CBCS SCHEME

USN 1 C R 2 2 C I 0 0 3

BCA701

# Seventh Semester B.E./B.Tech. Degree Examination, Dec.2025/Jan.2026

# Deep Learning

Time: 3 hrs. Max. Marks: 100

*Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.*
*2. M : Marks , L: Bloom's level , C: Course outcomes.*

| | | Module – 1 | M | L | C |
|---|---|---|---|---|---|
| Q.1 | a. | Define Neural Network. Explain briefly the benefits of neural networks. | 10 | L1 | CO1 |
| | b. | Explain the role of weights, bias and activation functions in a neuron model. | 10 | L2 | CO1 |
| | | **OR** | | | |
| Q.2 | a. | State and explain Perceptron Convergence Theorem. | 10 | L2 | CO1 |
| | b. | Discuss relation between the perceptron and bayes classifier for a gaussian environment. | 10 | L2 | CO1 |
| | | **Module – 2** | | | |
| Q.3 | a. | What is Multi Layer Perceptron (MLP) and explain how does it differ from a single layer perceptron. | 10 | L2 | CO2 |
| | b. | Explain Back propogation algorithm and its importance in training multilayer perceptrons. | 10 | L2 | CO2 |
| | | **OR** | | | |
| Q.4 | a. | Explain how a multilayer perceptron trained by Back-propogation can solve the XOR problem, where as a single layer perceptron cannot. | 10 | L2 | CO2 |
| | b. | Discuss Heuristics for making the back-propagation algorithm perform better. | 10 | L2 | CO2 |
| | | **Module – 3** | | | |
| Q.5 | a. | What is regularization in the context of deep learning? Explain L2 parameter regularization in detail. | 10 | L2 | CO3 |
| | b. | Discuss the following :<br>i) Dataset Augmentation<br>ii) Semi Supervised Learning. | 10 | L2 | CO3 |
| | | **OR** | | | |
| Q.6 | a. | Write a short note on :<br>i) Ill conditioning<br>ii) Plateaus, saddle points and other flat regions. | 10 | L2 | CO3 |
| | b. | Elaborate the challenges local minima, long term dependencies and Cliffs and exploding gradients involved in optimization for training deep model. | 10 | L2 | CO3 |

BCA701

| | | Module – 4 | | | |
|---|---|---|---|---|---|
| Q.7 | a. | Explain convolution neural network in detail. | 10 | L2 | CO4 |
| | b. | Discuss pooling in CNN. | 10 | L2 | CO4 |
| | | OR | | | |
| Q.8 | a. | Describe the variants of the basic convolution function in brief. | 10 | L2 | CO4 |
| | b. | With the context of convolutional networks, Explain structured output and data types. | 10 | L2 | CO4 |
| | | Module – 5 | | | |
| Q.9 | a. | Discuss Recurrent Neural Networks. | 10 | L6 | CO5 |
| | b. | Explain the idea of unfolding computational graph across a deep network structure. | 10 | L2 | CO5 |
| | | OR | | | |
| Q.10 | a. | Explain the following:<br>i) Bidirectional RNNs<br>ii) Encoder-Decoder sequence – to – sequence architectures. | 10 | L2 | CO5 |
| | b. | Explain the modes long short-term memory and network based on the gated recurrent unit. | 10 | L2 | CO5 |

* * * * *

# Deeplearning (BCA 701)

## Answerkey – January 2026

S.No

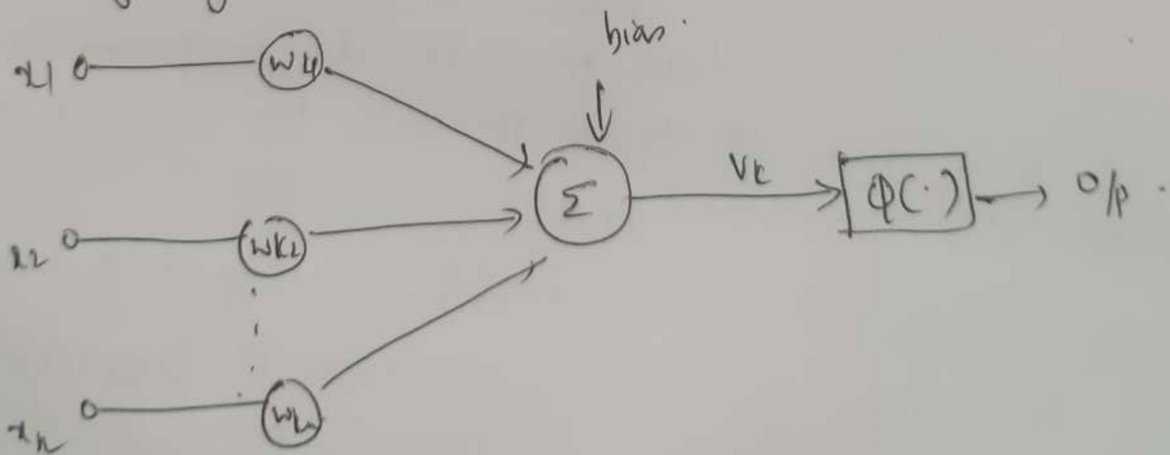
1a) Neural N/w → 5 Marks with diagram

→ is a massively parallel distributed processor made up of simple processing units that has a natural tendency for storing experiential knowledge and making it available for use.

Benefits — 5 Marks.

1. Non linearity
2. Input Output Mapping
3. Adaptivity
4. Evidential Response
5. Contextual Information
6. Fault Tolerance
7. VLSI Implementation.
8. Uniformity of Analysis & Design.
9. Neurobiology Analogy.

1.(b). Role of weights, bias and activation function.




Dig: 4M

Weight + bias + Activation fn – 2 Marks each.

