1

Multilayer Perceptrons

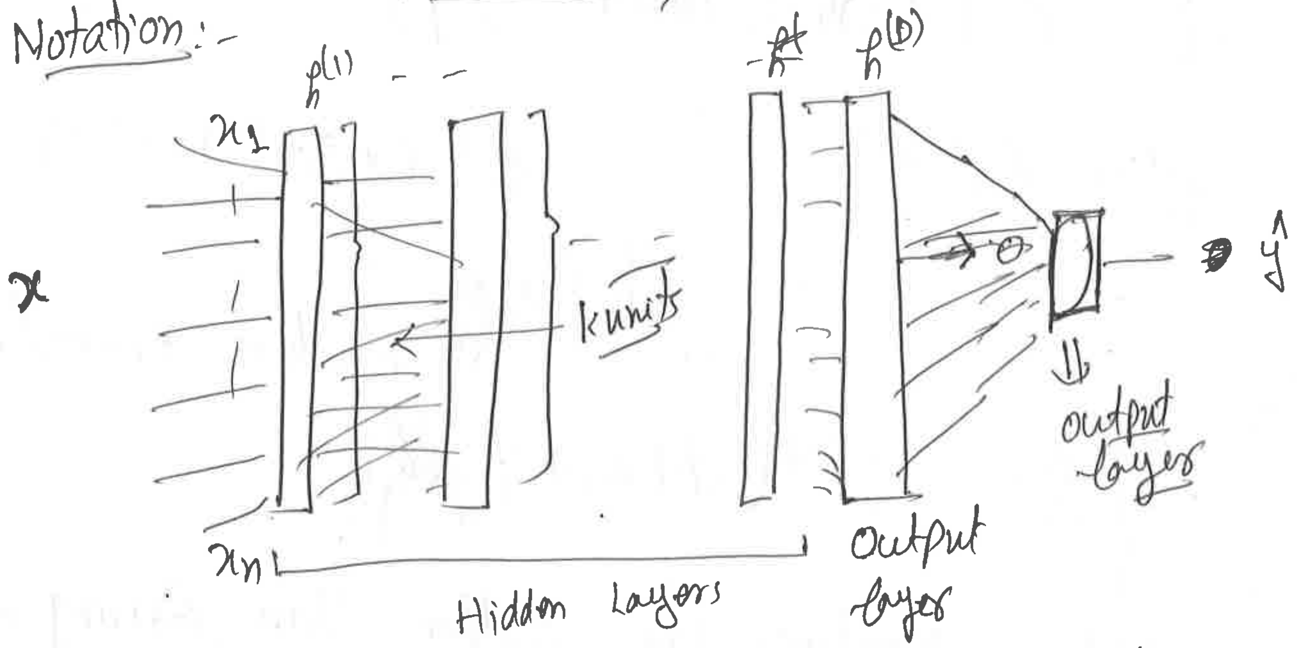
(Deep Learning)

COL 865

July 27, 2017

Forward Connections

Notation:-



Each layer transforms vector x onto a new vector x'. At the final layer, output $\hat{y}$ is produced. $\theta = (\theta^{(1)} \dots \theta^{(D)})$

$$\hat{y} = f(x; \theta, \theta^{(o)}) = f^{(o)}[h^{(D)}(h^{(D-1)}(\dots h^{(1)}(x, \theta^{(1)})\dots); \theta^{(o)}]$$

⇓

Approximate $y = f^*(x)$

$$\Theta = (\theta, \theta^{(o)})$$

Questions

Design questions

- What is the form of each $h^{(i)}$?] Activation function
- What is the form of $f^{(o)}$?] output function
- Cost [What do we want to optimize? i.e. $L[\hat{y}, y; \theta]$] Loss (e.g. likelihood)
- ~~How do we optimize?~~
- How do we optimize $L(\hat{y}, y)$?] Back propagation

