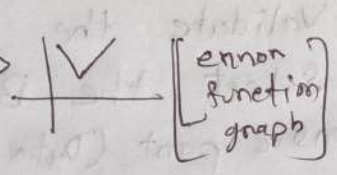
Deep Learning (Advanced)

#Pattern Recognition and Neural Networks

- Gradient Descent
- Importance of validation and test set.

pseudocode (old version)

Randomly initialize P_1 and other ... P_n
calculate error, e
while (not happy)
{ calculate $\frac{de}{dP_1}$
update $P_1 = P_1 - sign\left(\frac{de}{dP_1}\right) * \alpha$
calculate error e
}



x	1	2	3	4	5	Error
y	2	5	5	10	12	0

error, $e = \sum_{i=1}^{5} |\hat{f}(x_i) - y_i|$

$= \sum_{i=1}^{5} |P_1 x_i - Y_i|$

$= |P_1 - 2| + |2P_1 - 5| + |3P_1 - 5| + |4P_1 - 10| + |5P_1 - 12|$

$e_{P_1=1} = |-1| + |-3| + |-2| + |-6| + |-7| = 19$;

$\frac{de}{dP_1}(P_1=1) = ?$

$|P_1 x - y| = \begin{cases} P_1 x - y, & P_1 x - y \geq 0 \\ y - P_1 x, & P_1 x - y < 0 \end{cases}$

$\frac{de}{dP_1} = \begin{cases} x, & P_1 x - y \geq 0 \\ -x, & P_1 x - y < 0 \end{cases}$

$\frac{de}{dP_1} = -1 - 2 - 3 - 4 - 5 = -15$

sign(u)
- 1, u > 0
- -1, u < 0
- 0, u = 0

$P_1 = 1$, $e = 19$, $\frac{de}{dP_1} = -15$

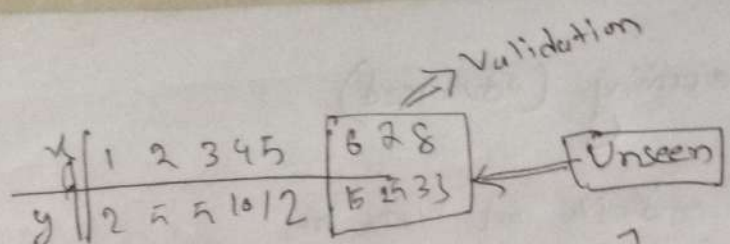
$P_1 = 2$, $e = 6$, $\frac{de}{dP_1} = -7$

$P_1 = 3$, $e = 11$, $\frac{de}{dP_1} = 15$

~~$P_1 = 4$, $e = 23$, $\frac{de}{dP_1} = 15$~~

(Assuming $\alpha = 1$)

[Consider using Non-linear function better approach]



⇒ No parameterization, selection. No Biased issue

- Chose a form of my function $\vec{f}$
- Find a parameterization of that reduces e on this data point

$y = f(x_i) + \eta$; It also learn noises
testdata

	e	
0.5x	-	-
1x	-	-
1.5x	-	-
2x		
2.5x	3.5	20.5 ⇒ Overfitted / Overtrained
3x	4	16 ⇒ good on test data of train data

(In training it performs well, but in test part it performs poor)

Validation data ⇒ Validate the Model
Select the Best Model
Expensive part (Data collection)

(10×10) Gray scale Image
Cat Dog (Binary classification)

$\vec{S} = M(\vec{X})$

$\vec{S} = \begin{bmatrix} S_{cat} \\ S_{dog} \end{bmatrix}$

if ($S_{cat} > S_{dog}$) then
"cat"
else
"dog"

$\vec{X}$ = Flattened vectorized from of the 10×10 image. $\vec{X} \in \mathbb{R}^{100}$

Training Data

Input → I	I_1	...	...	...	I_{2^k}
Label → C	c_0	c_1	...	...	D (C or D)