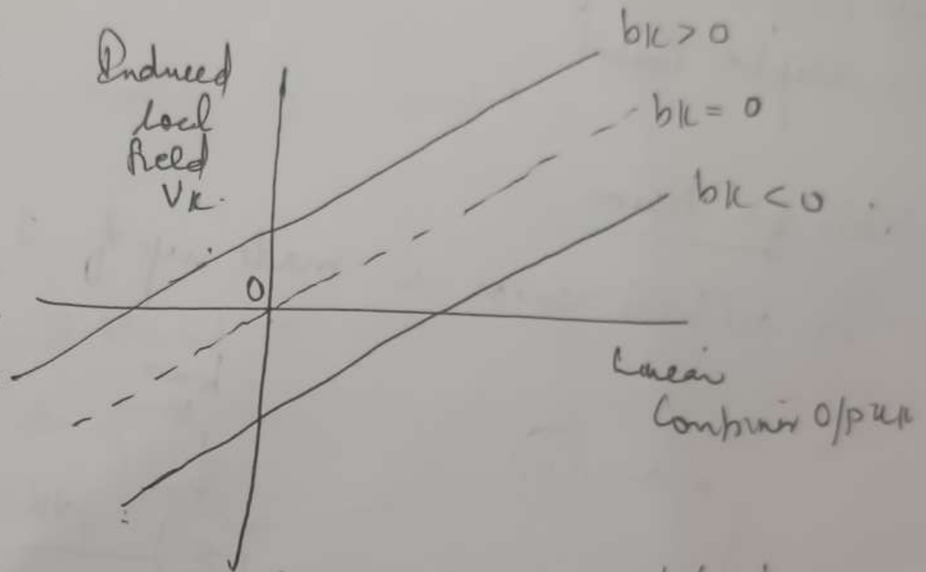
Shot on moto g54 5G
28 Jan 2026, 3:09 pm

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

$$y_k = \phi(\underbrace{u_k + b_k}_{v_k})$$

$\phi \rightarrow$ activation fn.

Affine Transformation
- linear transformation + a shift.
  $y = Wx + b$.
  ↑ move or shift it somewhere else.

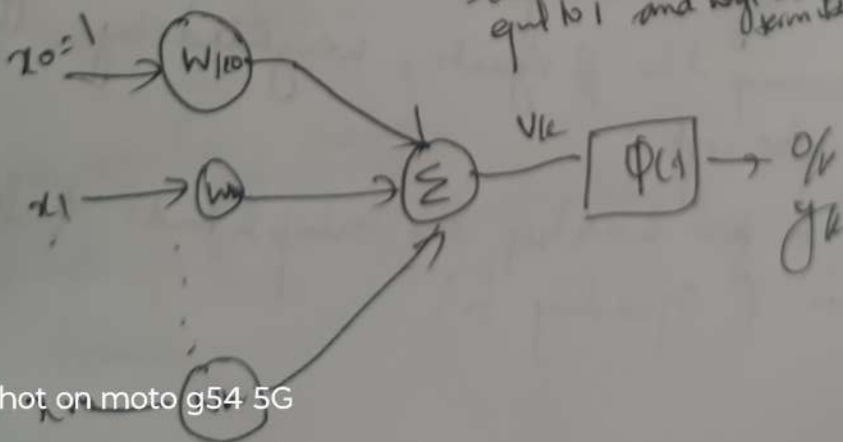



without bias, neurons always activate at the same pt (zero) so
Bias lets a neuron shifts its activation threshold so
it can learn patterns.

Bias can be modeled as an extra fixed i/p with value +1
($x_0 = 1$) and wgt equal to bias ($w_{k0} = b_k$).

(bias can be thought as a special i/p that is always equal to 1 and wgt = bias term itself)

Without bias, model learns that pass through (0,0). With bias, you can shift the o/p




$z = wx + b$

$z = w'x'$

$x' = [x_1, x_2 \ldots x_n, 1]$

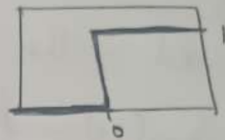
$w' = [w_1, w_2 \ldots, b]$

Shot on moto g54 5G
28 Jan 2026, 3:01 pm

## Types of Activation fn.

Threshold fn → also called Heaviside fn.

$$\phi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$



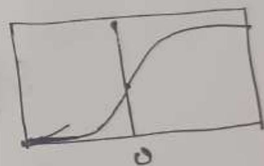

→ This model is the McCulloch and Pitts neural model.

→ all or none property $\quad v = \sum_j w_j x_j + b_k$

### Sigmoid fn

→ Commonly used fn. – Smooth curve b/w 0 and 1

→ increasing fn that balance b/w linear and nonlinear.




$$\phi(v) = \frac{1}{1 + e^{-av}}$$

a → slope parameter. $\quad$ a → scalar. $\quad$ v → i/p. $\quad$ e → 2.718 o/p. $\quad \sigma(av)$ 0 & 1

### Signum fn. → which is an odd fn → extracts sign of a real fn.

$$\phi(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases}$$

### Hyperbolic Tangent fn.

allows for an odd sigmoid type fn.

$\phi(v) = \tanh(v)$. range −1 to 1 $\quad \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$

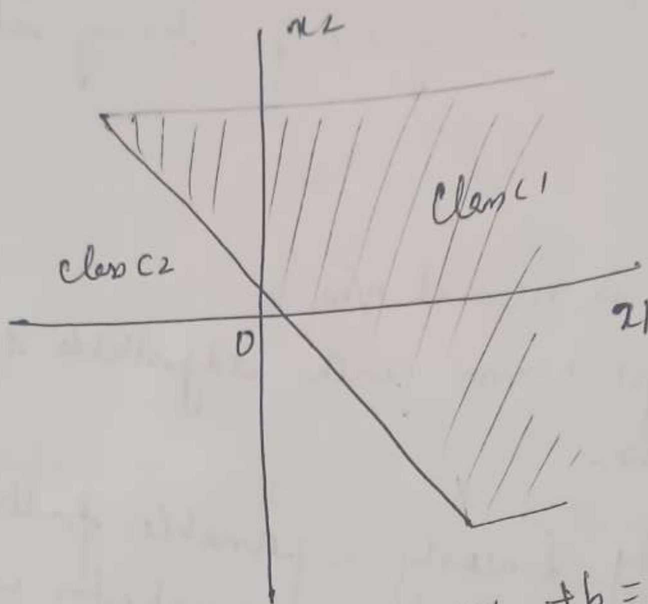
### ReLU fn – used in hidden layers

rectified linear unit

$\text{RELU}(x) = \max(0, x)$ $\quad x > 0 \;\; x$ $\quad x \leq 0 \;\; 0$


Shot on moto g54 5G
28 Jan 2026, 3:01 pm

decision regions separated by ...




