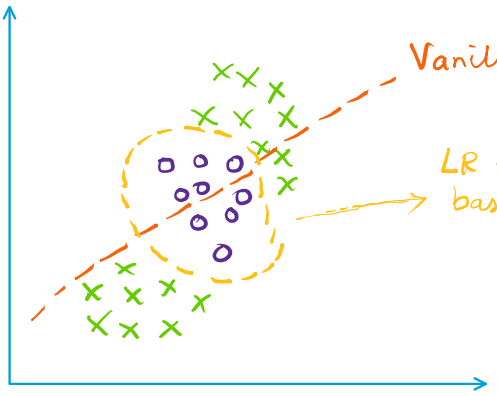
Lecture 11: Neural Networks

16 February 2024 11:06

Example 1:

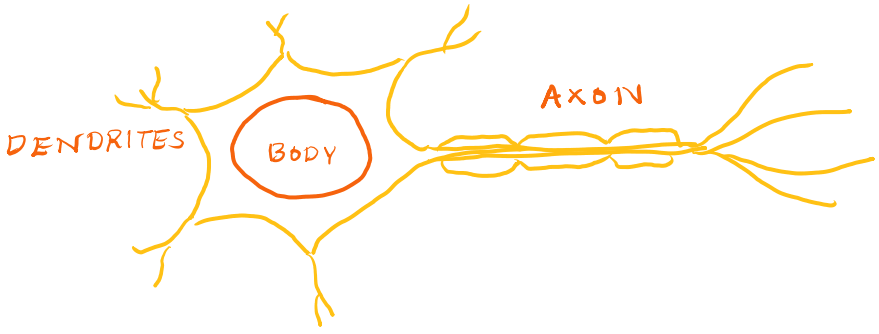


The underlying philosophy of Neural Networks is to come up with some basis fn $\phi(x)$ without explicitly programming it.

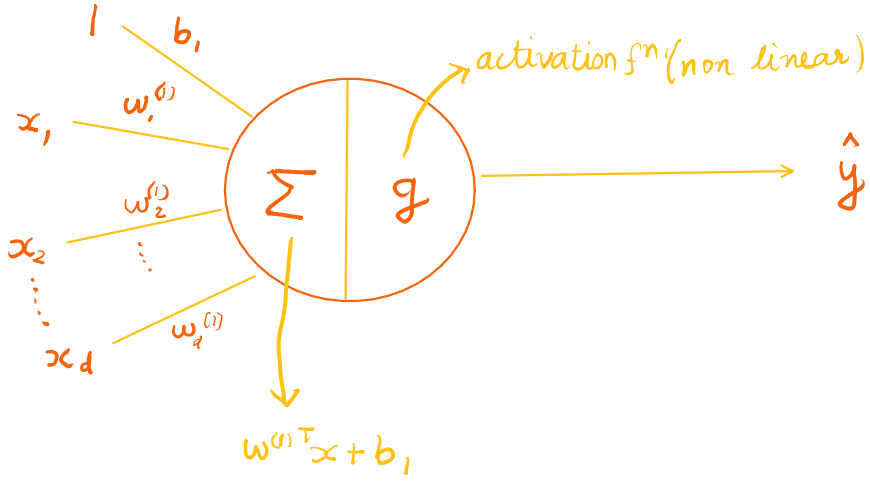
Origin of NN: Started in 1943 then progressed during 60's & 80's.

NN's are inspired by Neurons.

NEURON:



NEURAL NETWORK:



