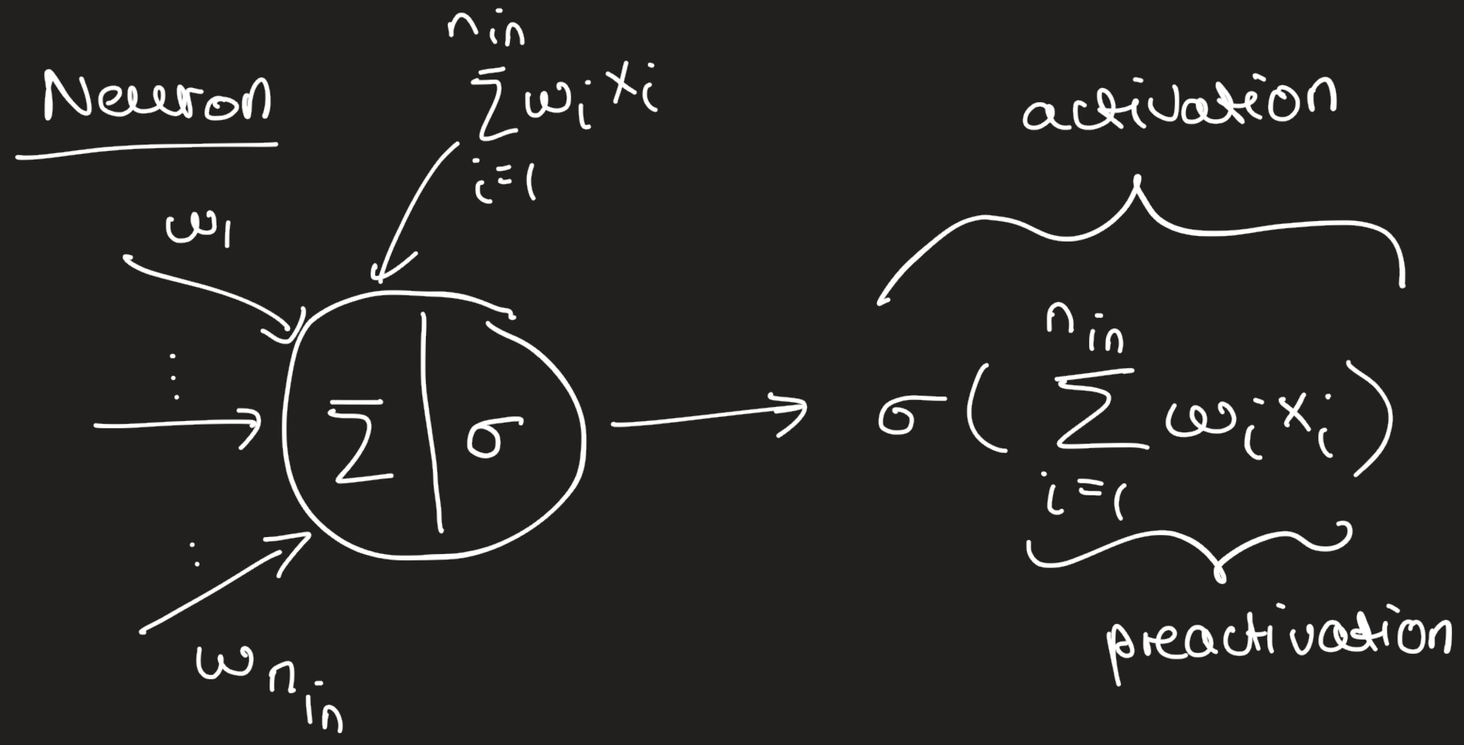
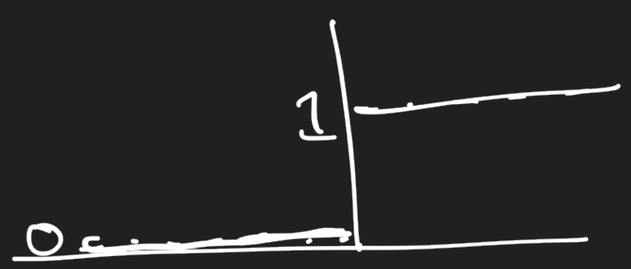
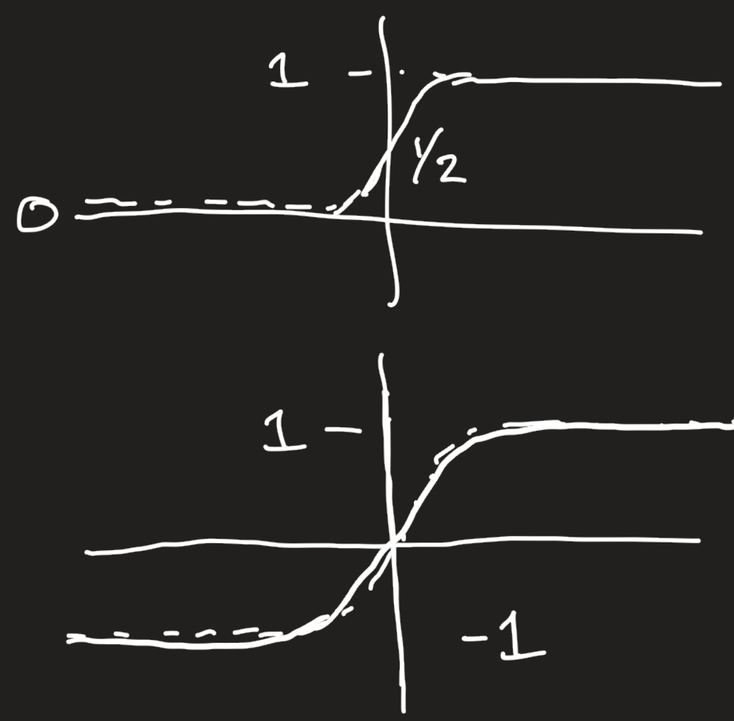