$w_1x_1 + w_2x_2 + b = 0$

$(x_1, x_2)$ lies above the boundary line $\rightarrow$ C1

$(x_1, x_2)$ " below " $\rightarrow$ C2

## The Perceptron Convergence Theorem

→ If the data is linearly separable, then the Perceptron algorithm is guaranteed to find a solution in a finite number of steps.

Initialize weights and bias

$w = 0 \quad b = 0$

Repeat for each training example

Compute activation

$$a = w^T x_i + b$$

Predict label :

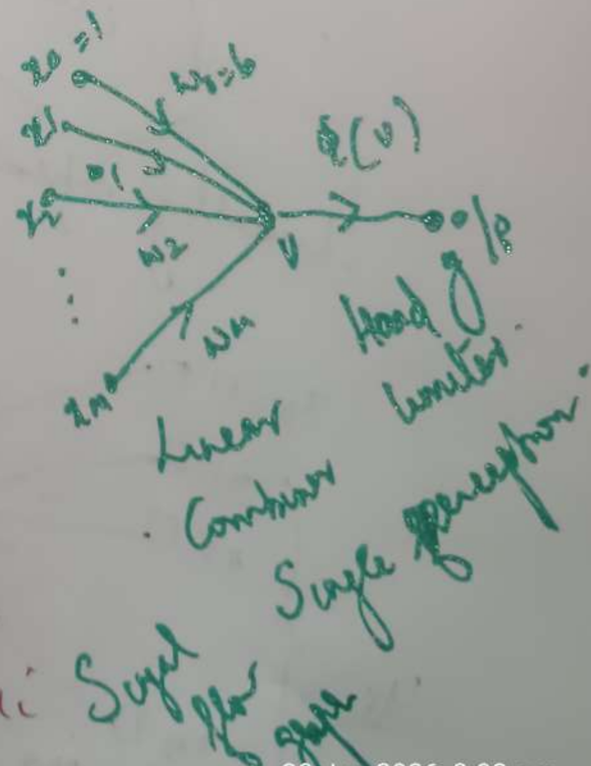
$$\hat{y}_i = \text{sign}(a)$$

if prediction is wrong update

$$w \leftarrow w + \eta \cdot y_i \cdot x_i$$

$$b \leftarrow b + \eta \cdot y_i$$

Repeat until all training examples are classified correctly




Shot on moto g54 5G
28 Jan 2026, 3:02 pm

<u>Perception Convergence Algorithm</u>

$$x(n) = [+1, x_1(n), x_2(n) \ldots x_m(n)]^T$$

$$w(n) = [b, w_1(n), w_2(n) \ldots w_m(n)]^T$$

$b$ = bias

$y(n)$ = actual response

$d(n)$ = desired response

$\eta$ = learning rate

(1) Initialization set $w(0) = 0$. Then perform the foll computations for time step $n = 1, 2 \ldots$

(2) Activation: At time step $n$, activate the perceptron by applying continuous valued input vector $x(n)$ and desired response $d(n)$

(3) Computation of Actual Response: Compute the actual response of the perceptron as

$$y(n) = \text{sgn}[w^T(n)\, x(n)]$$

where sgn(·) is the signum fn.

(4) Adaptation of Weight Vector

$$w(n+1) = w(n) + \eta[d(n) - y(n)]\, x(n)$$

$$d(n) = \begin{cases} +1 & \text{if } x(n) \text{ belongs to class } C_1 \\ -1 & \text{if } x(n) \text{ belongs to class } C_2 \end{cases}$$

(5) Increment time step $n$ by one and go back to step 2.

Shot on moto g54 5G
28 Jan 2026, 3:02 pm

## Relation b/w the Perceptron and Bayes Classifier for a Gaussian Environment

### Bayes Classification

→ is a probabilistic method used to decide which class an input belongs to. It is based on Bayes theorem which calculates the prob. of a class given some observed data.

email → spam/non spam words – higher probability.

### Gaussian Environment

- data for each class follows a bell shaped curve.
- normal distribution
- height of students.

### Perceptron

- model that tries to find a line to seperate two classes.
- doesn't use probabilities.
- adjust weights based on whether it got the prediction right or wrong.

If the data follows Gaussian distribution, the bayes classifier decision boundary is a linear function just like the Perceptron.

① Likelihood Ratio ($\Lambda(x)$)

class 1 and class 2

$\Lambda(x)$ is high $x \rightarrow$ class 1 (looks more likely to)

$\Lambda(x)$ is low $x \rightarrow$ class 2 (looks more likely to)

