Tutorial 7

26 February 2022 11:00

* ℓ_p are convex but not differentiable everywhere.

$f(x) = \|x\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$ $\quad x \in \mathbb{R}^n,\ p \geq 1$

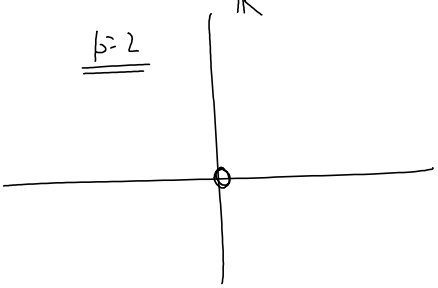
* $p > 1$: convex but not differentiable at $x = 0$

* $p = 1$

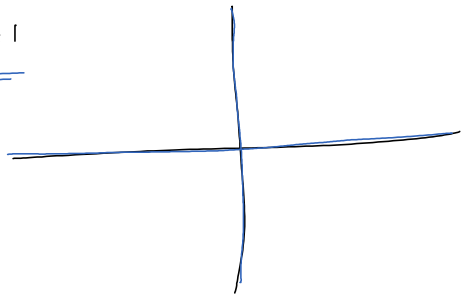
$f(x) = \sum_{i=1}^{n} |x_i|$

convex but not differentiable at any x such that $x_i = 0$ for some i.

$p = 2$ $\quad \mathbb{R}^2$



$p = 1$



Example $p = 2$

$f(x) = \|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$

$\nabla f(x) = \begin{bmatrix} \frac{2x_1}{2\|x\|_2} \\ \vdots \\ \frac{2x_n}{2\|x\|_2} \end{bmatrix} = \frac{x}{\|x\|_2}$ defined everywhere except $x = 0$

Convexity

$f(y) - f(x) \overset{?}{\geq} \nabla f(x)^T (y - x)$

$\|y\|_2 - \|x\|_2 \overset{?}{\geq} \frac{x^T}{\|x\|_2}(y - x) = \frac{x^T y}{\|x\|_2} - \frac{\|x\|_2^2}{\|x\|_2}$

$\|y\|_2 \overset{?}{\geq} \frac{x^T y}{\|x\|_2}$

$$\|y\|_2 \overset{?}{\geq} \frac{x^T y}{\|x\|_2}$$

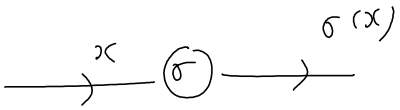
$$\|x\|_2 \|y\|_2 \overset{?}{\geq} x^T y$$

Neural Networks

Neuron

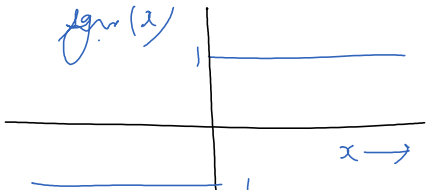
$\sigma : \mathbb{R} \longrightarrow \mathbb{R}$

(a non linear function)

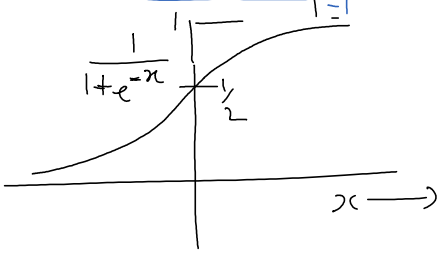


Examples

① Sign function

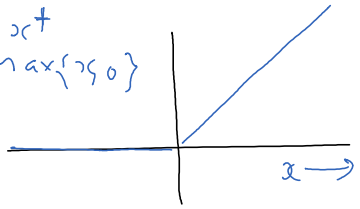


② sigmoid

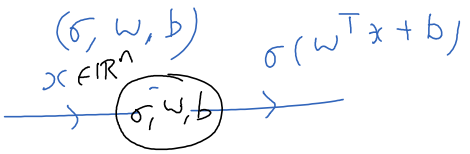


③ ReLU

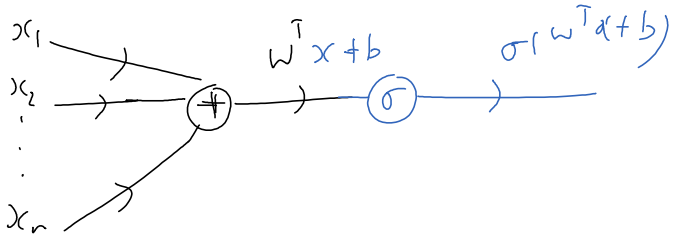
$x^+ := \max\{x, 0\}$



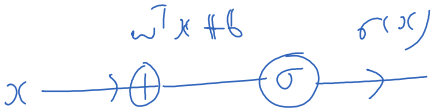
Neurons with vector inputs



$\sigma : \mathbb{R} \longrightarrow \mathbb{R}$



what does a neuron do



what does a neuron do

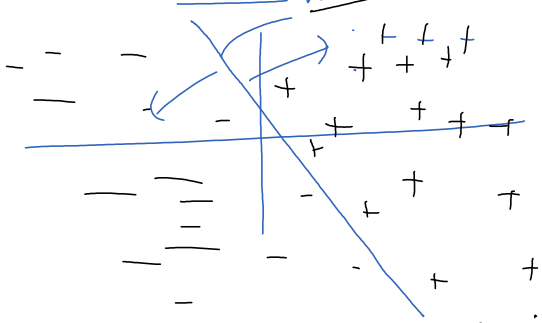
(σ, w, b)

