Nabila Tasnia
190104018

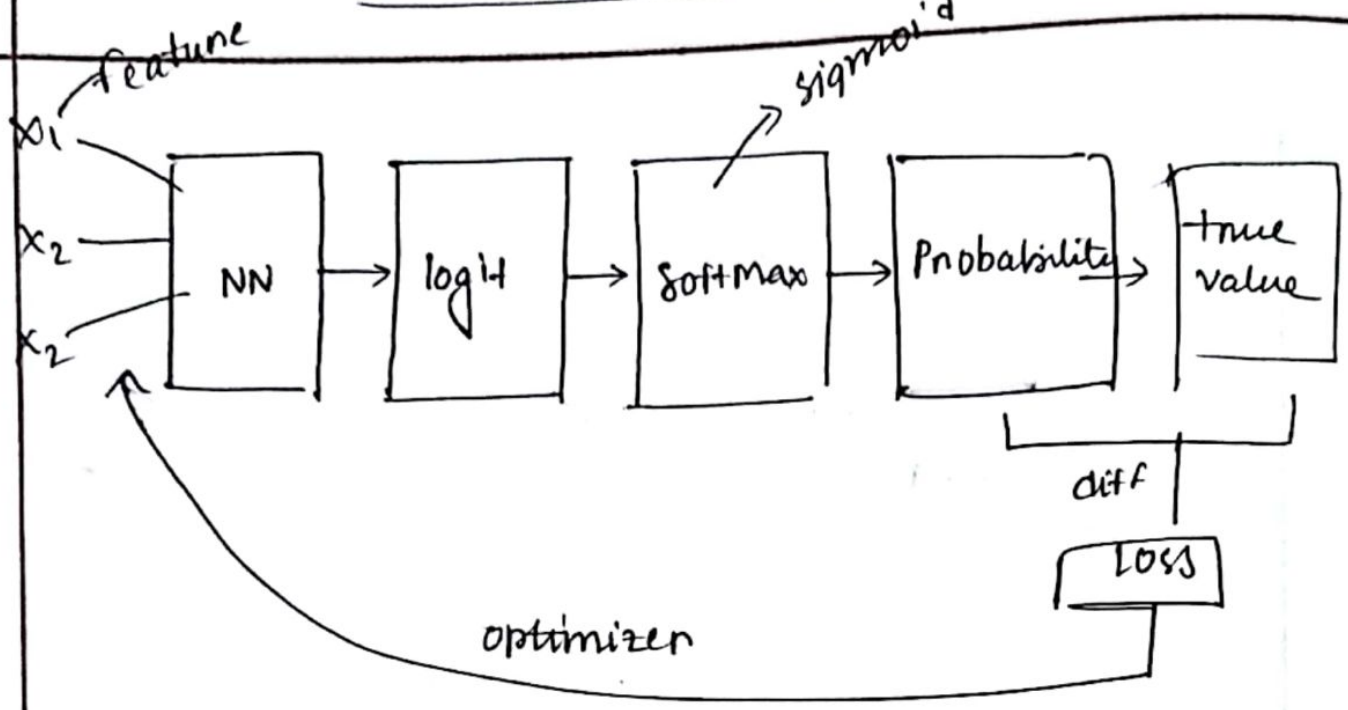
CSE: 4237

Topics:

1. Introduction to Neural Network
2. Neural Network Basics
3. Cross Entropy Loss
4. Shallow Neural Network
5. Hyperparameters and parameters
6. CNN
7. Parameter Calculation
8. Word Embedding
9. Natural Language Processing and Word Embedding
10. RNN, LSTM (incomplete)
11. HMM (incomplete)

- **Please do not rely on my calculations—there is a good chance they are wrong.**
- **If you don't understand any word, there is a good chance that I don't understand/remember it either.**

Introduction to Neural Network



softmax multiclass classification

→ Starts with a dataset that contains n input features.

→ The input features are fed into a neural network. The neural network can consist of several layers, including, input, hidden, output

→ A forward propagation is performed propagating the input features through the neural network.

$$y = x_1w_1 + x_2w_2 \dots + x_nw_n + b$$

After forward pass, the neural network produces output values called logits

→ Then the logits become input to activation function sigmoid / softmax. If multiclass softmax gives probability of each class. Sigmoid for pass output- multiple linear regression with logistic regression is.

difference between machine learning and deep learning

ML	DL
→ works on small amount of data	→ works on large amount of data
→ Dependent on low-end machine	→ dependent on high-end machine
→ divides tasks into subtask, solves individually and finally combines result	→ solves problem end to end
→ takes less time to train	→ long time to train
→ testing time may increase	→ less time to test data

Supervised learning

We are given a data set and we already know what our correct output should look like

→ We have some input(x) and we learn a function to map output(y)

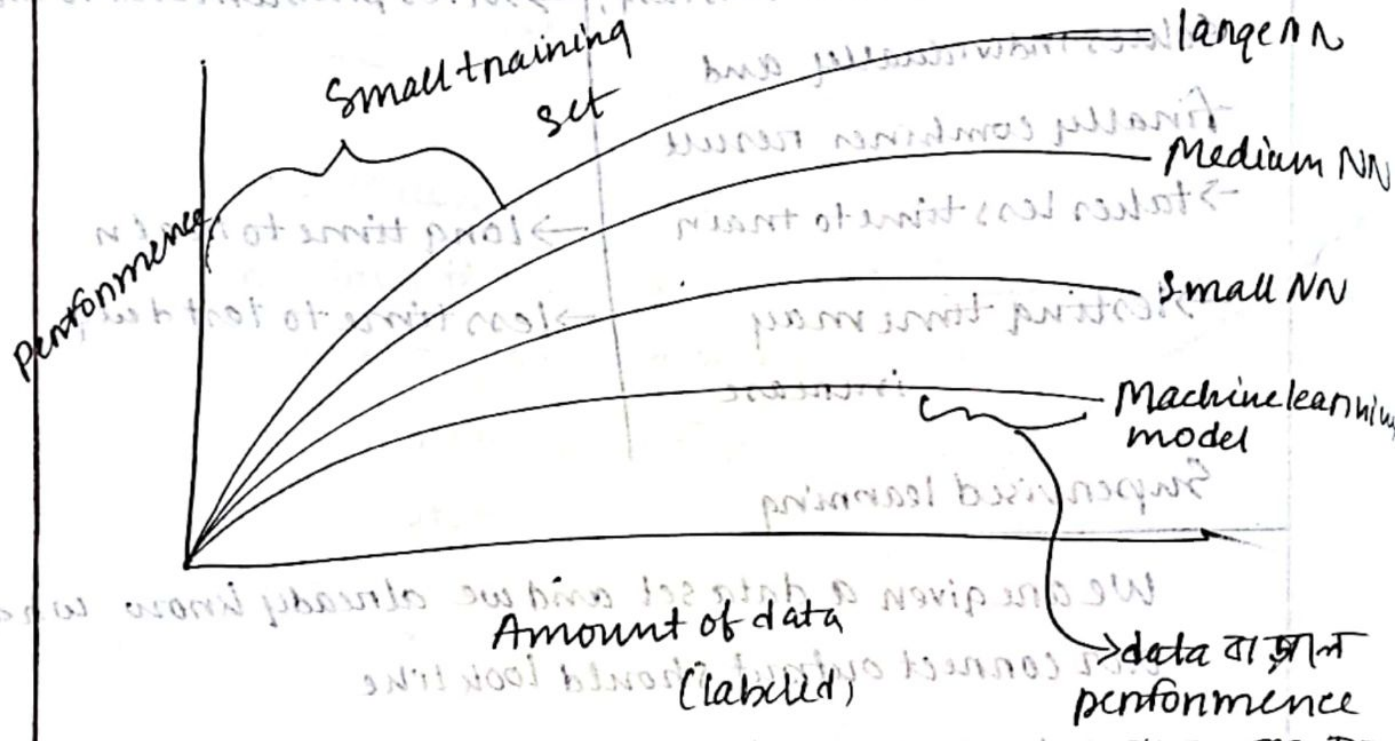
Structured data

→ databases and tables

Unstructured data

→ images vidio, audio, text

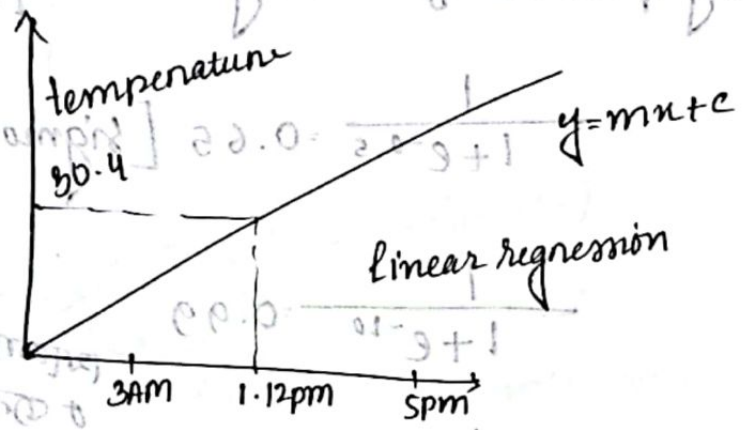
→ harden for computers to make sense of this kind of data



Single Neural Network

→ Given data about the size of houses, you want to fit a function that will predict the price of the houses.

→ it is linear regression because the price as a function of size is a continuous output



Regression → continuous value (যেকোন output আসবে)

Classification → discrete value (নির্দিষ্ট output আসবে)

Regression এ threshold value দিয়ে classification করা যায়।

This is logistic regression.

neural network unit = logistic regression

suppose here threshold is 25°

$y \geq 25$ ⟶ hot

$y < 25$ ⟶ cold

Logistic Regression: If we pass linear regression through sigmoid function, we get logistic regression, which is a classification.

$y = mx + c$ apply করে যা পাই, তা normalize করতে হয়। -

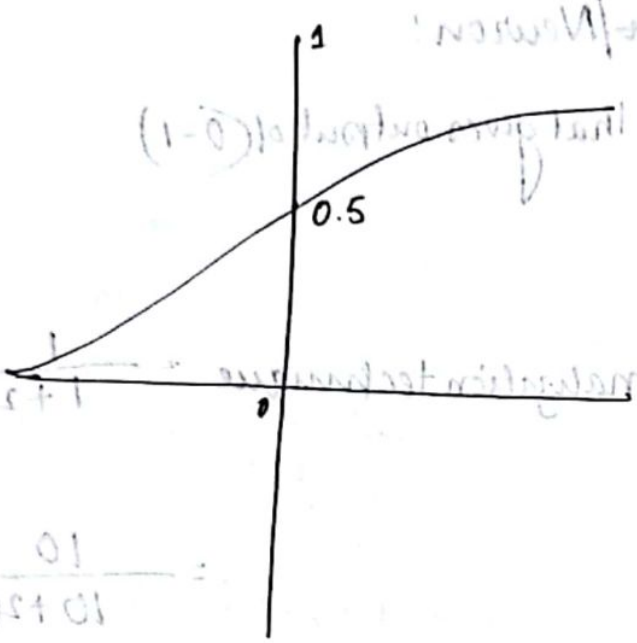
way of normalization → sigmoid $= \frac{1}{1+e^{-x}}$

$\frac{1}{1+e^{-0.5}} = 0.65$ [sigmoid এ threshold ধরা 0.5

$\frac{1}{1+e^{-10}} = 0.99$ যোগানো positive value এর জন্য threshold এর উপর আসবে।

$\frac{1}{1+e^{0.5}} = 0.37$ ranges between 0 to 1.

$\frac{1}{1+e^{100}} \approx 0$



6 (sigma)

6 (mx+c)

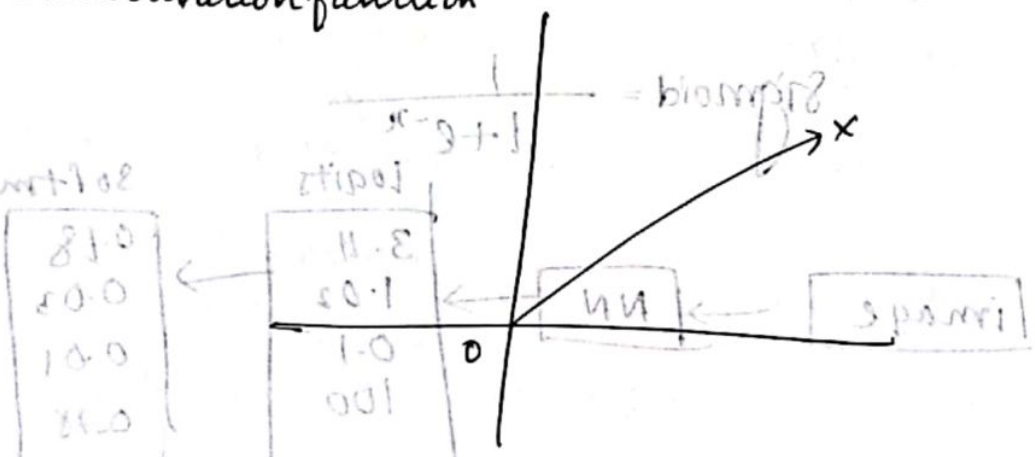
↳ sigm

mx+c → sigmoid (Activation function)

Activate or deactivate হয় কোন value based on value

→ Rectified Linear Unit

Relu activation function



#input এখা আসলে তা output এ দিবে তার negative এর value কে zero দিবে

Activation ~~function~~ / Neuron :

A function that gives output of (0-1)

~~Relu (0-~~

~~Normalization~~

Standard normalization technique $= \frac{1}{1+2} = \frac{1}{3}$

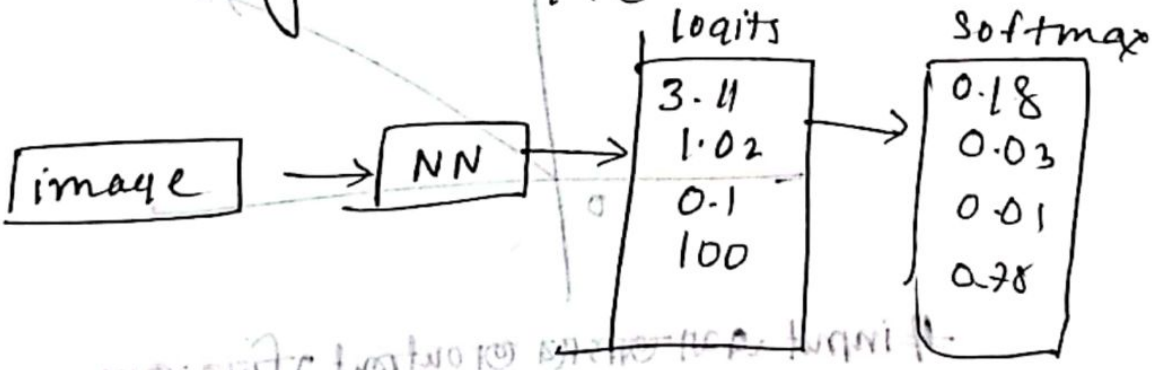
$= 0.33$

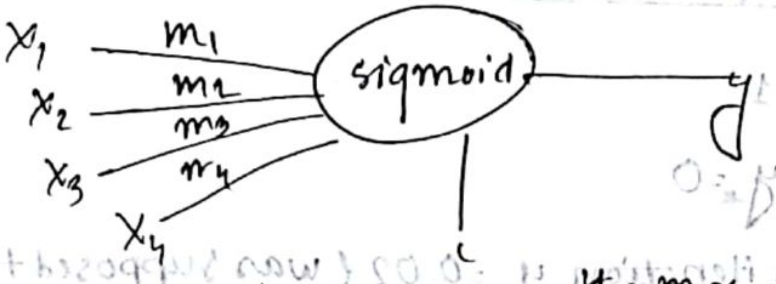
$= \frac{10}{10+20} = 0.33$

$$\text{softmax} = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}$$

softmax use করার কারণ হচ্ছে standard normalization এ সবকিছুতে summation দিবে।

$$\text{sigmoid} = \frac{1}{1+e^{-x}}$$





$y = x_1 m_1 + x_2 m_2 + x_3 m_3 \dots + c$

dimension = input + 1
↳ feature

(n+1) dimension হলে n সংখ্যক input and 1 টি output

$y = \sigma(x_1 w_1 + x_2 w_2 + x_3 w_3 + c)$

Image feature:

Image এর pixels এর feature.