Shot on moto g54 5G
28 Jan 2026, 3:02 pm

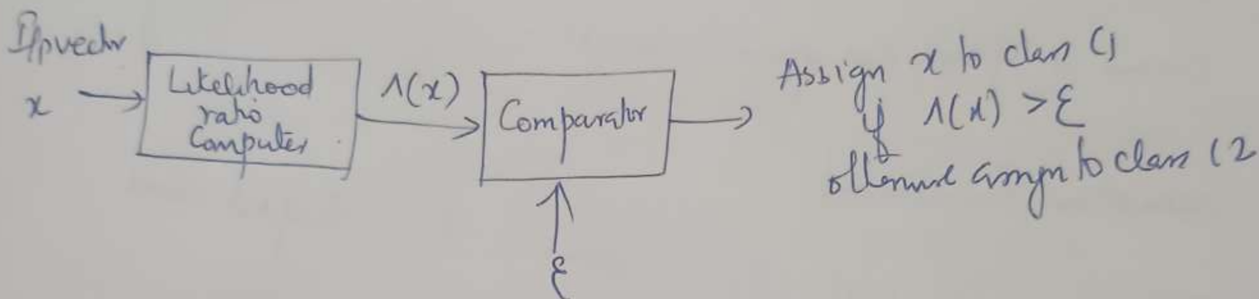
## Threshold $\xi$

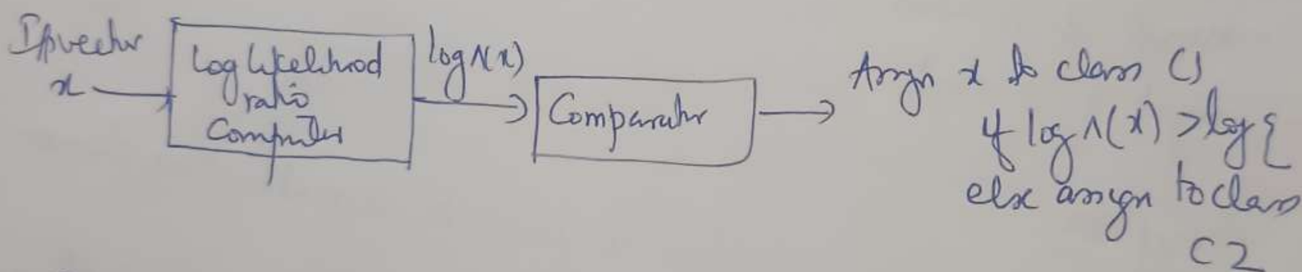
- cutoff point for deciding between the class

$\Lambda(x) > \xi$ assign $x$ to class 1 ($C_1$)

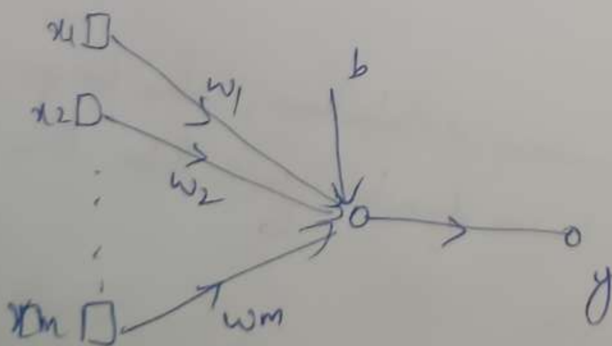
$\Lambda(x) \leq \xi$ assign $x$ to class 2 ($C_2$)

Likelihood ratio Test.




## Signal flow graph of Gaussian classifier




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

$$y = \log(\lambda_1/\lambda_2)$$

Log of likelihood ratio

O/p can be based on the ratio of prob. of 2 classes.

Shot on moto g54 5G

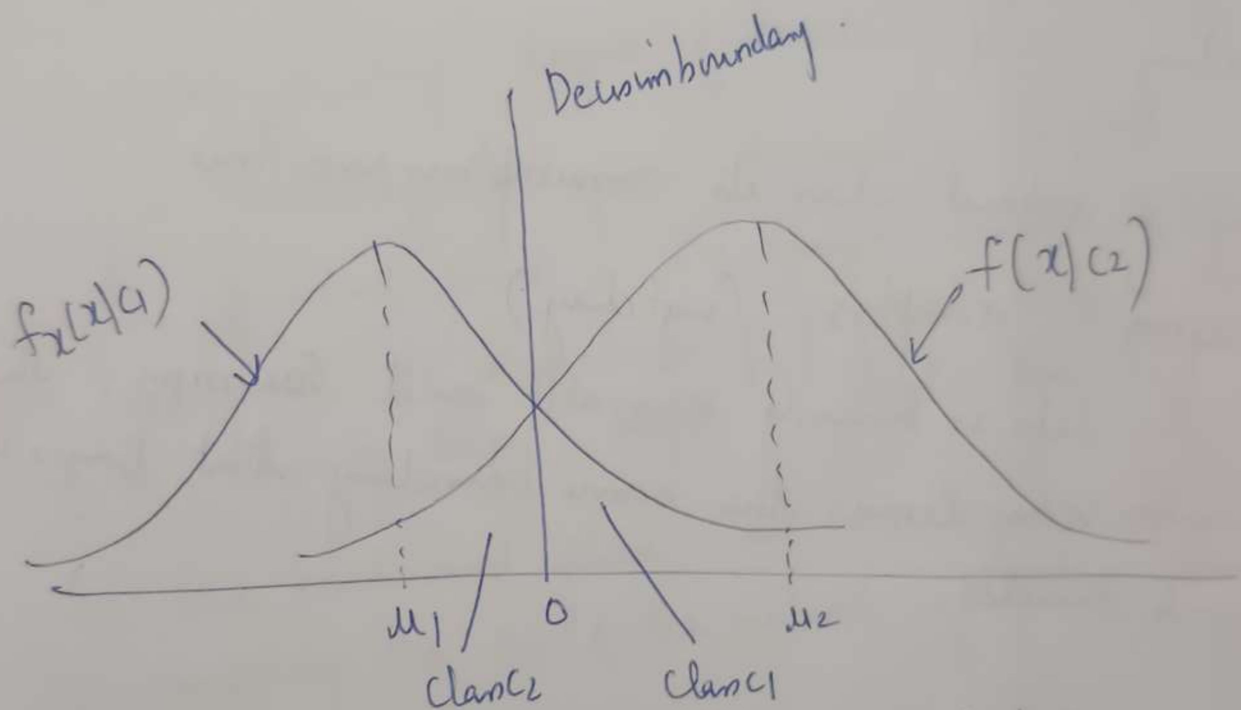
28 Jan 2026, 3:02 pm

$$W = C^{-1}(\mu_1 - \mu_2)$$

dif b/w avg. means of 2 classes adjusted by C (Covariance

$$b = \frac{1}{2}\left(\mu_2^T C^{-1} \mu_2 - \mu_1^T C^{-1} \mu_1\right)$$

bias values, which helps shift the decision boundary




Two overlapping gaussian distributions.

horizontal axis → i/p feature (height, weight)
vertical axis → prob. density
2 curves overlap → some values could belong to either class.
decision boundary → at $x = 0$ where classifier switches from predicting class 1 to Class C2.