$C \to 1$

$D \to 0$

$\vec{s}_{2\times 1} = W_{2\times 100}\,\vec{x}_{100\times 1}$

Training data $(I_J, 1/0)$

$$\vec{s} = W\vec{x}$$

$$\begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} w_{1,1} \cdots w_{1,100} \\ w_{2,1} \cdots w_{2,100} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{100} \end{bmatrix}$$

$$s_i = \sum_{J=1}^{100} w_{i,J} \cdot x_J \qquad \frac{ds_i}{dw_{i,k}} = x_k$$

$$s_1 = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 \qquad \frac{ds_1}{dw_{1,2}} = x_2$$

~~$s_2 = w_{1,2}$~~

$$s_2 = w_{2,1}x_1 + w_{2,2}x_2 + w_{2,3}x_3 \qquad \frac{ds_2}{dw_{2,3}} = x_3$$

$$\frac{ds_2}{dw_{1,1}} = 0$$

$\vec{s} = W\vec{x}$

↳ Error function e

$\frac{de}{dw_{i,J}}$ where $1 \le i \le 2 \quad i \in \mathbb{Z}^+$

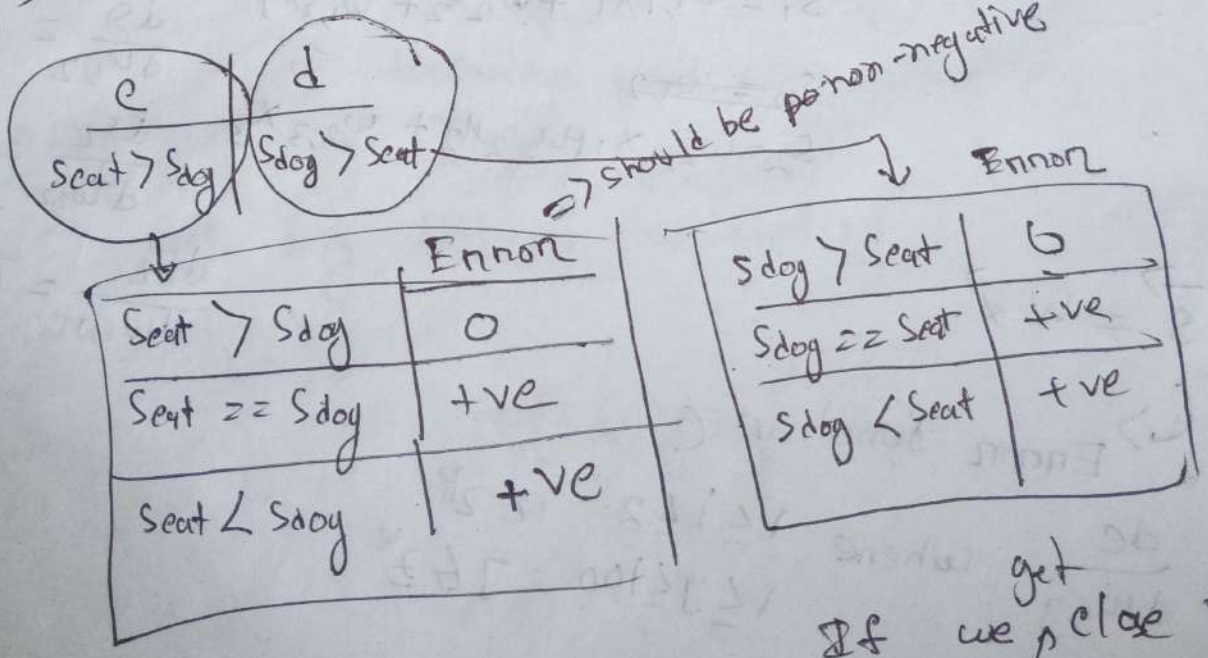
$1 \le J \le 100 \quad J \in \mathbb{Z}^+$

(P.T.O)

- Initialize the matrix W with random numbers
- Calculate error e

while (not happy)

```
{
    for (i=1; i<=2; i++)
    {   for (j=1; j<=100; j++)
        {
            calculate  de/dwi,j
            wi,j = wi,j - α de/dwi,j
        }
    }
    calculate errors;
}
```