Note:-

$f^{(k)}(f^{(k-1)} \cdots f^{(1)}(x)) = \phi(x;\theta)$

↓ (Non-linear) Feature transformation

$y = f^{(0)}(\phi(x;\theta);\omega)$

$f^{(k)}(f^{(k-1)} \cdots f^{(1)}(x;\theta^{(1)});\theta^{(2)}\cdots);\theta^{(k)})$

$= \phi(x;\theta)$

↓ Feature transformation

$$y = f^{(0)}(\phi(x;\theta);\omega)$$

Note:- Rather than having a single (non-linear) feature transformation, (e.g. SVMs), where ϕ is expressed as a composition of several non-linear transformations.

Kernel Function

$k(x,x') = \phi(x)\cdot\phi(x')$

what is form of each

Intuition:- Mimicks the computation in brain?

Basis of transfer learning

Two parts:- (a) Feature transformation] Hidden layer

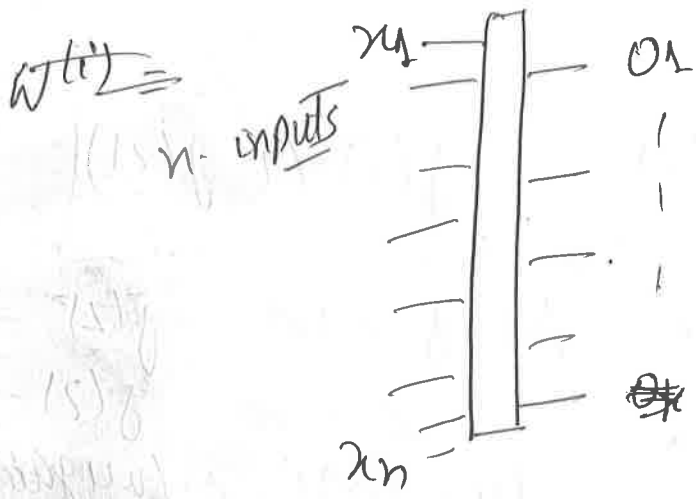
(b) Output processing] output layer

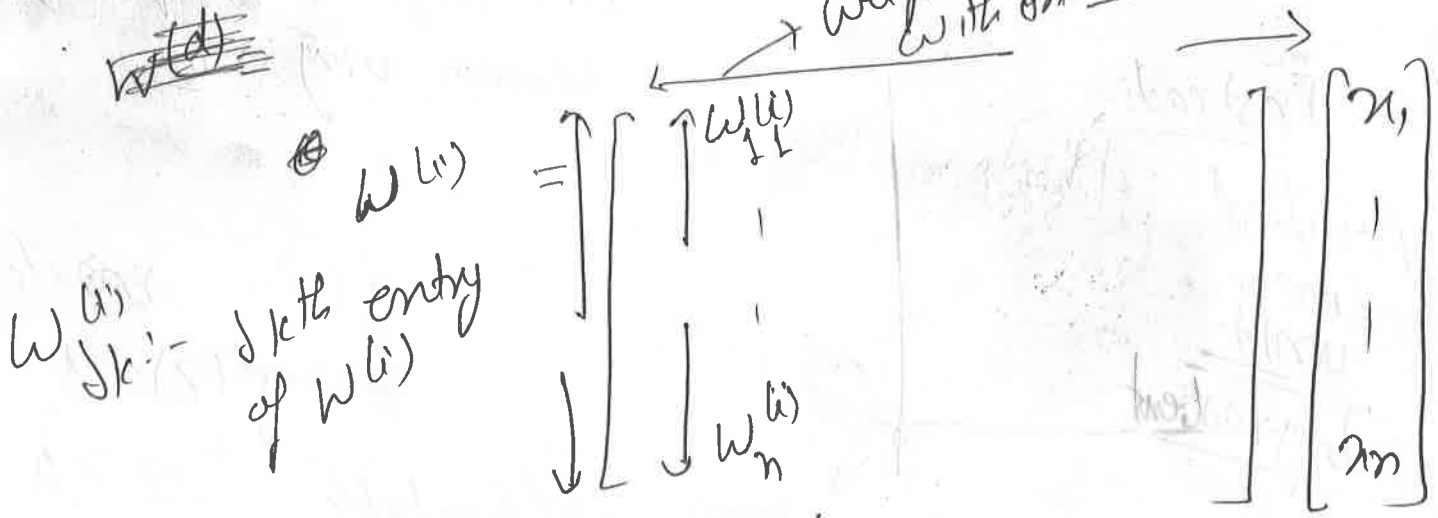
Each $f^{(i)}$ implements a non-linear function.