Shot on moto g54 5G
28 Jan 2026, 3:02 pm

Backpropagation algorithm

→ core method for training MLP

→ 2 phases — Forward Pass (Signal Propagation)
— Backward Pass (Error propagation & wgt adjustment)

→ 1 Initialization

- all wgts and threshold (bias) are initialized to small random values.

→ 2 Presentation of Training Examples.

Training data is given in epochs.

N/w performs — Forward Computation,
— Backward "

→ 3 Forward Computation

Input vector : $x(n)$ is fed into the i/p layer.

Desired o/p : $d(n)$ is provided at the o/p layer.

Local field of a neuron $j$ in layer $l$.

$$v_j^{(l)}(n) = \sum_i w_{ji}^{(l)}(n)\, y_i^{(l-1)}(n)$$

O/p of neuron $j$

$$y_j^{(l)}(n) = \phi_j(v_j^{(l)}(n))$$

→ 4 Error Computation

$$e_j(n) = d_j(n) - o_j(n)$$

Shot on moto g54 5G

28 Jan 2026, 3:03 pm

→ 5 Backward Computation

– Error is propagated backward to adjust weight

Local gradient δ

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\,\phi_j'(v_j^{(L)}(n)) & \rightarrow \text{for neuron } j \text{ in o/p layer } L \\ \phi_j'(v_j^{(l)}(n)) \sum_k \delta_k^{(l+1)}(n)\, w_{kj}^{(l+1)}(n) \end{cases}$$

→ 6. Weight Update Rule · generalized delta rule

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha\left[w_{ji}^{(l)}(n) - w_{ji}^{(l)}(n-1)\right] + \eta\, \delta_j^{(l)}(n)\, y_i^{(l-1)}(n)$$

→ 7 Iteration

Steps 3–6 are repeated until stopping criterion is m

Shot on moto g54 5G

28 Jan 2026, 3:03 pm

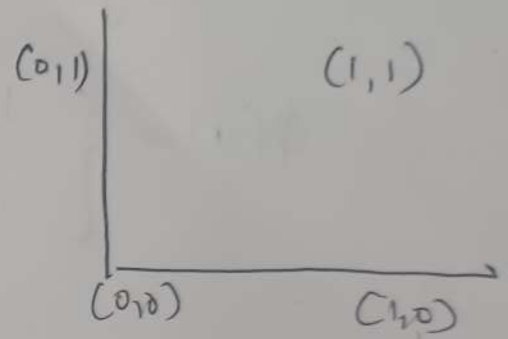
## XOR problem

→ In Rosenblatt's single layer perceptron, there are no hidden neurons.

→ it cannot classify i/p patterns that are not linearly separable.

XOR

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

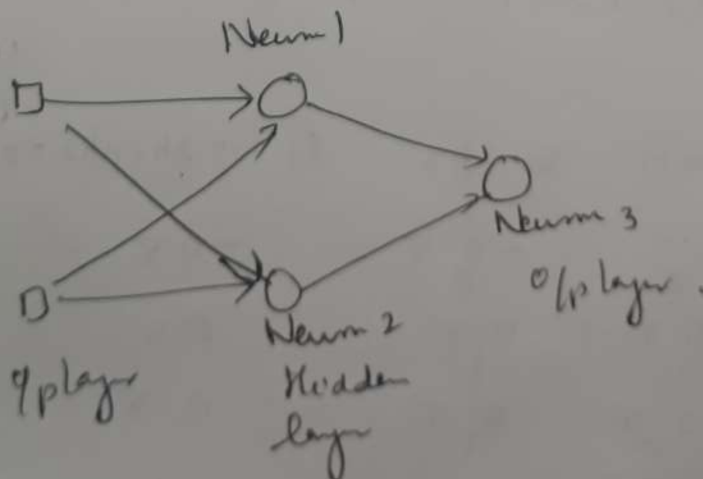



→ XOR not linearly separable.

→ We cannot draw a single straight line to seperate the i/p that yield different o/p.

→ A single layer perceptron due to its linear nature, fails to model the XOR fn.

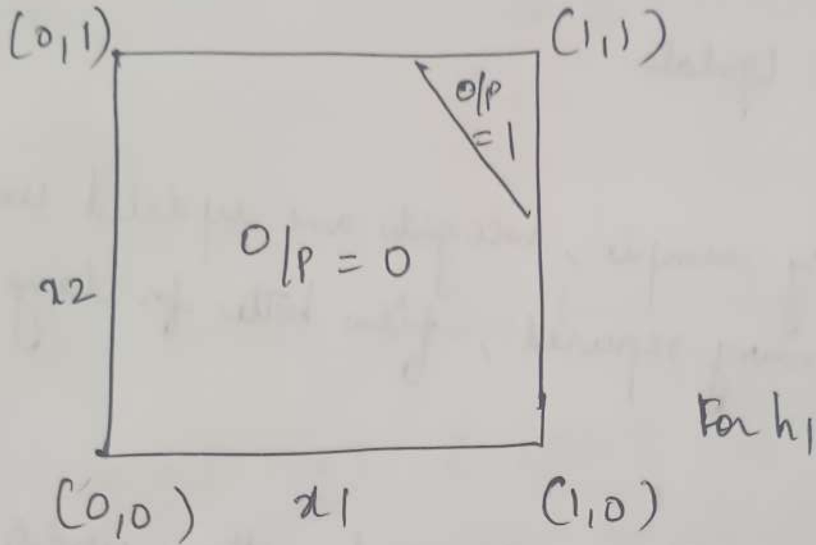
→ XOR problem can be solved using a single hidden layer with 2 neurons.

