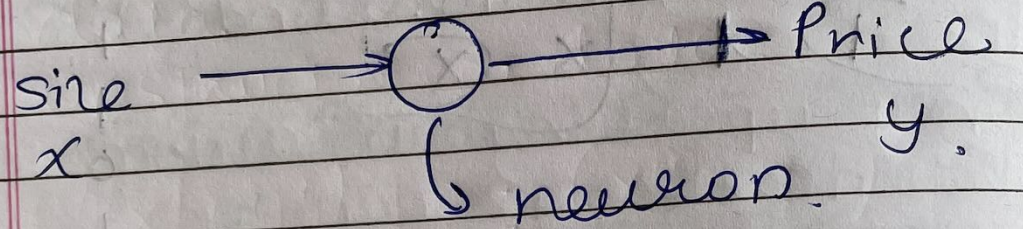
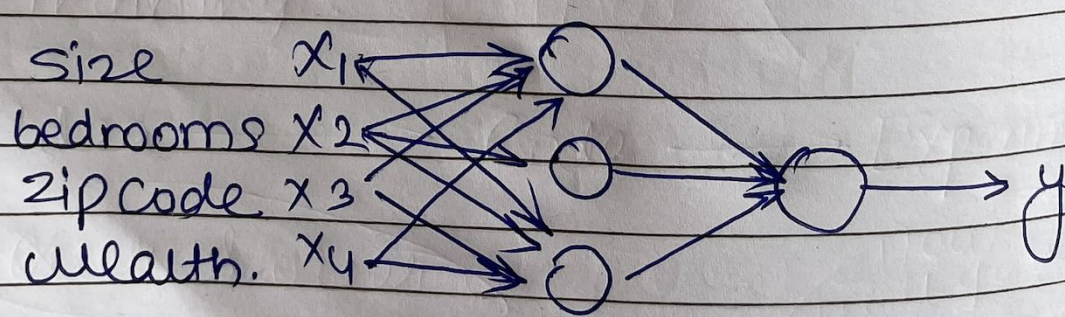
* Neural Networks.



neurons → stacking
↳ neural network.

Stacking neurons ?
helps in prediction.

give input (x)
output (y) > for ~~examples~~ in training.



PAGE No.
DATE / /

Lec 2.

(2) Supervised Neural Networks.

i) predicting ads (Neural Network).

1) Standard Neural Network
2) Image (Convolution NNs).
3) Audio (RNN) Recurrent neural network.
4) Language Sequence data
complex RNN

5) Autonomous driving custom, hybrid neutral networks.

Convolution NNs are used for image data.

Appl of machine L

(1) Structure data (Database) (Table).

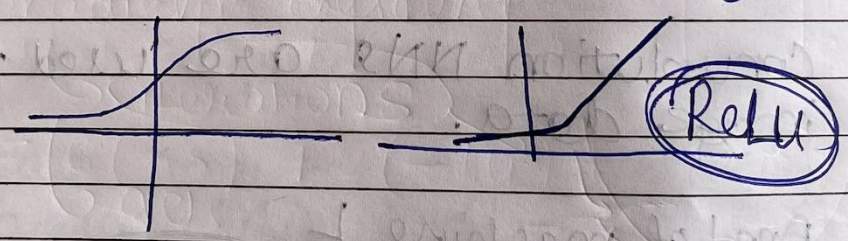
(2) Unstructure (Audio, Images, Text)

pixel values, words

Computers are now much better at understanding unstructured data.
Unstructured data.

m → size of training examples.

large neutral networks dominate over small, medium, traditional algorithms } due to data scale.



Training

Idea → Code → Experiment

over & over

How well your NN works.

PAGE No.
DATE / /

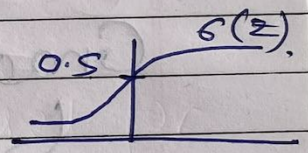
x is
$\hat{y}$ is probability that it is a cat picture.

x is a x dimensional vectors.

w is x dimensional vector
b is real number.

$$\hat{y} = \sigma(\underbrace{w^T x + b}_{z}).$$

sigmoid



$$\boxed{\sigma(z) = \frac{1}{1+e^{-z}}}$$

* Cost Function

To train the parameters w & b you need to define cost function.

