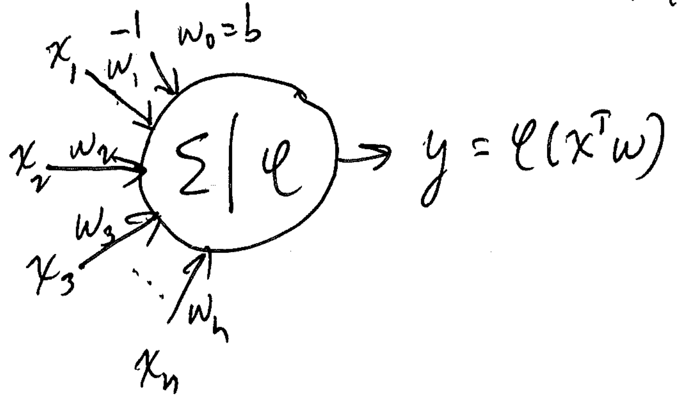
Deep Learning Fundamentals

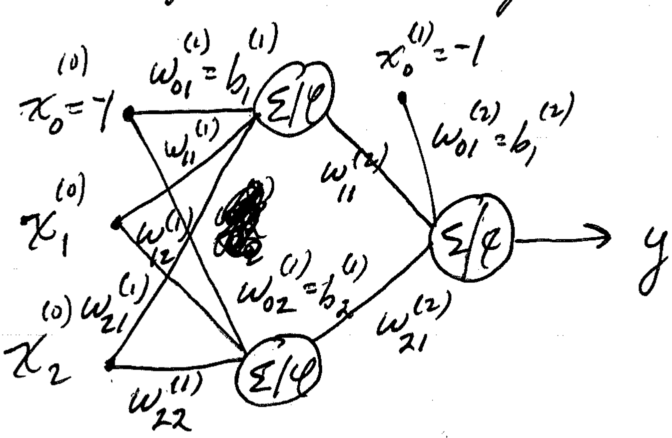
Abstract neuron with bias b and activation function φ [Let's try to build an artificial brain!]



$$\vec{x} \cdot \vec{w} = \vec{w} \cdot \vec{x} = \vec{x}^T \vec{w} = \vec{w}^T \vec{x} = \sum_{i=0}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n - b$$

linear neuron: $\varphi(x) = x$

neural network with one hidden layer (two neurons) and two inputs



$w_{ij}^{(k)}$ ← layer

input ↑ i, neuron (in layer k) ↑ j

If these are linear neurons

output of neuron 1 in layer 1 : $x_1^{(1)} = \varphi\left(\sum_{i=0}^{2} w_{i1}^{(1)} x_i^{(0)}\right) \overset{?}{=} \sum_{i=0}^{2} w_{i1}^{(1)} x_i^{(0)}$ (1)

" " 2 " " 1 $x_2^{(1)} = \varphi\left(\sum_{i=0}^{2} w_{i2}^{(1)} x_i^{(0)}\right) = \sum_{i=0}^{2} w_{i2}^{(1)} x_i^{(0)}$ (2)

output of neuron 1 in layer 2 or of network! $y = \varphi\left(\sum_{i=0}^{2} w_{i1}^{(2)} x_i^{(1)}\right) = \sum_{i=0}^{2} w_{i1}^{(2)} x_i^{(1)}$ (3)

Using (1) and (2) in (3)

$$\cancel{y = \sum_{i=0}^{2} w_{i1}^{(2)} \sum_{i=0}^{2} w_{i1}^{(1)} x_i^{(0)}}$$

