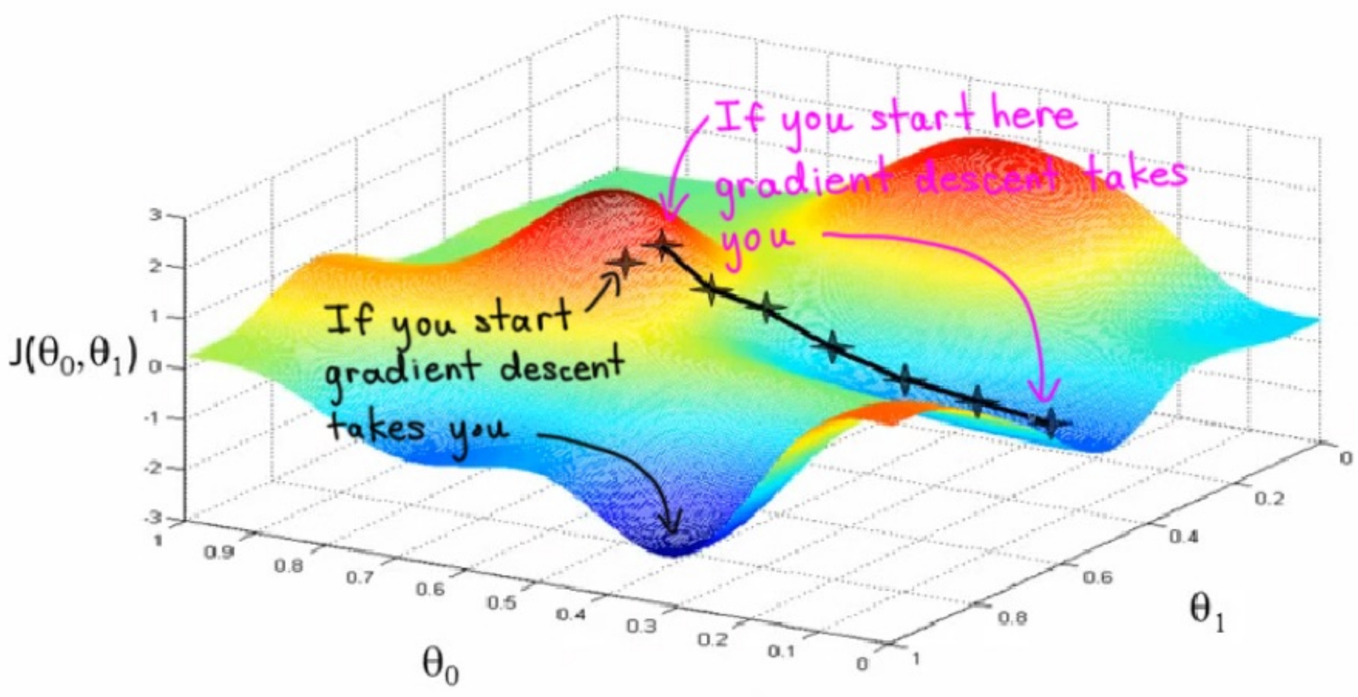
at iteration ("epoch") $\tau+1$

$$\vec{w}_{\tau+1} = \vec{w}_\tau - \eta \frac{dC(\vec{w})}{d\vec{w}}$$

C: cost function

η: learning rate



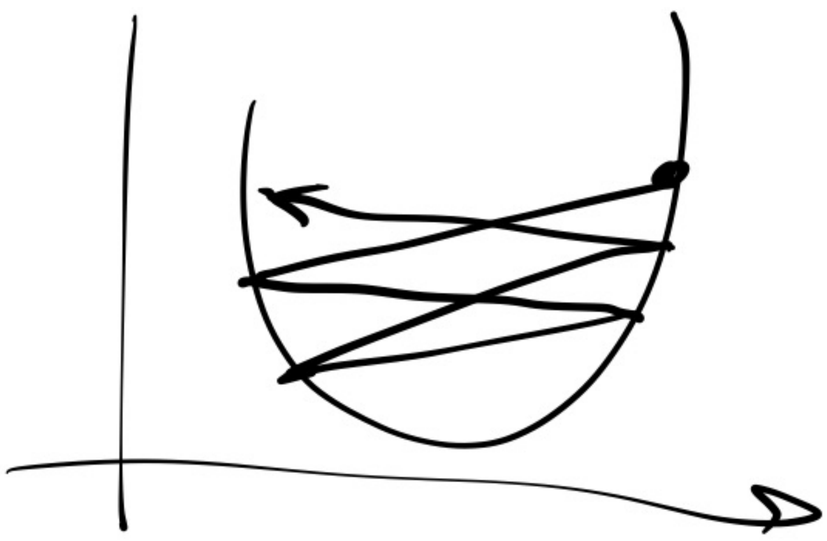


↳

↳ problem with local minima

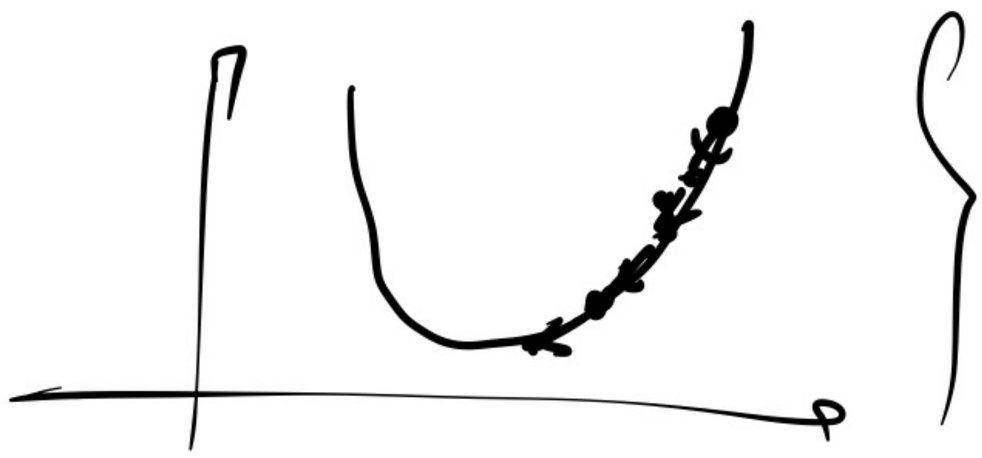
Depending on learning rate

large η



miss the minimum

too small



} too long training time

η can be a constant $\forall$ epochs (τ)

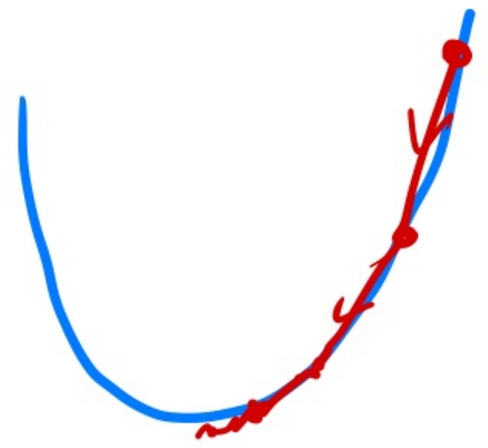
but sometimes

$$\eta \propto \frac{1}{\tau}$$

} hyperparameter that should be optimised

Stochastic G.D :