PAGE No.
DATE / /

$\hat{y} = \sigma(w^T x + b)$ $\sigma(z) = \frac{1}{1+e^{-z}}$

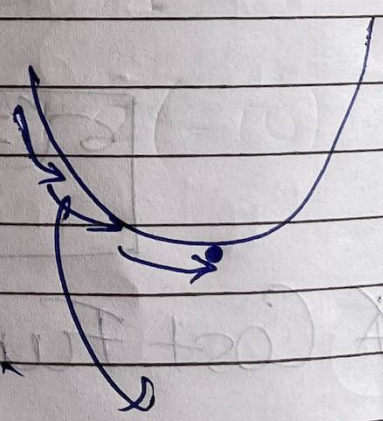
$\{(x^1, y^1) \cdots (x^m, y^m)\}$

cost function is average of loss (error) function.

Gradient descent

Iterations

gets close to global optimal.



$$w = w - \alpha \frac{dJ(w)}{dw}$$

α is learning rate

α controls steps of decline.

$$dw = \frac{dJ(w)}{dw}.$$

$$w = w - \alpha \frac{dJ(w,b)}{dw}.$$

$$b = b - \alpha \frac{dJ(w,b)}{db}$$

$$A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 3 \end{bmatrix}$$

$$a = 6 + 5 + 7$$

PAGE No.
DATE / /

$x_1 \rightarrow$
$x_2 \rightarrow$
$w_1 \rightarrow$
$w_2 \rightarrow$
$b \rightarrow$

$z = w_1 x_1 + w_2 x_2 + b$

$a = \sigma(z)$

$x_1 \rightarrow$
$w_1 \rightarrow$
$x_2 \rightarrow$
$w_2 \rightarrow$
$b \rightarrow$

$= w_1 x_1 + w_2 x_2 + b \rightarrow a = \sigma(z)$

$\downarrow$

$L(a, y)$

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{n} L(a^{(i)}, y)$$

Vectorization

i) getting rid of

$z = 0$
for i in range

non vectorized

Vectorized

$z = np.dot$

$\downarrow$

$w^T x$

$a = np.rand$

$\downarrow$

million dim

whenever pos
loops.

$$u_i = \sum_j A_{ij}$$

Vectorization

i) getting rid of explicit for loops.

```
z = 0
for i in range(n x):
```

non vectorized.

Vectorized

$$\boxed{z = np.dot(w, x) + b}$$

$w^T x$

```
a = np.random.rand(1000000)
```

million dim array.

whenever possible, dont use for loops.

$$u_i = \sum_j A_{ij} V_j$$

u = np.zeros((n,1))

for i

[up = np.dot(A, v)]

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \quad u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ e^{v_3} \end{bmatrix}$$

→ import numpy as np.

u = np.exp(v)
u = np.log(v)
u = np.abs(v)

element wise
1. exponent
2. log
3. abs.

sq. ← v**2 1/v inverse

Whenever you need to use a for loop check whether you can do that task using Numpy function.

$dw = np.zeros((n_x, 1))$

---

* Iterate over entire training set without using a single for loop.

$z^{(1)} = w^T x^{(1)} + b$. $z^{(2)} = w^T x^{(2)} + b$.
$z^{(3)} = w^T x^{(3)} + b$.
$a^{(1)}$

$$Z = w^T X + \underset{1 \times m}{\begin{bmatrix} b & b & b & b & b & b \end{bmatrix}}$$

$Z = np.dot(w.T, X) + b$.

PAGE No.
DATE / /

* A note on python/numpy vectors.

a = np.random.randn(s)

s random variables is a

(a.shape)

→ (s, 1)

a.T (transpose)

* a = np.random.randn(s, 1)

creat DS where a.shape = (s, 1)

assert((a.shape == (s,1)))

a = a.reshape((5,1))

Dont use rank 1 arrays use (n,1) column vectors or (1,n) row vectors.