$$y = w_{01}^{(2)} x_0^{(1)} + w_{11}^{(2)} x_1^{(1)} + w_{21}^{(2)} x_2^{(1)}$$

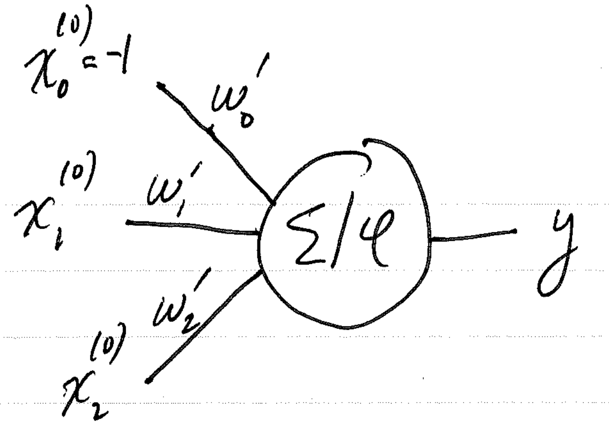
$$= w_{01}^{(2)} x_0^{(1)} + w_{11}^{(2)} \sum_{i=0}^{2} w_{i1}^{(1)} x_i^{(0)} + w_{21}^{(2)} \sum_{i=0}^{2} w_{i2}^{(1)} x_i^{(0)}$$

$$= \sum_{i=0}^{2} \underbrace{\left(w_{11}^{(2)} w_{i1}^{(1)} + w_{21}^{(2)} w_{i2}^{(1)}\right)}_{w_i'} x_i^{(0)} + w_{01}^{(2)} x_0^{(1)}$$

$$= \underbrace{\left(w_0' x_0^{(0)} + w_{01}^{(2)} x_0^{(1)}\right)}_{\text{constant!}} + \underbrace{w_1' x_1^{(0)} + w_2' x_2^{(0)}}$$ Linear!

$w_0' = b'$

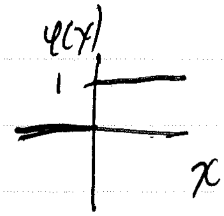
What is a new simplified network representation?



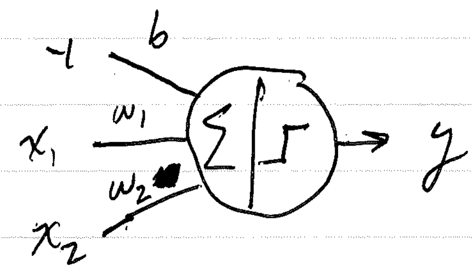
Network collapsed!

φ needs to be non-linear!

Perceptron $\quad \varphi(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$



Assume



which of the following logic gates can this implement (if any)?

AND

x_1	x_2	$y = x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

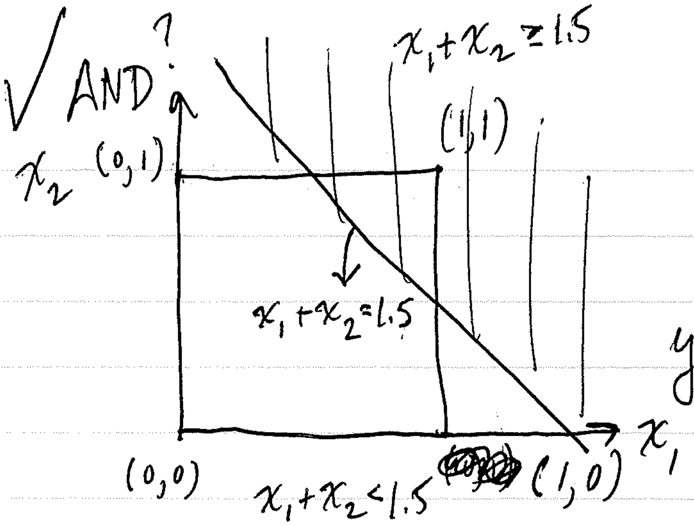
OR

x_1	x_2	$y = x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

XOR

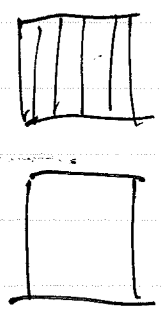
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

√ AND?