(2)

$h^{(i)}(x; \theta^{(i)}) = g[W^{(i)T} x + b^{(i)}]$

- $h^{(i)}$ ⇓ Vector valued function
- g ⇓ Non-linearity (activation function) ↳ can be shared across layers
- $W^{(i)T} x + b^{(i)}$ ⇓ Linear function





$W_{jk}^{(i)}$:- $j$kth entry of $W^{(i)}$

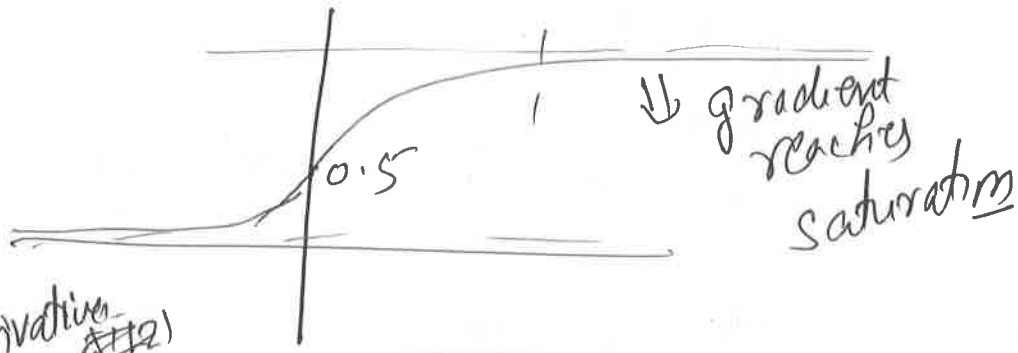
$b^{(i)} = \begin{bmatrix} b_1^{(i)} \\ \vdots \\ b_e^{(i)} \end{bmatrix}$

$x' = g(W^{(i)T} x + b^{(i)})$ element wise application

$$x'_k = g\left[\sum_{j=1}^{n} W_{jk}^{(i)} x_j\right] = g[W_{:k}^{(i)} x + b_k^{(i)}]$$

g: activation unit (non-linearity)

choices:- $g(z) = \frac{1}{1+e^{-z}}$:- sigmoid function



⇓ gradient reaches saturation

Left Derivative $\lim_{z \to 0^-} g'(z)$

Right derivative $\lim_{z \to 0^+} g'(z)$

$$g'(z) = g(z)(1-g(z))$$

Why is this not a problem?

As $z \to \infty$ $g(z) \to 0$

$z \to -\infty$ $g(z) \to 0$

For large z (weight), learning becomes very slow.

Instead:-

Rectified linear unit

sub gradient

⇓ non-differentiable

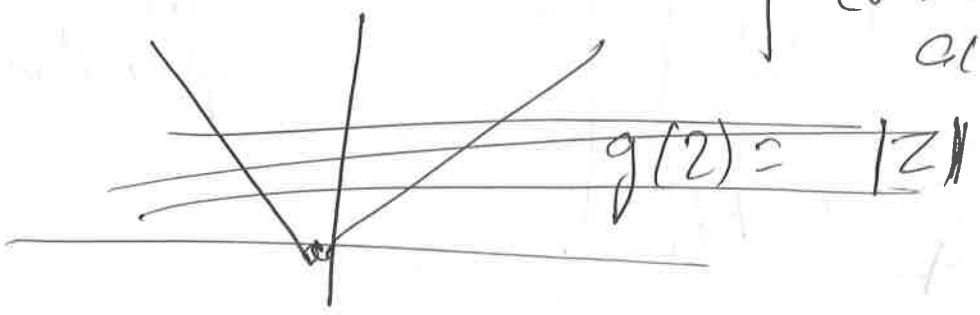
$g(z) = \max(0,z)$

$g'(z) = 1$ $z > 0$

gradient does not saturate (when unit is active)

