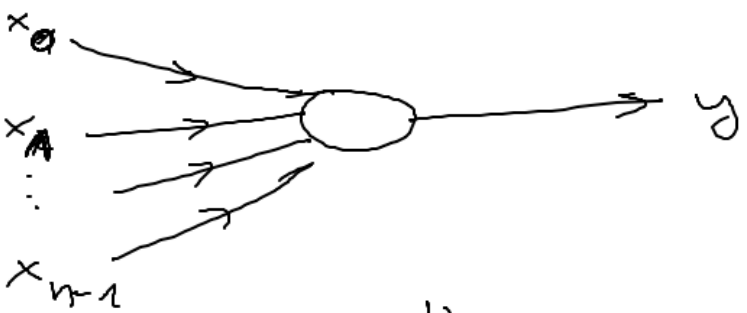
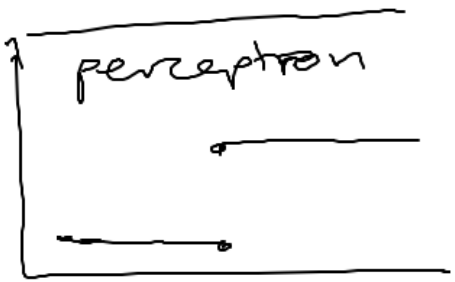
Neural nets

Neuron

is a function of a form





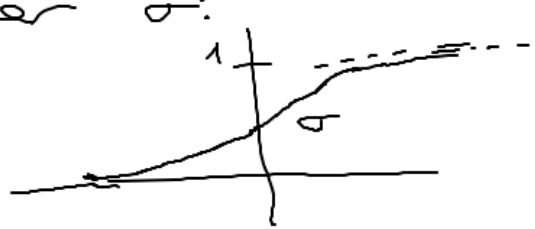
$$\mathbb{R}^n \ni (x_0, x_1, \dots, x_{n-1}) \longmapsto \sigma\left(\underbrace{\alpha_0 x_0 + \dots + \alpha_{n-1} x_{n-1}}_{\text{linear}} - \alpha_n\right) \in \mathbb{R}$$

$\alpha_0, \dots, \alpha_{n-1}$: weights; α_n: bias

$$\begin{cases} x_n = -1 \\ \alpha_n x_n = -\alpha_n \end{cases}$$

common choices for σ:

$(0,1) \ni \sigma(x) = \dfrac{1}{1+e^{-x}}$ sigmoid



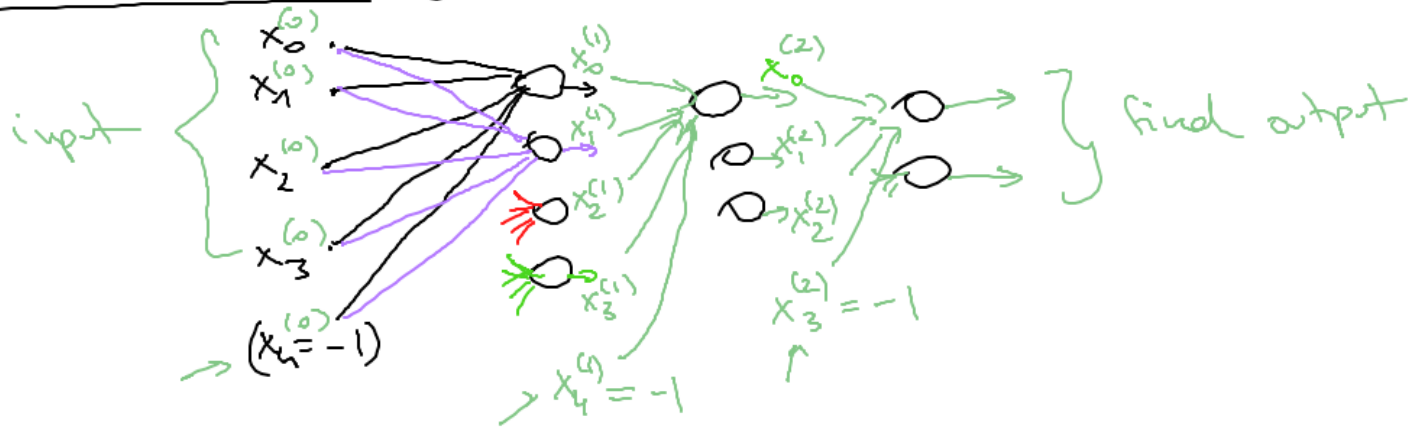
$\sigma(x) = \max(x, 0)$ ReLU

One usually takes $x_n := -1$ and then

$$(x_0, \dots, x_{n-1}) \longmapsto \sigma\left(\sum_{k=0}^{n} \alpha_k x_k\right) \leftarrow$$

so we get a nicer formula.