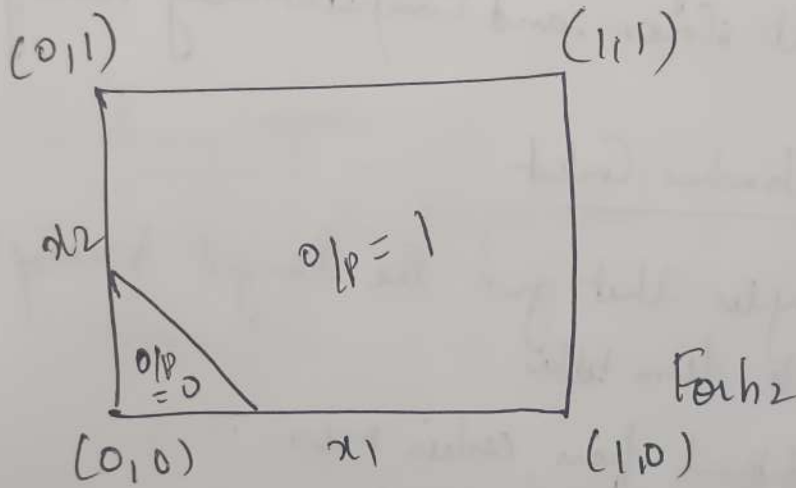


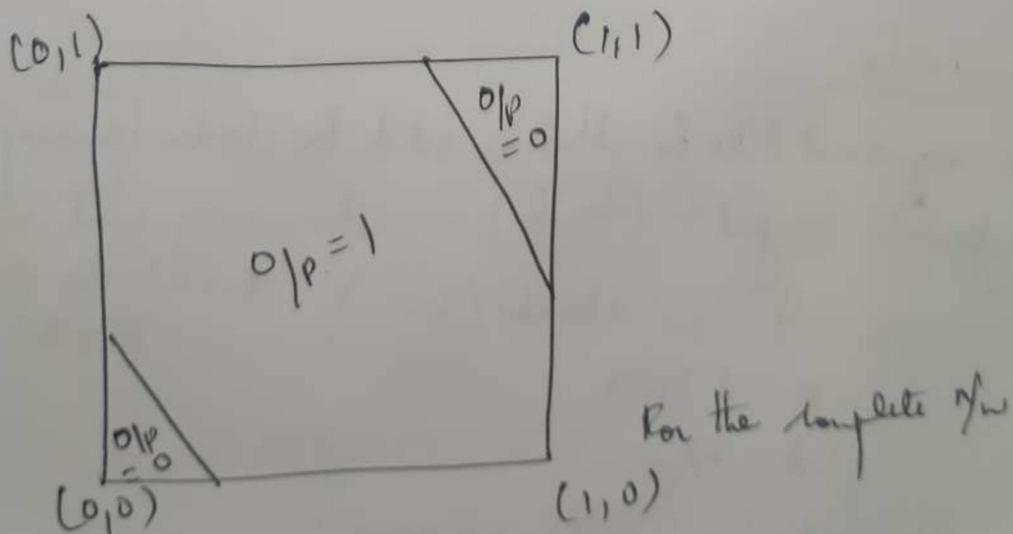
Shot on moto g54 5G

28 Jan 2026, 3:03 pm

Decision boundary

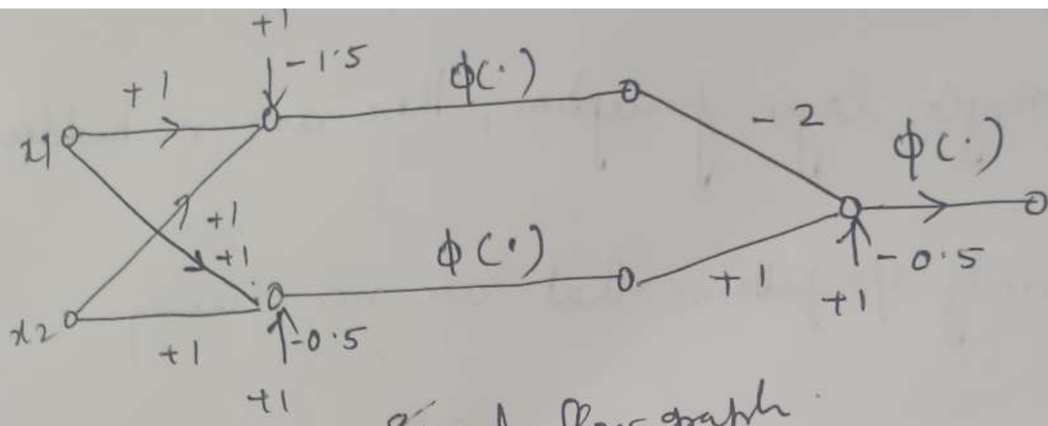


Shot on moto g54 5G

28 Jan 2026, 3:03 pm




Signal flow graph

$$\phi(v) = \begin{cases} 1 & v \geq 0 \\ 0 & v < 0 \end{cases}$$

$$s_1 = x_1(+1) + x_2(+1) + 1(-1.5)$$

$$= x_1 + x_2 - 1.5 \qquad h_1 = \phi(s_1)$$

$$s_2 = x_1(+1) + x_2(+1) + 1(-0.5)$$

$$= x_1 + x_2 - 0.5 \qquad h_2 = \phi(s_2)$$

$$s_3 = h_1(-2) + h_2(+1) + 1(-0.5) = -2h_1 + h_2 - 0.5$$

$$y = \phi(s_3)$$

| $x_1$ | $x_2$ | $s_1 \to h_1$ | $s_2 \to h_2$ | $s_3 = -2h_1 + h_2 - 0.5$ | $y$ |
|---|---|---|---|---|---|
| 0 | 0 | -1.5 → 0 | -0.5 → 0 | -0.5 | 0 |
| 0 | 1 | -0.5 → 0 | 0.5 → 1 | 0.5 | 1 |
| 1 | 0 | -0.5 → 0 | 0.5 → 1 | 0.5 | 1 |
| 1 | 1 | 0.5 → 1 | 1.5 → 1 | -1.5 | 0 |

Shot on moto g54 5G
28 Jan 2026, 3:03 pm

# Heuristics for making the Backpropagation Algorithm

① Stochastic VS Batch Update

Stochastic

After each training sample, weights are updated immediately

→ Faster, less memory required, often better for large datasets

Batch Update

Wait until all samples are processed, then update weights

More stable but slower and Computationally heavy

② Maximizing Information Content

Use training samples that give the largest training error

forces the n/w to learn better

Use examples different from earlier ones.