approximate $\frac{dC(\vec{w})}{d\vec{w}}$



$$C(\vec{w}) = \frac{1}{N} \sum_{i=1}^{N} C_i(\vec{w})$$

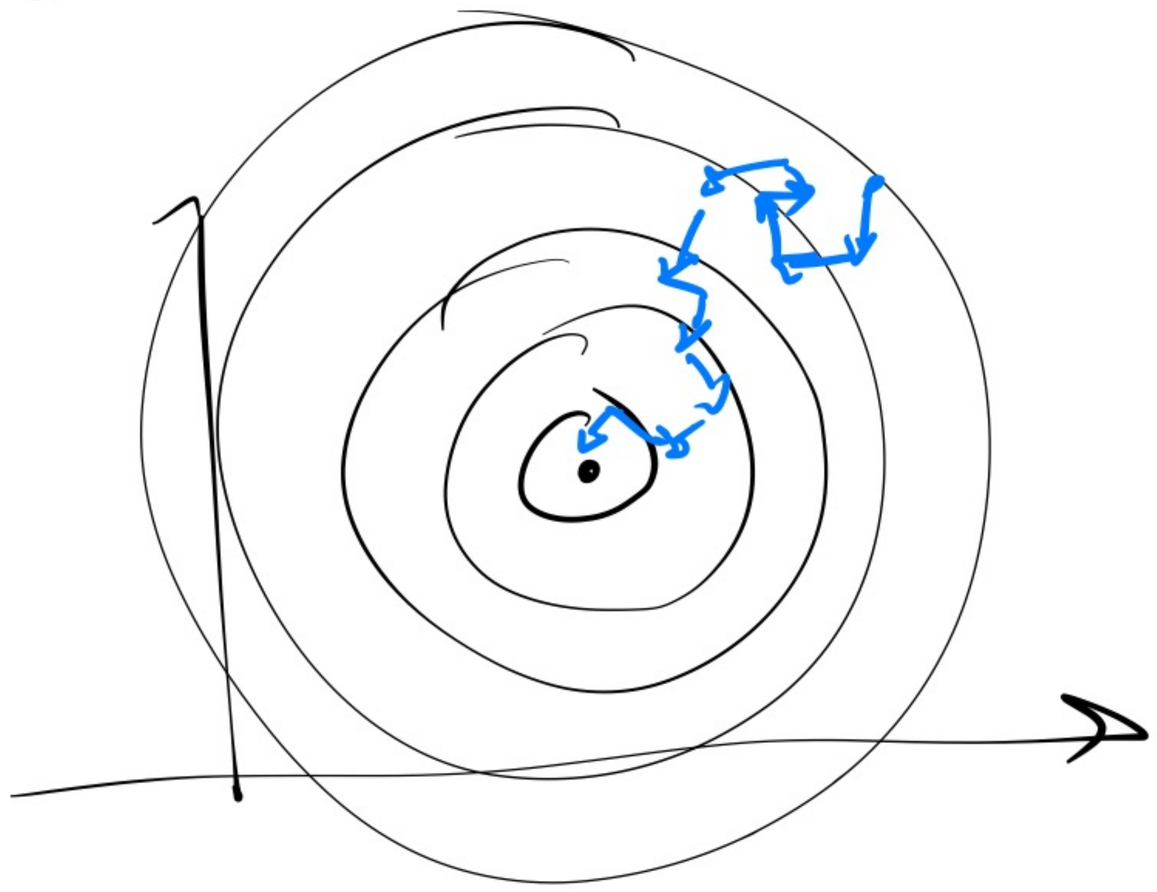
↳ expensive for large N

$$\frac{dC(\vec{w})}{d\vec{w}} \approx \frac{dC_i(\vec{w})}{d\vec{w}}$$

↳ chosen randomly

works because, on average

$$\frac{1}{N}\sum_i \frac{dC_i}{d\vec{w}} = \frac{dC}{d\vec{w}}$$



Back to perceptron

SGD

$$\vec{w}_{\tau+1} = \vec{w}_\tau + \eta \vec{\phi}_i t_i$$

$$\vec{w}_{\tau+1} = \vec{w}_\tau + \vec{\phi}_i t_i$$

η η

$$\Rightarrow \vec{w}_{\tau+1} = \vec{w}_\tau + \vec{\phi}_i t_i$$

