Characteristics of NN

1) Mapping capability — input patter to their associal output patter.

2) learn by example — train then used to infer or predict new object

3) can generalized: therefore can predict new outcomes from past trends.

4) Robust & fault tolerant: can recall full pattern from partial or noisy pattern.

5) can process informat in parallel in a distributed manner.

LEARNING METHODS

Supervised

input feature	output labels

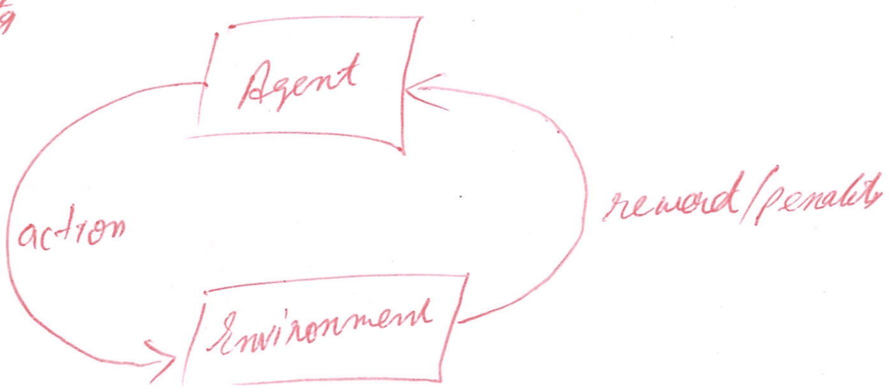
labelled data

Unsupervised

un labelled data

Reinforce.

in absence of data



Robotics / Drone stability / etc

Hebbian learning: Fire together - wire together

based on correlative weight adjustment

oldest learning mechanism inspired by biology.

Gradient decent

base on minimization of error E defin in terms of weigh & the activation function of the networ.

→ activate fun should be differencia as weigh update depends on gradient of eror E

weig upa $\Delta W_{ij} = \eta \frac{\partial E}{\partial W_{ij}}$

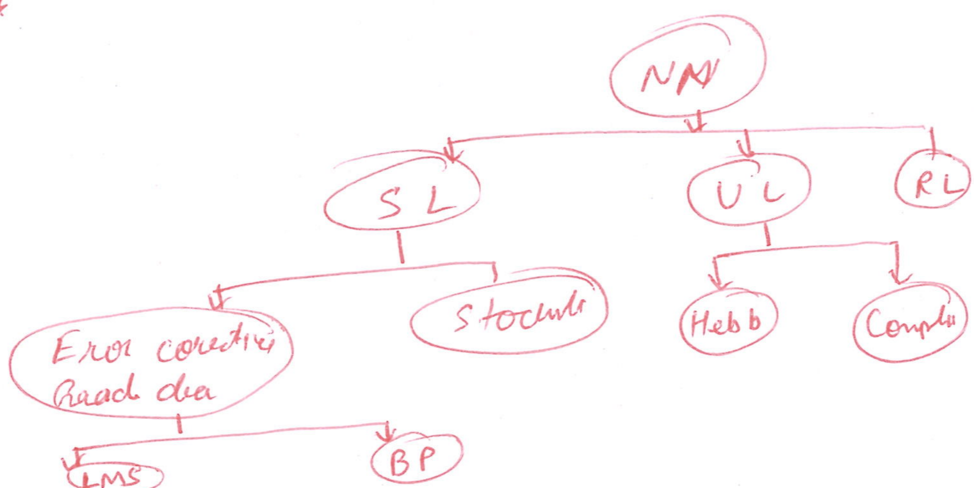
η is learning rate & $\frac{\partial E}{\partial W_{ij}}$ is error gradien wrt weig W_{ij}

Competetive learning winner - takes - all

those neuron which responds strongly to input stimuli have their weigh updated

when input is presnet all neura compete & the winner neuron undergoes weigh upat

Stochastic Learning



Loss Function

Euclidean distance

~~Hamilton~~

Manhattan distance

MSE

Single Training Img ①

$$C_x(w,b) = \|\hat{y}(x) - y(x|w,b)\|^2$$

desired Target — Network output

For each Train Data

$$C(w,b) = \frac{1}{n}\sum_{n} C_x(w,b)$$

No of imgs..