24x24 image এর feature = 24x24x3 = 1728 (→ RGB (3 frames))

Neural network এ input দেয়ার জন্য feature matrix কে 1D করতে লাগবে.

$x = \begin{bmatrix} 255 \\ 231 \\ 42 \\ \vdots \\ 255 \\ 134 \\ 202 \\ \vdots \\ 255 \\ 134 \\ 93 \\ \vdots \end{bmatrix}$ (red, green, blue)

64x64x3 = 12284

for 64x64 image = 1D ⇒ (1, 64x64), (1, 64x64)

1 row, 12288 column

transpose করলে হবে 1 column, 12288 rows

neural network এ data "flatten" করে দিতে লাগবে

To create feature vector x, the pixel intensity values are unrolled and shaped for each color.

Neural Network Basics

cat ☺; not cat ~~prediction~~ classification

if cat, $y_1 = 1$

if not cat, $y_2 = 0$

after first iteration $y_1 = 0.02$ (was supposed to be 1)

loss = target $- y_1$

$= 1 - 0.02 = 0.98$

~~Optimization~~

Logistic regression

Logistic regression is a learning algorithm used in supervised learning problem when the output y are all ~~eq~~ either zero or one

→ The goal of logistic regression is to minimize the error between its predictions and training data

→ Given an image represented by feature vector x, the algorithm will evaluate the probability of a cat being in the picture

Given x, $\hat{y} = P(y=1|x)$, where $0 \leq \hat{y} \leq 1$

The parameters

The input feature vector: $x \in \mathbb{R}^{n_x}$, where n_x is the number of features.

The training label $y \in 0, 1$

parameters | The weights: $w \in \mathbb{R}^{n_x}$ where n_x is the number of features

the bias $b \in \mathbb{R}$

The output $\hat{y} = \sigma(w^T x + b)$

↳ weight vector transposed

Sigmoid function $s = \sigma(w^T x + b) = \sigma(z) = \frac{1}{1 + e^{-z}}$

Role of bias

→ The bias helps the activation function to be shifted to left or right, to better fit the data.

→ changes in weights alter the steepness of the sigmoid curve, whilst the bias offsets it, shifting the curve entirely so it fits better.

→ Bias can only influence the output values, it does not interact with the actual input data.

Bias is a measure of how easy it is to get a node to fire

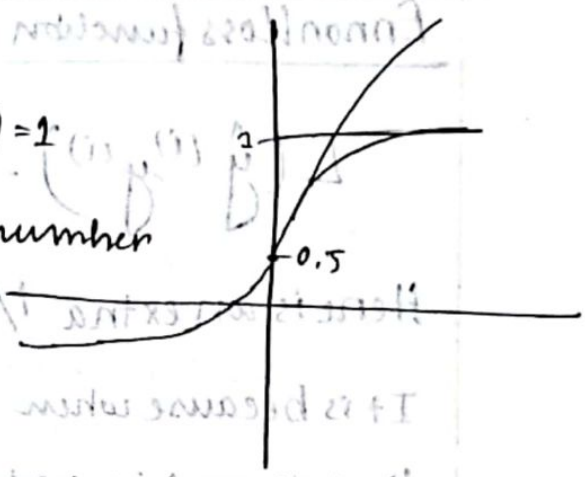
→ for a node with high bias, the output tends to be intrinsically high, with small positive weights and inputs producing large positive output (near to 1)

→ Bias can be negative, leading to sigmoid output near 0

→ If bias too small, the output will be decided by the value of weights and input alone

If z is a large positive number, then $\sigma(z) = 1$.

If z is a small or large negative number then $\sigma(z) = 0$



If $z = 0$, then $\sigma(z) = 0.5$.

Cost function

To train the parameters w and b, we need to define a cost function.

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

Given $\{(x^1, y^1), \ldots, (x^n, y^n)\}$ we want $\hat{y}^{(i)} \approx y^{(i)}$

↳ আমরা চাই predicted value আর actual value প্রায় কাছাকাছি হোক।

Loss function

Loss function measures the discrepency between the prediction ($\hat{y}^{(i)}$) and the desired output ($y^{(i)}$)

→ In other words, the loss function computes the error for single training sample.

Error/loss function

$$L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$$

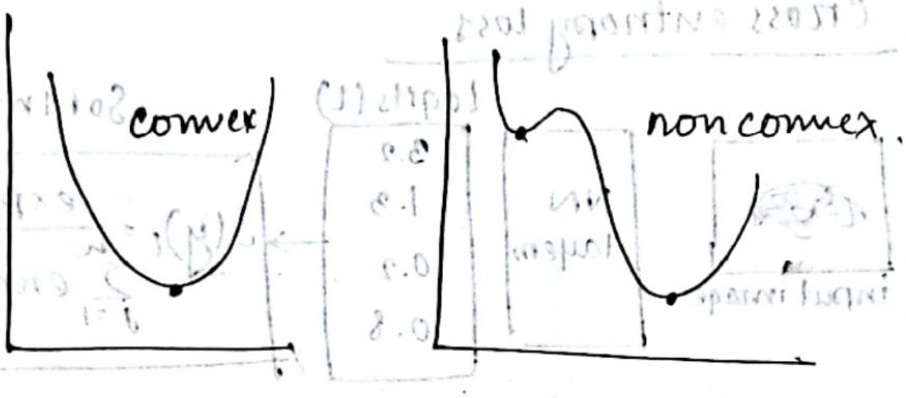
Here is an extra 1/2 in the equation.

It is because when we take the derivative of cost function, that is used in updating the parameter during gradient descent, that 2 in the power get cancelled with the 1/2 multiplier.

Is squared error function a good choice ~~for~~ ?

→ The square error function (commonly used function for linear regression) is not very suitable for logistic regression.

- In case of logistic regression, the hypothesis/prediction is non-linear (sigmoid function), which makes the square error function to be non convex
- On the other hand, logarithmic function is a convex function for which there is no local optima, so gradient descent works well.



If you are doing binary classification, squared error function generally also penalizes examples that are correctly classified but are still near decision boundary, thus creating a 'margin'

Gradient descent waste a lot of time getting prediction very close to {0,1}

Is it necessary for loss function to be convex?

Convexity is a desirable property for loss function in optimization problems. A convex loss function guarantees the existence of a unique global minimum. It simplifies the optimization by ensuring that the solution is not trapped in local optima, making it easier to find best possible solution.

Gradient Descent

Logistic regression: cross entropy loss

$$L(\hat{y}^{(i)}, y^{(i)}) = -\left(y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)})\right)$$

If $y^{(i)}=1$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(\hat{y}^{(i)})$ where $\log(\hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 1

If $y^{(i)}=0$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(1-\hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 0.

Cost function

Cost function is the average of loss function of the entire training data set. We are going to find the parameters w and b that minimises the overall cost function.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)})\right]$$

So far we know

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

$$J(w,b) = \frac{1}{m}\sum L(\hat{y}^{(i)}, y^{(i)})$$

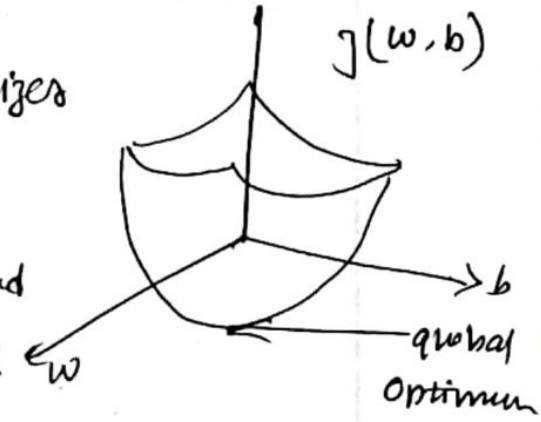
$$= -\frac{1}{m}\sum y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-y^{(i)})$$

Gradient descent

Our cost function is a convex.

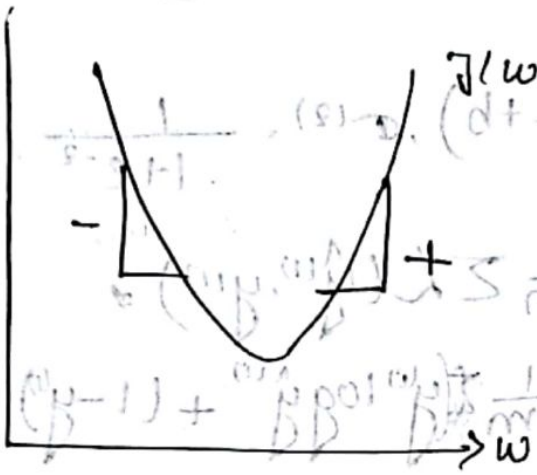
→ first we initialize w, b to 0, 0 or random values in the convex function and try to improve the values to reach minimum value

we want parameters that minimizes $J(w,b)$



Gradient starts at the initial point and take a step in the steepest downhill direction after each iteration.

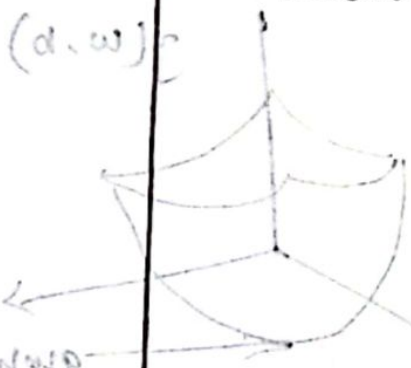
It will try to reach global optimum.



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

↳ learning rate

α - learning rate, how biggen steps we choose at each iteration of gradient descent



Actual update rule:

We want panameters w, b to minimize J(w,b)

$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$ (partial derivative)

J(w,b)

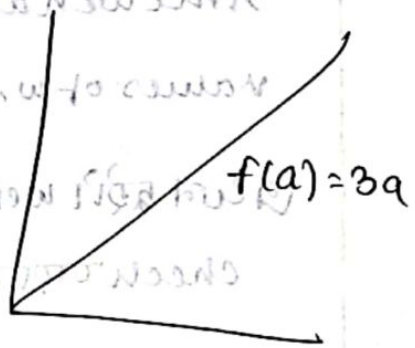
$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

derivative

a

a = 2 , f(a) = 6

a = 2.001 f(a) = 6.003



If we shift a by 0.001, f(a) shifts by 3 times 0.001

a = 5 f(a) = 15

a = 5.001 f(a) = 15.003

here slope is 3

$$\frac{df(a)}{da} = 3$$

It mean rate of change is 3

Do we actually need gradient descent?

Let's say, we have 1 weight. To find the ideal weight to minimize our cost, we need to try a bunch of values ~~for~~ for w, (1000 values)

→ It takes 0.04 seconds to check 1000 different weights values for our neural network.

Since we have computed the cost for a wide range values of w, we can just pick one with the smallest cost

এভাবে দুইটা weight নিয়ে আমরা 1000x1000 values check করতে পারতাম।

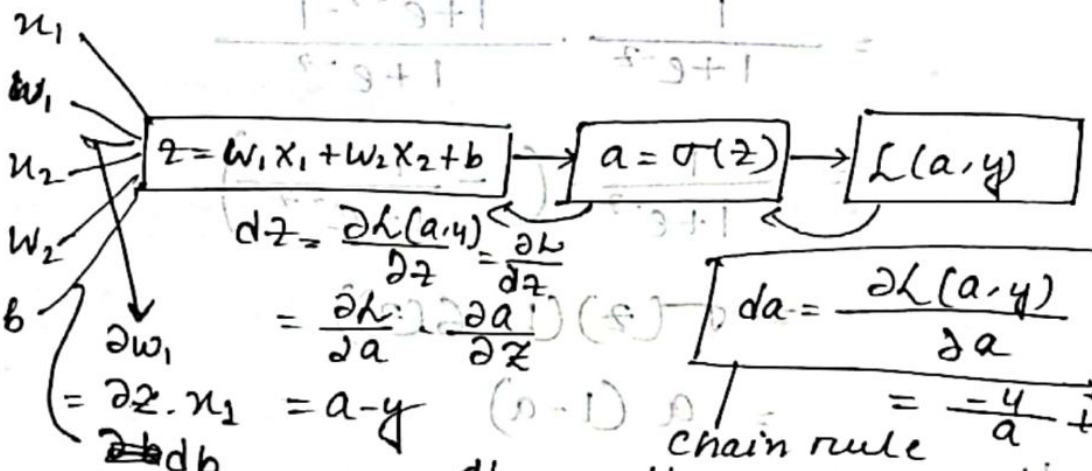
Logistic Regression: Forward Propagation

$z = w^T x + b$

$\hat{y} = a = \sigma(z) \rightarrow$ Prediction / Sigmoid output

$L(a, y) = -(y \log a + (1-y) \log(1-a)) \rightarrow$ cross entropy loss

computing loss of a single training example



$$\frac{dL}{dw_1} = \frac{dL}{da} \cdot \frac{da}{dz} \cdot \frac{dz}{dw_1}$$

$$= \frac{dL}{dw_1} \cdot$$

$$L(a, y) = -(y \log a + (1-y) \log(1-a))$$

$$\frac{dL}{da} = \frac{d}{da}\left(-y \log_2 a - (1-y) \log(1-a)\right)$$

$$= -\frac{y}{a} + \frac{(1-y)}{1-a}$$

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

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

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

$$= -1 \times \left(1+e^{-z}\right)^{-2}\left(-e^{-z}\right)$$

$$= \frac{e^{-z}}{\left(1+e^{-z}\right)^2}$$

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

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

$$= \sigma(z)(1-\sigma(z))$$

$$= a(1-a)$$

$$\frac{dL}{da} \cdot \frac{da}{dz}$$

$$= \left(\frac{-y}{a} + \frac{1-y}{1-a}\right) \times a(1-a)$$

$$= \frac{-y(1-a) + a(1-y)}{a(1-a)} \times a(1-a)$$

$$= -y + ay + a - ay$$

$$= a - y$$

$z = w_1 x_1 + w_2 x_2 + b$

$\frac{dz}{dw_1}$

$\frac{\partial z}{\partial w_1} = x_1$

$\frac{dL}{dw_1} = \frac{dL}{da} \times \frac{da}{dz} \times \frac{dz}{dw_1}$

$= (a-y)x_1$

$\frac{dL}{dw_2} = (a-y)x_2$

Logistic regression, gradient descent on m examples

$J=0,\ dw_1=0,\ dw_2=0,\ db=0,\ w1=0,\ w2=0,\ b=0$

for i=1 to m

$z = w_1 x_1(i) + w_2 x_2(i) + b$

$a(i) = \text{sigmoid}(z)$

$J += y(i)\log a(i) + (1-y(i))\log(1-a(i))$

Backward pass

$dz(i) = a(i) - y(i)$

$dw1 += dz(i) \times x1(i)$

$dw_2 += dz(i) \times x2(i)$

$db += dz(i)$

$J /= m$

$dw_1 /= m$

$dw_2 /= m \quad db /= m$

$Z^{[1](i)}_{(a)}$ → No of layer; (i) → No of sample; (a) → No of node

Gradient descent

$w1 = w1 - alpha \times dw_1$

$w2 = w_2 - alpha \times dw2$

$b = b - alpha \times db$

Here, we have done one step of gradient descent, we need to repeat it multiple times in order to take multiple steps of gradient descent.