f(a) is invarent under rescaling of $\vec{w}$

At each iteration τ,

find a random misclassified point,

$$\vec{w}_{\tau+1} = \begin{cases} \vec{w}_\tau + \vec{\phi}_i & \text{if } i \in C_1 \\ \vec{w}_\tau - \vec{\phi}_i & \text{if } i \in C_2 \end{cases}$$

in iteration $\tau+1$ the $E_p(\vec{w})$ will be

smaller than in τ

$$-\vec{w}_{\tau+1}\,\vec{\phi}_i\,t_i = -(\vec{w}_\tau + \vec{\phi}_i\,t_i)^T\,\vec{\phi}_i\,t_i$$

$$= -\vec{w}_\tau^T\,\vec{\phi}_i\,t_i$$

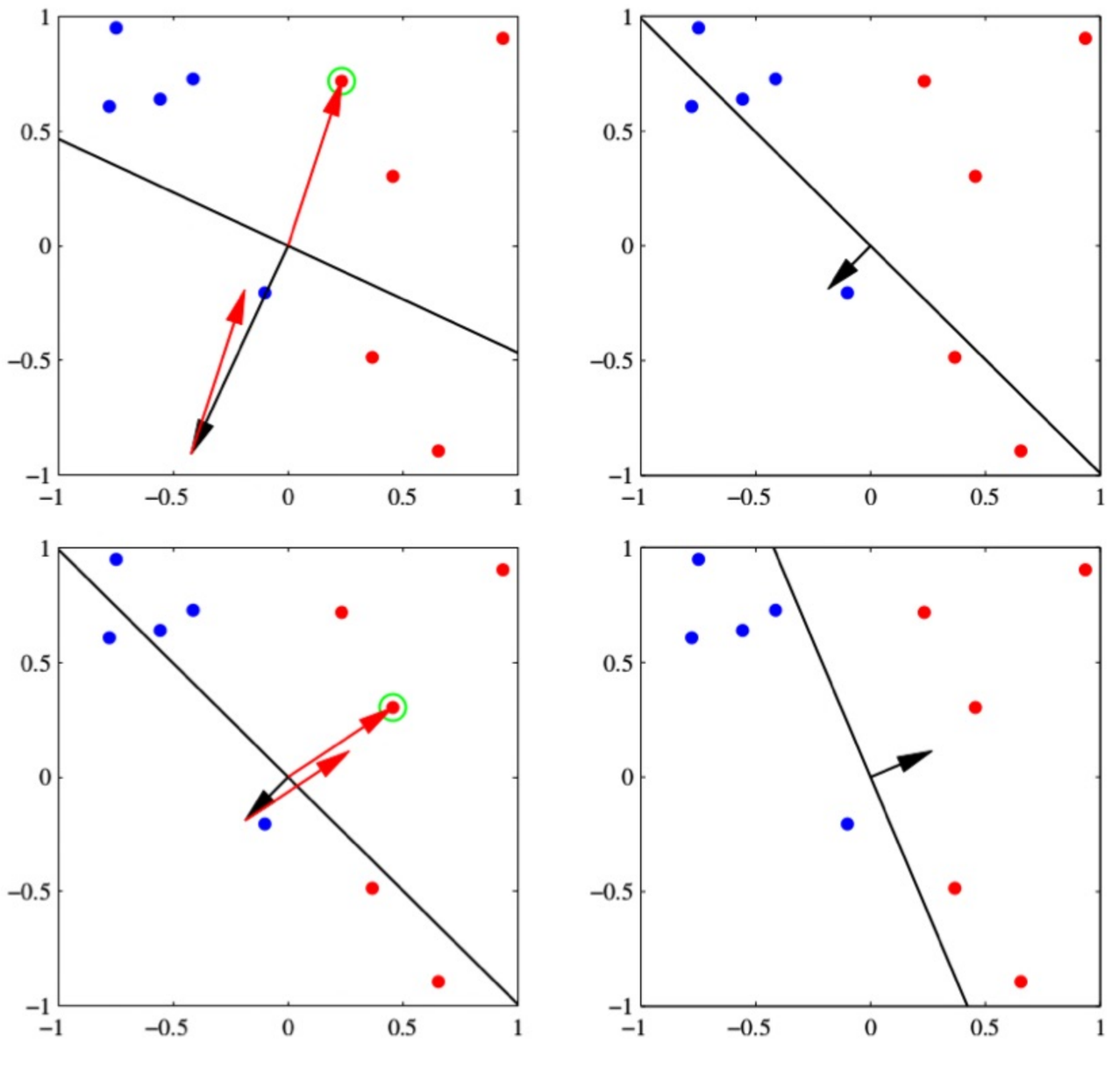
$$-(\vec{\phi}_i\,t_i)^T(\vec{\phi}_i\,t_i)$$