Datasets are shuffled to keep variety.

③ Activation fn

Use sigmoid like fn that is odd for faster learning

Hyperbolic tangent fn (tanh) - Commonly used in neural n/w due to its symmetry and bounded o/p

$$\phi(v) = a \tanh(bv)$$

$a = 1.7159$

$b = 2/3$

O/p range $[-1.7159, +1.7159]$

At $v=0$ slope is 1

$\phi(-v) = -\phi(v)$ Symmetric which helps faster

Shot on moto g54 5G
28 Jan 2026, 3:03 pm

4. Target Values

- Desired o/p should not be exactly at the limits of the activation fn ($+a$ or $-a$) – enters a saturated state where gradients becomes small – slow learning
- $a - \varepsilon$ or $-a + \varepsilon$
- This prevents neurons from saturation. – when the o/p of the activation fn becomes close to its max/min, gradient becomes very small or zero
- $a = 1.7159$  $\varepsilon = .7159$

Target values can then be conveniently chosen as $+1$ and $-1$.

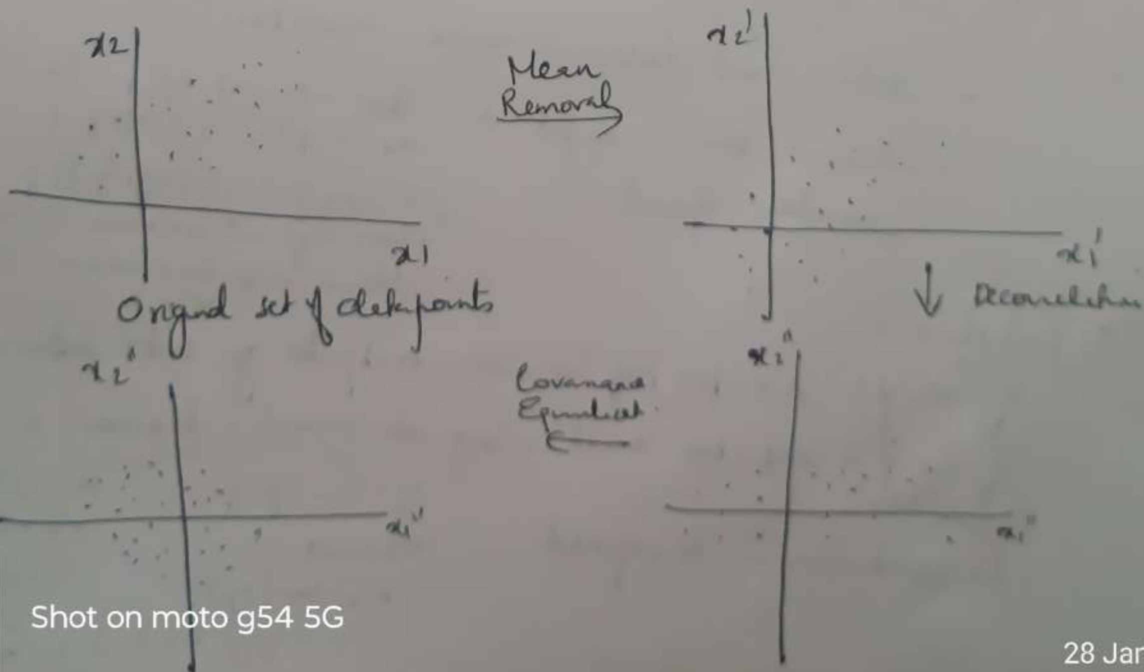
5. Normalizing Inputs.

Inputs should be preprocessed

Mean = 0

S.D = 1

If all inputs are large and always +ve, wgts may only grow in one direction – slow down learning. Normalization ensures balanced & faster convergence.




Shot on moto g54 5G
28 Jan 2026, 3:03 pm

1. Original set of data points

Input features $x_1, x_2$ are scattered.

They may have different scales, means and correlations.

2. <u>Mean Removal</u>

Each feature's mean is shifted to zero. $x'_i = x_i - \mu_i$

after this the data cloud is centered around the origin $(0,0)$.

But variables may still be correlated.

3. <u>Decorrelation</u>

Using PCA, features are transformed.

The cloud looks like an ellipse aligned with the coordinate axes.

4. <u>Covariance equalization</u>

- scale the data so that variances along each axis are equal.
- data cloud becomes more circular.
- ensures each weight in the neural n/w learns at a similar speed.

6. <u>Initialization</u>

If weights are too big neuron outputs go into saturation

If weights are too small - signals vanish, learning is very slow.

So initialization is required.

$\mu_v = E[v_j] = E[\sum_{i=1}^{m} w_{ji} y_i]$

$\rightarrow E[w_{ji} y_i] = E[w_{ji}] \cdot E[y_i]$

Choose wgts with zero mean $E[w_{ji}] = 0$

Normalized i/p have zero mean $E[y_i] = 0$

$\mu_v = 0$

$v_j = \sum_{i=1}^{m} w_{ji} y_i$

Variance,

$\sigma^2_v = Var(v_j)$

$\mu_v = 0 \leftarrow$ prediction mean is zero

$\sigma^2_v = E[v_j^2]$

$\sigma^2_v = m$ ... $\rightarrow$ Variance

Shot on moto g54 5G
28 Jan 2026, 3:03 pm

7. <u>Learning from hints</u>

→ giving network extra guidance – certain properties of the target fn $f(\cdot)$

→ Instead learning only from raw training data, the n/w can also learn from prior knowledge.