In this code, we implement two for loops, which is not very efficient. Solution = vectorization technique

Modified LR Gradient Descent

$J=0,\ dw1=0,\ dw2=0,\ db=0,\ w_1=0,\ w_2=0,\ b=0$

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

In backward pass

previously

$dw1 += dz(i) \times x_1(i)$

$dw2 += dz(i) \times x_2(i)$

modified

$dw\,t = x(i) \times dz(i)$

In Gradient Descent,

previously	modified
$dw1 = dw1/m$	$dw = dw/m$
$dw2 = dw2/m$	

Vectorizing Logistic regression (forward)

$$X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$$ while $\mathbb{R}^{n \times m}$

$x^{(1)}$ → 1st sample, $x^{(2)}$ → 2nd sample

$w = n \times 1$

$X = n \times m$

$$\begin{bmatrix} w^T \end{bmatrix} \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

$w^T X = (1 \times n) \cdot (n \times m)$

$= (1 \times m)$ → 1 no dim

1st training example	2nd training example	3rd training example
$z^{(1)} = w^T x^{(1)} + b$	$z^{(2)} = w^T x^{(2)} + b$	$z^{(3)} = w^T x^{(3)} + b$
$a^{(1)} = \sigma(z^{(1)})$	$a^{(2)} = \sigma(z^{(2)})$	$a^{(3)} = \sigma(z^{(3)})$

$$[z^{(1)} z^{(2)} z^{(3)} \ldots z^{(m)}] = W^T X + [b \ldots b] = [w^T x^{(1)} + b \quad w^T x^{(2)} + b \quad \ldots \quad w^T x^{(m)} + b]$$

↳ $1 \times m$

$$Z = np.dot(w.T, X) + b$$
↳ (1,1)
(broadcasting)

$$A = [a^{(1)} a^{(2)} \dots a^{(m)}] = \sigma(Z)$$

$$dz = \frac{dL}{da} \cdot \frac{da}{dz} \quad dz^{(1)} = a^{(1)} - y^{(1)} \quad dz^{(2)} = a^{(2)} - y^{(2)}$$

$$dz^{(3)} = a^{(3)} - y^{(3)}$$

$$dz = [dz^{(1)} dz^{(2)} \dots dz^{(m)}]$$

$$A = [a^{(1)} a^{(2)} \dots a^{(m)}]$$

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

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

gradient computation

previously

$dw = 0$

$dw += X^{(1)} dz^{(1)}$

$dw += X^{(2)} dz^{(2)}$

$\vdots$

$dw += X^{(m)} dz^{(m)}$

$dw /= m$

$db = 0$

$db += dz^{(1)}$

$db += dz^{(2)}$

$\vdots$

$db += dz^{(m)}$

$db /= m$

modified

$$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$$

$$db = \frac{1}{m} np.sum(dZ)$$

$$dw = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)} \cdot x^{(i)}$$

$$dw = \frac{1}{m} \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ dz^{(2)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$$

$$dw = \frac{1}{m} \left[X^{(1)} dz^{(1)} + X^{(2)} dz^{(2)} \cdots X^{(m)} dz^{(m)} \right]$$

Single iteration of gradient descent

$$Z = w^T X + b$$

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

$$A = \sigma(Z)$$

$$dZ = A - Y$$

$$dw = \frac{1}{m} X \, dZ^T$$

$$db = \frac{1}{m} np.sum(dZ)$$

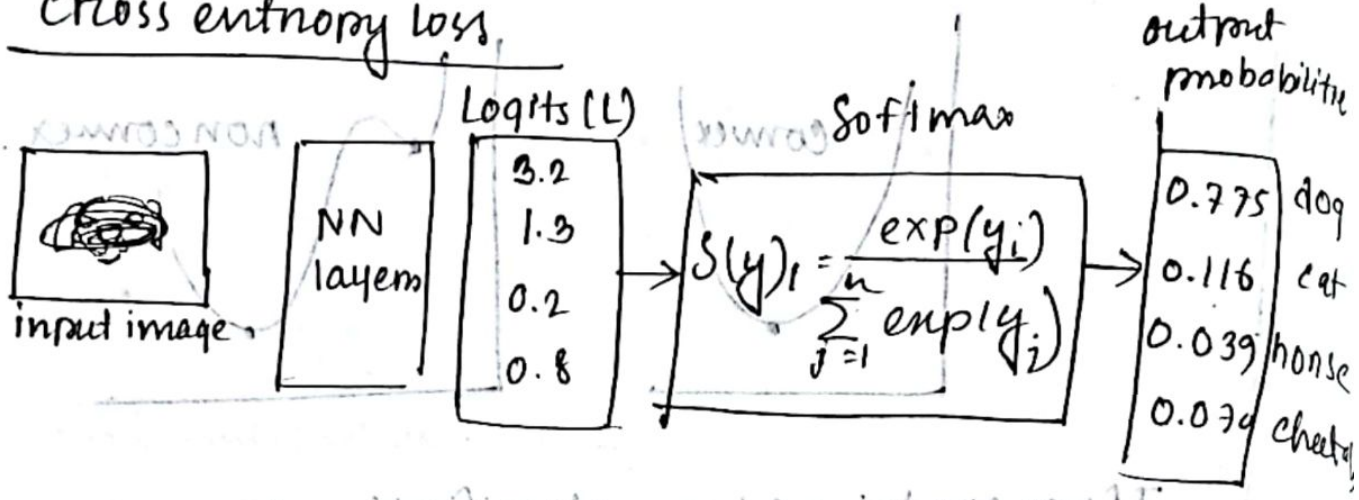
$$w := w - \alpha \, dw$$

$$b := b - \alpha \, db$$

[gradient update]

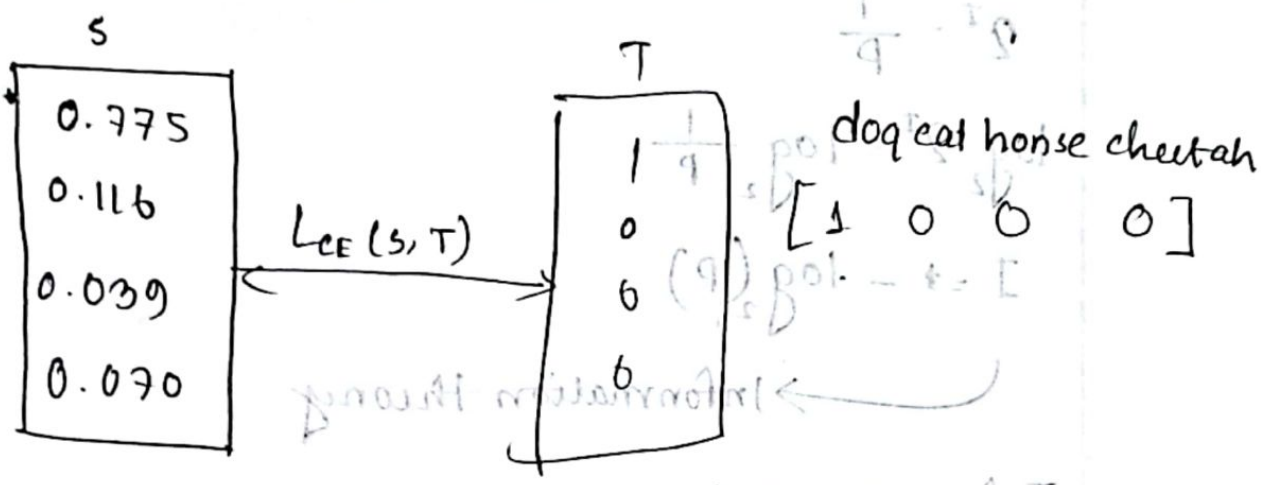
Slide: Cross entropy loss

Cross entropy loss



Logits

The vector of raw (non-normalised) predictions that a classification model generates, which is ~~ordinary~~ ordinarily then passed to a normalization function. If the model is solving a multi-class classification problem, logits typically become an input to the softmax function. The softmax function then generates a vector of (normalized) probabilities with one value for each possible class

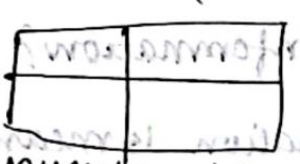


Entropy

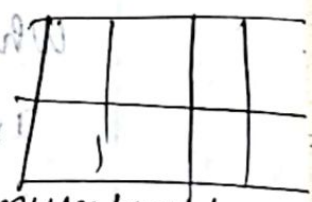
Entropy of a ~~non~~ random variable x is the level of uncertainty inherent in the variable possible outcome.

$$H(X) = \begin{cases} \int_x P(x) \log P(x) & \text{if } x \text{ is continuous} \\ -\sum_x P(x) \log P(x) & \text{if } x \text{ is discrete.} \end{cases}$$

number of box = 2
number of ques / info = 1



number of box = 4
number of ques = 2



number of box = 8
number of ques = 3

$$2^{\text{info}} = \text{number of box}$$

$$\text{number of box} = 2^I$$

$$P = \frac{1}{\text{number of box}}$$

$$\text{number of box} = \frac{1}{P}$$

$$2^I = \frac{1}{P}$$

$$\log_2 2^I = \log_2 \frac{1}{P}$$

$$I = -\log_2(P)$$

→ Information theory

Information Vs Entropy

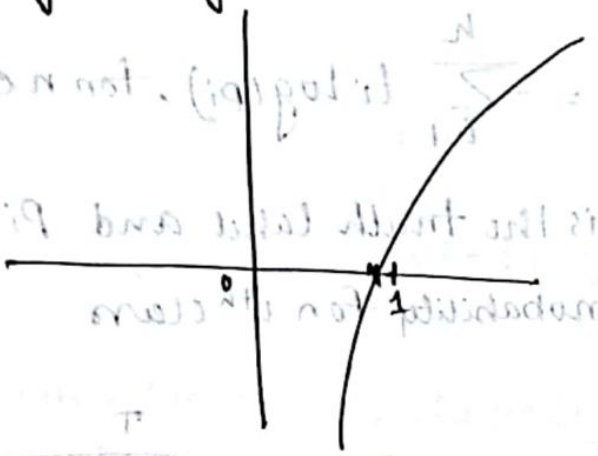
Information provides a way to quantify the amount of surprise for an ~~an~~ event measured in bits.

Entropy provides ~~a way~~ to measure of the average amount of information needed to represent an event drawn from a probability distribution for a random variable.

What is information?

Information is measured in bits. When measuring information, we are measuring the number of bits or the number of "yes" "no" questions we would have to ask to reach the same conclusion.

Reason of negative sign



$\log(p(x)) < 0$ for all $p(x)$ in $(0,1)$.

Cross entropy loss

→ Cross entropy loss is also called logarithmic loss, log loss, logistic loss.

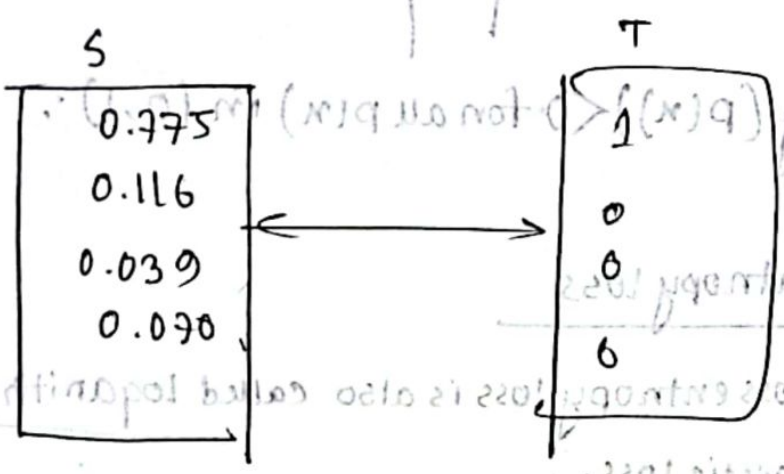
→ Each predicted class probability is compared to the actual class desired output of 0 or 1.

→ A score / loss is calculated that penalises the probability based on how far it is from the actual expected value.

Cross entropy loss

$$L_{CE} = -\sum_{i=1}^{n} t_i \log(p_i), \text{ for } n \text{ classes},$$

where t_i is the truth label and p_i is the softmax probability for i^{th} class



$$L_{CE} = \sum_{i=1}^{n} T_i \log(S_i)$$

$$= -[1 \log_2(0.775) + 0 \log_2(0.116) + 0 \log_2(0.039) + 0 \log_2(0.070)]$$

$$= -\log_2(0.775)$$

$$= 0.3677$$

Binary cross entropy loss

$$L = -\sum_{i=1}^{2} t_i \log(P_i)$$

$$= [-t \log(P) + (1-t) \log(1-P)]$$

When using log base 2, the unit of entropy is bits, whereas with natural log the unit is nats.

⁂ লোগারিদম - আপনার চেয়ে better না, - its like the difference between km/hour and m/s

Categorical crossentropy loss | Sparse categorical cross entropy

Both categorical cross entropy and sparse categorical cross entropy have the same loss function. The only difference between the two is how truth labels are defined.

Categorical cross entropy is used when true labels are one-hot encoded. [1, 0, 0]

In sparse categorical cros entropy, truth labels are integer encoded : [1, 2, 3]

Loss vs cost

The loss function computes the error for a single training example, while the cost is the average of the loss function of the entire ~~do~~ training set.

$$L = -\frac{1}{N}\left[\sum_{j=1}^{N}\left[t_j \log p_j + (1-t_j)\log(1-p_j)\right]\right]$$

WORDLE

The goal of wordle is to guess 5 ~~too~~ letter words, with 6 guesses. After answering, each letter turns a green, amber or grey

Green => correct spot

Amber = a letter is is word, but not in correct spot

Grey = not in the word.

5757 five letter words in english.

At beginning = there are $\log_2(5757)$ or 12.49 bits of hidden info.

As you play, you acquire more information about the hidden word.

The goal is to create an algorithm that in each guess chooses the word that adds the most info.

(and reduces uncertainty the most)

→ the initial guess is 'tares'

→ there are only 12 words in english language that matches this criteria

→ we have reduced the uncertainty from 12.5 bits to 3.6 bits

→ Our guess made us gain 8.9 bits of info

→ after the second guess "orate", the only possible word is great

---

On the other hand, if we enter the word 'fuzzy', there are 62% (5 letter words) in English that do not contain 'f' 'u' 'z' 'y'; so 62% of time we will end up with a wordle that's all grey.

all gray is the outcome with the least amount of information gained.

If we put 'lanes' as the initial word, in only 7% of the times we will get a wordle all grey.

93% of the time we will know at least one letter.

fuzzy reduces our possible word list from

5757 to 1185

lanes reduces it to 78.

Quiz-10

Slide: Shallow Neural Networks

Activation functions

#activation functions can be used in hidden layer and output layers.

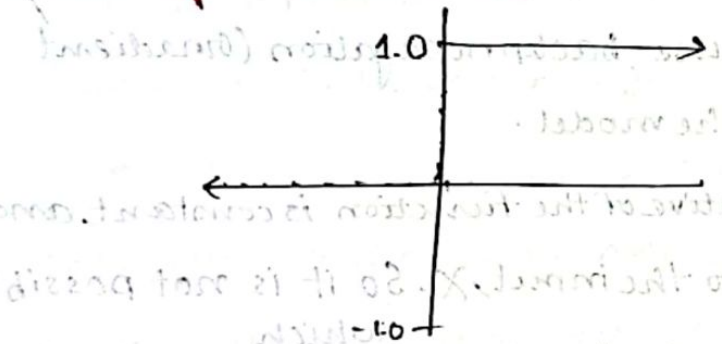
$z^{[1]} = w^{[1]} x + b^{[1]}$

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

$z^{[1]} = w^{[2]} a^{[1]} + b^{[2]}$

$a^{[2]} = \sigma(z^{[2]})$

Binary step function



threshold based activation function

- If the input value is above on below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer.

Problem

⇒ does not allow multi value outputs.

⇒ It is not recommended to use in hidden layer because it does not represent derivative learning value

Types of activation function

Linear activation function.