$$>0$$

$$< -\vec{w}_\tau^T\,\vec{\phi}_i\,t_i$$

perceptron convergence theorem

↳ data linearly separable



Neural nets

Start with linear model

$$y(\vec{x}, \vec{v}) = f\left(\sum_{h=1}^{H} v_h \phi_h(\vec{x}) \right)$$

more flexible model

$f(a) = a$ for regression problems

$f(a)$: non-linear function
e.g. sigmoid

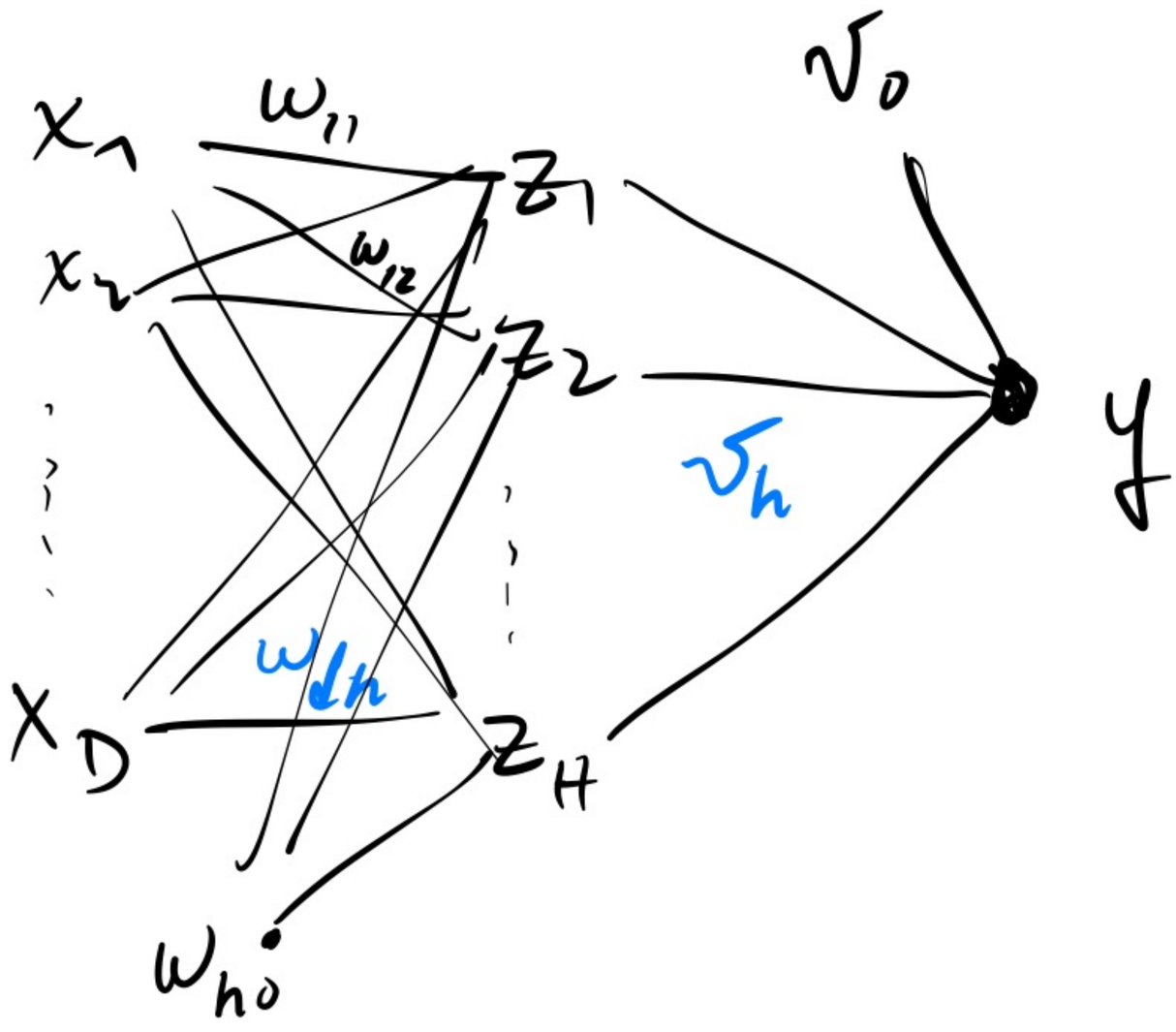
$$p(C_1 | \vec{x}, \vec{w})$$

$$\phi_h(\vec{x}, \vec{w_h}) = g\left(\sum_{d=1}^{D} w_{hd} x_d + w_{h0} \right)$$

↳ non-linear function (activation)

$$y(\vec{x}, \vec{v}, \vec{w}) = f\left(\sum_{h=1}^{H} v_h \underbrace{g\left(\sum_{d=1}^{D} w_{hd} x_d + w_{h0} \right)}_{\equiv z_h} + v_0 \right)$$