Ex activation functions



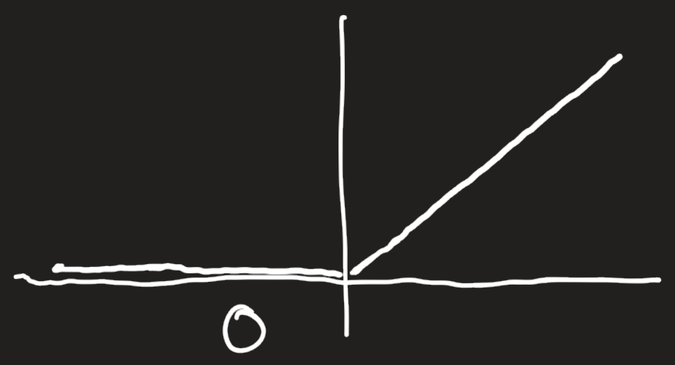
$\sigma(x) = \operatorname{sgn}(x)$ — perceptron



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

logistic sigmoid

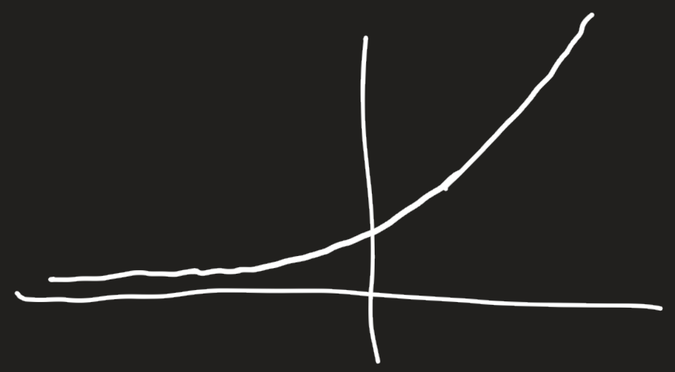
$$\sigma(x) = \tanh(x)$$



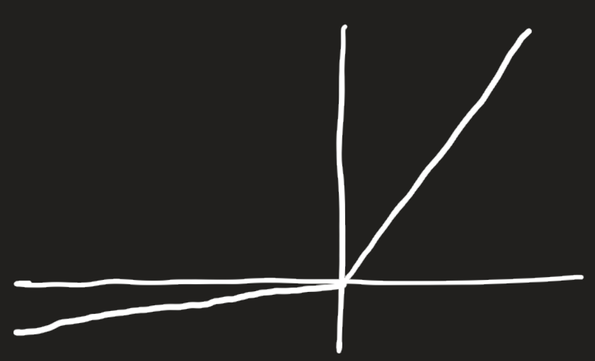
$$\sigma(x) = x_+$$

ReLu – rectified linear unit

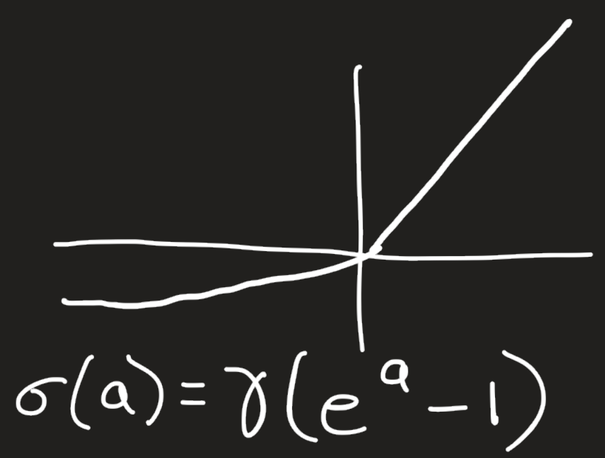
≈2015



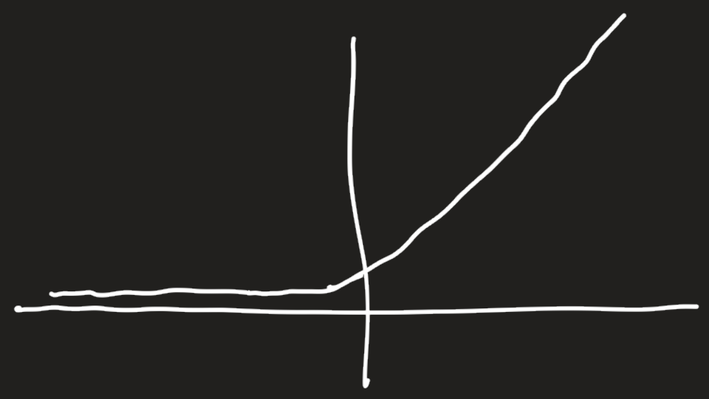
$\sigma(x) = \ln(1 + e^q)$ softplus

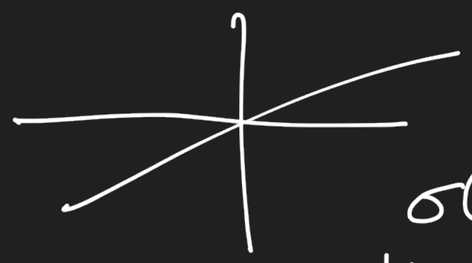


leaky ReLU $\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ \beta x & \text{if } x < 0 \end{cases}$



exponential LU (ELU)
or
scaled ELU (SELU)





$\sigma(x) = x$
linear

$$\sigma(x) = x \cdot \sigma(\beta x)$$

logistic sigmoid

"swish function" 2017

found by searching over expression space for best performing activation functions

Takeaway: the activation function choice is a hyperparameter that should be chosen in the same way you choose the optim alg, etc. by taking the application domain & dataset into account

Expressivity of MLPs ← Multilayer perceptrons

What class of functions can be represented using a NN with L layers and an arbitrary number of neurons?