⇒ form $A = cx$

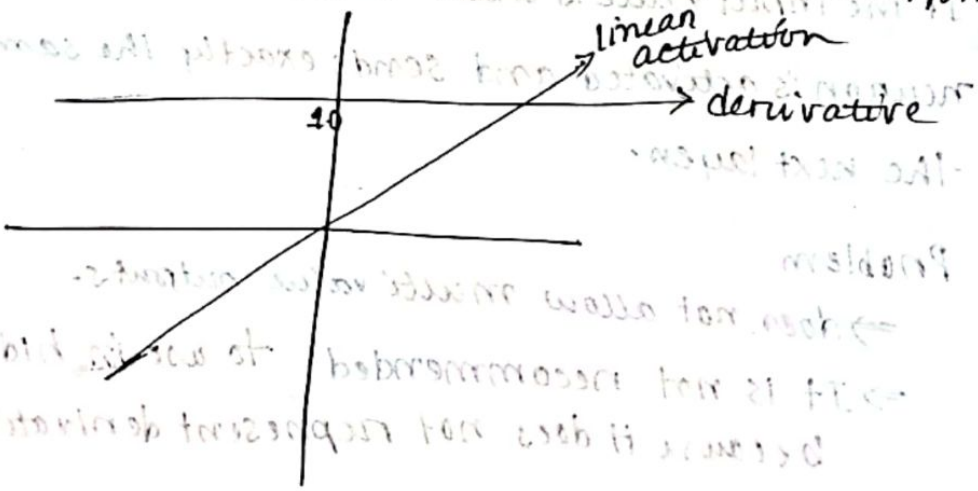
⇒ It takes input multiplied by weights of each neuron and creates an output signal proportional to the input.

Linear function is better than step function because it allows multiple outputs, not just yes and no.

Problems:

→ not possible to use backpropagation (Gradient Descent) to train the model.

- The derivative of the function is constant, and has no relation to the input, X. So it is not possible to go back and understand ~~why~~ which weights in the in the input neurons can provide a better prediction



- When $A = cx$ is derived from x, we reach c. ~~That~~ As there is ~~not~~ no relationship with x in the derivative, therefore there is no learning.

⇒ All layers of neural network will collapse into one -

- With linear activation function, no matter how many layers in the neural network, the last layer will be a linear function of the first layer (because a linear combination of linear functions is still a linear function). So a linear activation function turns the neural network, on even deep neural network into just one layer

⇒ ~~A linear acti~~ A neural network with ~~a~~ linear activation function on without any activation function is simply a linear regression model. It has limited power and ability to handle complexity varying parameters of input data.

with linear activation function on no activation function

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

$$a^{[1]} = z^{[1]}$$

Why do we need non linear activation function?

→ If linear activation function or no activation function is used, the output signal becomes a simple linear function.

⇒ Linear functions are only single grade polynomials

⇒ A non activated neural network will act as a linear regression with limited learning power.

⇒ We want our neural network to learn non-linear states. Because, we will give it complex real world information such as images, videos, texts and sounds which are non linear and have high dimensionality to learn to your neural network.

⇒ Multilayered deep neural network can learn meaningful features from data.

⇒ It allows backpropagation because they have a derivative function, which is related to the input.

→ They allow stacking of multiple layers of neurons to create a deep neural network. Multiple hidden layers of neurons are needed to learn complex datasets with high level of accuracy.

- simple linear function
- single grade polynomial
→ linear regression with limited learning power
- need to process complex data
→ allows backpropagation
→ allows stacking of multiple layers of neuron
⇒ can learn meaningful features from data

for example

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

$$a^{[2]} = z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$$

$$= w^{[2]}a\left(w^{[1]}x + b^{[1]}\right) + b^{[2]}$$

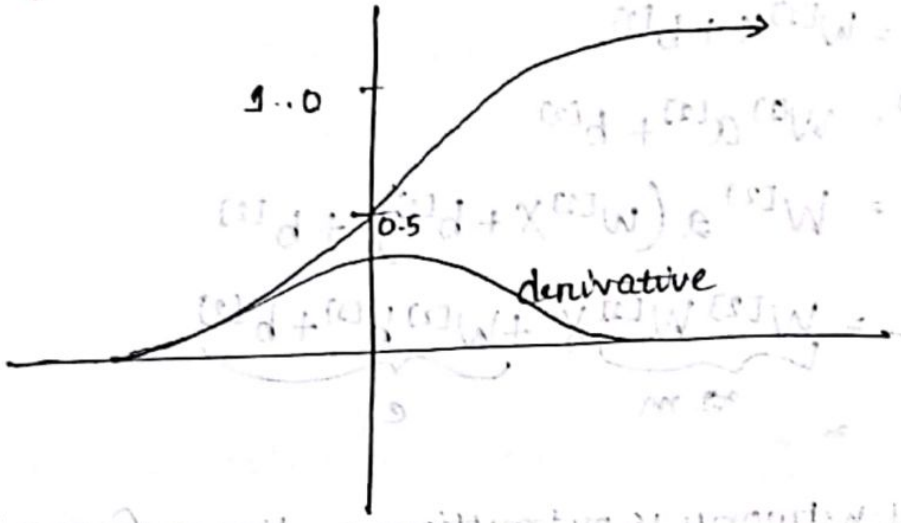
$$= \underbrace{w^{[2]}w^{[1]}}_{m}x + \underbrace{w^{[2]}b^{[1]} + b^{[2]}}_{c}$$

$$=$$

Neural network is outputting a linear function of inputs. Composition of two or more linear function is also a linear function.

⇒ One place you can use a linear activation function when you are solving a regression problem, meaning the output is a real number. But only in the last hidden layer and not in the intermediate layers.

Sigmoid / Logistic function



It can is also derived because it is different from step function, which means <u>Learning can happen!</u>

⇒ <u>smooth gradiant</u>, prevents jumps in the output values.

⇒ output values bound between 0 and 1.

⇒ if X is above 2 or below -2, tends to bring y value (prediction) to the edge of the curve, very close to 1 or 0. This enables clear prediction.

⇒ The sigmoid function is the most frequently used activation function, but there are alternatives

Problem:

Vanishing Gradient: for very high and very low values of X, there is almost no change to the prediction. The derivative values in these ranges are very small and converges to 0, This is called vanishing gradient and learning is minimal.

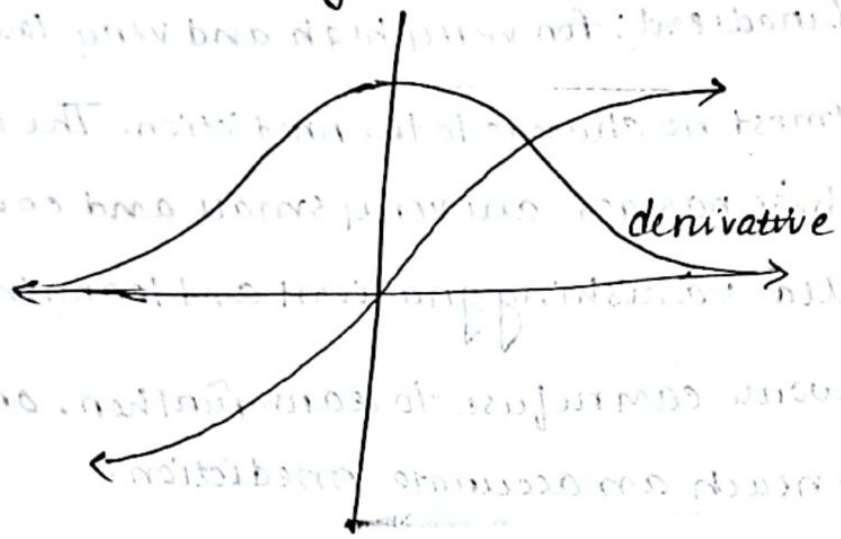
→ The network can refuse to learn further, on being too slow to reach an accurate prediction.

→ When slow learning occurs, the optimization algorithm that minimizes error can be attached to local minimum values and can not get maximum performence from the ~~antifica~~ artificial neural network.

⇒ Output is not zero centered. So output of all the neurons will be of same sign.

⇒ Computationally expensive

Hyperbolic Tangent function (-tanh)



similan structure to sigmoid function.

zero centered - making it easier ~~for~~ to model inputs that have strongly negative, neutral and strongly positive values. The range of values in this case is from -1 to +1

⇒ The advantage over sigmoid function is that its derivative is more steep, which means it can get more values.

This means it will be more efficient because it has a widen range for faster learning and grading.

The problem of gradients at the end of function continues.

Linear function: $f(x) = x \quad (-\infty, \infty)$

Step function: $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \quad \{0, 1\}$

Sigmoid function: $f(x) = \sigma(x) = \frac{1}{1+e^{-x}} \quad (0, 1)$

hyperbolic tan: $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (-1, 1)$

derivative of tanh.

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

$$\frac{d}{dz} g(z) = \frac{(e^z + e^{-z})(e^z + e^{-z}) - (e^z - e^{-z})(e^z - e^{-z})}{(e^z + e^{-z})^2}$$

$$= \frac{(e^z + e^{-z})^2 - (e^z - e^{-z})^2}{(e^z + e^{-z})^2}$$

$$= \frac{(e^z + e^{-z})^2}{(e^z + e^{-z})^2} - \frac{(e^z - e^{-z})^2}{(e^z + e^{-z})^2}$$

$$= 1 - \tanh(z)^2$$

$$= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2}$$

$$= 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2$$

$$= 1 - a^2$$

$d/dx\left(\frac{u}{v}\right) = \frac{vu' - uv'}{v^2}$

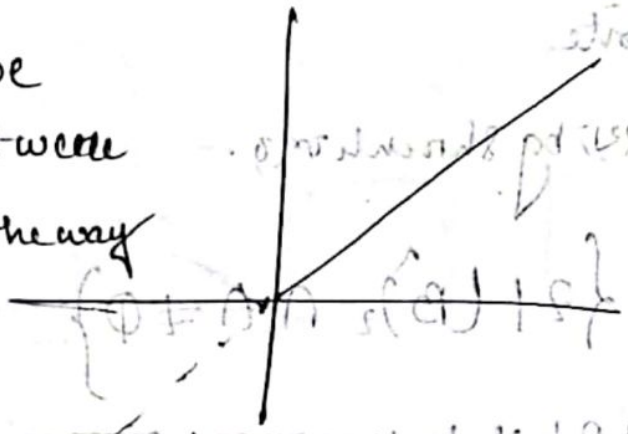
Reference

dropout = randomly dropping some neurons of neural network

~~Hyperbolic tangent function~~

linear ও backpropagation এর

it would be linear, if it were linear all the way



looks like linear, acts like non linear.

$q = u$ for $u > 0$

$q = 0$ for $u \leq 0$

ReLU

advantage

⇒ positive এ linear হওয়াতে loss function এ ঝামেলা, so ~~derivative~~ weight update করা যায় (for posidives)

→ ~~ই~~ neural network fast converge করে, dropout এর মতো করে, negative value drop করার সার্থকতা;

→ computational load কমানো যায় calculation easy

dying ReLU problem:

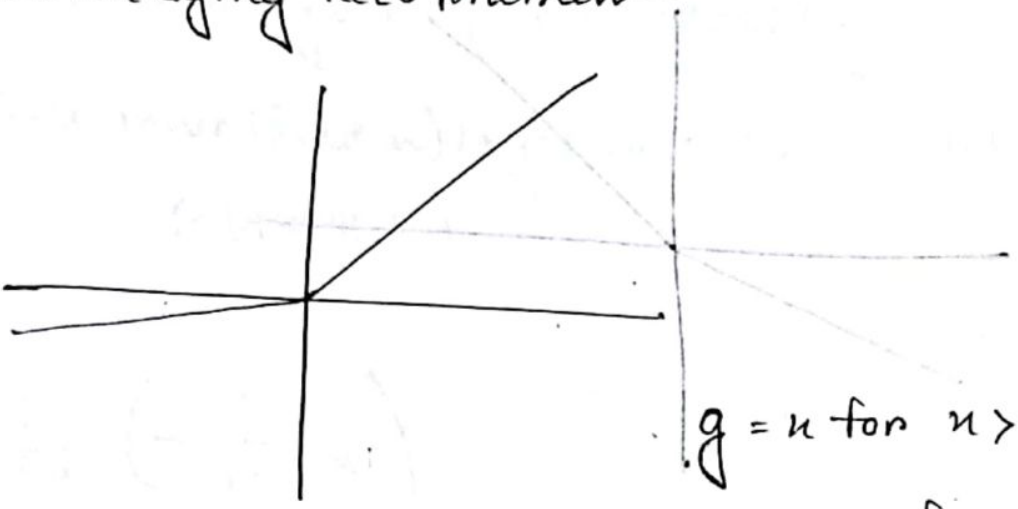
disadvantage

→ negative এ zero হয়;

→ positive এ linear

→ negative value র জন্য back propagation হবে না; ∴ derivative 0 হয়ে যাবে;

Solution for dying ReLU problem



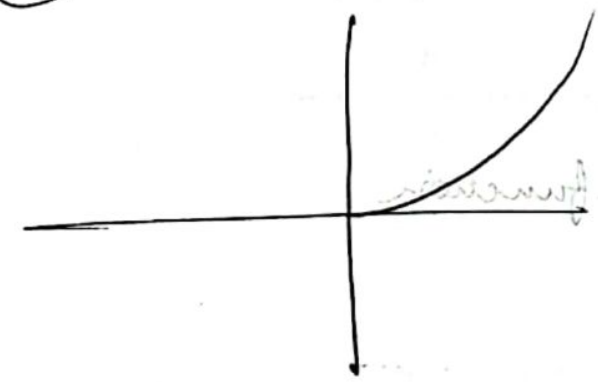
$g = n$ for $n > 0$

$g = 0.1\,n$ for $n \leq 0$

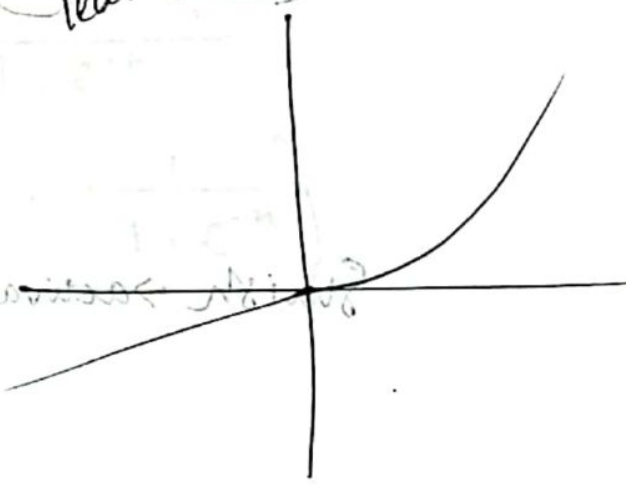
leaky ReLU activation function

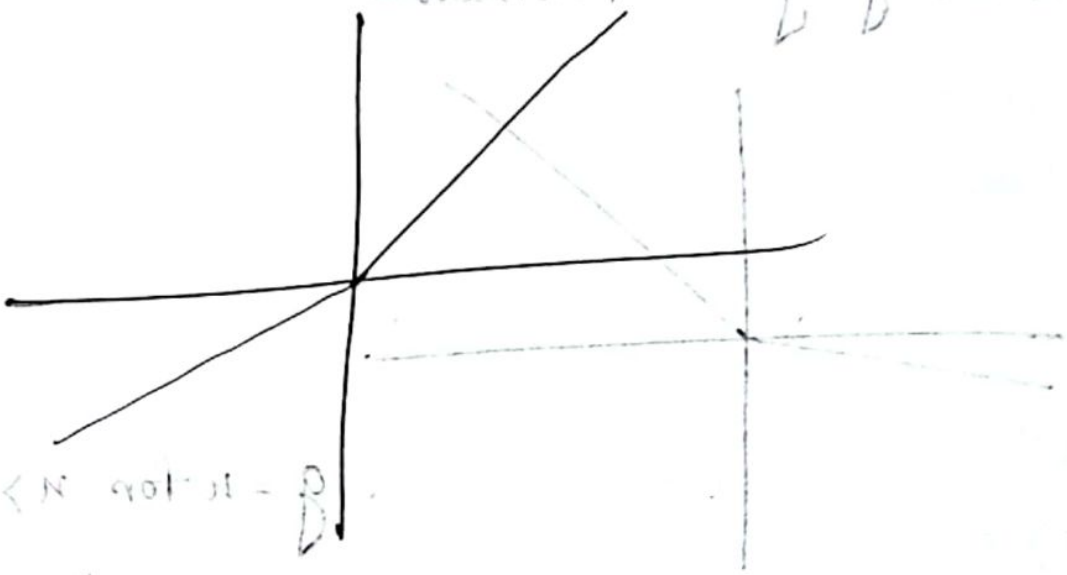
which activation function is suitable depends on data