$g(z) = \max(0,z) + \alpha \min(0,z)$

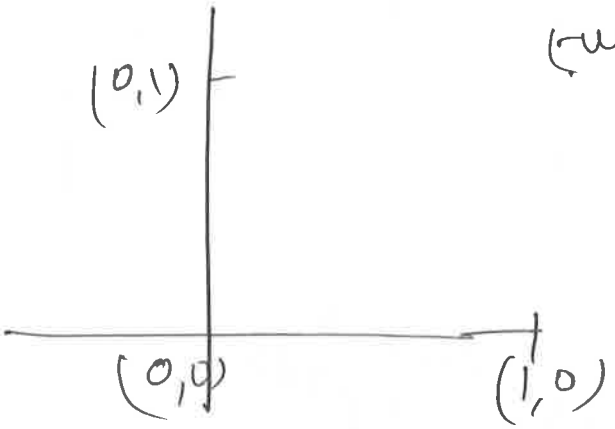
⇓ variations

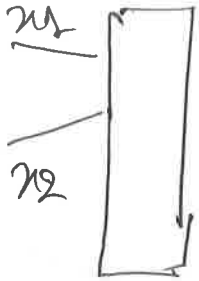


$g(z) = |z|$

(3) Example:- XOR

Not separable





Hidden Layer (2 units)

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$X = \begin{bmatrix} x^{(1)} & x^{(2)} & x^{(3)} & x^{(4)} \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

(+ve) (+ve) (2,2) +ve

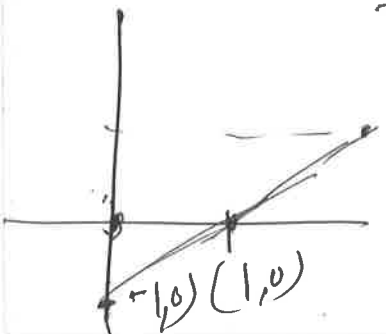
+ve (1,1)

(0,0)

$$W^T X + b = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

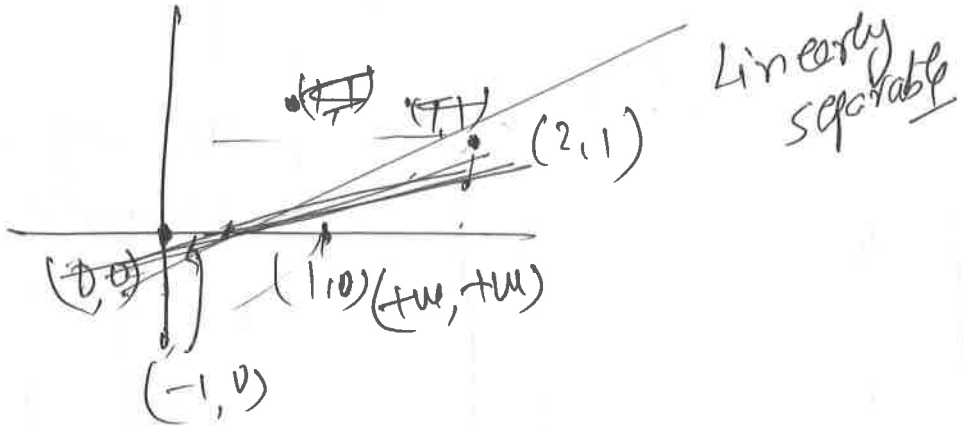
Not linearly separable

$$= \begin{bmatrix} x^{(1)} & x^{(2)} & x^{(3)} & x^{(4)} \\ 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$



⇒ Now apply $g(z) = \max(0, z)$

$$\Rightarrow g(w^T x + b) = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$



(3) (4) Output Unit:-

$f^{(o)}(\phi(x;\theta); \theta^{(o)})$ ~~$\theta^{(o)}$~~

what is form of $f^{(o)}$?