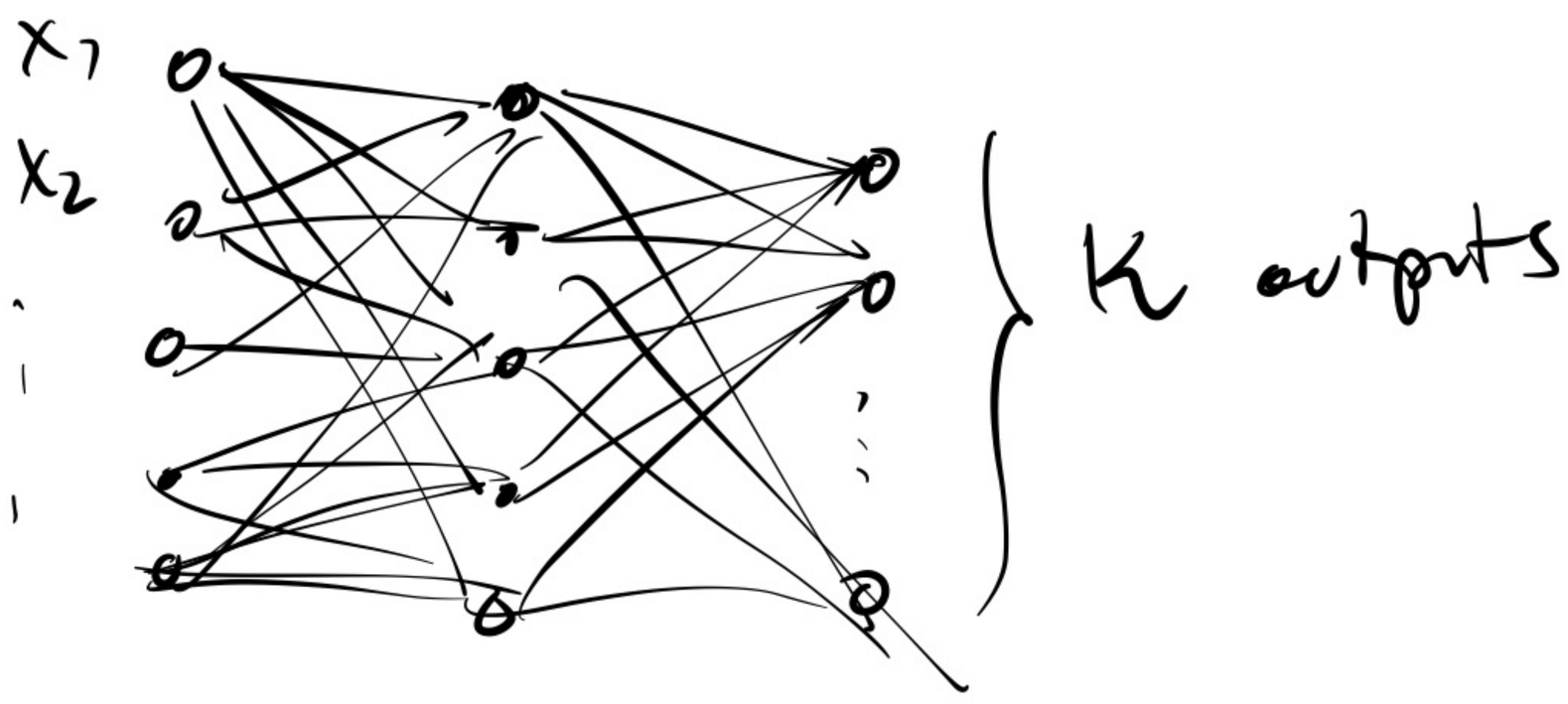
↳ one-hidden layer neural network (feedforward)



For classification (output is k dimensional)

$$\vec{t}_i \to \{t_1, \dots, t_k\}$$
$$k = 1, \dots, k$$

$$y_k(\vec{x}, \vec{v}, \vec{w}) = f\left(\sum_{h=1}^{1+} v_{hk}\, g\left(\sum_d w_{hd} x_d + w_{h0}\right) + v_{k0}\right)$$



$$y_k(\vec{x}, \vec{v}, \vec{w}) = f\left(\sum_{h=1}^{H} v_{hk}\, g\left(\sum_{d} w_{hd} x_d + w_{h0}\right) + v_{k0}\right)$$