Neural network (dense)



A neural network is a function which is a composition of neurons

Universality theorem (there are many versions)

Thm. (Cybenko)

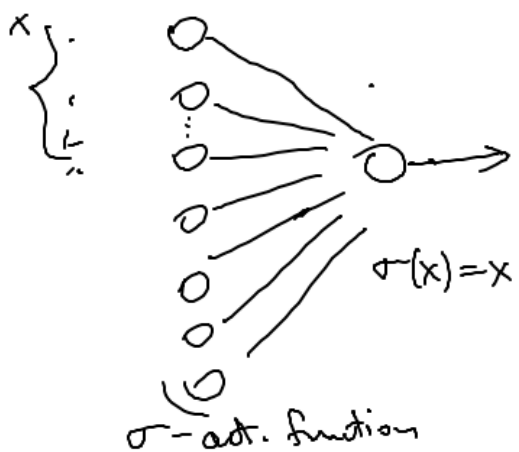
Let σ – sigmoidal, continuous function.
($\sigma(-\infty)=0$, $\sigma(\infty)=1$)

Then the finite sums of the form

$$G(x)=\sum_{j=1}^{N} \alpha_j \, \sigma(\underbrace{y_j^T}_{\text{weights}} x + \underbrace{\theta_j}_{\text{bias}}) \quad , x \in [0,1]^n$$

are dense in $\left(C([0,1]^n), \|\cdot\|_{\sup}\right)$.

Proof is short, but some background in Functional Analysis is needed.



Neural nets are often used in classification problems:

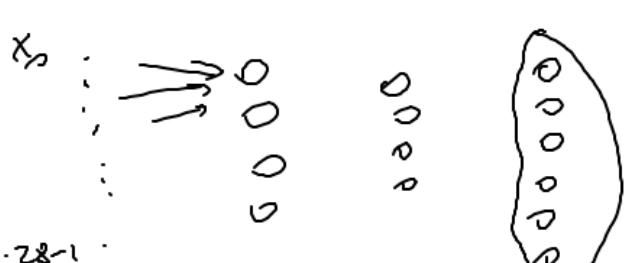
the input is of one of the given N classes

the aim: given input, decide to what class it belongs to

MNIST database of hand-written digits:

the picture is a point in $\mathbb{R}^{28 \cdot 28}$

60k pictures



expected output for digit "7"

MNIST: last layer

$0 \to (0,1)$

$1 \to (0,1)$

7

$9 \to (0,1)$

0.1

0.05

0.8 ← answer: "7"

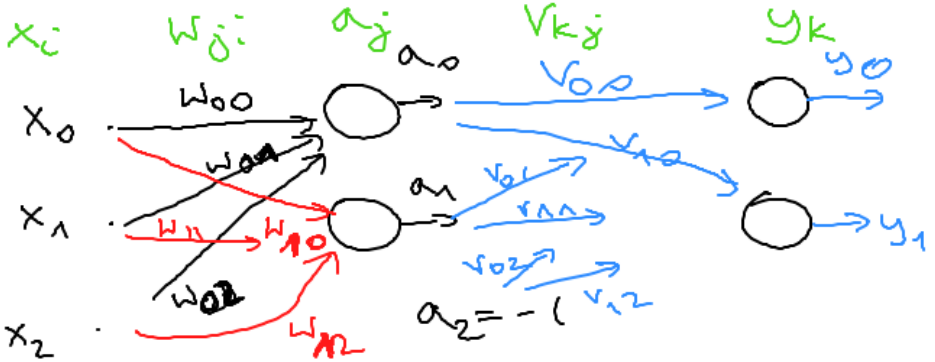
0.2

last layer is of size N

(and sigmoid is used as an activation function)

argmax

Neural net with 2 layers with 2 neurons in each layer.