(1) If $\theta^{(o)} = (w^{(o)}, b^{(o)})$

$f^{(o)}(\phi(x;\theta); \theta^{(o)})$

$= w^{(o)T}\phi(x;\theta) + b^{(o)} = \hat{y}$

Linear Transformation

~~Not a cte~~

$L(y, \hat{y}; \theta) = \|y - \hat{y}\|^2$

$= \|y - f(x;\theta)\|^2$

Total Loss:-

$$J(\theta) = \sum_{x \in X} \|y - f(x;\theta)\|^2$$

Not a very good metric. Especially for classification when $y \in \{0, 1\}$.

All possible instances

Instead Use $f^{(0)}$ to define a distribution over ~~y (given~~ y|x y (given (as a function of θ)).

Assume- $y \in \{0, 1\}$

$$f^{(0)}(\phi(x;\theta); \theta^{(0)})$$

$$= g^{(0)}(w^{(0)T} \phi(x;\theta) + b^{(0)})$$

⇓

sigmoid function.

can define Bernoulli rand distribution

~~$p(y=1) =$~~ $g^{(0)}(z) = \frac{1}{1+e^{-z}}$

$$p(y=1) = g^{(0)}(z)$$

$$p(y=0) = 1 - g^{(0)}(z) = g^{(0)}(-z)$$

Generic in nature

$$p(y) = g^{(0)}[(2y-1) z]$$

why is it good thing to do? -ve log likelihood

Finally:- ~~L(y,ŷ;θ) =~~

$$L(y,\hat{y};\theta) = -\log[p(y)] \Rightarrow$$

Approximate by training loss $\leftarrow$ ~~L~~ $J(\theta) = E_{(x,y \sim \text{Data})} - \log[P_{model}(y)]$