$Y : N \times K$, $V = H \times K$, $V_0 = N \times K$

$X : N \times D$ $W : D \times H$ $W_0 : N \times H$

$$Y = f\left(g\left(XW + W_0\right)V + V_0\right)$$

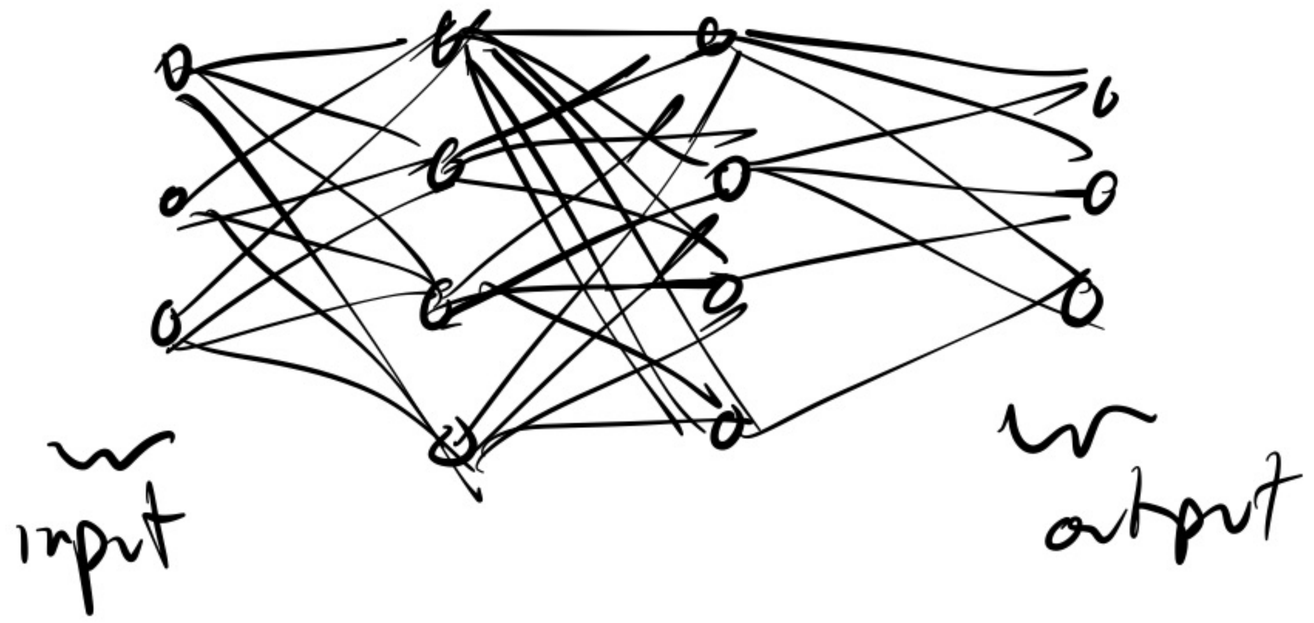
$N \times H$ $N \times H$

$(N \times H)$ $H \times K$

$N \times K$ $N \times K$

$N \times K$

A NN with 2 hidden layers



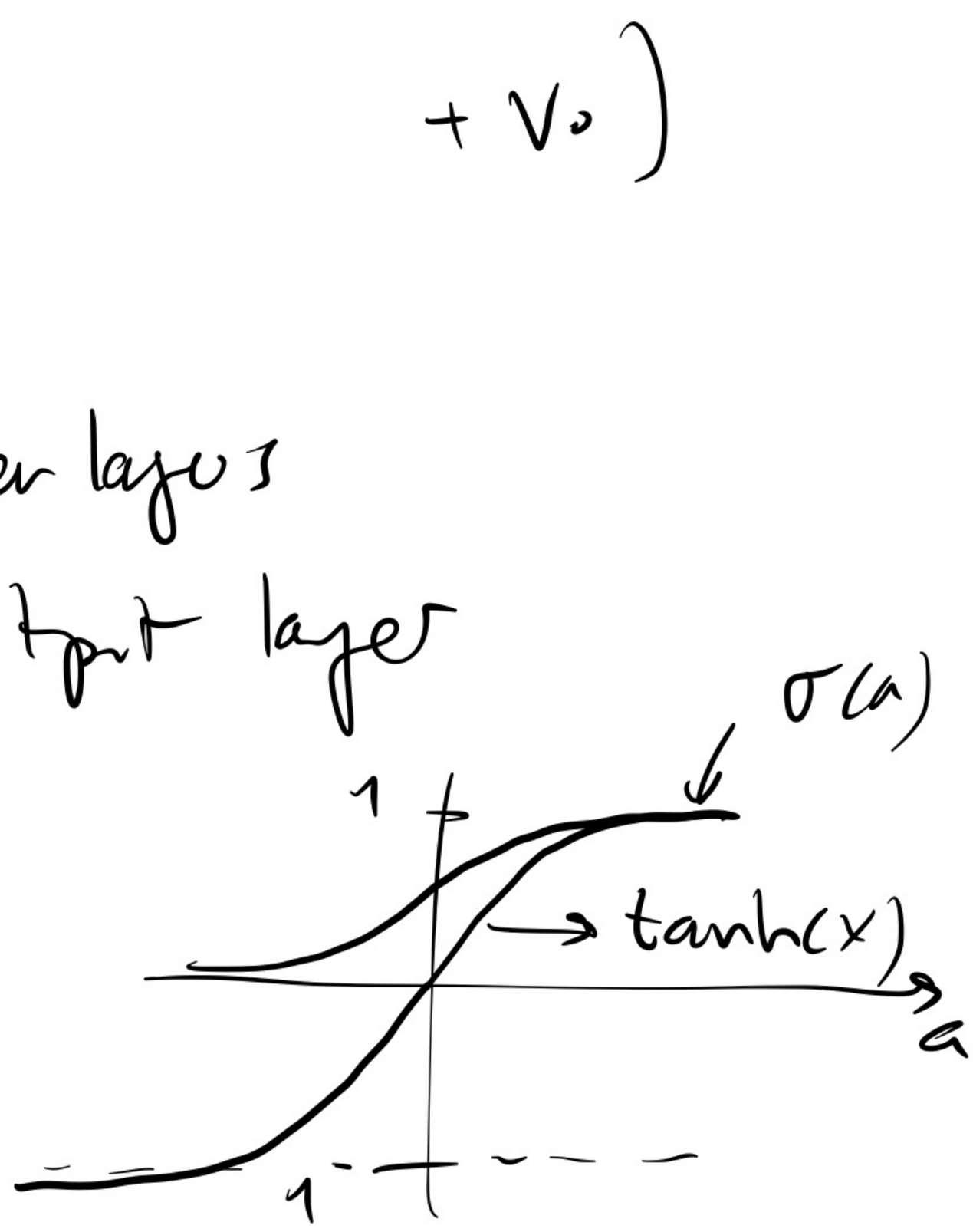
$$Y = f\left(g_2 \left[g_1 (X W^{(1)} + W_0^{(1)}) W^{(2)} + W_0^{(2)} \right] V + V_0 \right)$$