Fact: for any activation function that is nonlinear & smooth
monotonic, it is the case that any function f can approximated arbitrarily well in L_2 norm by a two-layer NN w/ this activation function.

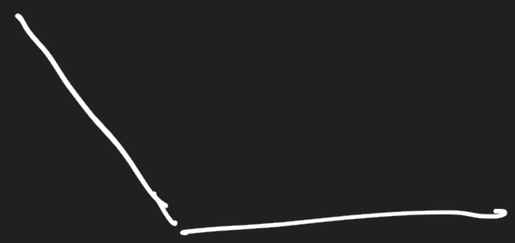
That is, there exists a 2-layer NN $g(x)$ such that

$$\|g - f\|_2^2 = \int |f(x) - g(x)|^2 \, dx < \varepsilon.$$

(g may use an exponential # of neurons)

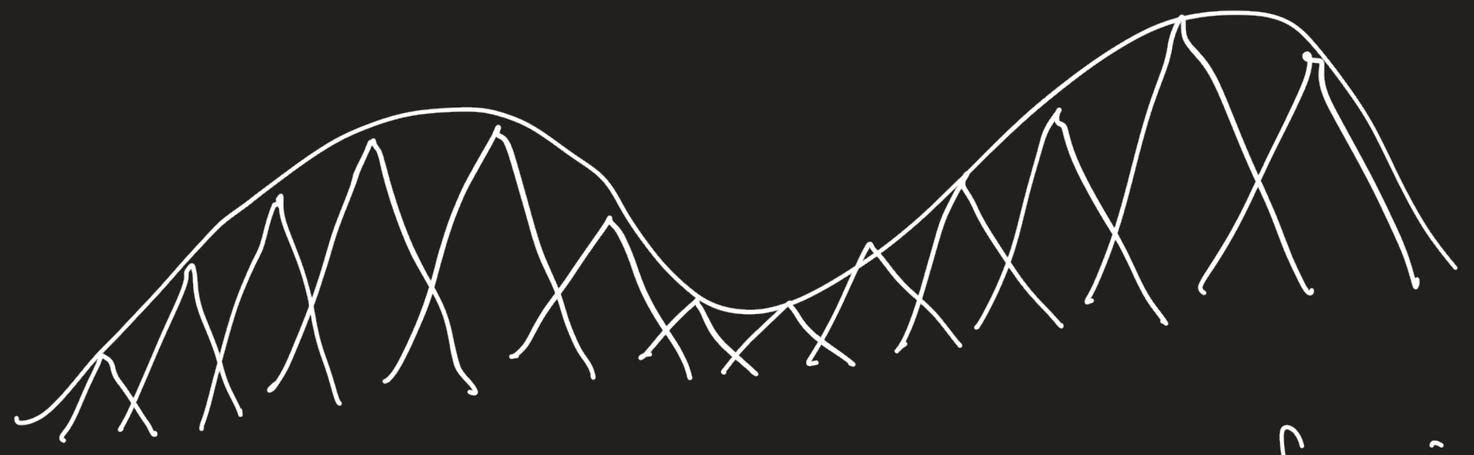
Idea sketch:

$\sigma(x)$



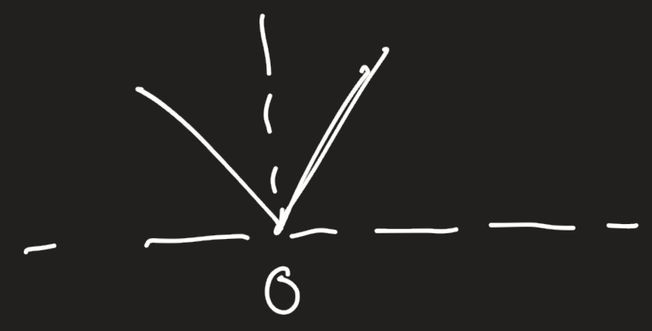
$\sigma(-x)$

use two layers to get a triangle



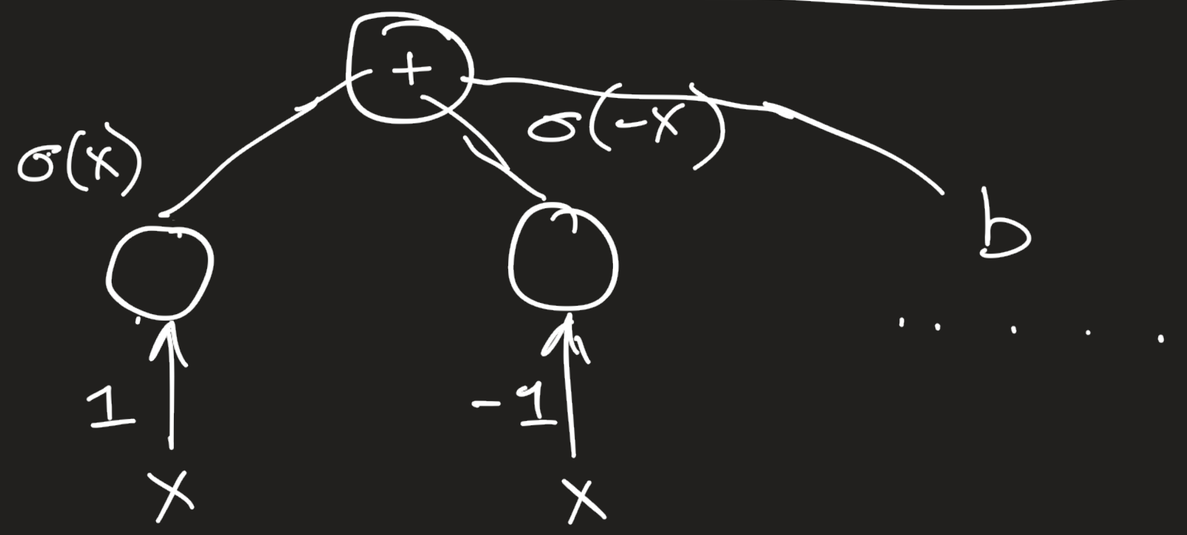
approx any function w/ sum of triangles

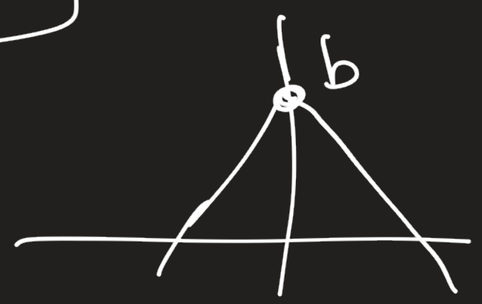
$$\sigma(x) + \sigma(-x)$$



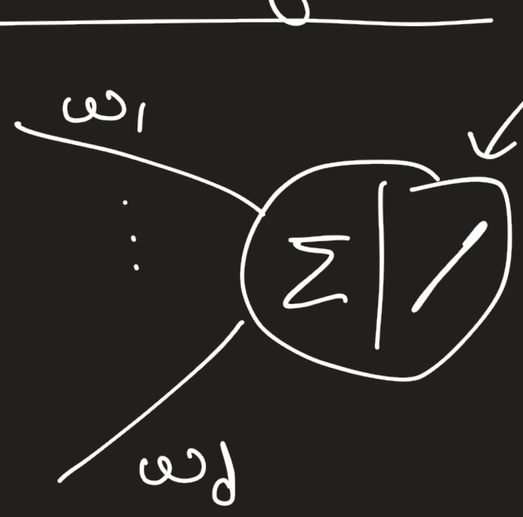
$$x = \sigma(x) - \sigma(-x) \quad - \quad \sigma(x) + \sigma(-x)$$