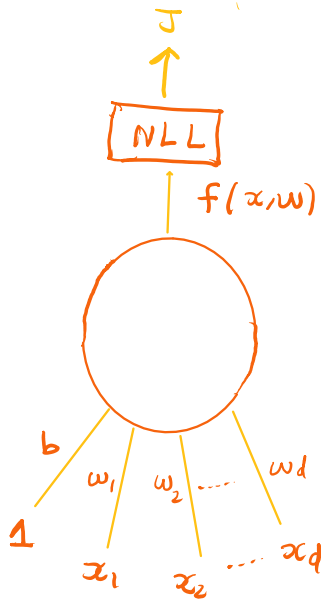
Logistic regression using NN

$$f(x,w) = g(w^T x + b)$$

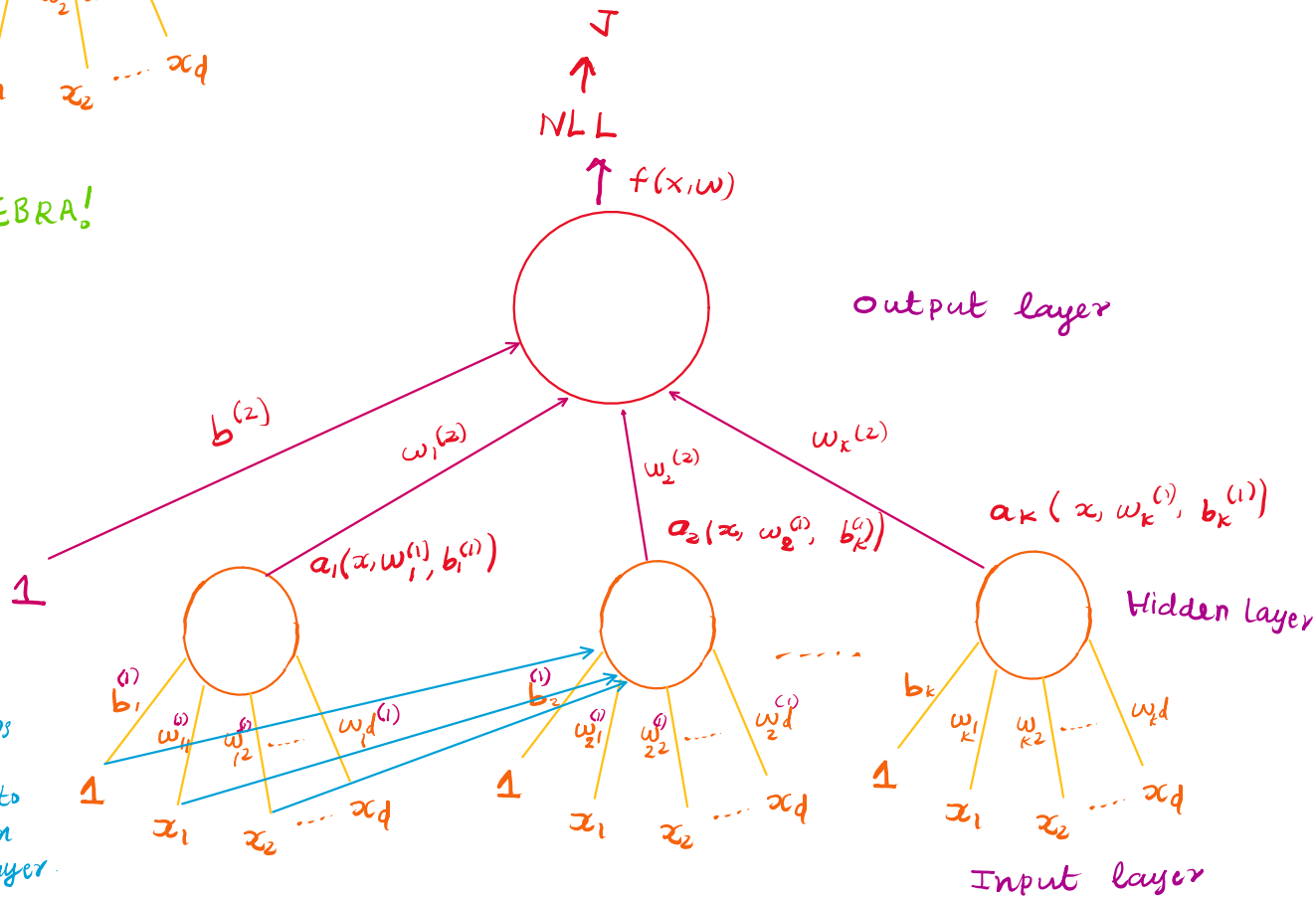
$$g = \sigma(\text{sigmoid})$$

$$NLL_i = -\left[y_i \log f(x_i,w) + (1-y_i)\log(1-f(x_i,w))\right]$$

(cross entropy)



Use GEOGEBRA!

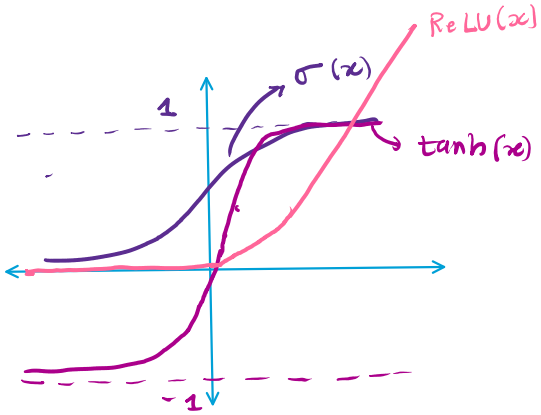


GOAL: To learn the w's and b's of every layer to minimize the loss.