Activation functions

sigmoid ⇒ hidden layers

softmax ⇒ output layer

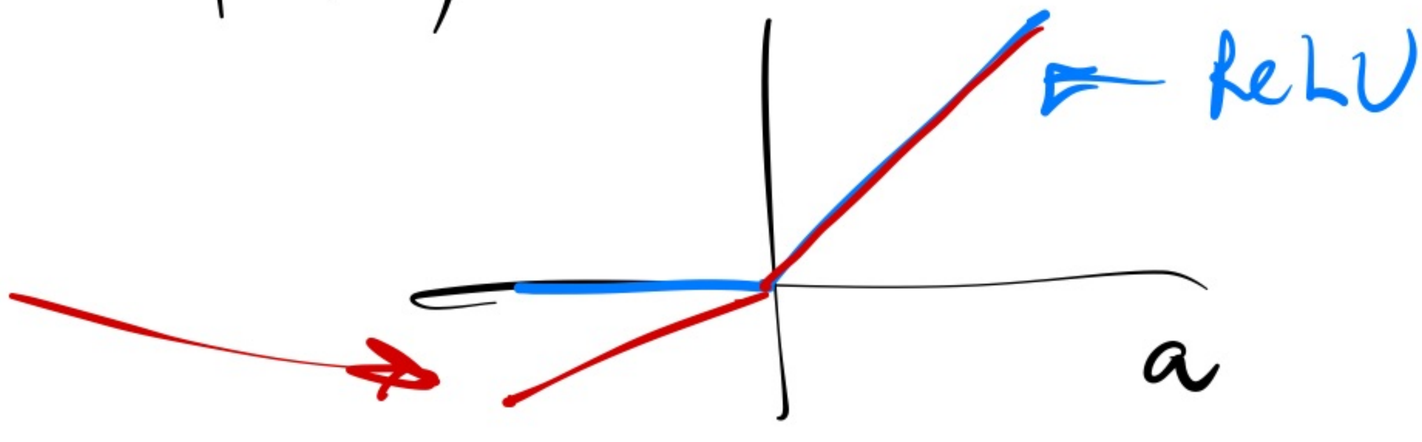
tanh, ...



more recently

ReLU = max(0, a)

