Lecture 05 - 28.4.2022 DL

4.3.1

layer l $\quad 1 \leq l \leq L$:

$$\underline{x}_{l-1} \in \mathbb{R}^{M_{l-1}} \longrightarrow \boxed{\begin{matrix} \underline{\underline{W}}_l \\ \underline{b}_l \\ \phi_l \end{matrix}} \longrightarrow x_l \in \mathbb{R}^{M_l}$$

$$\underline{\underline{W}}_l \in \mathbb{R}^{M_l \times M_{l-1}}$$

$$\underline{b}_l \in \mathbb{R}^{M_l}$$

$$\underline{a}_l = \underline{\underline{W}}_l \cdot \underline{x}_{l-1} + \underline{b}_l \quad \in \mathbb{R}^{M_l}$$

$$\underline{x}_l = \phi_l(\underline{a}_l) \quad \in \mathbb{R}^{M_l}$$

$M_l(M_{l-1} + 1)$ unknown parameters to be learnt by training eg by minimising cost f^n

- No shortcuts are possible i.e no direct connection b/w $\underline{x}_{l-1}$ and $\underline{x}_l$
- No ~~feeb~~ feedbacks i.e. $\underline{x}_l$ ~~and~~ dont feed data or i/p to $\underline{x}_{l-1}$

Network :

$$\underline{x}_L = f(\underline{x}_0 \, ; \underline{\theta}) : \mathbb{R}^{M_0} \to \mathbb{R}^{M_L}$$

•) parameter vector

$$\underline{\theta} = \begin{bmatrix} \mathrm{vec}(\underline{\underline{W}}_1) \\ \underline{b}_1 \\ \vdots \\ \mathrm{vec}(\underline{\underline{W}}_L) \\ \underline{b}_L \end{bmatrix} \in \mathbb{R}^{N_p}$$

, learnt from data during training.

N_p : No of parameters

$$= \sum_{l=1}^{L} M_l \times (M_{l-1} + 1)$$

No of multiplications $N_X = \sum_{i=1}^{L} M_l M_{l-1} \simeq N_p \to$ computational complexity

" " additions $\approx N_p$

•) No of layers L and no of neurons in each layer $\{M_1 \dots M_L\}$

•) activation fn $\{\phi_1 .. \phi_L\}$

hyper parameters - chosen by you

see 6.7

4.4.1 First neural network in history:

(liner) perceptron by Rosenblatt, 1957

$$y = \underline{w}^T \underline{x} + b$$

*) no non-linear activation function $\phi()$

*) no hidden layers

$\Rightarrow$ Only for linear tasks

4.4 Activation function

Mild requirements on $\phi_L()$:

*) non-linear in general $\rightarrow$ fundamental

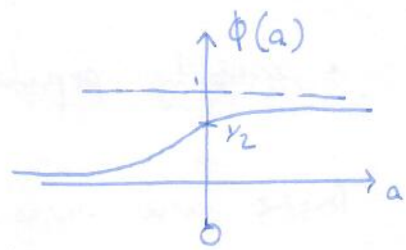
*) smooth, differentiable $\rightarrow$ for training

*) simple calculation $\rightarrow$ low computational complexity

Slide 4.6

Sigmoid function

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



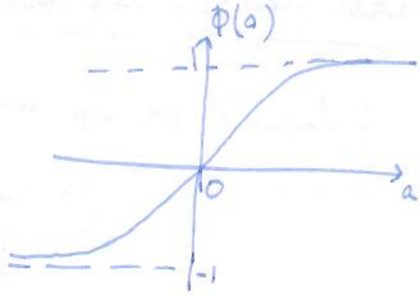
- $0 < \phi(a) < 1 \; \hat{=}$ probability
- symmetrical $\phi(-a) = 1 - \phi(a)$
- derivative

$$\frac{d}{da}\phi(a) = \frac{e^{-a}}{(1+e^{-a})^2} = \phi(a)\phi(-a) = \phi(a)\left[1-\phi(a)\right] \in (0,1)$$

- widely used in conventional N.N.