$E = S_{dog} - Seat$

$E = \begin{cases} 0, & S_{dog} - Seat < 0 \\ S_{dog} - Seat, & S_{dog} - Seat \geq 0 \end{cases}$

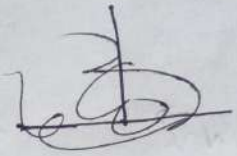
If we get close to 0, It may get overfitted

$E_{Real} = \begin{cases} E, & E \geq 0 \\ 0, & E < 0 \end{cases}$ $\quad E = S_{dog} - S_{cat} + m$ (where $m \geq 0$)

$Loss_C = \max(0, S_{dog} - S_{cat} + m)$ $\quad m > 0$

$Loss_D = \max(0, S_{cat} - S_{dog} + m)$ $\quad m > 0$

$y = \max(0, x)$



$Loss_{Total} = label \cdot Loss_{cat} + (1 - label) Loss_{dog}$

⇒ label = 1 if Cat.
label = 0 if Dog.

Do math and Exercises

L_{Total} = Aggregation L_{sample}

Gradient Descent → L_{sample}

$$\frac{1}{k} \sum_{k=1}^{n} k L_{sample}$$

Randomly 1 samples and calculate the loss and further update parameters → SGD (stochastic Gradient Descent)

For taking k samples instead of 1 sample → (Batch Gradient Descent) stochastic

SGD & BSGD have several pros & cons.

Epoch = Countsen : Iteration = of Total samples trained

Consider thinking Mini Batch Gradient Descent Algorithm instead of SGD, BSGD so that model gets benefits overcoming the cons of these algorithms.

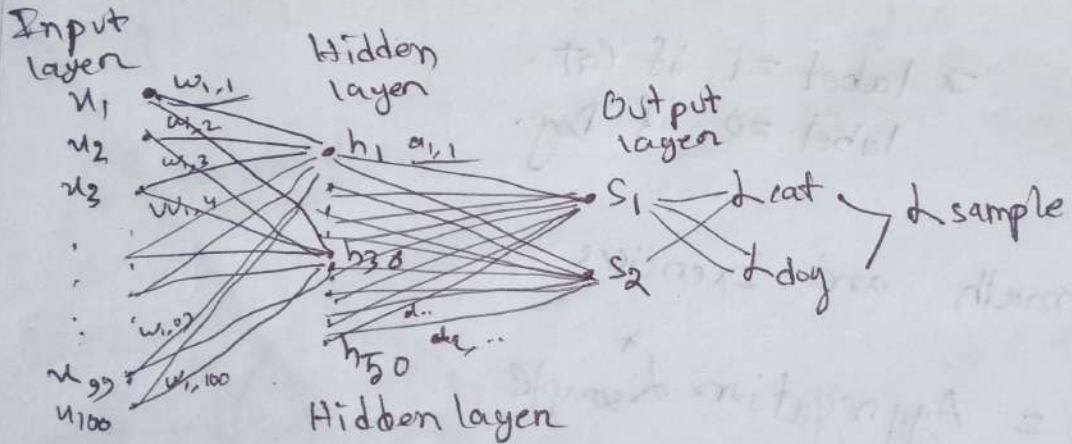
$$\vec{S} = AW\vec{x}$$

$\vec{x} \in \mathbb{R}^{100}$

$W \in \mathbb{R}^{50 \times 100}$

$A \in \mathbb{R}^{2 \times 50}$

$\vec{S} \in \mathbb{R}^{2}$



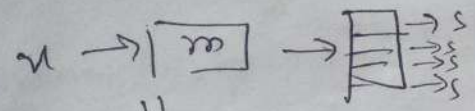
Fully connected Neural Networks

We can calculate $\frac{d\hat{y}_{sample}}{dw_{i,j}}$, $\frac{d\hat{y}_{sample}}{da_{i,j}}$

~~Cater~~

Categorical Cross Entropy

Classification
- Fixed number of class / label
- Usually predict a score for each class



one out of K classes