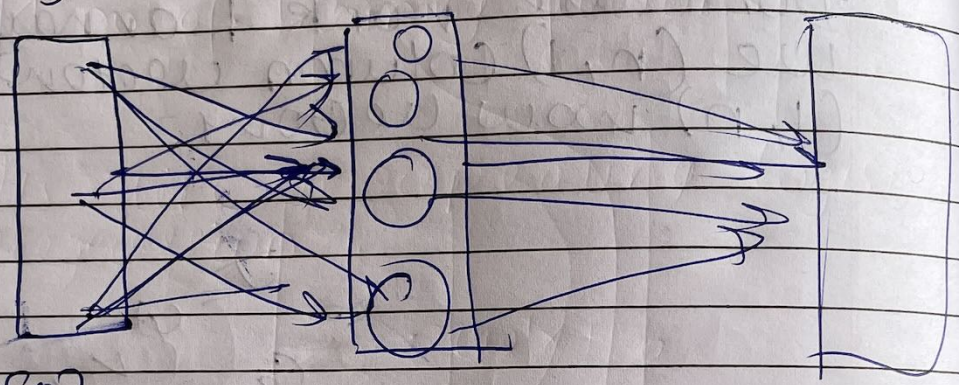
$$J = (v + w)$$

$$= (a+b)(a-b)$$

$$= a^2 - b^2 + (b+c)(b-c)$$

$$= a^2 - c^2 + b^2 - c^2$$

Input layer → Hidden layer → Output layer



$a^{[0]} = x$

a is passing to subsequent layer.

2-layer NN → NN with 1 hidden layer

0 + 1 + 2

layer 1

$y = wx + b$

$y = w^T x + b$.

↳ if x is a vector.

$y = \text{sigmoid}(w(\text{transpose})x + b)$

Learning

Genetic. mini project

courses

could add features

NPTEL Learning.

$L(\hat{y}, y) =$

PAGE No.
DATE / /

* Neural Network Representation

M no of training vectors

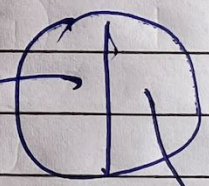
Its like a repeating logistic regression

$$z = w^T x + b$$

$$a = \sigma(z)$$

↳ sigmoid.

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$



$$a_1^{[1]} = \sigma(z_1^{[1]})$$

$a^{[l]}_{i}$

$[l] \rightarrow$ layer no

$i \rightarrow$ node no

$$\overbrace{\begin{bmatrix} - w_1^{[1]T} - \\ - w_2^{[2]T} - \\ - w_3^{[1]T} - \\ - w_4^{[1]T} - \end{bmatrix}}^{w^{[1]}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \overbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[2]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}^{b^{[1]}} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix}$$

$$\therefore = \begin{bmatrix} z_1^{[1]} \\ z_2^{[2]} \\ z_3^{[3]} \\ z_4^{[4]} \end{bmatrix}$$

$$(4,1) \quad (4,3) \quad (3,1) \quad + (4,1)$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$(4,1) \longrightarrow (4,1)$$

$$w^T b$$

$$\hat{y} = a = \sigma(z)$$

[* Vectorize across multiple training examples.]

$$z^{[1]} = w^{[1]T} x + b^{[1]}$$
$$a^{[1]} = \sigma[z^{[1]}]$$
$$z^{[2]}$$
$$a^{[2]}$$

for $i = 1$ to m,

$$z^{[1][i]} = w^{[1]} x^{[i]} + b^{[1]}$$
$$a^{[1][i]} = \sigma(z^{[1][i]})$$
$$z^{[2][i]} = w^{[2]} a^{[1][i]} + b^{[2]}$$
$$a^{[2][i]} = \sigma(z^{[2][i]})$$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$a^{[2](i)}$$

Training example no

Layer no

for i=1 to m,

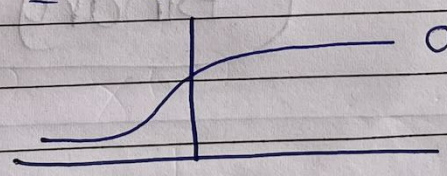
{Repeat ...}

$$w_1^{[1]} x^{(1)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad w^{[2]} x^{(2)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