$$a_0 = \sigma(\underbrace{w_{00}x_0 + w_{01}x_1 + w_{02}x_2}_{net_0})$$

$$a_1 = \sigma(\underbrace{w_{10}x_0 + w_{11}x_1 + w_{12}x_2}_{net_1})$$

$$W = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \end{bmatrix}$$

$$W \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} net_0 \\ net_1 \end{bmatrix}$$

$$y_0 = \sigma(v_{00}a_0 + v_{01}a_1 + v_{02}\underbrace{a_2}_{-1})$$

Let's suppose that for some input $\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$ we expect the net to output $\begin{bmatrix} t_0 \\ t_1 \end{bmatrix}$ but actually it outputs $\begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$

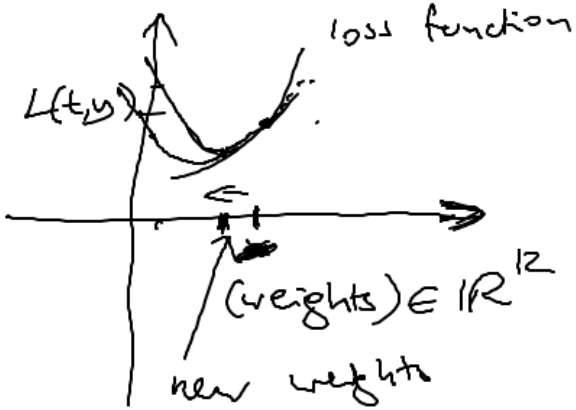
We introduce a loss function

$$L(t, y) = \frac{1}{2}\|t-y\|_2^2 = \frac{1}{2}\sum_{k=0}^{N-1}(t_k - y_k)^2$$

(t: target, y: actual output)

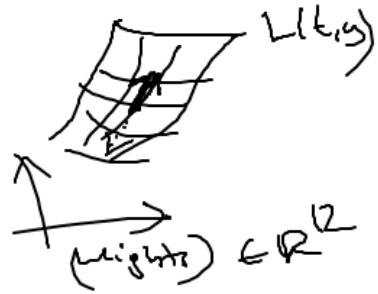
We want to adjust the weights so that the loss function gets smaller.

general idea: calculate the gradient of L with respect to the weights $(w_{00}, \dots, v_{12})$ and then:



$$\text{(new weights)} = \text{(old weights)} - c\,(\quad)$$

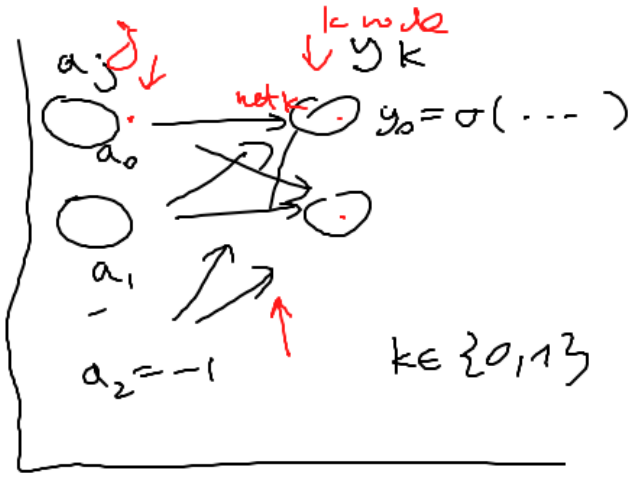
↑ learning rate, e.g. $c = 0.01$



$$\text{(new } v_{kj}) = \text{(old } v_{kj}) - c\,\frac{\partial L}{\partial v_{kj}}$$