Hyperbolic tangent $\tanh()$

$$\phi(a) = \tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

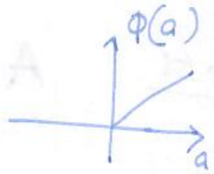


$$= 2\sigma(2a) - 1$$

- like sigmoid, another output range

Rectifier Linear Unit (ReLU)

$$\phi(a) = ReLU(a) = \max(a, 0) = \begin{cases} a & a \geq 0 \\ 0 & a < 0 \end{cases}$$



- $\hat{=}$ diode
- Simple calculation
- $\frac{d\phi}{da} = \begin{cases} 1 & a > 0 \\ 0 & a \leq 0 \end{cases} = u(a)$ with $u(0) = 0$
- mostly popular in Deep Learning

There are various variants of ReLU like leaky ReLU and Parametric leaky ReLU.

Softmax

$\phi(\underline{a}) : \underline{a} = [a_i] \in \mathbb{R}^C \longrightarrow \mathbb{R}^C$

$$\phi(\underline{a}) = \text{softmax}(\underline{a}) = \begin{bmatrix} \phi_1(\underline{a}) \\ \vdots \\ \phi_c(\underline{a}) \end{bmatrix} \quad \text{with}$$

$$\phi_i(a) = \frac{e^{a_i}}{\sum_{j=1}^{c} e^{a_j}} \in (0,1) \quad ; \sum_{i=1}^{C} \phi_i(a) = 1$$

- a_i large $\rightarrow \phi_i(a)$ close to 1
  a_j '' $\rightarrow \phi_j(\underline{a})$ close to 0

- a non-linear normalization of $\underline{a}$

- used in the o/p layer ~~of~~ for classification tasks.

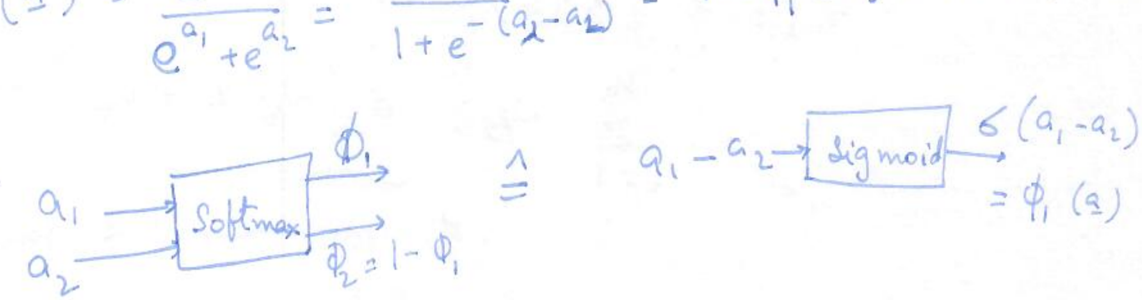
Slide 4-8

Special case : $C = 2$

$$\phi_1(\underline{a}) = \frac{e^{a_1}}{e^{a_1} + e^{a_2}} = \frac{1}{1 + e^{-(a_1 - a_2)}} = \sigma(a_1 - a_2)$$

$$\phi_2(\underline{a}) = \frac{e^{a_2}}{e^{a_1} + e^{a_2}} = \frac{1}{1 + e^{-(a_2 - a_1)}} = 1 - \phi_1(a) = \sigma(a_2 - a_1)$$

i.e.



one o/p of sigmoid is sufficient for binary classification

Derivative of softmax :

$$\frac{\partial \phi_i(\underline{a})}{\partial a_j} = \dots = \begin{cases} \phi_i(\underline{a})\left(1-\phi_i(\underline{a})\right) & i=j \\ -\phi_i(\underline{a})\,\phi_j(\underline{a}) & i \neq j \end{cases}$$

4.5 Universal approximation theorem

Why can neural network be used everywhere ?