ELU → exponential



leaky ELU





Parameterized ELU

GELU

Gaussian

Swish → activation function

Quiz-3

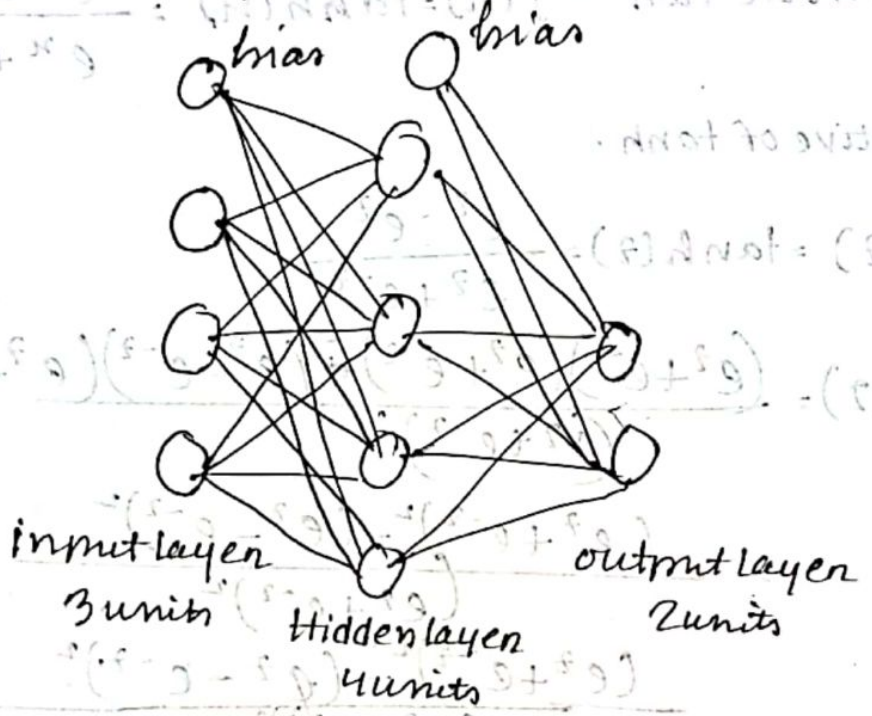
Hyperparameters and parameters

Parameter

weight, bias এর training time এ set হওয়ার জন্য change হয়;-

⇒ not set by users

⇒ learned from data



Input layer এর প্রতি node hidden layer এর 4 node এর সাথে connected. প্রতি connection এ weight থাকে,-

এই 2 layer এ total weight = 3 × 4

Hidden layer এর each node output layer এর 2 node এর সাথে connected, so, for 4 hidden layer = 4 × 2

one bias for 4 nodes in hidden layer = 4 × 1

one bias for 2 nodes in output layer = 2 × 1

total weights = 12 + 8 + 4 + 2 = 26

Hyperparameter tuning

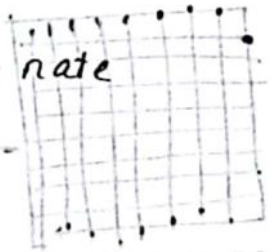
⇒ External to model

⇒ can not be estimated from data, set before training

⇒ determines network structure

Example, dropout, # of hidden layers, weight initialization, Active functions

- Epoch, Batch size, optimizer, learning rate



Tuning techniques

⇒ Randomly ~~chooses~~ search

Randomly chooses from all possible combinations of hyperparameters

⇒ Grid Search

Tests all possible combination of hyperparameters of given machine learning algorithm.

⇒ Bayesian optimizer

Based upon bayes' rule and considers previously known knowledge to help narrow down search space of good hyperparameter combinations

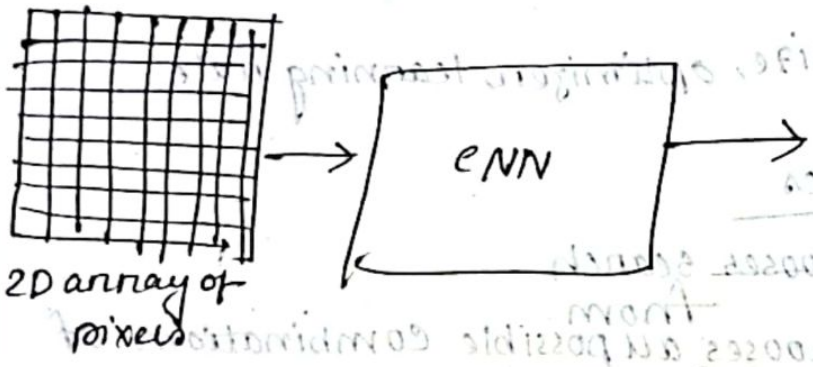
⇒ Gradient-based

A methodology to optimize several hyperparameters,

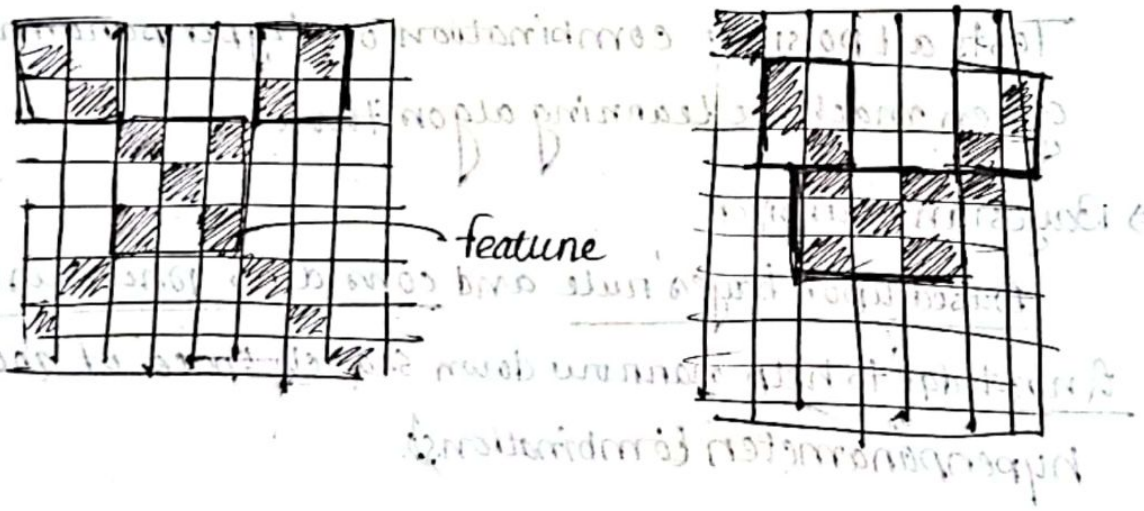
Based on the computation of the gradient of machine learning model selection criterion w.r.t hyperparameters

CNN

Convolutional Neural Network



⇒ Feature engineering এর মতো।



Filtering

This model does feature engineering by itself. So it finds some 'filters/features' and runs it through the input image

- X dim - filter size + 1 → valid padding
- X dim → same (original img তে 0 padding করে)

Pooling:

Shrinking the image stack

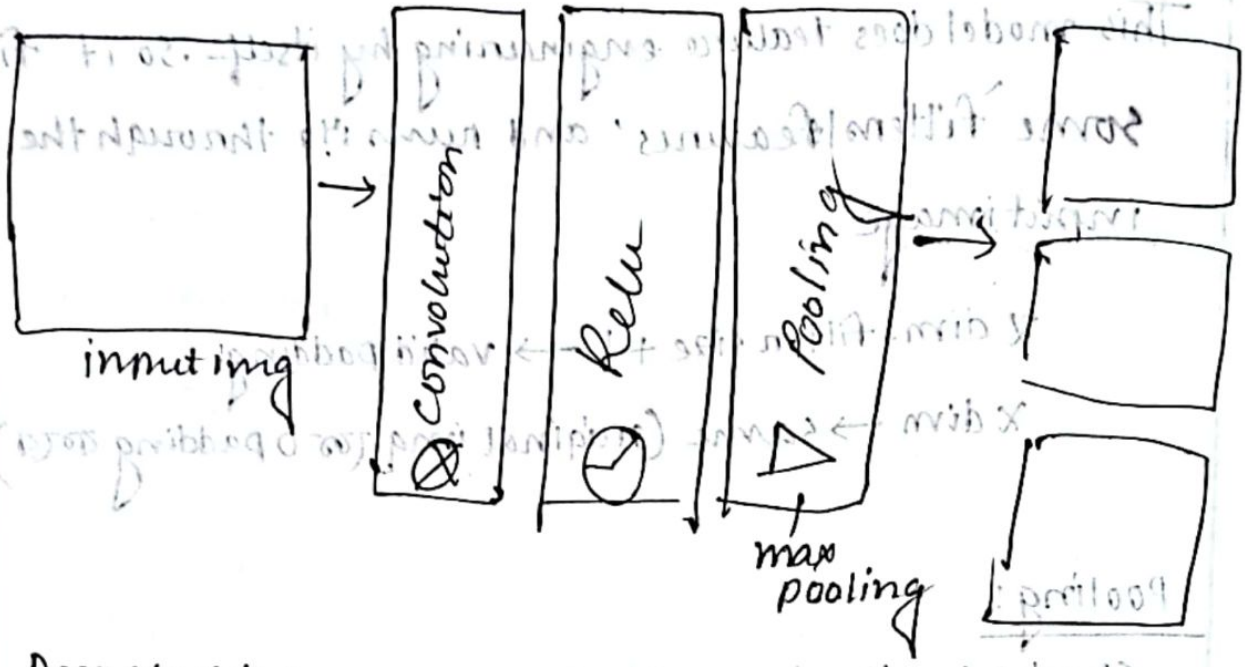
- ~~filt~~ specific stride pick করবে
  - ↳ ~~filter~~ কত ঘর যাবে window
- ~~filter~~ window তে filtered image এর উপর go through করে
- taking the max value from each window

Normalize

Negative value 0 করে (ReLU)

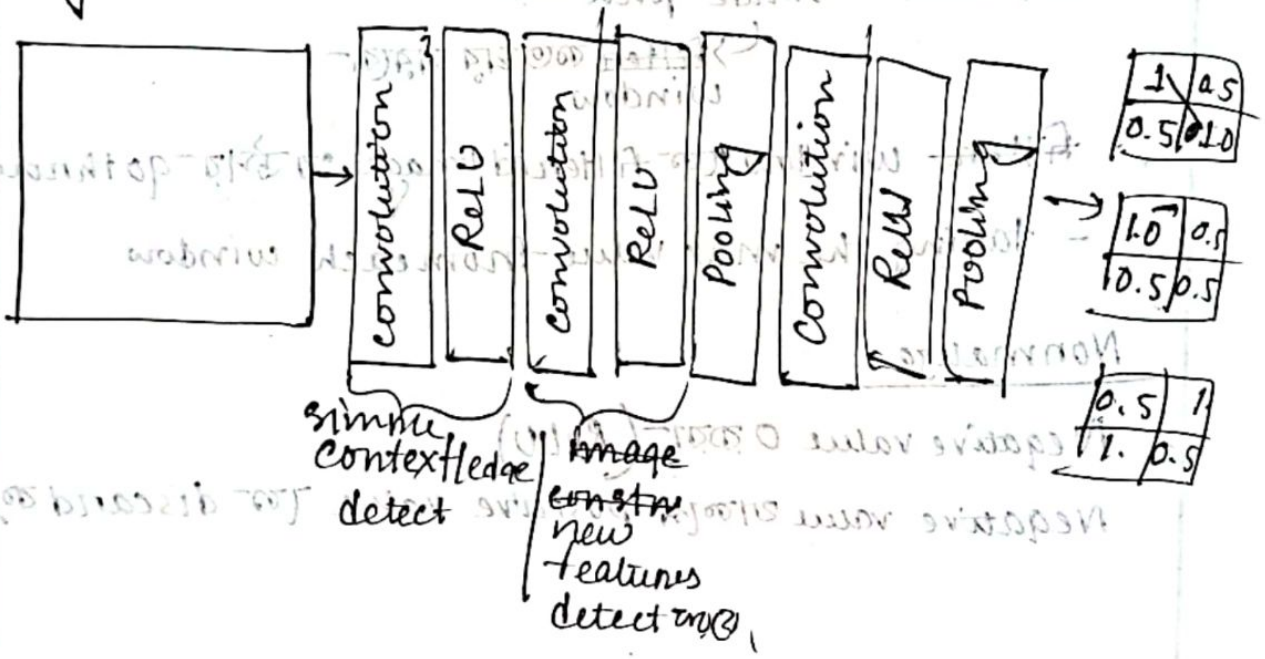
Negative value থাকলে positive value তে discard করা যায়।