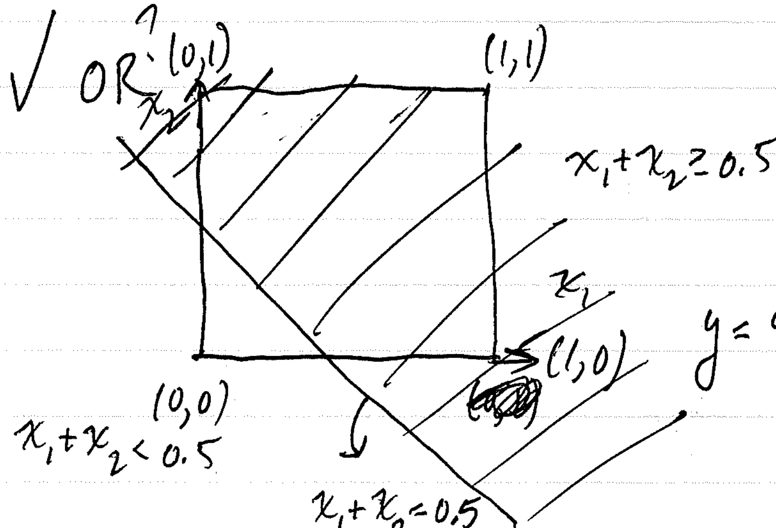


let $w_1 = w_2 = 1$ and $b = 1.5$

$$y = \varphi(x_1 + x_2 - 1.5) = \begin{cases} 0, \text{ if } x_1 + x_2 < 1.5 \\ 1, \text{ if } x_1 + x_2 \geq 1.5 \end{cases}$$



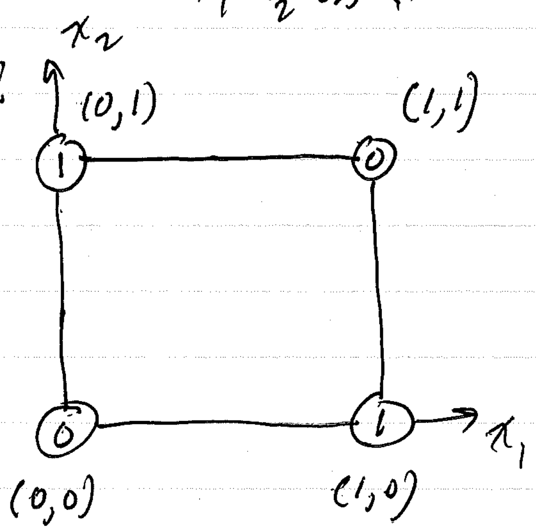
√ OR?



let $w_1 = w_2 = 1$ and $b = 0.5$

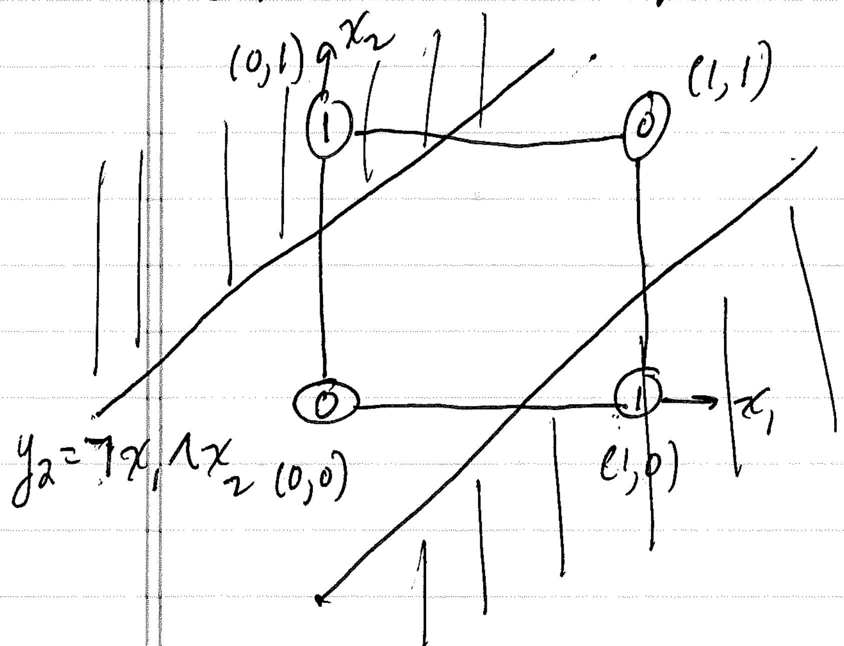
$$y = \varphi(x_1 + x_2 - 0.5) = \begin{cases} 0, \text{ if } x_1 + x_2 < 0.5 \\ 1, \text{ if } x_1 + x_2 \geq 0.5 \end{cases}$$

X XOR?



No line can separate the 0 and 1 output classes!