$$-\sigma(x) + \sigma(-x) + b$$



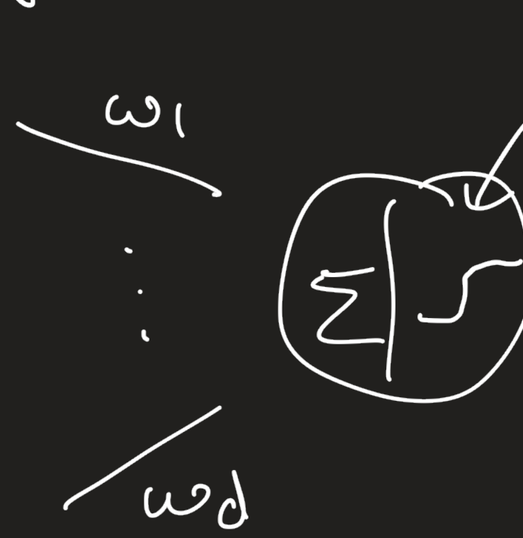


Linear regression



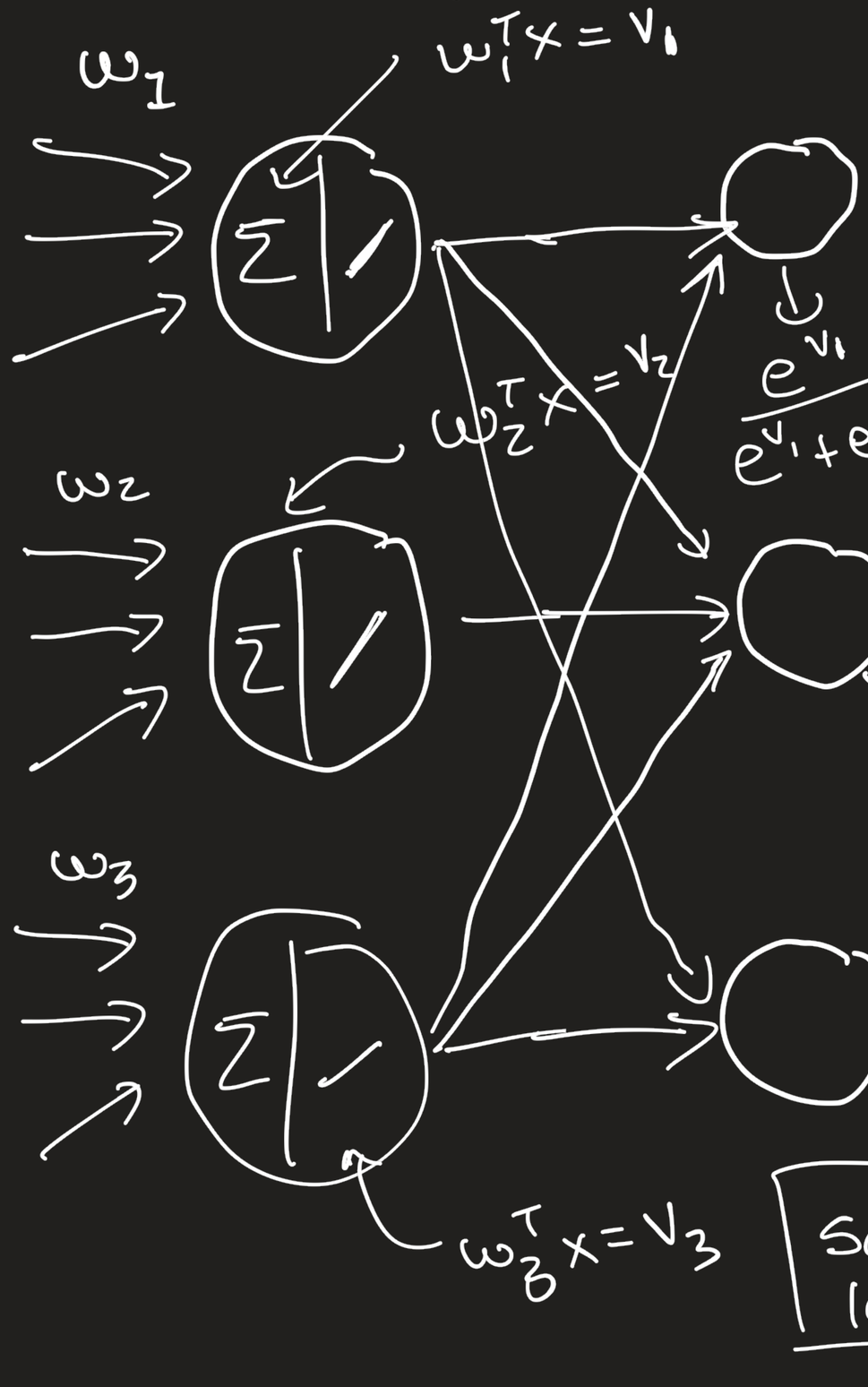
$$\hat{y} = \langle w, x \rangle \qquad \text{loss} = (y - \hat{y})^2$$

Logistic Regression (Binary)



$$\hat{y} = \sigma(\langle w, x \rangle) \qquad \text{loss} = \log(1 + e^{-y \cdot \hat{y}})$$

Multinomial Logistic Reg (3-class)



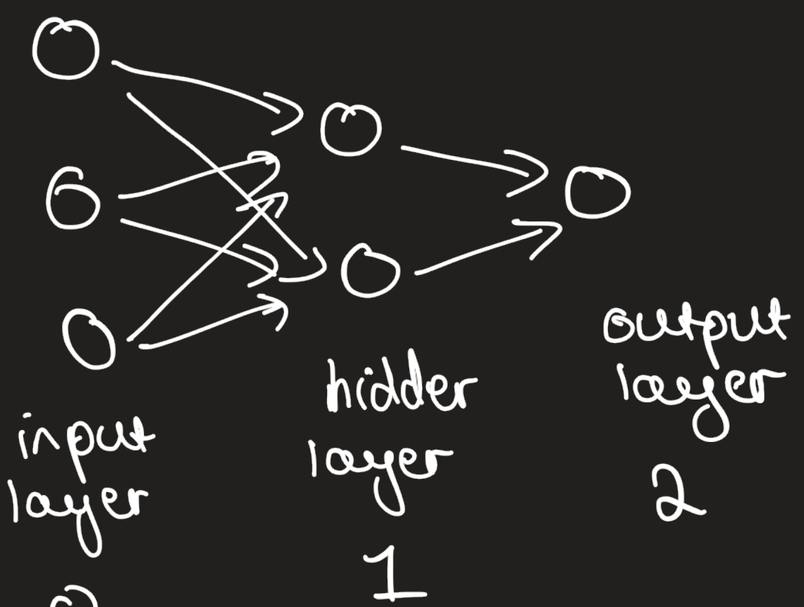
$$p_j = \frac{e^{\omega_j^T x}}{\sum_{i=1}^{3} e^{\omega_i^T x}}$$

$$\text{logits} = Wx = \begin{bmatrix} \omega_1^T x \\ \omega_2^T x \\ \omega_3^T x \end{bmatrix}$$

$$p = \text{softmax}(\text{logits})$$

Vector notation for MLPs

Let our MLP have L layers (L is the output layer, 0 is the input layer)



$L = 2$

and we say layer l has n_l neurons ($n_0 = 3$, $n_1 = 2$, $n_2 = 1$)

if our inputs are in $\mathbb{R}^d$ then $n_0 = d$

if our outputs are in $\mathbb{R}^k$ then $n_L = k$

We write the preactivation values of the neurons on layer l as vectors of length n_l, $a^l \in \mathbb{R}^{n_l}$