Layers get stacked

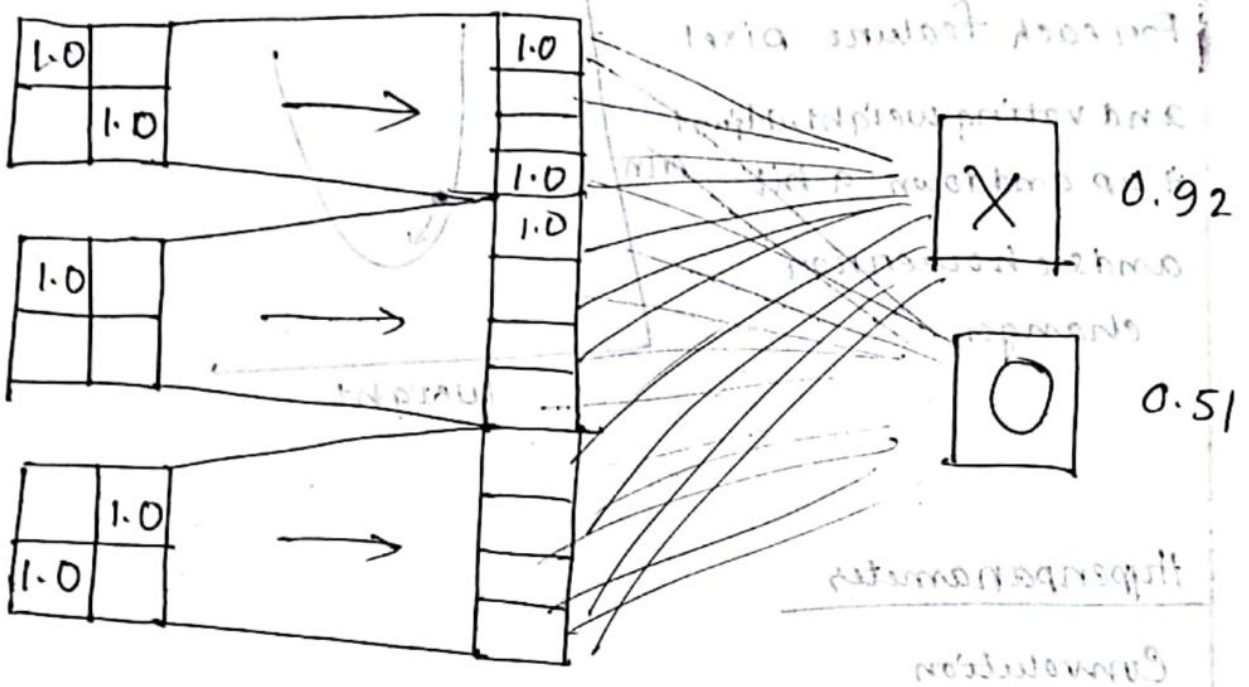


Deep stacking

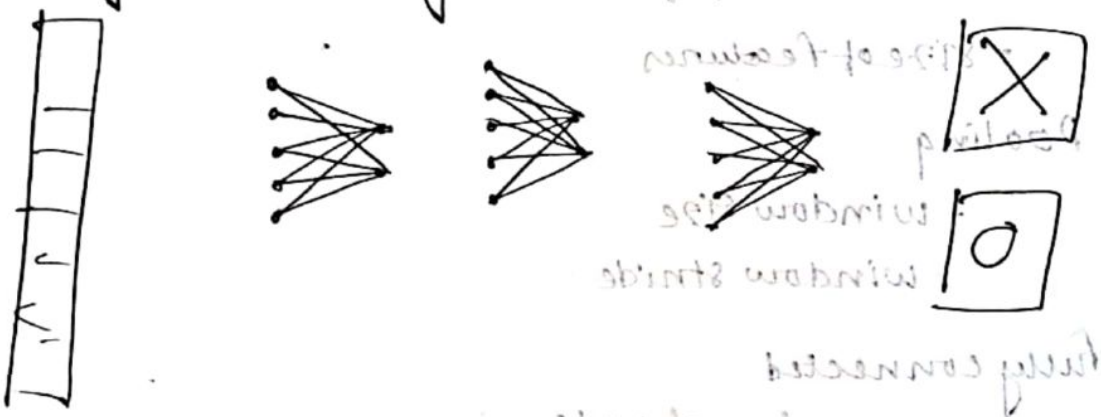
Layers can be repeated several times



Fully connected layer



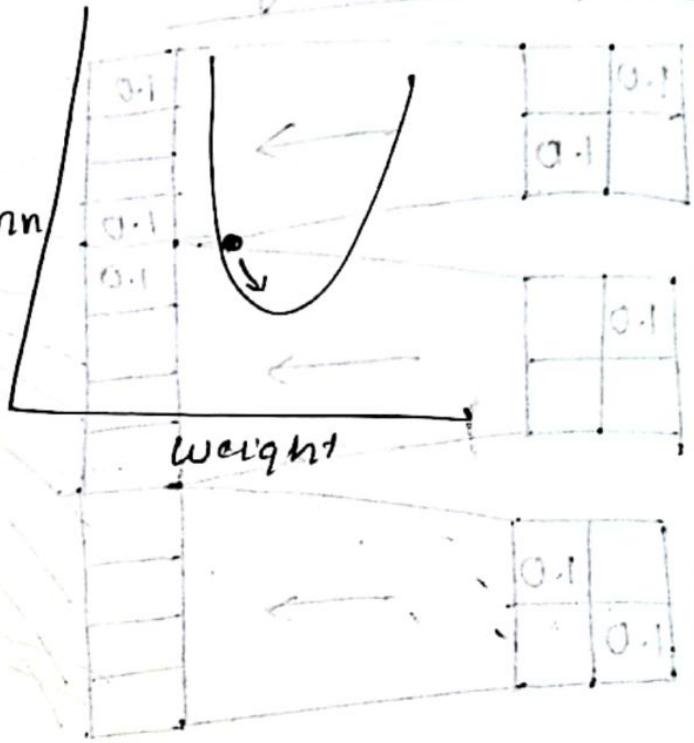
Fully connected layers can be stacked too -



Backpropagation = filter 의 weight learning -

Gradient descent

For each feature pixel and voting weight, adjust it up and down a bit and see how error changes.



Hyperparameter

Convolution
- number of features
- size of features

Pooling
- window size
- window stride

fully connected
- number of node

Limitation

- only captures local 'spatial' pattern in data. If the data can not be made to look like an image, ~~cnn~~ convnet is less useful.

Rule of thumb

If shuffling columns does not affect data, then convnet can't be used

Slide: Parameter calculation of NN

Parameter calculation in NN

```
myinput = Input(shape=(10,10,1))      # image height, image width, chanel
myoutput = Convolution2D(1, 3)(myinput)   # 1 -> no of convfilter, 3 -> 3x3 filter
                                          # height/width
```

output will become $10 - 3 + 1 = 8$ (8X8)

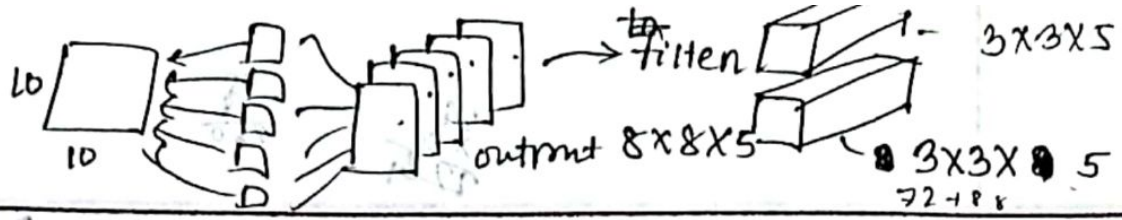
```
myoutput = Activation('sigmoid')(myoutput)
model.summary
```

Layer	Output shape	Param
input_17 (InputLayer)	[(none, 10, 10, 1)]	0
conv2d_19 (Conv2D)	(none, 8, 8, 1)	10 (3X3 filter = 9 values + 1 bias)
activation(	(none, 8, 8, 1)	0)

total param: 10

trainable param = 10

non trainable = 0



my input = Input (Shape = (10, 10, 1))

my output = Convolution2D (5, 3) (my-input) → 5 3X3 filter

my output = Activation ('sigmoid') (my output)

my output = Convolution 2D (2, 3) (my outpu

my output = Ac . . .

3X3X5 +1 = 46

layer	output shape	param
input_19	[(none, 10, 10, 1)]	0
conv2d_22	[(none, 8, 8, 5)]	50
activation 22	(none, 8, 8, 5)	0
conv 2d_23	(none, 6, 6, 2)	46 X 2 = 92
activation	(none, 6, 6, 2)	0

8(3X3X5+1)

Layer	Output	Param
Input_20	[(none, 100, 100, 3)]	0
conv2D_24	(none, 98, 98, 8) $8 \times [3 \times 3 \times 3 + 1]$	224

// no of filter = 8
each filter dim = $3 \times 3 \times 3 = 27$
each filter param = $(3 \times 3 \times 3) + 1 = 28$
total filter param = 28×8

activation	(none, 98, 98, 8)	0
conv2d_25	(None, 96, 96, 1) $3 \times 3 \times 8 + 1$	73
activation		0

1D convolution

layers	Output shape	Param
input_25	[(none, 100, 5)]	0
conv1d	(none, 98, 8) filter (3 X 5) no of filter 8 param is 1 filter: $(3 \times 5) + 1 = 16$ total param = $8 \times 16 = 128$	128
activation		0
conv1d_5	(none, 96, 1) $((3 \times 8) + 1)$	25

Number of parameters in conv layer would be :

$$((m \times n \times d) + 1) \times k)$$

((shape of the width of the filter x shape of height of filter x number of filter channel in prev. layer) + 1) x number of filter.

Output shape

$$\text{floor}\left(\frac{(w - F + 2P)}{s} + 1\right)$$

26
16
156
260
416

Example

28
28
224
560

(3X3)+1)

(4X4)
5X5
26X16

layer	output	param
Conv2d	[-1, 16, 28, 28]	16x((5X5)+1) = 416
dropout	[-1, 16, 28, 28]	0
ReLU	[-1, 16, 28, 28]	0
Maxpool	[-1, 16, 14, 14]	0
Conv2D	[-1, 32, 14, 14]	32((5X5X16)+1) = 12832
Dropout-6	[-1, 32, 14, 14]	0
Relu	[-1, 32, 14, 14]	0
Maxpool	[-1, 32, 7, 7]	0
flatten-9	[-1, 1568]	0
Linear-10	[-1, 512]	(1568+1)X512 = 803,328
Dropout-11	[-1, 512]	0
ReLU	[-1, 512]	0
Linear-13	[-1, 128]	(513X128) = 65664
Dropout-14	[-1, 128]	0
ReLU-15	[-1, 128]	0
Linear-16	[-1, 10]	1296

28
-14
14

16((15X15)+1)

$$\text{Output shape} = \left(\frac{W/H - f + 2P}{S} + 1\right)$$

Example:

Input shape = 1024

Layers	Output	Param
1024	1024	0
FcWB(256)	256	(1024 X 256) + 256 = 262400
dropout	256	0
Fc(128)	128	(256 X 128) = 32768
tahh	128	0
maxpool(3)	floor((128 − 3 + 2 X 0)/3 + 1) = 42 (height/width, filtersize, stride, padding)	0
Fc (3)	3	42 X 3 = 126
output	3	0
		295294

Slide: Word Embedding (Shaown Sir)

Natural Language Processing and word embedding

Bag of words

We create a vocabulary with the words that are present in our dataset.

Review 1: This movie is very scary and long
Review 2: This movie is not scary and is slow.
Review 3: This movie is spooky and good

	This	movie	is	very	scary	and	long	not	slow	spooky	good
Review 1	1	1	1	1	1	1	1	0	0	0	0
Review 2	1	1	1	0	0	1	0	1	1	0	0
Review 3	1	1	1	0	0	1	0	0	0	1	1

~~word ના presence ને one hot encoding~~

presence calculate કરે છે.

context નો

vocabulary નો size ઘણો મોટો.

BOW example

D-1: It was the best of times

D-2: It was the worst of times

D-3: It is the time of stupidity

D-4: It is the age of foolishness

	It	was	the	best	of	times	worst	stupidity	is	age	foolishness
D-1	1	1	1	1	1	1	0	0	0	0	0
D-2	1	1	1	0	1	1	1	0	0	0	0
D-3	1	0	1	0	1	1	0	1	1	0	0
D-4	1	0	1	0	1	0	0	0	1	1	1

vocabulary size অনেক লম্বা; but each document has only few of them.

The result is a vector with lots of zeros, called sparse representation

Solution#1 for BOW

Stemming Vs Lemmatisation

Stemming	Lemmatisation
→ removes last few characters from a word often leading to incorrect meaning and spelling	→ considers the context and converts the word to meaningful base form
Caring ⇒ can	caring ⇒ care
→ used in large dataset where performance is issue	→ computationally expensive as it involves lookups tables

Solution 2

using n gram

for example using trugram

Term frequency (TF)

$$TF_{t,d} = \frac{n_{t,d}}{\text{Total number of terms in document } d}$$

$n_{t,d}$ → doc এ কতবার word আসছে

D-1 : It was the best of the time

D-2 : It was the worst of times

D-3 : It is the time of stupidity

It was the best of time worst is stupidity.

D-1	1	1	2	1	1	1	0	0	0	(7)
D-2	1	1	1	0	1	1	0	0	0	(6)
D-3	1	0	1	0	1	1	0	1	1	(6)

Calculating TF

It was the best of time worst is Stupidity

D-1	1/7	1/7	2/7	1/7	1/7	1/7	0	0	0	
D-2	1/6	1/6	1/6	0	1/6	1/6	1/6	0	0	
D-3	1/6	0	1/6	0	1/6	1/6	0	1/6	1/6	

Inverse document frequency

$$IDF_t = \log\left(\frac{\text{total number of doc}}{\text{The number of doc with term } t}\right)$$

rare words are more useful and domain specific

Calculating IDF

It = ~~0~~ log(3/3) = 0

was = log(3/2) = 0.18

the = log(3/3) = 0

best = log(3/1) = 0.48

of = log(3/3) = 0

time = log(3/3) = 0

worst log(3/1) = 0.48

is = log(3/1) = 0.48

stupidity = log(3/1) = 0.48

TF-IDF

$$W_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF for each Document

	Doc-1	Doc-2	Doc-3
It	0	0	0
was	0.026	0.03	0
the	0	0	0
best	0.068	0	0
of	0	0	0
time	0	0	0
worst	0	~~0~~ 0.08	0
stupidity	0	0	0.08
is	0	0	~~0.08~~ ~~0.48~~ 0.08

Word embedding

Word representation:

vocabulary तयार करा containing all words

each word one hot encoding वापरून represent करा.

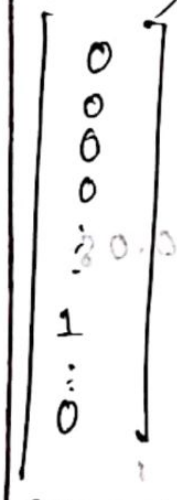
vector length = vocabulary size

1 hot representation.

Size of vocabulary [V] = 10,000

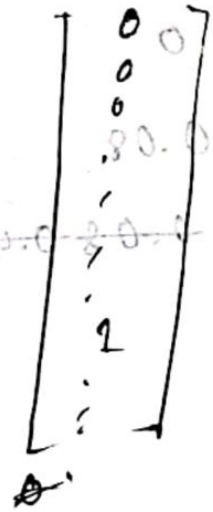
V = [a, aaron ... zulu, <UNK>]

Man (5391)



O(5391)

Woman 9853



meaning नाही.

I want a glan of orange

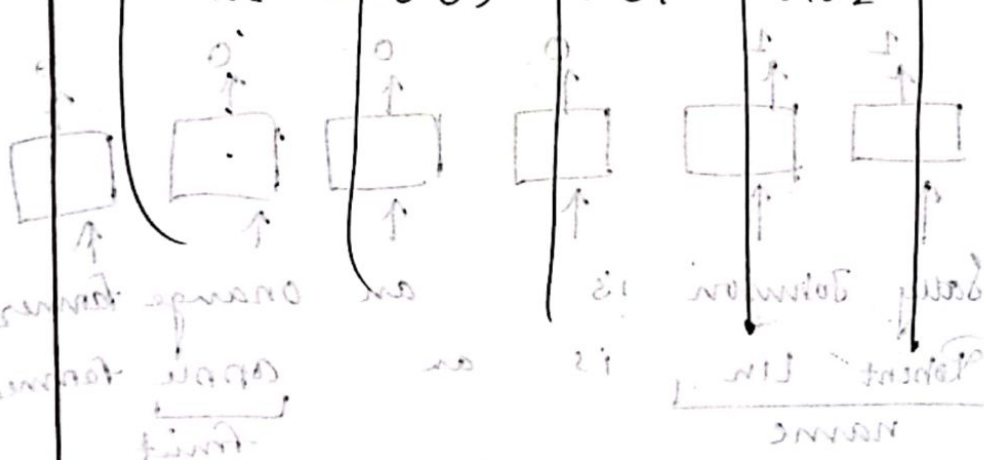
I want a glan of apple

one hot encoding apple and onange relate করতে পারবে না,
কারণ তাদের inner product 0.