What about our 3 neuron model with linear neurons replaced by perceptrons?



$y_1 = x_1 \wedge \neg x_2$

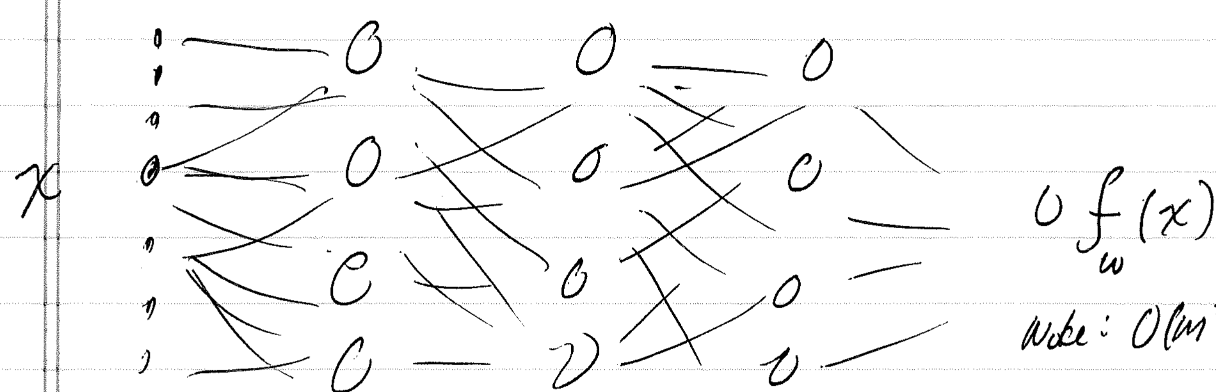
$y = y_1 \vee y_2$

$= (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$

$=$ XOR ✓

Non-linear neurons and multiple/hidden layered network yields complex decision boundaries!

Universal Function Approximator!



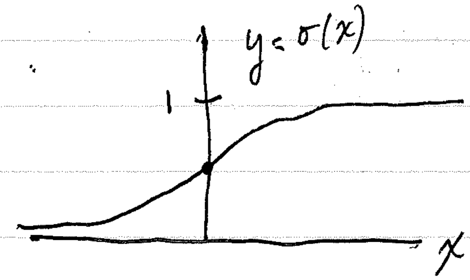
Note: $O(m^2)$ for SVMs

So are SVMs but NN training $O(m)$ m samples

What are some other non-linear neurons?

Sigmoid

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



What does this represent: $y = \sigma(x^T w)$?

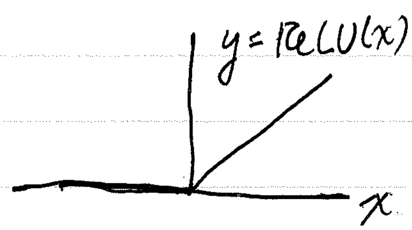
Logistic Regression! $\Rightarrow$ 1 Sigmoid neuron

Early versions of NNs used sigmoid neurons, but because they saturate lead to inability to learn! Known as "vanishing gradient problem" or "barren plateaus".

How to solve this problem? ReLU(x)!

Rectified Linear Unit (ReLU)

$$\varphi(x) = \mathrm{ReLU}(x) = \max\{0, 0\} = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$$



For multi-class output that takes a vector $\vec{x} = (x_1, \ldots, x_K)$ as input, use softmax

$$\varphi(\vec{x}) = \sigma(\vec{x}) = \left(\underbrace{\frac{e^{x_1}}{\sum_{k=1}^{K} e^{x_k}}}_{p(y=1|\vec{x})}, \ldots, \underbrace{\frac{e^{x_K}}{\sum_{k=1}^{K} e^{x_k}}}_{p(y=K|\vec{x})} \right)$$

$\sigma(\vec{x})_i = p(y=k|\vec{x})$

$$\sum_{k=1}^{K} p(y=k|\vec{x}) = \sum_{k=1}^{K} \frac{e^{x_k}}{\sum_{k'=1}^{K} e^{x_{k'}}} = \frac{\sum_{k=1}^{K} e^{x_k}}{\sum_{k'=1}^{K} e^{x_{k'}}} = 1$$

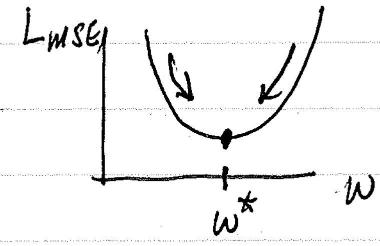
where $\log \sigma(\vec{x})_i = x_i - \log \sum_{k=1}^{K} e^{x_k}$

This avoids vanishing gradient ~~pr~~ problem!