Fix k_0, j_0 and let $v = v_{k_0 j_0}$

$$L(t,y) = \frac{1}{2}\sum_k (t_k - y_k)^2 = \frac{1}{2}\sum_k \left(t_k - \varphi\left(\sum_j v_{kj} a_j\right)\right)^2$$



$$\frac{\partial L}{\partial v} = \sum_k \left(t_k - \varphi\left(\sum_j v_{kj} a_j\right)\right) \cdot \frac{\partial}{\partial v}\left(-\varphi\left(\sum_j v_{kj} a_j\right)\right) =$$

$$= -\sum_k \left(t_k - \varphi\left(\sum_j v_{kj} a_j\right)\right) \cdot \varphi'\left(\sum_j v_{kj} a_j\right) \cdot \frac{\partial}{\partial v}\left(\sum_j v_{kj} a_j\right) =$$

ex. $k_0 = 0, j_0 = 1$

$\sum_{k=0}^{1} \sum_{j=0}^{2} v_{kj}$

v_{01}

$$= \left(\varphi\left(\sum_j v_{k_0 j} a_j\right) - t_{k_0}\right) \varphi'\left(\sum_j v_{k_0 j} a_j\right) \cdot a_{j_0}$$

"error" at node k_0 — $\varphi'(\text{net}_{k_0})$ — for sigmoid $= y_k(1-y_k)$

$\frac{\partial}{\partial v_{01}} v_{02} = 0$

Fix i_0, j_0, let $w = w_{j_0 i_0}$

$$a_j = \sigma\left(\sum_i w_{ji} x_i\right) \Leftarrow$$

$$\frac{\partial L}{\partial w} = \sum_k \left(t_k - \varphi\left(\sum_j v_{kj} a_j\right)\right) \cdot \frac{\partial}{\partial a_{j_0}}\left(-\varphi\left(\sum_j v_{kj} a_j\right)\right) \cdot \underbrace{\frac{\partial a_{j_0}}{\partial w}}_{\text{0 if } j \neq j_0} =$$

$\frac{\partial}{\partial x} y = 0$

$\frac{\partial}{\partial x} x = 1$

"δ signal" (k)

$$= \sum_k \left(t_k - \varphi\left(\sum_j v_{kj} a_j\right)\right) \varphi'\left(\sum_j v_{kj} a_j\right) v_{kj_0} \cdot \sigma'\left(\sum_i w_{j_0 i} x_i\right) \cdot x_{i_0}$$