$\downarrow$

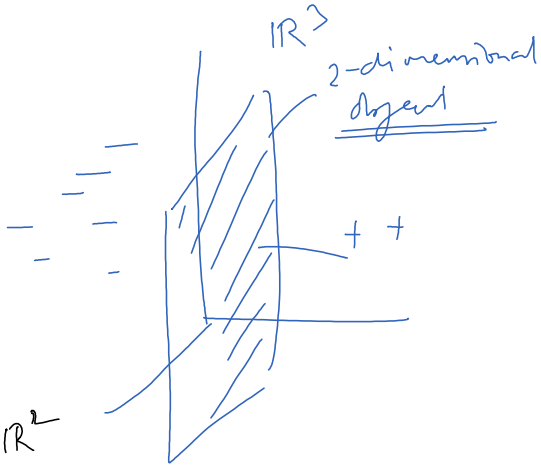
$\sigma(\cdot)$

$$\sigma_{w,b}(x) = \begin{cases} 1 & \text{if } w^T x \geq -b \\ -1 & \text{if } w^T x < -b \end{cases}$$

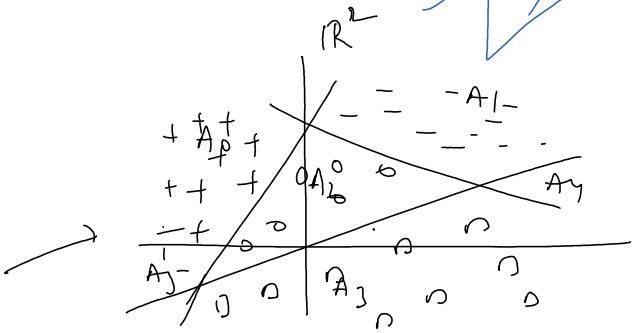
$\in \mathbb{R}^n$



A single neuron has divided $\mathbb{R}^n$ into two $(n-1)$ dimensional spaces by drawing a hyperplane through it.



what we want

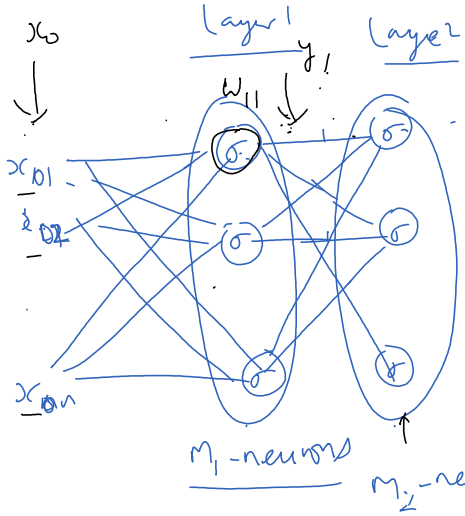


equivalent to implementing $f(x) = i \quad \forall x \in A_i \quad \forall i = 0, 1, \ldots, 4$

Multilayer NNs allow us to divide n-dimensional space in more complicated ways.

in more complicated ways.

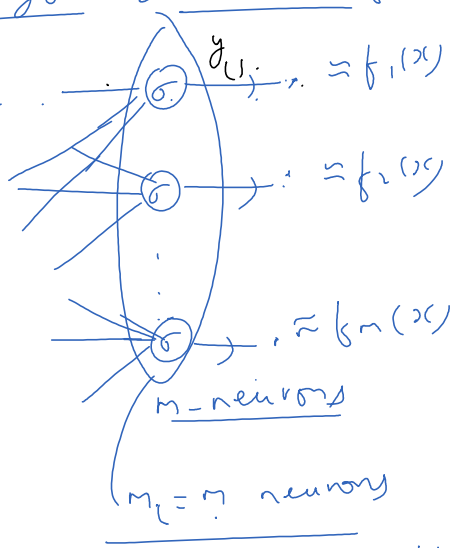
Layers



$x \in \mathbb{R}^n$

$f: \mathbb{R}^n \to \mathbb{R}^m$

- goal is: realize f.