How do we train a NN for an arbitrary dataset?

How do we fit a linear model with multiple parameters?

Find the weights $\vec{w}$ that minimize the (MSE) mean square error loss

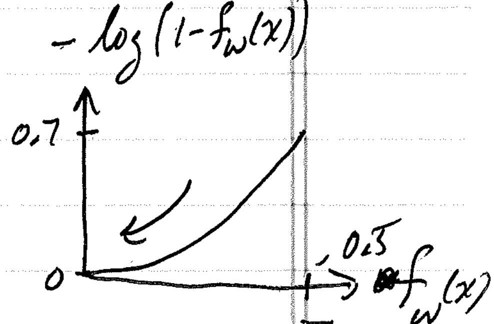


$$L_{MSE} = \frac{1}{m}\sum_{i=1}^{m}\left(y_i - f_w(x^i)\right)^2$$

where $f_w(x) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$

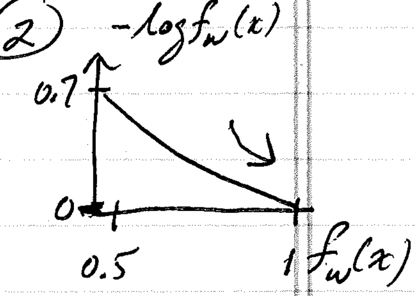
Do the same for NNs! MSE is used when output of NN is real number.

What is a good loss function for a classifier (e.g. binary classifier) where output is 0 or 1?



Binary Cross Entropy:

$$L_{BCE} = -\frac{1}{m}\sum_{i=1}^{m}\Big[\underbrace{y_i \log f_w(x^i)}_{①} + \underbrace{(1-y_i)\log\left(1-f_w(x^i)\right)}_{②}\Big]$$

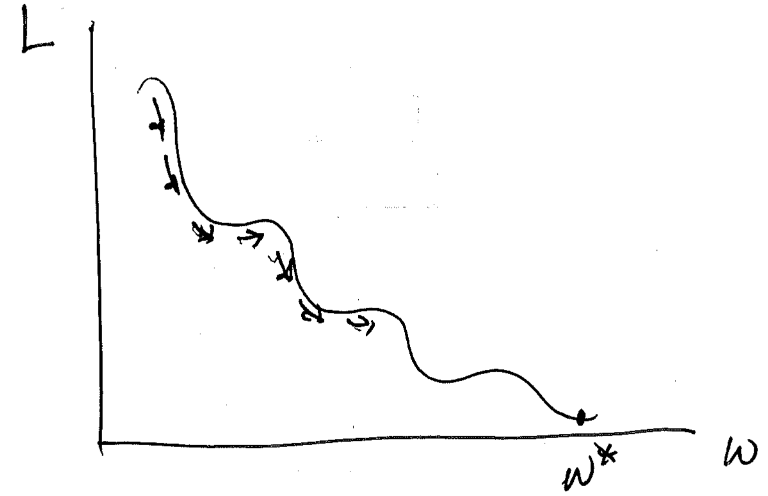


if $y_i = 1$ ② $= 0$ ① $= -\log f_w(x^i)$

if $y_i = 0$ ① $= 0$ ② $= -\log(1 - f_w(x^i))$

How do we minimize loss functions for a neural network?