net$_{j_0}$

(counterpart of) the 'error' at node j_0

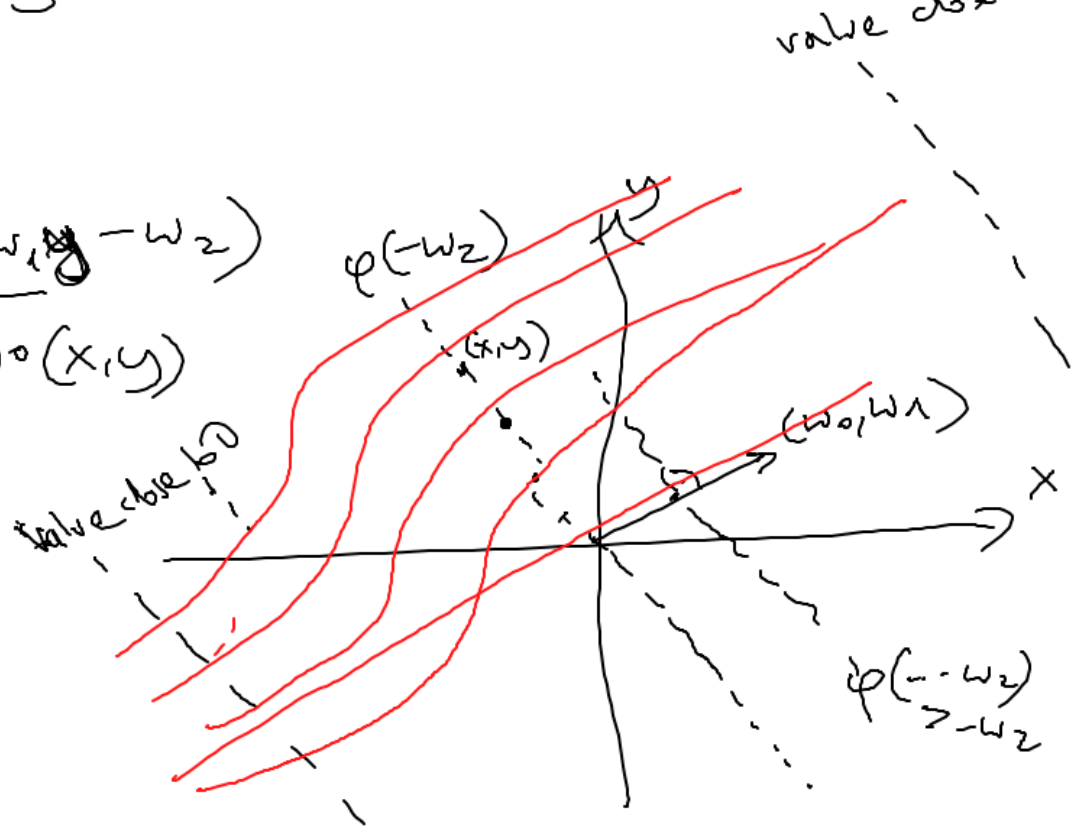
backpropagation

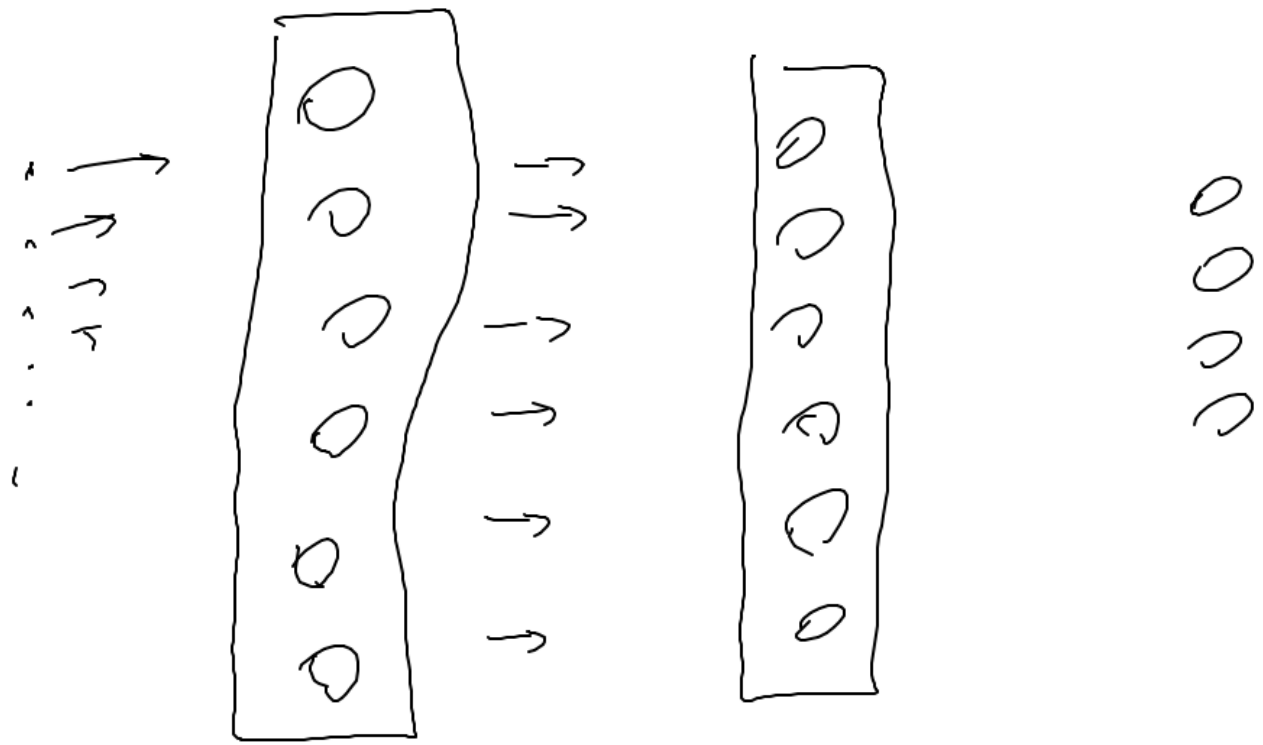
For $\varphi(x) = \frac{1}{1+e^{-x}}$: $\varphi'(x) = \varphi(x)(1-\varphi(x))$

On the ~~webpage~~: an example of forward and back propagation

---

How does the function defined by a neuron look like?





layer is a (convenient) basic building block