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

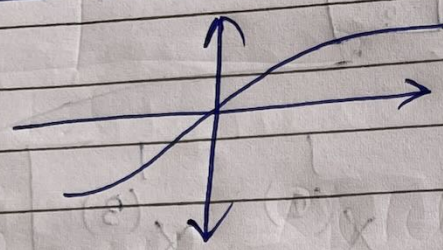
$$\therefore \; w^{[1]} X = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$$

$$Z = \begin{bmatrix} | & | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & z^{[1](4)} \\ | & | & | & | \end{bmatrix}$$



$$a = \frac{1}{1+e^{-z}}$$

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



$a = \tanh(z)$

$$g(z^{(1)}) = \tanh(z^{(1)})$$

tanh superior for layer transfers.

except

final output layer

Binary classification

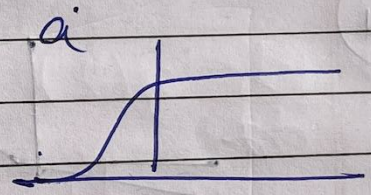
Actuation fn

1. $\tanh(z^{(i)})$
2. $\sigma(z^{(i)})$

3. $a = \max(0, z)$

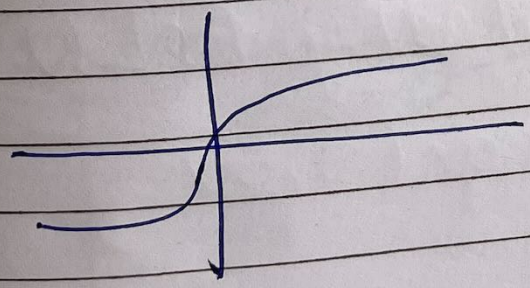
Pros & cons of actuation

1. $a = \frac{1}{1+e^{-z}}$



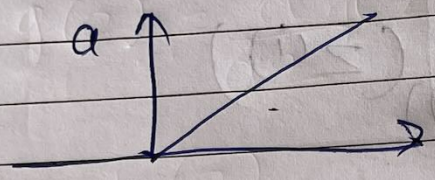
never use sigmoid unless output layer (final).

2. tanh : $a = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$



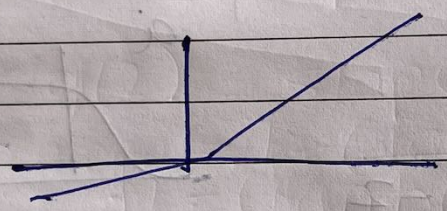
PAGE No.
DATE / /

(3)



def
default Relu.

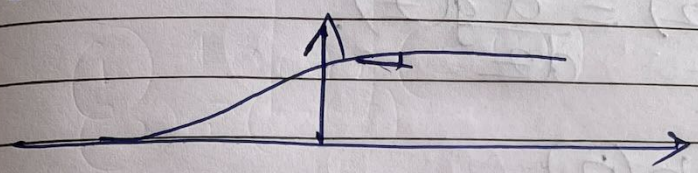
(4)



Leaky Relu a=

Choice of Activation Functions.

Sigmoid activation function



$$g(z) = \frac{1}{1+e^{-z}}$$

$$\frac{d}{dz} g(z) = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

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

$z = 10 \quad g(z) \approx 1$

$$\therefore \boxed{\frac{d}{dz} g(z) \approx 0}$$

g

z very large $\therefore$ slope $= 0$

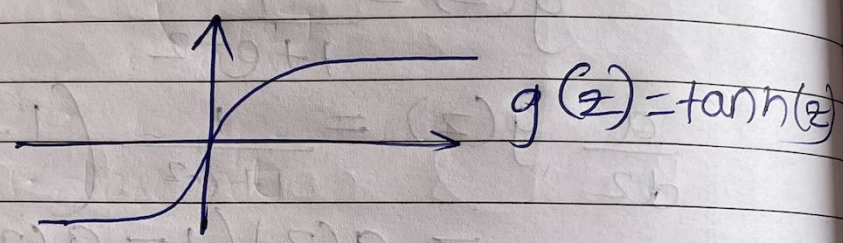
$z = -10$ $\therefore$ slope $= 0$

$z = 0$

$z = 0 \rightarrow \frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{1}{4}$

$$g'(z) = \frac{d\,g(z)}{dz}$$

② $\tanh(z)$.