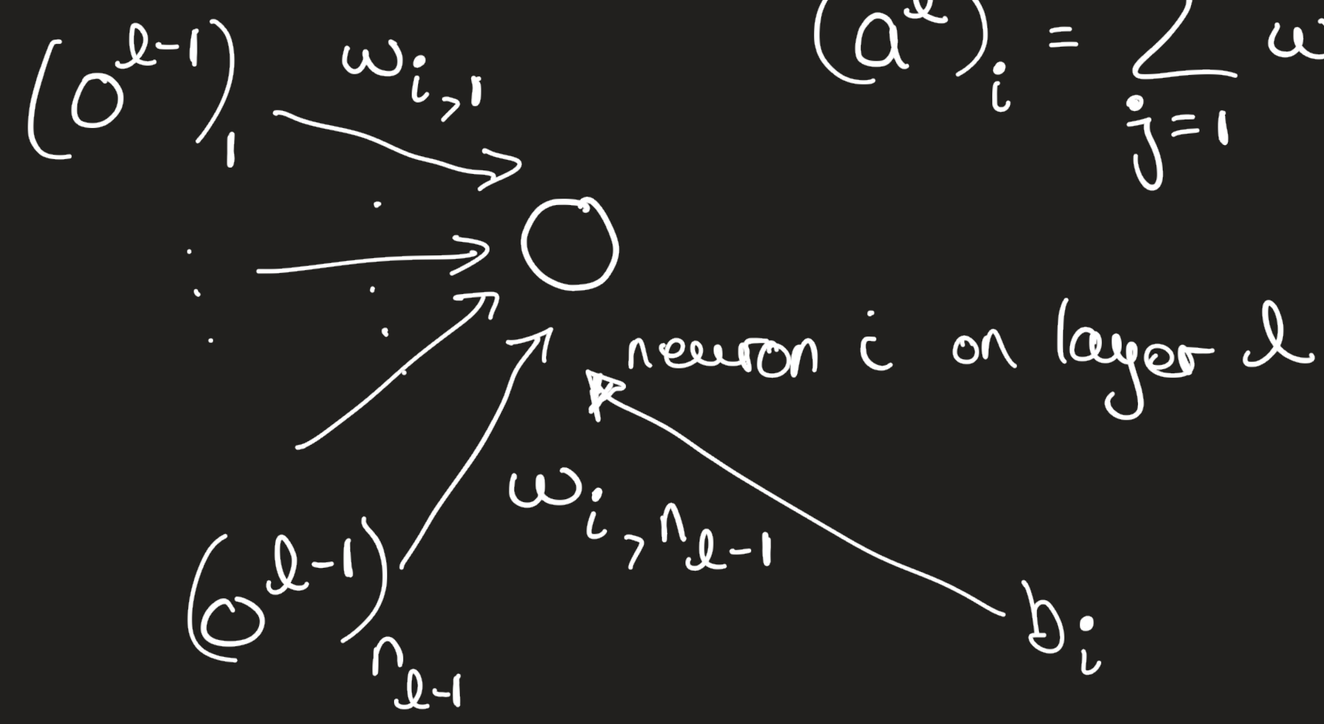
and the activation values on layer l as $o^l \in \mathbb{R}^{n_l}$

We write

$$o^l = \sigma(a^l)$$ meaning entrywise application of the activation function

Expression for a^l

$$(a^l)_i = \sum_{j=1}^{n_{l-1}} w_{i,j}(o^{l-1})_j + b_i$$



In vector notation

$$a^l = W^l o^{l-1} + b^l \quad \text{where} \quad W^l \in \mathbb{R}^{n_l \times n_{l-1}}$$

$a^l \in \mathbb{R}^{n_l}$, $o^{l-1} \in \mathbb{R}^{n_{l-1}}$, $b^l \in \mathbb{R}^{n_l}$

where

$$w^l = \left[w_{ij} \right]_{\substack{i=1,\cdots,n_l \\ j=1,\cdots,n_{l-1}}}$$

so

$$o^l = \sigma(a^l) = \sigma(w^l o^{l-1} + b^l)$$

With this notation, an L-layer MLP is a function of the form

$$\hat{y}(x) = o^L = \sigma(a^L) = \sigma(w^L o^{L-1} + b^L)$$
$$= \sigma(w^L \sigma(w^{L-1} o^{L-2} + b^{L-1}) + b^L)$$
$$= \cdots$$
$$= \sigma(w^2 \sigma(w^{L-1} \cdots (\sigma(w^1 o^0 + b^1) + b^2) \cdots + b^L)$$

- it is parametrized by the kernel/weight matrices of the L-layers and the bias vectors