leaky ReLU



All the activation function give in general good results

⇓

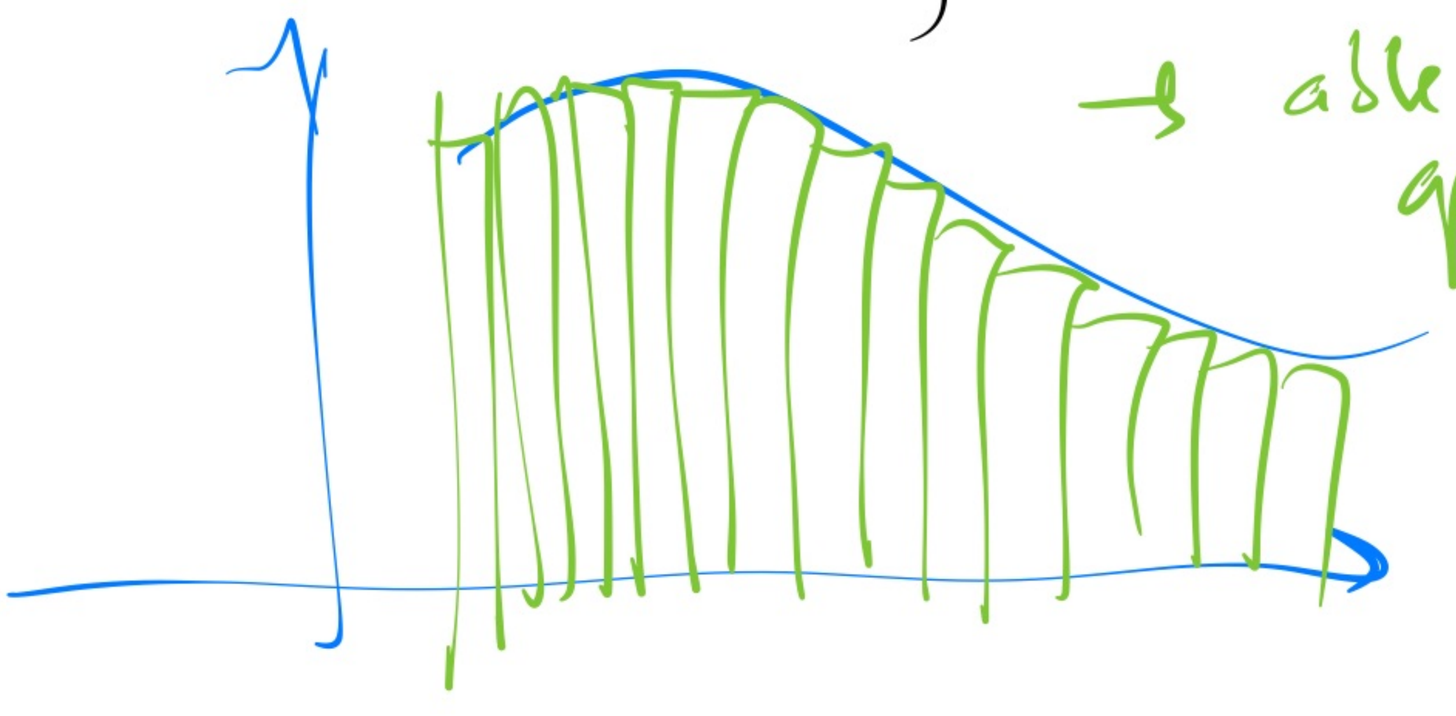
The Universal Approximation Theorem

A feedforward NN with a single hidden layer with finite # of units can approximate any continuous function under mild assumptions of the activation function

originally Proven with $\sigma(a) = \begin{cases} 1 & a \to \infty \\ 0 & a \to -\infty \end{cases}$

but

eg. ReLU works equally well



→ able to approx with rectangles

⇓

Good

Training the network

↳ find optimum weights that maximise likelihood (minimise cost function)

↓

Classification

$$N \quad k$$

$$\mathcal{E}(\vec{w}) = -\sum_{i=1}^{N} \sum_{k=1}^{k} t_{ik} \ln \underbrace{y_k(\vec{x_i}, \vec{w})}_{\equiv y_{ik}}$$

softmax ↗ sigmoid ↗

$$Y = \sigma_y\left(\sigma(XW + W_0)V + V_0\right)$$

$$\equiv \sigma_y\left(ZV + V_0\right)$$

sigmoid

$$\sigma(a) = \frac{1}{1+e^{-a}} \Rightarrow \frac{d\sigma}{da} = \sigma(1-\sigma)$$

Softmax

$$\sigma(a_k) = \frac{e^{a_k}}{\sum_j e^{a_j}} \Rightarrow \frac{d\sigma(a_k)}{da_j} = \sigma_k(1-\sigma_k)\delta_{kj}$$

$$= \sigma_k(\delta_{kj} - \sigma_j)$$

$$\frac{\partial E}{\partial v_{ok}} = -\sum_i \sum_{k'} t_{ik'} \frac{1}{y_{ik'}} \frac{\partial y_{ik'}}{\partial v_{ok}}$$

$$= -\sum_i \sum_{k'} t_{ik'} \frac{1}{y_{ik'}} y_{ik'} (\delta_{kk'} - y_{ik}) \frac{\partial a_k}{\partial v_{ok}}$$

$\frac{\partial a_k}{\partial v_{ok}} \to 1$

$$= -\sum_i \sum_{k'} t_{ik'} (\delta_{kk'} - y_{ik})$$

$$= -\sum_i \sum_{k'} (t_{ik'} y_{ik}) - \sum_i t_{ik}$$

$$= -\sum_i y_{ik} \underbrace{\sum_{k'} t_{ik'}}_{=1} - \sum_i t_{ik}$$

$$= \sum_i (y_{ik} - t_{ik}) = \sum_i (Y - T)_{ik}$$

Next

$$\frac{\partial E}{\partial v_{hk}} = -\sum_i \sum_{k'} t_{ik'} (\delta_{kk'} - y_{ik}) \frac{\partial a_{ik}}{\partial v_{hk}}$$

$\frac{\partial a_{ik}}{\partial v_{hk}} = z_{ih}$

$$= -\sum_i (y_{ik} - t_{ik}) z_{ih}$$

$$\frac{\partial \mathcal{E}}{\partial V} = Z^T (Y - T)$$

$$\frac{\partial \mathcal{E}}{\partial w_{ok}} = -\sum_i \sum_k t_{ik} \frac{1}{y_{ik}} y_{ik}(1 - y_{ik}) \frac{\partial a_{ik}}{\partial z_{ih}} \frac{\partial z_{ih}}{\partial w_{oh}}$$

$$= \sum_i \sum_k (t_{ik} - y_{ik}) v_{hk} z_{ih} (1 - z_{ih})$$

$$= -\sum_i \sum_k (T - Y)_{ik} (V^T)_{kh} z_{ih}(1 - z_{ih})$$

$$= -\sum_i \left[(Y - T) V^T * Z * (1 - Z) \right]_{ih}$$

* = Schur product (element-by-element)

finally

$$\frac{\partial \mathcal{E}}{\partial w_{jh}} = \sum_i \left[(Y - T) V^T * Z * (1 - Z) \right]_{ij} x_{ij}$$

$$= X^T \left[(Y - T) V^T * Z * (1 - Z) \right]$$