Method of Steepest Descent or Gradient Descent



Taylor Expansion (Linear Approximation):

$$\Delta L = L(w^0 + \eta v) - L(w^0) = \sum_{i=1}^{n} \frac{\partial L}{\partial w_k}(w^0)\,\eta v^k + O(\eta^2)$$

Change weights from w^0 in v ~~dire~~ (unit vector) direction with step size η (which we assume is small).

$$\Delta L = \eta \underbrace{\vec{\nabla} L \cdot \vec{v}}_{\text{projection of } \vec{\nabla} L \text{ onto } \vec{v}} + O(\eta^2)$$

Cauchy inequality

$$-\|\vec{\nabla} L(w^0)\|\,\|\vec{v}\| \le \vec{\nabla} L \cdot \vec{v} \le \|\vec{\nabla} L(w^0)\|\,\|\vec{v}\|$$

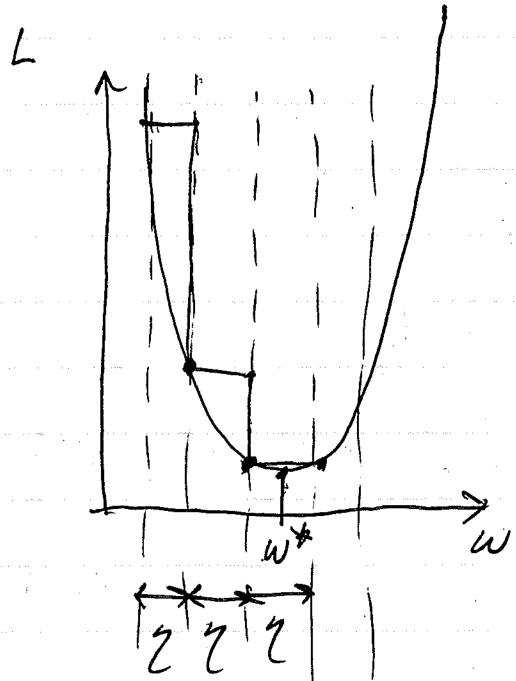
$\vec{\nabla} L$ & $\vec{v}$ anti-parallel $\qquad$ $\vec{\nabla} L$ & $\vec{v}$ parallel

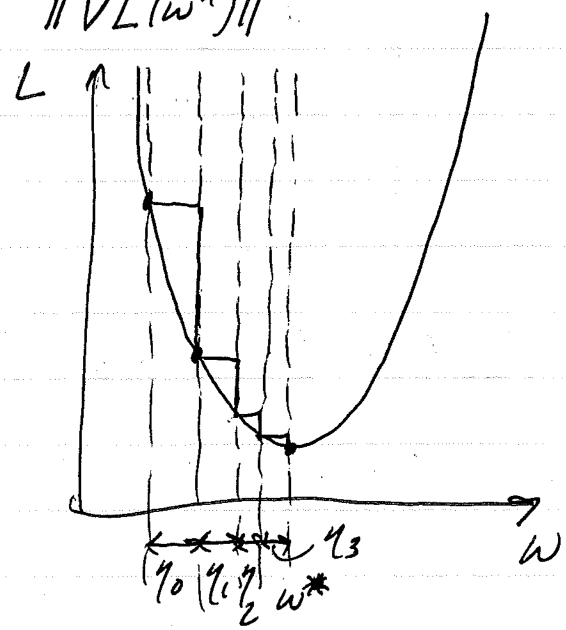
Maximum drop in L occurs for unit length $\vec{v}$ ($\Rightarrow \|\vec{v}\|=1$) when it's anti-parallel to $\vec{\nabla} L$

$$\vec{v} = -\frac{\vec{\nabla} L(w^0)}{\|\vec{\nabla} L(w^0)\|}$$

$\therefore$ Update weights as follows

$$w^{n+1} = w^n - \eta \frac{\vec{\nabla} L(w^n)}{\|\vec{\nabla} L(w^n)\|}$$





Problem: Overshoots!

Fixed: Learning rate $\propto$ gradient

$$\text{let } \eta_n = \delta \|\vec{\nabla} L(w^n)\|$$

$$w^{n+1} = w^n - \eta_n \frac{\vec{\nabla} L(w^n)}{\|\vec{\nabla} L(w^n)\|} = w^n - \delta \cancel{\|\vec{\nabla} L(w^n)\|} \frac{\vec{\nabla} L(w^n)}{\cancel{\|\vec{\nabla} L(w^n)\|}}$$

$$w^{n+1} = w^n - \delta \nabla L(w^n)$$

How calc ∇L w/ $f_w(x^i)$ representing a NN? Back Propagation