- what is in our control

W_{11} - weight vector of the first neuron of the first layer

b_{11} bias of the first neuron of the first layer

- $m_1, \dots, m_{L-1}$
- w_{kl^i}, $1 \le k \le L$, $1 \le i \le m_k \ \forall k$
- b_{ki} $1 \le k \le L$, $1 \le i \le m_k \ \forall k$

} we can configure them

- we assume that $m_1, \dots, m_{L-1}$ are given
- W_{ki}, b_{ki} are configured using gradient descent

Neural Network implements a function $\tilde{f}: \mathbb{R}^n \to \mathbb{R}^m$

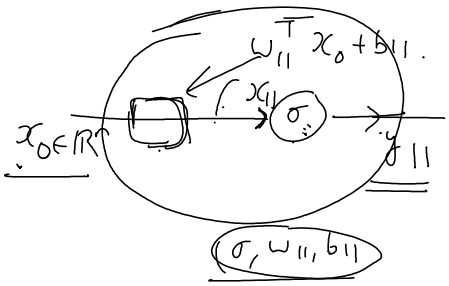
goal: $\tilde{f}(\cdot) \approx f(\cdot)$ via tuning w_{ij} and b_{ki}

understanding $\tilde{f}(\cdot)$

$$\begin{pmatrix} \cdots & w_{11}^T & \cdots \end{pmatrix} \begin{pmatrix} x_{01} \\ \vdots \end{pmatrix} \begin{pmatrix} b_{11} \\ \vdots \end{pmatrix} \begin{pmatrix} x_{11} \\ \vdots \end{pmatrix}$$

$$\begin{bmatrix} -w_{11}^T- \\ -w_{12}^T- \\ \\ -w_{1m_1}^T- \end{bmatrix}}_{W_1} \begin{bmatrix} x_{01} \\ \vdots \\ x_{0n} \end{bmatrix} + \underbrace{\begin{bmatrix} b_{11} \\ b_{12} \\ \\ b_{1m_1} \end{bmatrix}}_{b_1} = \begin{bmatrix} x_{11} \\ x_{12} \\ \\ x_{1m_1} \end{bmatrix}$$

$$\begin{bmatrix} x_{11} \\ x_{12} \\ \\ x_{1m_1} \end{bmatrix} \xrightarrow{\sigma} \begin{bmatrix} y_{11} \\ y_{12} \\ \\ y_{1m_1} \end{bmatrix}$$



x_1: input to the first layer

y_1: output of the first layer

$$\left.\begin{aligned} x_1 &= w_1 x_0 + b_1 \\ y_1 &= \sigma(x_1) \end{aligned}\right\}$$ mathematical representation of the first layer.

following similar steps $\forall\ 1 \le k \le L$

$$\left.\begin{aligned} x_k &= w_k y_{k-1} + b_k \\ y_k &= \sigma(x_k) \end{aligned}\right\}$$ (with the understanding that $y_0 = x_0$)

$$\tilde{f}(x_0) = y_L$$

How are we converging f?

$f(x) =$ $x \in \mathbb{R}^n$

We are given N input-output pairs

$x_0^{(1)}$ $f(x_0^{(1)})$

$x_0^{(2)}$ $f(x_0^{(2)})$

$\vdots$

$\cdots$

} training data

$x_0^{(N)}$ $f(x_0^{(n)})$

* we want outputs $y_L^{(1)}, y_L^{(2)}, \ldots, y_L^{(N)}$ to be close to $f(x_0^{(1)}), \ldots$ $f(x_0^{(n)})$

$$J(W_1, \ldots, W_L, b_1, \ldots, b_L) = \sum_{n=1}^{N} L\left(y_L^{(n)}, f(x_0^{(n)})\right) \qquad \sum_{i=1}^{N} f_i(x)$$

total discrepancy

objective would be to minimize this

Gradient descent start with $W_1(0), \ldots, W_L(0), b_1(0), \ldots, b_L(0)$

and

$$W_l(t+1) = W_l(t) - \epsilon_t \nabla_{W_l} J(W_1(t), \ldots, W_L(t), b_1(t), \ldots, b_L(t))$$
$$b_l(t+1) = b_l(t) - \epsilon_t \nabla_{b_l} J(W_1(t), \ldots, W_L(t), b_1(t), \ldots, b_L(t))$$

- we will focus on gradient of one term
- will drop the superscript (n) for convenience

systematic gradient computation is called back propagation

$$\frac{d}{dx} f(g(x)) = g'(x) f'(g(x))$$

Back propagation

(I) compute $\frac{\partial L}{\partial y_{Li}}$ $i = 1, \ldots, m_L$

(II) For $l = L, L-1, \ldots, 1$ compute

For $l = L$ we

$$\frac{\partial L}{\partial W_{l,ij}} = \frac{\partial L}{\partial y_{li}} \sigma'(x_{li}) y_{l-1,j} \qquad i = 1, \ldots, m_l \quad j = 1, \ldots, m_{l-1}$$

$l = L$

we can compute

$$\frac{\partial \tilde{L}}{\partial w_{lij}} = \frac{\partial \tilde{L}}{\partial y_{li}} \sigma'(x_{li}) u_{l-1,j} \qquad j = 1, \ldots, m_{l-1}$$

$$\frac{\partial \tilde{L}}{\partial b_{li}} = \frac{\partial \tilde{L}}{\partial y_{li}} \sigma'(x_{li}) \qquad i = 1, \ldots, m_l$$

$$\frac{\partial \tilde{L}}{\partial y_{(l-1)i}} = \sum_{k=1}^{m_l} \frac{\partial \tilde{L}}{\partial y_{lk}} \sigma'(x_{lk}) w_{lki} \qquad i = 1, \ldots, m_{l-1}$$