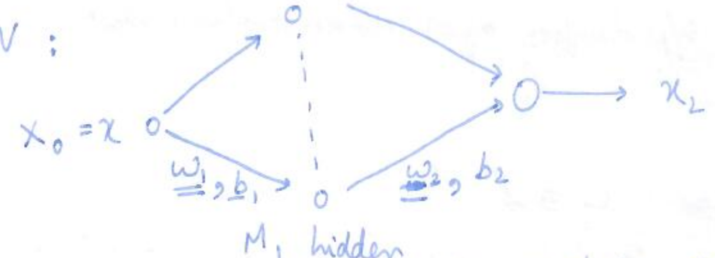
E4.3 Regression with one hidden layer

True f^n : $f_0(x)$

Given : $x(n)$ and noisy measurements $y(n) = f_0(x(n)) + z(n)$

$1 \leq n \leq N$

NN :

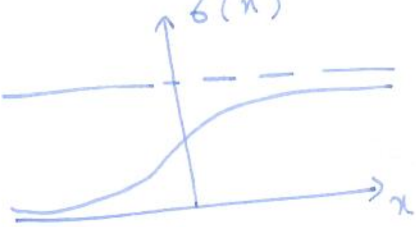


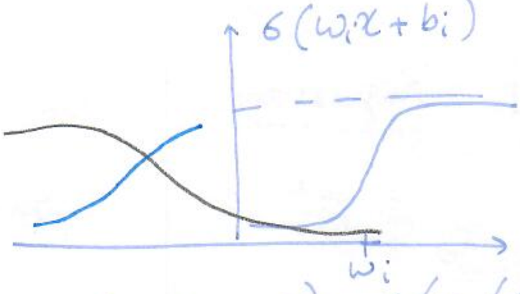
M_1 hidden neurons, $\underline{x}_1$

sigmoid $\phi_1(\underline{a}) = \sigma(\underline{a})$

$$x_2 = f(x;\underline{\theta}) = \underline{w}_2^T \sigma(\underline{w}_1 x + \underline{b}_1) + b_2$$

$$= \sum_{i=1}^{M_1} w_{2,i} \underbrace{\sigma(w_{1,i} x + b_{1,i})}_{M_1 \text{ non-linear basis functions of } x} + b_2$$





$$\sigma(w_i x + b_i) = \sigma\left(w_i\left(x + \frac{b_i}{w_i}\right)\right)$$

- new centre at $-\frac{b_i}{w_i}$
- new slope w_i

Universal approx theorem guarentees that there is a $f^{\underline{n}}$ that exists but doesn't tell us more about it. Optimal is found using deep NN.

No theory on which activation $f^{\underline{n}}$ is better than the others. It has to be found experimentally.

4.6 Loss and cost functions

$$x_0 = x \longrightarrow \boxed{DNN;\ \underline{\theta}} \longrightarrow \underline{\hat{y}} = \underline{x}_L = f(\underline{x};\underline{\theta}) \longrightarrow \boxed{loss} \longleftarrow \underline{y}$$

ch 3.4

$$\min_{\underline{\theta}} L(\underline{\theta}) = \frac{1}{N}\sum_{n=1}^{N} l(\underline{x}(n), \underline{y}(n); \underline{\theta})$$ overall cost $f^{\underline{n}}$ to be minimized from D_{train}.

$$l(\underline{x}, \underline{y}; \underline{\theta}) = -\ln q(\underline{y}|\underline{x};\underline{\theta})$$ ← NLL loss for one $\underline{x}, \underline{y}$

$$\underline{q}\ () \xleftrightarrow{\text{to}} DNN$$

4.6.1 Regression

$\underline{x} \in \mathbb{R}^d$ random input

$\underline{y} \in \mathbb{R}^c$: desired " o/p

Assumption : DNN estimates the mean of $\underline{y}$ i.e.

$$\underline{y} = f(\underline{x}; \underline{\theta}) + \underline{z} \quad ; \underline{z} \text{ - noise}$$