Training consists of learning $\{w^l, b^l\}$

Training MLPs

Given an L layer MLP

$$f(W^1, \dots, W^L, b^1, \dots, b^L)$$

$$= \frac{1}{n} \sum_{i=1}^{n} \ell(\hat{y}(x_i), y_i) + R(\{W^i, b^i\})$$

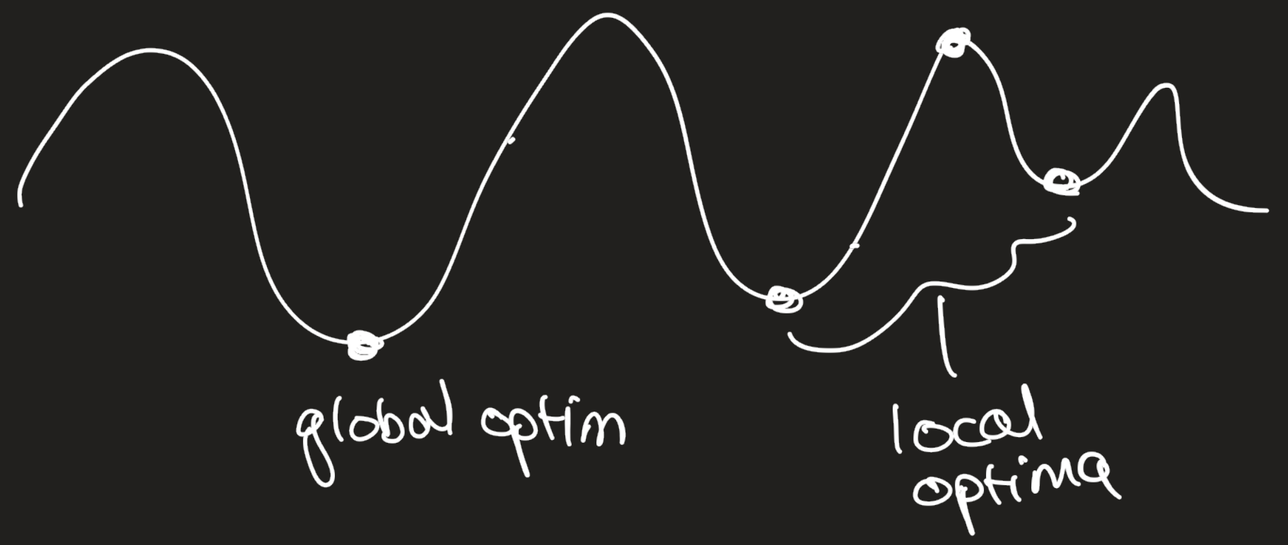
where $\hat{y}(x_i) = O^L(x_i)$

As before (in the CVX case) to train we use a stochastic first-order method: minibatch SGD or minibatch Adam, etc.

Caveat: this is a nonconvex optimization problem in general

We can't guarantee (usually) convergence to optimally performing parameters

Instead we can often guarantee convergence to local optima.



these have different qualities, so initialization matters.

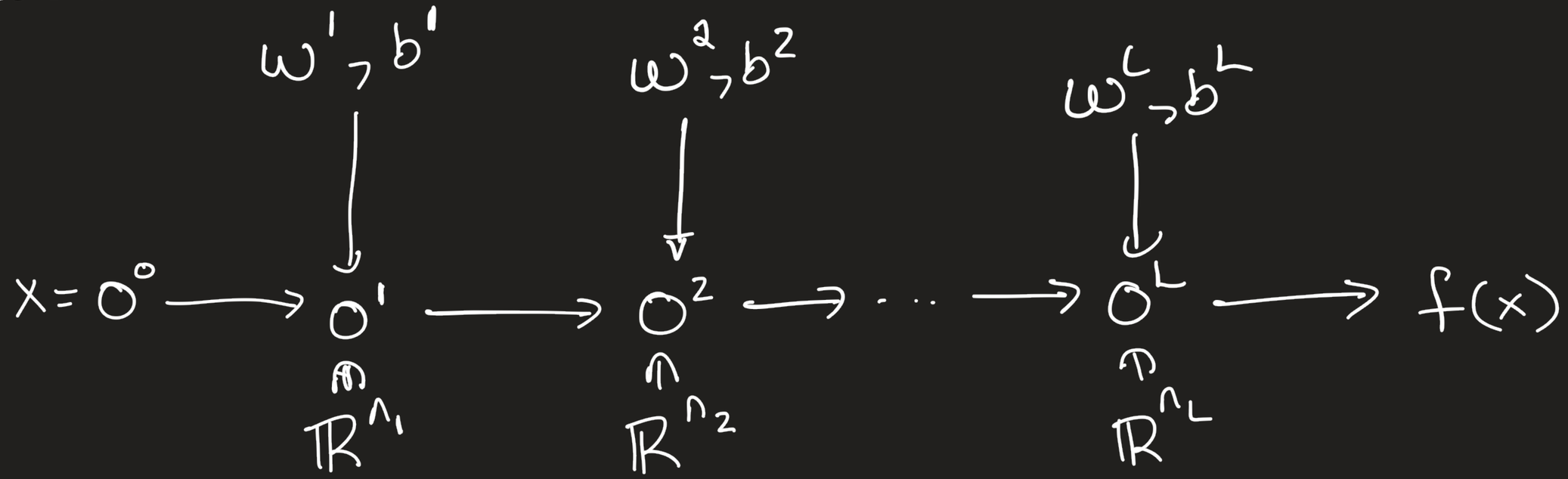
Convergence for nonconvex probs is measured by the size of the gradient $\|\nabla f(w)\|_2 \to 0$

To use these stochastic first-order methods we need to know how to compute

$$\nabla_{W^\ell} f \;\rightarrow\; \nabla_{b^\ell} f \qquad \text{for } \ell = 1, \dots, L$$

$$\nabla_{W^\ell} f \in \mathbb{R}^{n_\ell \times n_{\ell-1}}, \qquad \nabla_{b^\ell} f \in \mathbb{R}^{n_\ell}$$

Idea:



Note that f depends on W^ℓ, b^ℓ only through O^ℓ

So we can the chain rule to go <u>backward</u> along this chain of dependencies to calculate $\nabla_{W^\ell} f, \nabla_{b^\ell} f$

"<u>back propagation</u>" (backprop)