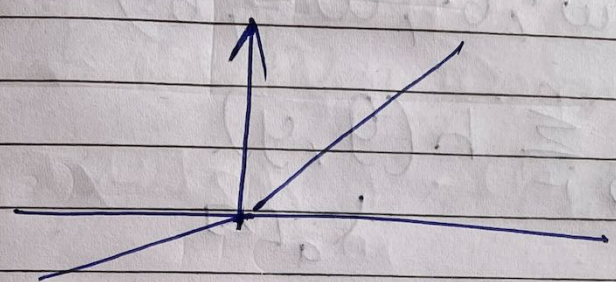
$$\frac{d}{dz} g(z) = \text{slope of } g(z)$$
$$= 1 - (\tanh(z))^2$$
$$= \boxed{1 - (\tanh(z))^2}$$

$$a = g(z)$$
$$g'(z) = 1 - a^2$$

③ ReLU and Leaky ReLU.

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

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$



$$g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

if we implement,

* Gradient Descent for One layered Neural Networks.

Param $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}$.

$n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1$.

PAGE No.
DATE / /

* cost function :

$J(w^1, b^1, w^2, b^2)$

$= \frac{1}{m} \sum_{i=1}^{n} h(\hat{y}, y)$

$a^{[2]}$

* Gradient descent.

Repeat :

compute predictions

$dw^1 = \frac{dJ}{dw^1}, db^1 = \frac{dJ}{db^1}, \ldots$

$\vdots$

$w^1 = w^1 - \alpha dw^{[1]}$

$b^1 = b^1 - \alpha db^{[1]}$.

* Formulas for derivatives

* Forward Propagation:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]}) \qquad (\longrightarrow)$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[1]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$$

* Back propagation:

np.sum($dZ^{[1]}$, axis=1(hori)).

Back propagation

$$dZ^{[2]} = A^{[2]} - Y \qquad Y = [y^{(1)} y^{(2)} \dots y^{m}]$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} \text{np.sum}(dZ^{[2]}, \text{axis}=1, \text{keepdims}=\text{True})$$

$$dz^{[1]} = \underbrace{w^{[2]T} dz^{[2]}}_{n^{[1]}, m} * g^{[1]\prime}(z^{[1]})$$

matrix

$\rightarrow$ $dw^{[1]} = \frac{1}{m} dz^{[1]} x^T$

$\rightarrow$ $db^{[1]} = \frac{1}{m} np.sum(dz^{[1]}, axis=1)$

* NN $\longrightarrow$ initialize random parameters.

If we initialize to 0,

For any example,

$$a_1^{[1]} = a_2^{[2]}$$

$$\therefore dz_1^{[1]} = dz_2^{[2]}$$

$$dw = \begin{bmatrix} y & \checkmark \\ y & \checkmark \\ y & \checkmark \\ y & \checkmark \\ \vdots & \vdots \end{bmatrix}$$

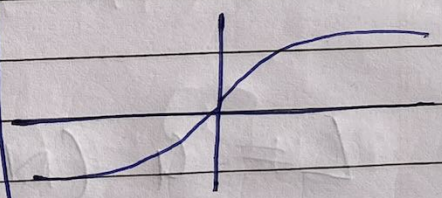
$w^{[1]} = np.random.randn((2,2)) * 0.01$

$b^{[1]} = np.zero$ (no problem to initialize to 0)

⇓

(no symmetry breaking problem).

for very small random values



$z^{[1]} = w^{[1]}x + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

For training deep networks

0.01 suitable is not always

ReLu $g'(z) = \begin{cases} 0 & z<0 \\ 1 & z>0 \\ \text{undefined} & z=0 \end{cases}$

PAGE No.
DATE / /

example stack = X matrix

Vectorization → same operation to multiple examples at once.

$L = 4$ (NN)

layers = Hidden + Output

$n^{[1]} = n^{(1)} = 5$

$n^{[2]} = 5$

$n^{[3]} = 3$

$n^{(4)} = 1.$

$\boxed{n^{[0]} = n_x = 3}$