Lower cost value

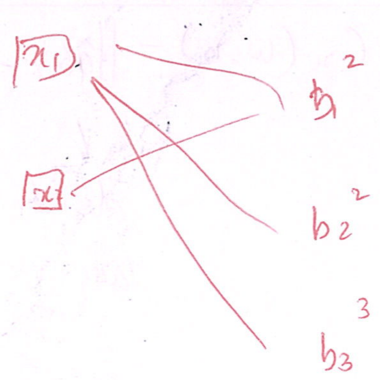
Init Weight & Bias of Network Random Var

2 Compute Activation for each Training Img

3) Compute Cost for Single Train Data

4) Update Weight & Bias using Gradient Descent

5) Repeat Steps 2 - 4 until Cost is Reduced to an acceptable level



$\boxed{b_1^3}$ Signore Neur

Input layer

MNIST (hand written Digits) 60,000 images

28 x 28

Notes by Nielsen 2015

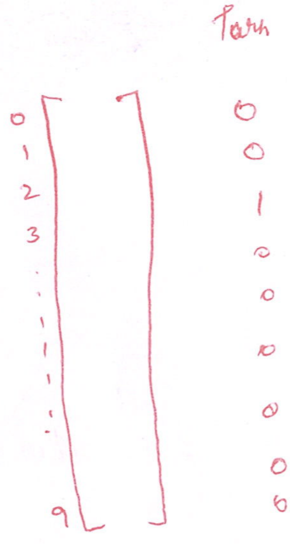
784 neur 30 neuron 10 output neur.

x_1

⋮

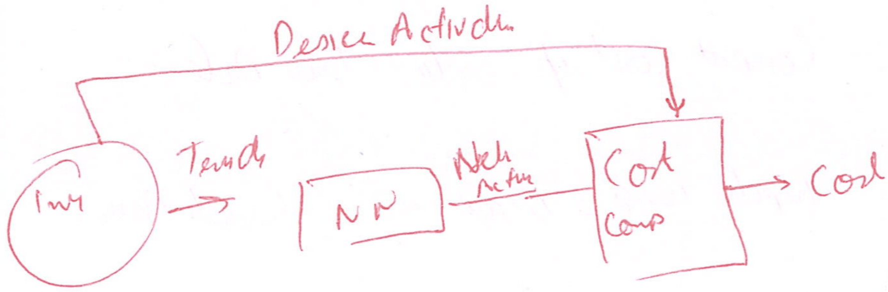
2

x_{784}



Actual.

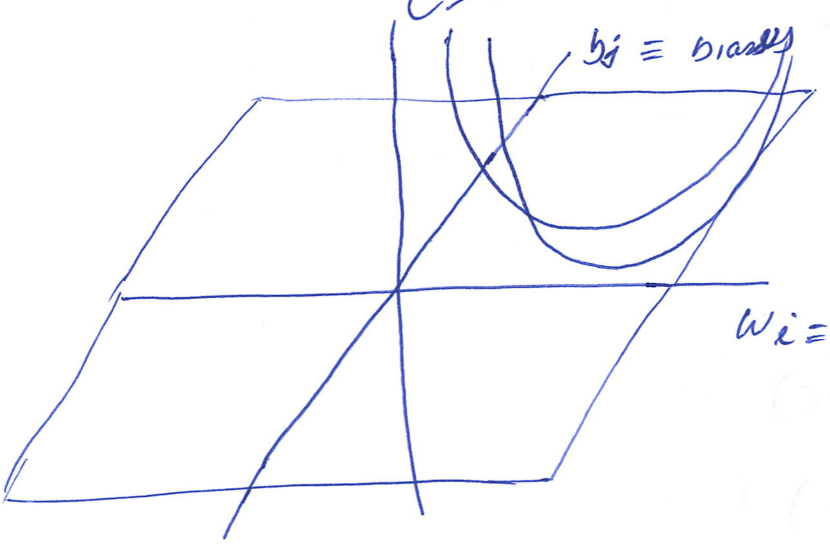
Inp	1	2	3
Label	1	2	3
Actual			
$\hat{y}(x)$			

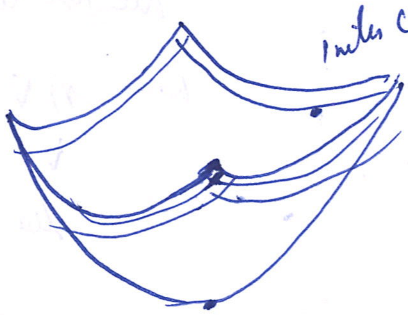


(1)