<u>Intuition</u>

(1) "$\dfrac{\partial f}{\partial W^\ell} = \dfrac{\partial f}{\partial O^\ell}\dfrac{\partial O^\ell}{\partial W^\ell}$" ← chain rule

this equation is not mathematically correct

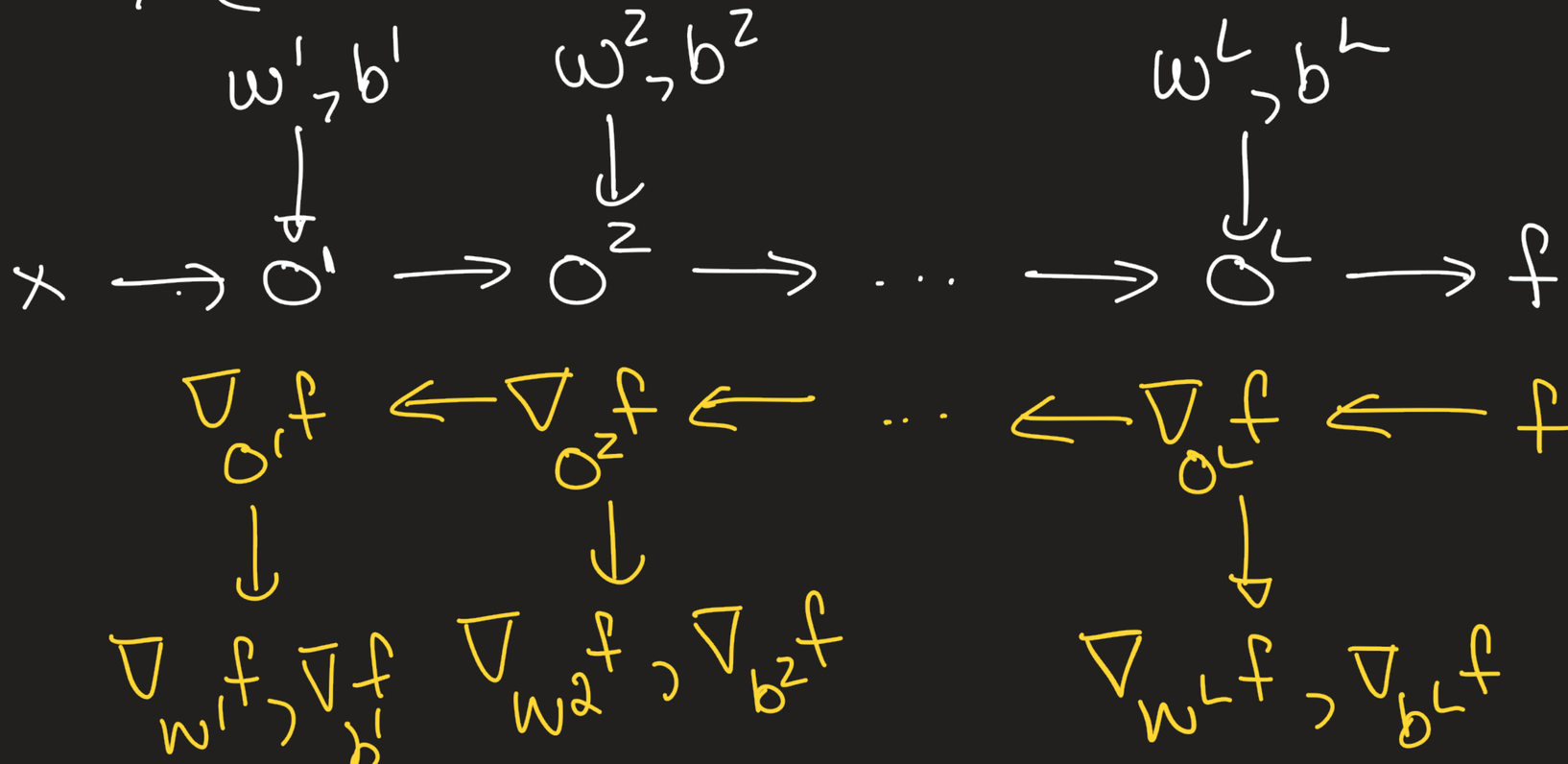
$\ell = 1, \dots, L$

(2) "$\dfrac{\partial f}{\partial b^\ell} = \dfrac{\partial f}{\partial O^\ell}\dfrac{\partial O^\ell}{\partial b^\ell}$"

(3) "$\dfrac{\partial f}{\partial O^\ell} = \dfrac{\partial f}{\partial O^{\ell+1}}\dfrac{\partial O^{\ell+1}}{\partial O^\ell}$" ←

This suggest computing partial derivatives w.r.t. O^L, then $O^{L-1}, \dots, O^1$ working backwards and when we have the dependence on O^l we can compute the dependence on w^l, b^l

To make these relationships mathematically correct, start w/ (3)



Consider $\nabla_{o^\ell} f \in \mathbb{R}^{n_\ell}$

recall $o^{\ell+1} = \sigma(a^{\ell+1}) = \sigma(w^{\ell+1} a^\ell + b^{\ell+1})$

now to compute $\nabla_{o^\ell} f \in \mathbb{R}^{n_\ell}$, use the chain rule:

$$\left[\nabla_{o^\ell} f\right]_i = \sum_{j=1}^{n_{\ell+1}} \left[\frac{\partial f}{\partial o^{\ell+1}}\right]_j \frac{\partial (o^{\ell+1})_j}{\partial (o^\ell)_i}$$

$\vdots$ we will see next time

$$= (w^{\ell+1})^T \cdot \left[\sigma'(a^{\ell+1}) \odot \nabla_{o^{\ell+1}} f\right]$$