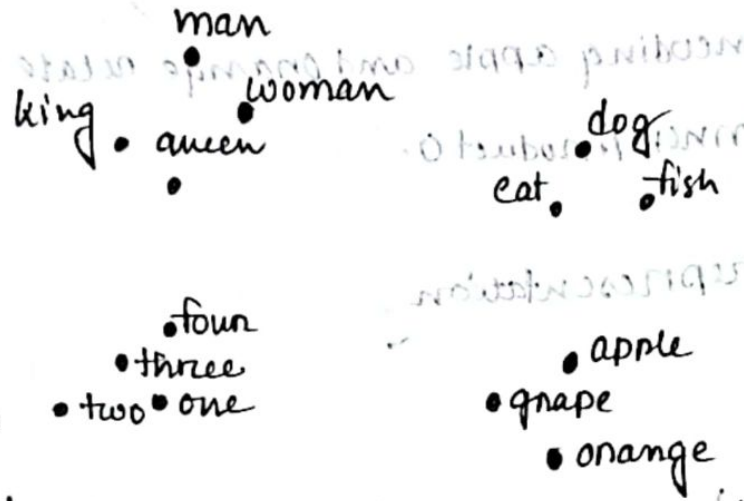
featurized representation

300

different features	Man (5391)	Woman (9853)	King (4914)	
Gender	-1	1	-0.95	
Royal	0.01	0.02	0.93	
Age	0.03	0.02	0.70	
food	0.09	0.01	0.02	



Visualizing word embedding

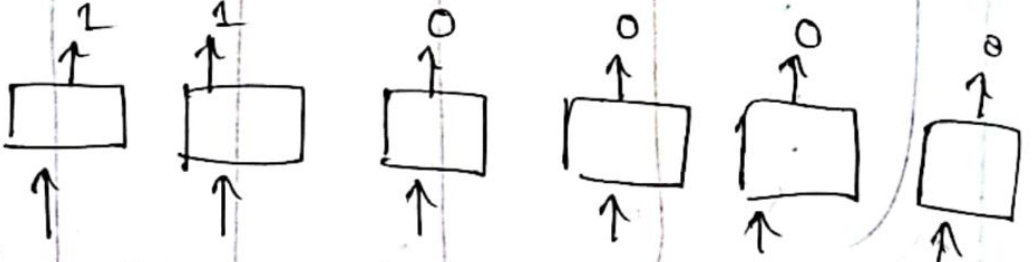


t-SNE algorithm to visualize word embedding in

(300D) → 2D

→ similar words are together

Named entity recognition problem



labeled Sally Johnson is an orange farmer

unlabeled Robert Lin is an apple farmer

name

fruit

Slide: Natural language processing and word embedding

Transfer learning and word embeddings

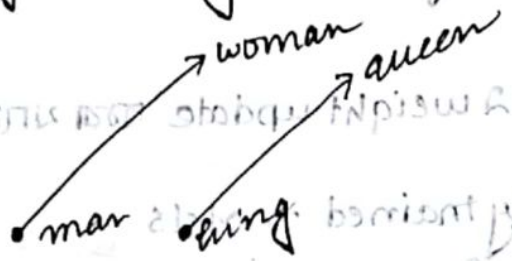
- transfer learning এ weight update করা যাবে না।—
  - ⇒ will re-use already trained models
  - ⇒ আমাদের task অনুযায়ী smaller labeled training dataset এ trained embedding apply করবো।—
  - ⇒ continue to fine tune যদি আমাদের training dataset অনেক বড় হয়।

Advantage

→ 10,000 one hot sparse >> 300 features vector (dense)

⇒ helpful when we have smaller dataset

Analogies using word vectors



$e(man) - e(woman) \approx e(king) - e(w)$

$\text{argmax}_w \; sim(e_w, e_{king} - e_{man} + e_{woman})$

Similarity function

Euclidean distance.

$$\sqrt{\sum_{i=1}^{k} (u_i - v_i)^2}$$

cosine similarity

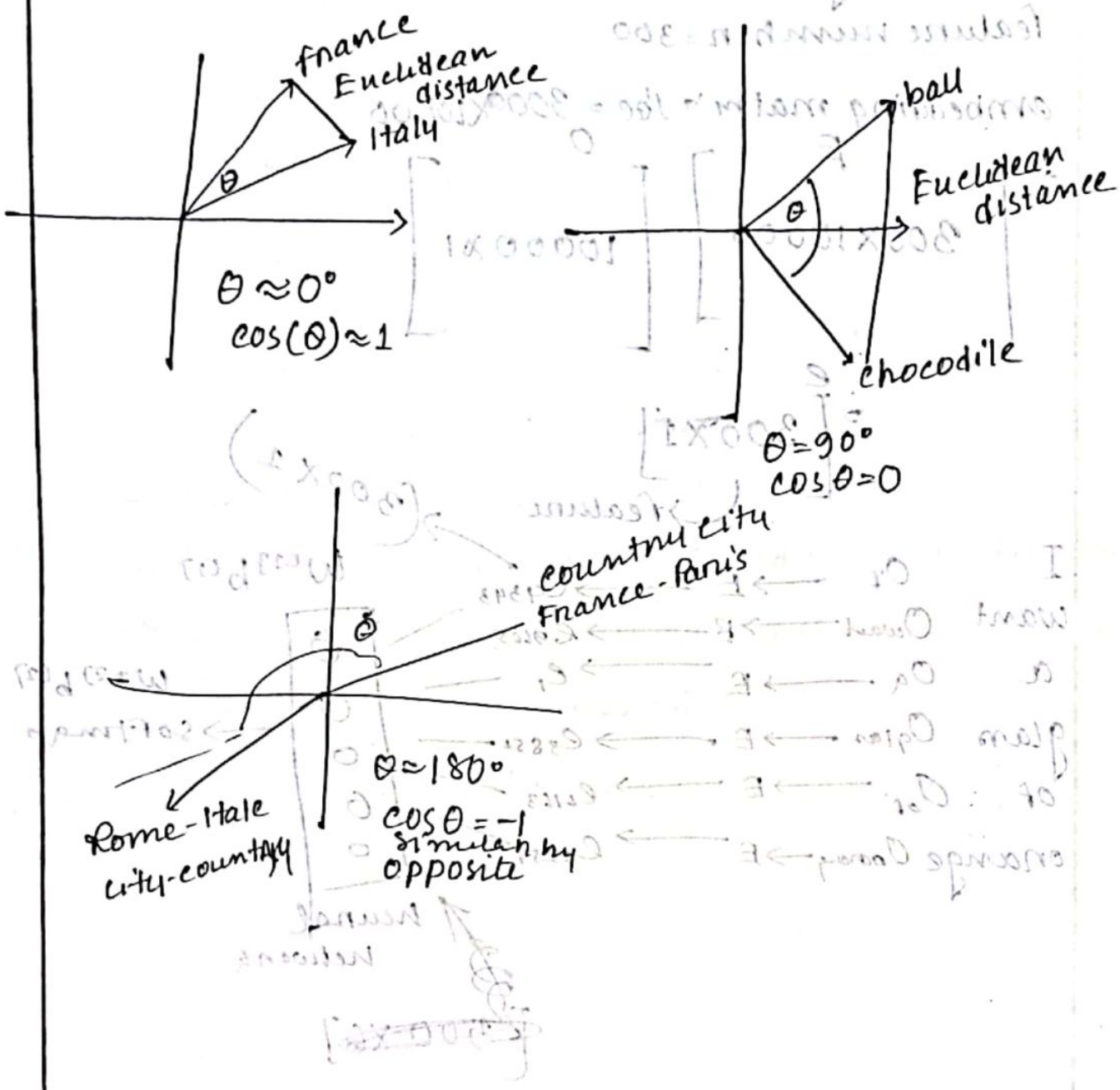
$$\text{cosinesimilarity}(u,v) = \frac{u \cdot v}{\|u\|_2 \|v\|_2} = \cos\theta$$

$\|u\|_2 \|v\|_2$ → we can distand

$[-1, 1]$

cosine similarity $= \frac{u \cdot v}{\|u\|_2 \|v\|_2} = \cos\theta$

$u \cdot v$ dot product



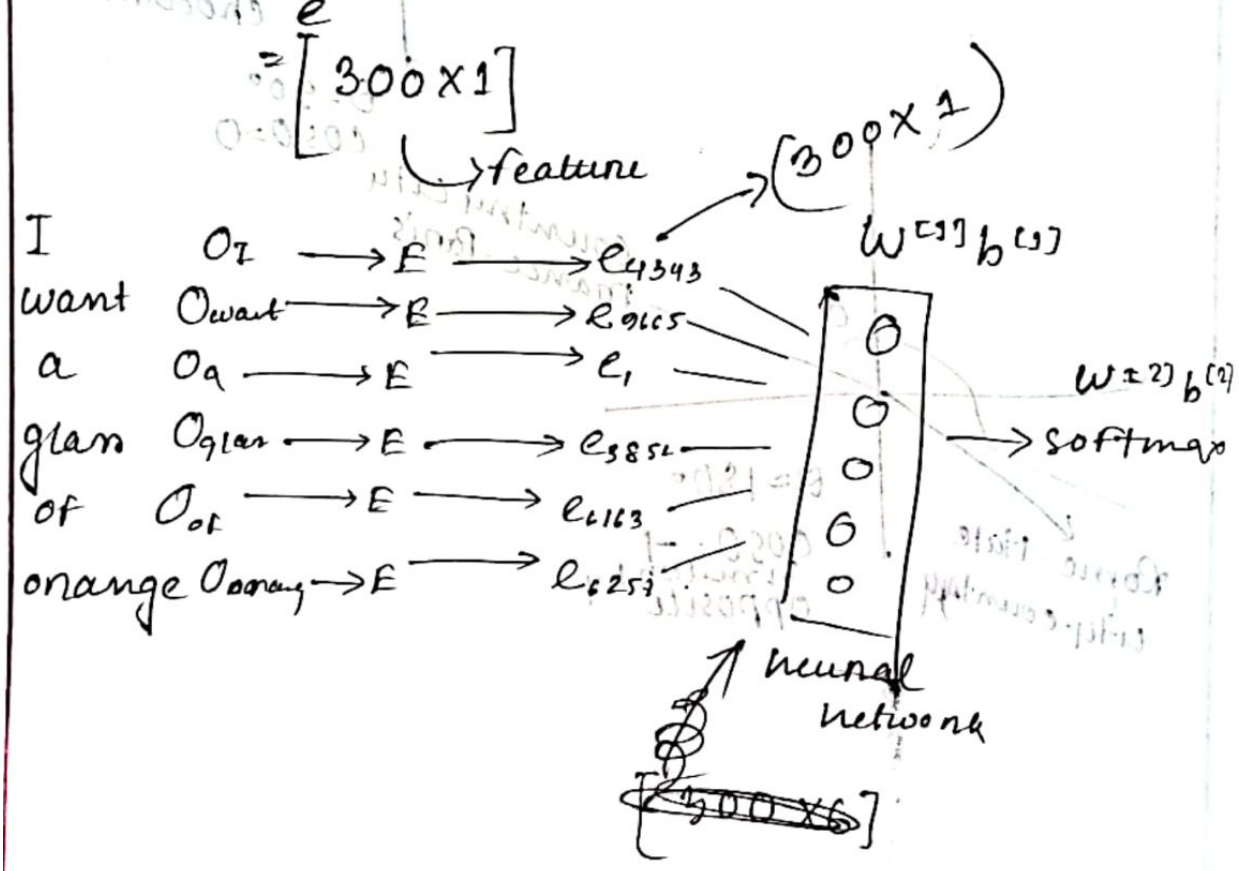
Embedding matrix

Vocabulary size = 10000

feature number = 300

embedding matrix size = 300 × 10000

$$E_{300\times10000} \times O_{10000\times1} = e_{300\times1}$$



Word2Vec : Skip gram

I want a glass of orange juice to go along with my cereal

Context	Target	How far
Orange	juice	+1
Orange	glass	-2
Orange	my	+6

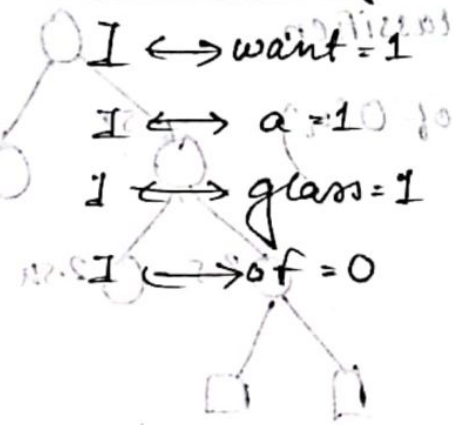
→ Randomly pick context word

→ target is chosen randomly based on window of specific size.

→ We have converted the problem into a supervised problem to predict the target word given the context word.

window size = 4

I want a glass of orange juice



Vocabulary size = 10,000

context word c ("orange") target word t ("juice")

X ⟶ Y

context c [6257] ("orange") ⟶ target t [4834] ("juice")

$O_c \longrightarrow E \longrightarrow e_c \longrightarrow O \longrightarrow \hat{y}$ (Softmax)

$$\text{Softmax } p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}}$$

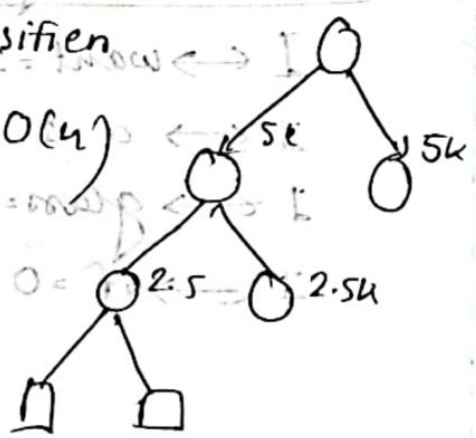
↳ very complex

$$\text{Loss } \mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{10,000} y_i \log \hat{y}_i$$

Solution:

Hienanchical softmax classifier

O(log(n)) instead of O(n)



Negative sampling

Random sampling এর সমস্যা => context word এর sample এ অপ্রয়োজনীয় word (such as is, the) আসতে পারে, আবার frequent words প্রয়োজনীয় word ও বাদ পড়তে পারে,-

=> sample তৈরির সময় আমরা less frequent word এর বেশি weight দিব-

এর জন্য skipgram model

=> skipgram এ problem ছিলো- softmax এ high computation power দরকার ছিলো-

Negative sampling এ concept হলো- for a context word আমরা k negative sample এ 1 টা positive sample নিব,-

$$P(y=1 \mid e, t) = \sigma(\theta_t^T e_c)$$

context	word	target
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0

e t y

k

How do we solve the softmax complexity?

আমরা task টা binary logistic regression এ নামিয়ে approach করবো, so, 10,000 way softmax use না করে আমরা প্রতি step এ k+1 classifier use করবো.

How to choose negative sample

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10000} f(w_j)^{3/4}}$$

frequency of a particular word in a corpus

Practice

D1- A woman finds a pot of treasure on the road while she is returning from work

D2- Delighted with her luck, she decides to keep it..

D3- However, enthusiam refused to fade away

Idf('luck) $= \log \frac{3}{1} = 0.48$

Idf(enthusiasm) $= \log \frac{3}{1} = 0.48$

Idf(to) $= \log \frac{3}{2} = 0.176$

tf(luck) $= \frac{1}{18}$

tf(enthusiam) $= \frac{1}{7}$

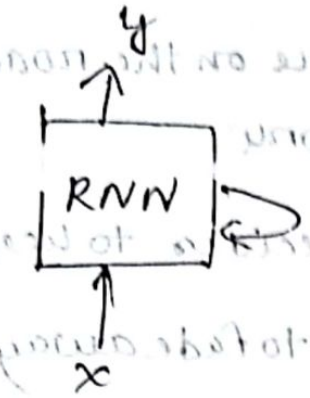
tf(to) $= \frac{1}{7}$

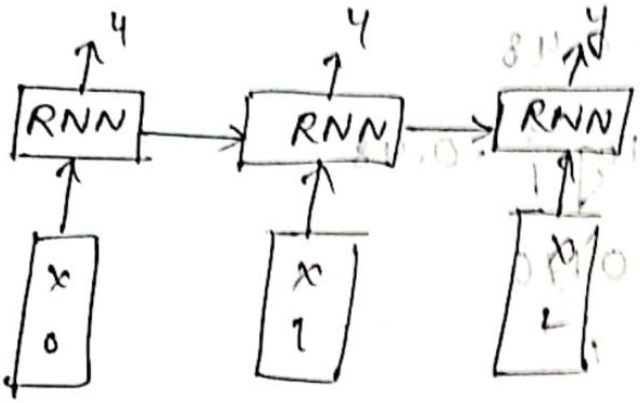
TF-IDF$_{(luck)} = \frac{1}{18} \times 0.48 = 0.027$