Extra

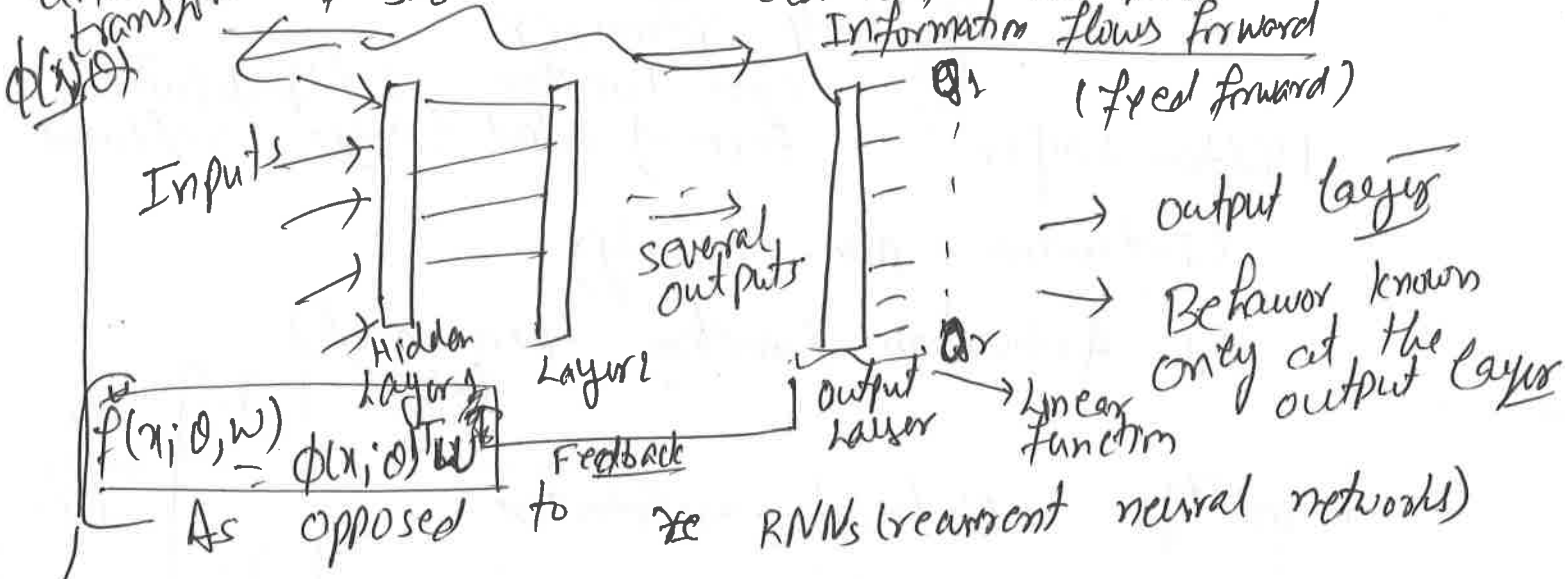
Deep Feedforward Networks

Also known as:- feedforward neural networks
multilayer perceptrons

Goal:- $y = f^*(x)$

Define a mapping $y = f(x;\theta)$ → parameters of the model

f should be as close to f^* as possible



As opposed to RNNs (recurrent neural networks)

→ Foundational building blocks for CNNs, RNNs, many other deep learning models.

Network:- network of connections (going forward)

depth of the model

$y = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ — n layered network.

compositional structure

network structure

implements a compositional structure.

Neural:- Resembles the functioning of neuron in the brain. Each neuron in a layer acts in parallel.

Each layer:- vector-valued function.

Theme in Deep Learning:- Figure out the transformation $\phi(x;\theta)$

⇓ hidden Layer

Feature transformation

Design choices:-

Output layer:-

(1) ~~Cost Function $L(y, \phi(x)^T w)$~~

(2) ~~Outp~~ Optimizer

(2) Cost Function $L(y, \phi(x)^T w)$

(3) form of output layer:- softmax.

Hidden Layer:-

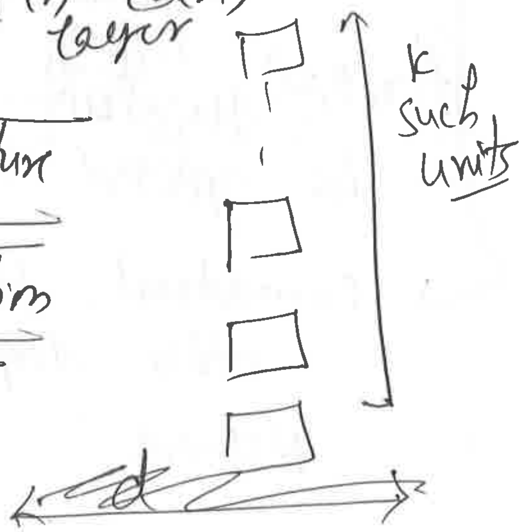
(Introduce non-linearity)

↳ Activation Function (in each layer)

Finally:- Network architecture

For training:- Backpropagation

(gradient based) & generalizations



Hidden Layer:-

$h^{(1)} = f^{(1)}(x; W, c)$

⇓

$= g(W^T x + c)$

vector valued function — Activation fn ⇓ — Linear function

$W \in \mathbb{R}^{d \times k}$

~~$h^{(1)} = g(\ldots)$~~ (Element-wise)

$$W^T = \begin{bmatrix} -- & w_{(1)}^T & -- \\ & | & \\ & | & \\ -- & w_{(k)}^T & -- \end{bmatrix} \begin{bmatrix} | \\ x \\ | \end{bmatrix} + \begin{bmatrix} | \\ c \\ | \end{bmatrix}$$