784   30   10 $\underbrace{y \quad t}_{\text{Cost func}}$

$W_i \; b_j$





func $= f(x, y)$ Taylor Series

& to find $f(x+\Delta x, x+\Delta y)$ with linear approx...

$$\cong f(x,y) + \left\{ \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y \right\}$$

$$= f(x,y) + \Delta f(x,y)$$

where, $\Delta f(x,y) = \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y$

Similarly

$$\Delta C = \Delta C(w_i, b_j) = \frac{\partial C}{\partial w_i}\Delta w_i + \frac{\partial C}{\partial b_j}\Delta b_j$$

In vector form:

$$\Delta C = \begin{bmatrix} \frac{\partial C}{\partial w_i} & \frac{\partial C}{\partial b_j} \end{bmatrix} \cdot \begin{bmatrix} \Delta w_i \\ \Delta b_j \end{bmatrix}$$

GRADIENT · change in weight & bias

$$= \nabla C \cdot \Delta v$$

$\Delta C = \nabla C \cdot \Delta v$

∇C

we want to move opposite to gradient or $-\nabla C$

(change in weights & bias)

Let $\Delta v = -\nabla C$

$\therefore$ we use a fraction of this

i.e. $-\eta \nabla C$

η → learn rate

$$\therefore \Delta C = \nabla C \cdot (-\eta \nabla C)$$

$$= -\eta (\nabla C \cdot \nabla C)$$

$$= -\eta \| \nabla C \|^2$$

$\|\nabla C\|^2$ → Magnitude

$\therefore$

$$\boxed{\begin{aligned} w_i \rightarrow w_i' &= w_i - \eta \frac{\partial C}{\partial w_i} \\ b_j \rightarrow b_j' &= b_j - \eta \frac{\partial C}{\partial b_j} \end{aligned}}$$

Update rule

→ need to find deriv of Cost w.r.t all parameter

η → small more computation slow convergence

η → large less compute faster conv { but may overshoot

From Taylor Series

$$f(x+\Delta x, y+\Delta y) \simeq f(x,y) + \overbrace{\frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y}^{\Delta f(x,y)}$$

$$\Delta f(x,y) \simeq \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y$$

Applying to Cost func (Loss fun)

$$\Delta C \simeq \frac{\partial C}{\partial w_i}\Delta w_i + \frac{\partial C}{\partial b_j}\Delta b_j$$

$$= \begin{bmatrix} \frac{\partial C}{\partial w_i} & \frac{\partial C}{\partial b_j} \end{bmatrix} \begin{bmatrix} \Delta w_i \\ \Delta b_j \end{bmatrix}$$

Change in Cost:

$$\Delta C = \nabla C \cdot \Delta v$$

∇C ↓ Gradient

Δv ↘ Change in weight & bias

we need opposit direc of ∇C

Let $\Delta v = -\eta \nabla C$

η ↘ Learn rate

$$\therefore \Delta C = \nabla C \cdot -\nabla C = -\|\nabla C\|^2$$

$$\boxed{\begin{aligned} w_i \rightarrow w_i' &= w_i - \eta \frac{\partial C}{\partial w_i} \\ b_j \rightarrow b_j' &= b_j - \eta \frac{\partial C}{\partial b_j} \end{aligned}}$$

Update rule

→ need to find derivat of cost wrt all the parameters

η small $\rightarrow$ more computat (slow convergen)

η - large $\rightarrow$ fasters (fast convergen) but may overs shoot.

GRADIENT DECENT for Entire network

$$\Delta v = \begin{bmatrix} \Delta w_{jk}^{(l)} \\ \vdots \\ \Delta b_j^{(l)} \\ \vdots \end{bmatrix} ; \quad \nabla C = \begin{bmatrix} \dfrac{\partial C}{\partial w_{jk}^{(l)}} \\ \vdots \\ \dfrac{\partial C}{\partial b_j^{(l)}} \\ \vdots \end{bmatrix} ; \quad \begin{array}{l} j : \text{neuron} \\ k : \text{inputs} \\ l : \text{layer.} \end{array}$$

$\boxed{\Delta v = -\eta \nabla C}$ is still valid

$$w_{jk}^{(l)} \longrightarrow w_{jk}^{(l)'} = w_{jk}^{(l)} - \eta \frac{\partial C}{\partial w_{jk}^{(l)}}$$

$$b_j^{(l)} \longrightarrow b_j^{(l)'} = b_j^{(l)} - \eta \frac{\partial C}{\partial b_j^{(l)}}$$