Popular Activation fns

i) Sigmoid $\sigma(x) = \dfrac{1}{1+e^{-x}}$

ii) Hyperbolic tan : $\tanh(x) = \dfrac{e^{2x}-1}{e^{2x}+1}$
(scaled sigmoid)



ii) Rectified linear units:

$$ReLU(x) = \max\{0, x\}$$

Vanishing Gradient problems: For deeper neural networks, the gradient vanishes for large 'x'.

Gradient doesn't vanishes for ReLU fn, hence its desirable for large x.

Training NN:

Given a data set $D = (x_i, y_i)_{i=1}^{n}$ find the values of w's and b's that minimize the loss fn. (Fit forward NN)

STEP 1: Define a loss function $J(\underbrace{w, b}_{\theta})$. (eg: cross entropy loss)

STEP 2: $J(\theta) = \sum_{i=1}^{n} \ell(NN(x_i, \theta), y_i)$ (here, $f(x_i, \theta) = NN(x_i, \theta)$)

{eg: i) $p(y_i = 1 | x_i, \theta)$ ii) soft max fn.}

$$\left\{\text{eg: } \ell(NN(x_i, \theta), y_i) = \left[-y_i \log(NN(x_i, \theta)) + (1 - y_i) \log(1 - NN(x_i, \theta))\right]\right\}$$

Once the loss fn is defined, we use SGD to optimize it.

NN training algorithm:

- Inputs: $NN(x, \theta)$, training examples $x_1, x_2, \ldots x_n$, labels $y_1, \ldots y_n$ and a loss function ℓ
- Randomly intialize θ
- Do until stopping criteria:
  - Pick randomly an example (x_i, y_i) → choose a batch B of examples $B \subseteq \{1, \ldots n\}$
  - Compute gradient of ℓ, $\nabla_\theta \ell(x_i, y_i)$ → $\sum_{i \in B} \nabla_\theta \ell(x_i, y_i)$
  - $\theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta \ell$

Mini-batch GD, instead of SGD

- Return θ_{t+1}

How will we compute $\nabla_\theta \ell$ efficiently? Using backpropagation, which uses the chain rule of differentiation in a clever way!

Scalars: $\frac{dg}{dx} = \frac{du}{dx}\frac{dg}{du}$, where $g = g(u)$ and $u = u(x)$

Scalars: (g, x, u are scalars)

$$\frac{dg}{dx} = \frac{du}{dx}\frac{dg}{du}$$, where $\bar{g} = g(u)$ and $u = u(x)$

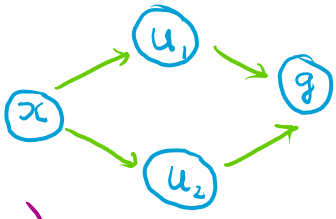
Vectors: (g, x are scalars, but $u = (u_1, u_2)$ is vector)

$$\frac{\partial g}{\partial x} = \frac{\partial u_1}{\partial x}\cdot\frac{\partial g}{\partial u_1} + \frac{\partial u_2}{\partial x} + \frac{\partial g}{\partial u_2}$$

$$= \frac{\partial u}{\partial x}\cdot\frac{\partial g}{\partial u}$$

$\frac{\partial u}{\partial x}$ → $\left(\frac{\partial u_1}{\partial x}, \frac{\partial u_2}{\partial x}\right)$

$\frac{\partial g}{\partial u}$ → $\left(\frac{\partial g}{\partial u_1}, \frac{\partial g}{\partial u_2}\right)$



Assume g, x, u to be all vectors:

$$\frac{\partial g}{\partial x} = \frac{\partial u}{\partial x}\cdot\frac{\partial g}{\partial u}$$

where, $\frac{\partial u}{\partial x} = \begin{bmatrix} \frac{\partial u_1}{\partial x_1} & \frac{\partial u_2}{\partial x_1} & \cdots & \frac{\partial u_k}{\partial x_1} \\ \frac{\partial u_1}{\partial x_2} & \cdots & \cdots & \frac{\partial u_k}{\partial x_2} \\ \vdots & & & \end{bmatrix}$

and, $\frac{\partial g}{\partial u} = \begin{bmatrix} \frac{\partial g_1}{\partial u_1} & \cdots & \frac{\partial g_p}{\partial u_1} \\ \vdots & & \vdots \\ \frac{\partial g_1}{\partial u_k} & \cdots & \frac{\partial g_p}{\partial u_k} \end{bmatrix}$

Back propagation: $\frac{\partial g_1}{\partial x_1} = \sum_{j=1}^{k} \frac{\partial u_j}{\partial x_1}\cdot\frac{\partial g_1}{\partial u_j}$