TF IDF$_{(enthu)} = \frac{1}{7} \times 0.48 = 0.08$

TF IDF$_{(to)} = \frac{1}{7} \times 0.176 = 0.025$

Recurrent Neural Networks





$$h_t = f_w(h_{t-1}, x_t)$$

h_t — new (hidden) state

f_w — function

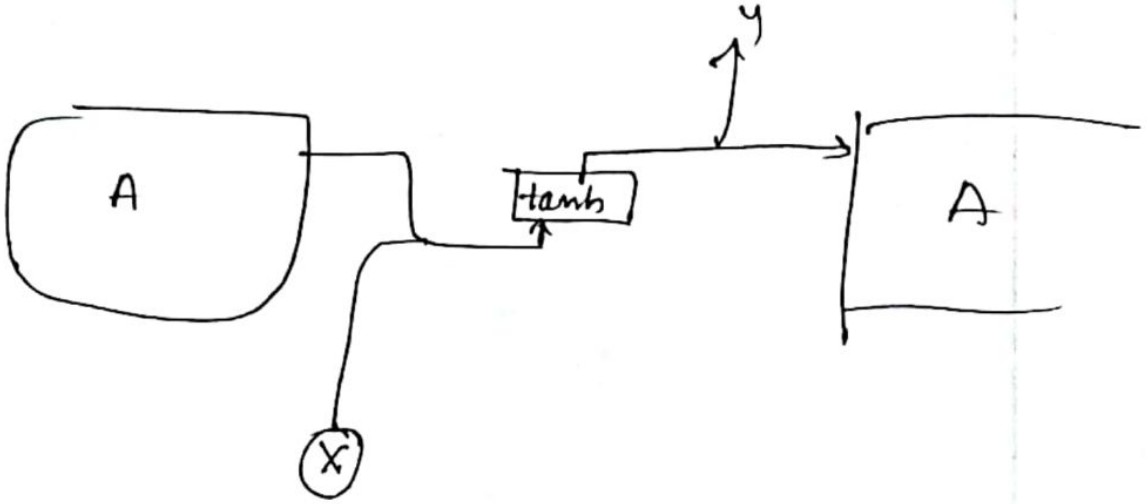
h_{t-1} — old state

x_t — input vector at time step t

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

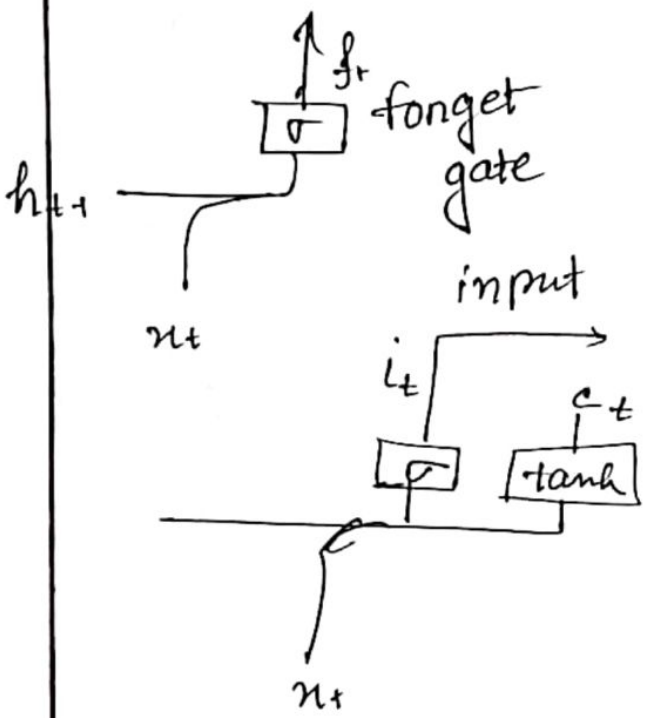
$$y_t = W_{hy} h_t$$

$$\frac{12-3}{2}+1$$



LSTM

C_{t-1} ——(X)—— cell state



$$f_t = \sigma\left(W_f [h_{t-1}, x_t] + b_f\right)$$

$$i_t = \sigma\left(W_i [h_{t-1}, x_t] + b\right)$$

$$c_t = \tanh\left(W_c [h_{t-1}, x_t] + B_c\right)$$

$$C_t = f_t * C_{t-1} + i_t \times \tilde{c}_t$$

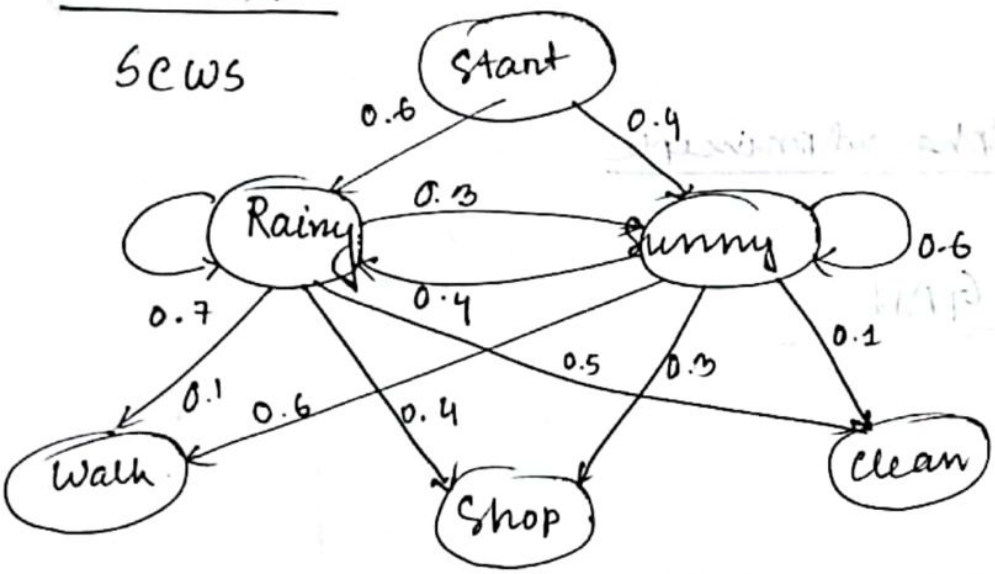
$$O_t = \sigma\left(W_o [h_{t-1}, x_t] + b_o\right)$$

$$h_t = O_t \times \tanh(C_t)$$

HMM

Softcomp

SCWS



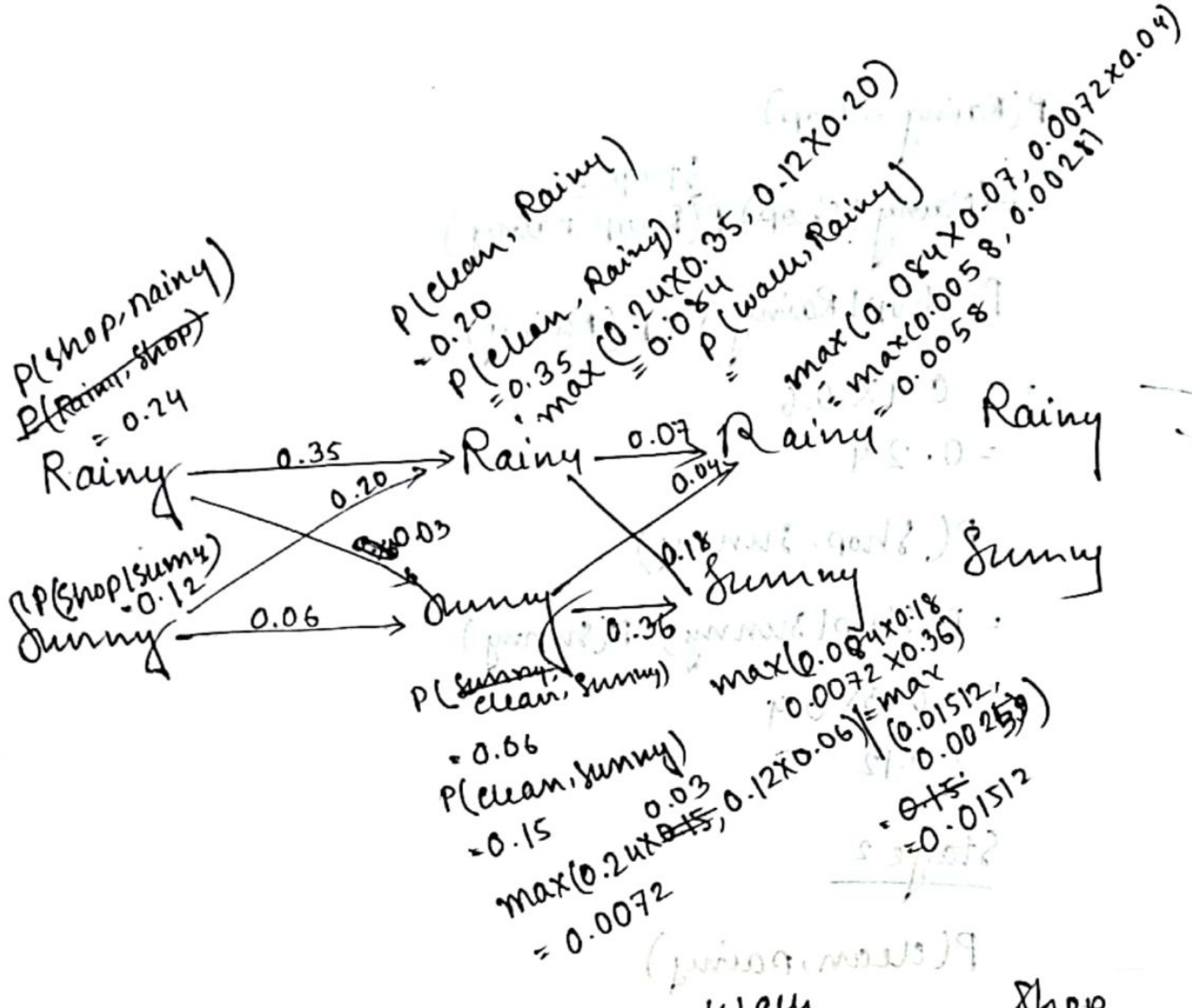
Initial probability [0.6 0.4]

Transition matrix

	Rainy	Sunny
Rainy	0.7	0.3
Sunny	0.4	0.6

Emission matrix

	walk	shop	clean
Rainy	0.1	0.4	0.5
Sunny	0.6	0.3	0.1


P(shop, rainy)
P(Rainy, shop)
= 0.24
Rainy
Sunny
P(shop|sunny)
= 0.12
0.35
0.20
0.03
0.06
Rainy
Sunny
P(clean, Rainy)
= 0.20
P(clean, Rainy)
= 0.35
max(0.2x0.35, 0.12x0.20)
= 0.084
P(clean, sunny)
= 0.06
P(clean, sunny)
= 0.15
max(0.24x0.15, 0.12x0.06)
= 0.0072
0.07
0.04
0.18
0.36
Rainy
Sunny
P(wall, Rainy)
= max(0.084x0.07, 0.0072x0.04)
= max(0.0058, 0.00028)
= 0.0058
max(0.084x0.18
0.0072 x0.36)
= max
(0.01512,
0.00259)
= 0.01512
Rainy
Sunny
Shop
Clean
Wall
Shop

~~P(Raing, shop)~~

~~P(Raing | shop)~~ P(shop, raing)

Stage 1:

P(shop | Raing) x P(Raing)

= 0.4 X 0.6

= 0.24

P(shop, sunny)

= P(shop | sunny) x P(sunny)

= 0.3 X 0.4

= 0.12

Stage 2

P(clean, raing)

P(clean | raing) x P(raing | raing)

= 0.5 X 0.7

= 0.35

P(clean, sunny)

= P(clean | sunny) x P(sunny | sunny)

= 0.1 X 0.6

= 0.06

P(clean, Rainy)

= P(clean | Rainy) P(Rainy | Sunny)

= 0.5 X 0.4

= 0.20

P(clean, sunny)

= P(clean | sunny) P(sunny | Rainy)

= ~~0.5~~ 0.1 X 0.3

= ~~0.15~~ 0.03

Stage 3

P(walk, Rainy)

= P(walk | Rainy) P(Rainy | Rainy)

= 0.1 X 0.7

= 0.07

P(walk, Sunny)

= P(walk | Sunny) P(Sunny | Sunny)

= 0.6 X 0.6

= 0.36

P(walk & Rainy)

= P(walk | Rainy) P(Rainy | Sunny)

= 0.1 X 0.4 = 0.04

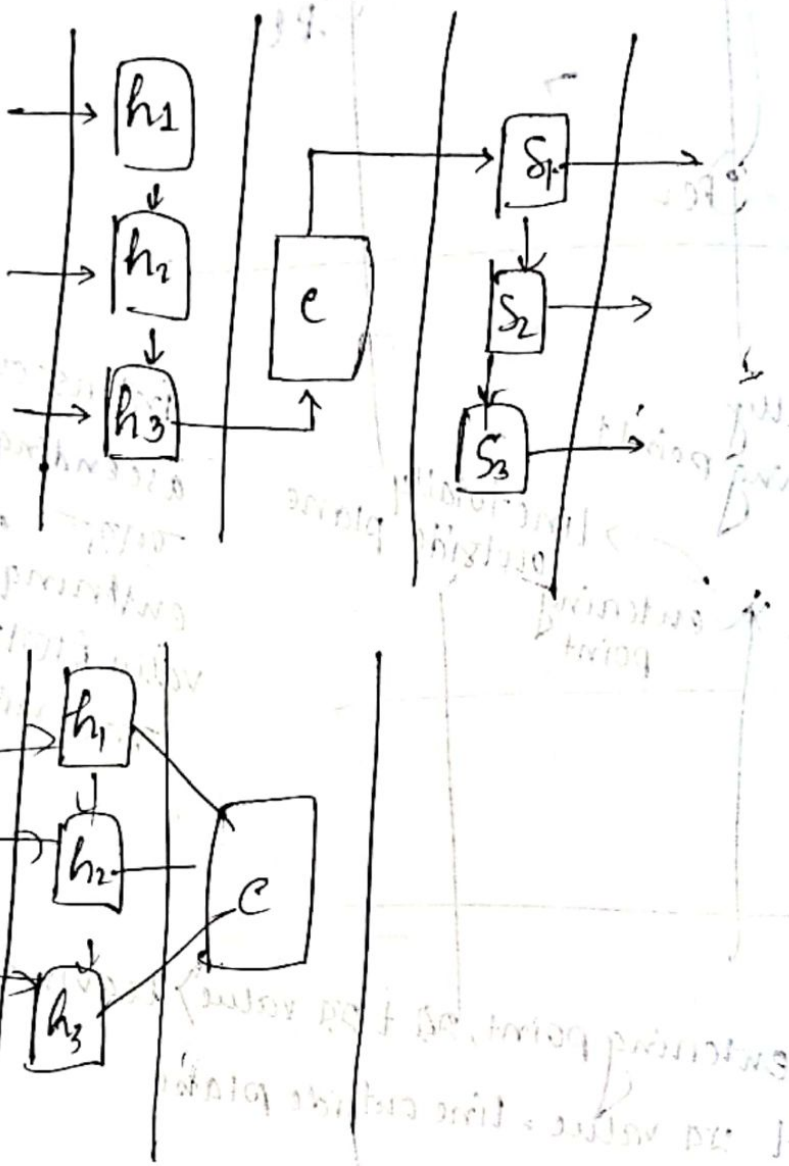
P(walk, Sunny)

= P(walk | Sunny) X P(Sunny | Rainy)

= 0.6 X 0.3 = 0.18

Soft Comp

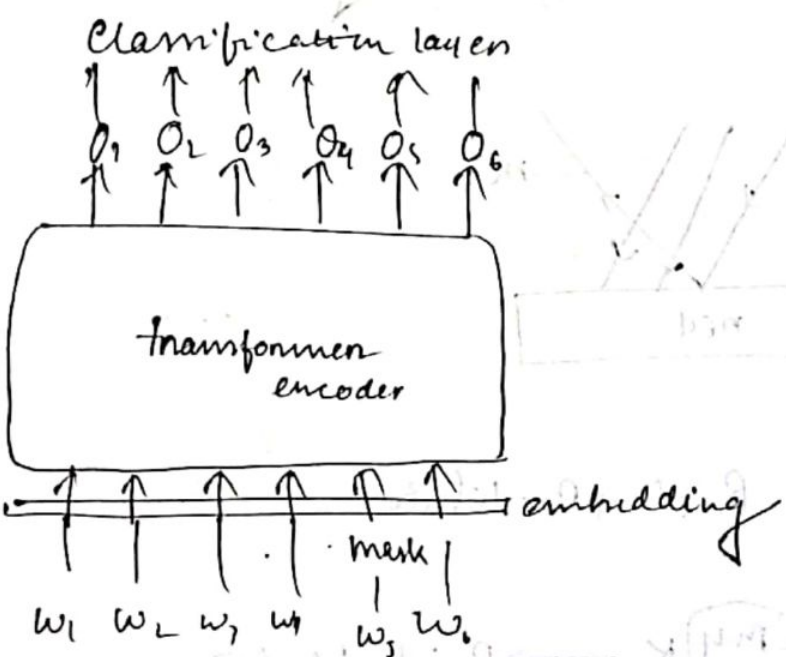
Attention mechanism



BERT

- Bidirectional — দুইদিকের context maintain হয়
- Encoder — transformer এর encoder part ব্যবহার
- Representation — encoder এর output
- transformer

vanilla BERT এ decoder ব্যবহার নাই



BERT - Mashed LM