Symmetries, invariances, constraints → Rules that o/p must satisfy.

- Symmetries: fn remain the same when i/p are rearranged
- Invariances: o/p of the fn don't change under certain transformation.
  - Translation Invariance – object in a image stays same if shifted
  - Rotation Invariance
  - Scale Invariance.

→ If fn is symmetric, we can design the n/w to respect that.

→ If i/p are rotation invariant, we can add hints.

→ learning from hint → not only learn from example but using clues about the target fn $f(\cdot)$

8. <u>Learning rate</u>

Controls how fast weights are updated.

$\eta$ is too large – oscillations, divergence.

$\eta$ is too slow – very slow learning

Neurons closer to the o/p need smaller learning rates.

" " input need larger "

Shot on moto g54 5G

28 Jan 2026, 3:04 pm

Regularization in DL is a set of techniques used to redu[ce] overfitting by adding constraints or penalties to the model's parameters or by modifying the training process, so that the model generalizes better to unseen data.

Types of Regularization in DL

(1) L1 Regularization (Lasso)

- It adds absolute value of weights to the loss fn.
- Encourages weights to become sparse (few features are kept rest are dropped)
- model becomes minimalistic.

(2) L2 Regularization (Ridge)

- adds squared value of weights to the loss.
- encourages weights to be small but not exactly zero.
- Model becomes balanced & calm.

(3) Dropout

- Randomly turns off some neurons during training
- Prevents neurons from becoming too dependent on each other.

(4) Data Augmentation

- Modify training images / text / audio
- Gives model more variety without collecting new data.
- Makes model more [illegible] more imaginative

Shot on moto g54 5G
28 Jan 2026, 3:04 pm

## Early Stopping

- stops training when validation error starts going up.
- prevents overfitting
- model dont overheat.

Suppose we have 3 points.

| $x$ | $y$ |
|---|---|
| 1 | 3 |
| 2 | 5 |
| 3 | 7 |

$y = wx + b$

**Without Regularization**

$$w = \frac{\Sigma (x_i - \bar{x})(y_i - \bar{y})}{\Sigma (x_i - \bar{x})^2}$$

$b = \bar{y} - w\bar{x}$

$\bar{x} = 2 \quad \bar{y} = 5$

$w = 4/2 = 2$

$b = 1 \quad wx + b$

$y = 2x + 1$

Fits perfectly

**With L2 Regularization**

$$L = \frac{1}{n} \Sigma (y_i - (wx_i + b))^2 + \lambda w^2$$

$\lambda = 1$

$$w = \frac{4}{2+1} = \frac{4}{3} \approx 1.333$$

$b = \bar{y} - w\bar{x}$

$= 2.334$

With regularization $y \approx 1.333x + 2.334$

weight shrinks, avoids overfitting

Perform well with the unseen data.

Shot on moto g54 5G

28 Jan 2026, 3:04 pm

Regularization → – A central [illegible] algo. that will perform well not just on the training data but also on new inputs.

- strategies used in ML.
- is defined as any modification we make [to] a learning algo. that is intended to reduce its generalization error but not its training error.

Parameter Norm Penalty.

- Parameter → weights.
- Norm → size or magnitude. L1 or L2 measures how big weights are.
  - L1 norm $|w_1| + |w_2| + \dots$
  - L2 norm $w_1^2 + w_2^2 + \dots$
- Penalty → extra cost.

Loss with Penalty = Original Loss + $\lambda \cdot$ Norm of weights

Regularization approaches are based on limiting the capacity of the models by adding norm penalty $\Omega(\theta)$ to the objective fn.

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha \Omega(\theta)$$

- $\tilde{J}(\theta; X, y)$ → Regularized obj. fn.
- $J(\theta; X, y)$ → Original obj. fn. (It defines what model is learning)
- $\alpha \Omega(\theta)$ → Penalty Term.

$\alpha \in [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term relative to std. obj. fn $J$.

$\alpha = 0$ → no regularization.

Shot on moto g54 5G

28 Jan 2026, 3:04 pm

Penalty is also known as Weight decay.

when our training algo minimizes the regularized obj fn $\hat{J}$ will decrease both original objective J on the training data and some measure of the size of the parameters $\theta$.

## $L^2$ Parameter Regularization

$L^2$ parameter norm penalty → weight decay.

Regularization term $\Omega(\theta) = \frac{1}{2}\|w\|_2^2$ to the obj. fn.

Ridge Regression or Tikhonov regularization.

$\Omega(\theta)$ → regularization penalty.

$\theta$ → refers to the weights.

$\|w\|_2^2$ $\quad \|w\|_2 = \sqrt{w_1^2 + w_2^2 + \dots + w_n^2}$

$$\|w\|_2^2 = \sum_{i=1}^{n} w_i^2$$

Sum of squares of all weights.

$1/2$ → added for mathematical [illegible] curve.

$\frac{1}{2}w^2 \quad \frac{1}{2} \times 2w = w$

Weights are kept as small

Simple model

Better generalization

Reduced overfitting.

Shot on moto g54 5G

28 Jan 2026, 3:04 pm

# Dataset Augmentation

- improve a ML model - is by training it on more d
- To overcome ~~data~~ limited data, you can create fake (s data and add it to your training set.
- A classifier maps a complex high dimensional input xx a single category label yyy. The main challenge is m the classifier invariant to many possible transformations of th input.
- By transforming the i/p data xxx, you can easily generate new (x,y) (x,y) (x,y) pairs for training.

Cat Image → Rotate Cat, Flip Cat, Mirror Cat, Zoom Cat

## Limitations

→ approach doesn't extend to all tasks.
→ generating fake data for density estimation is hard.

- Data augmentation is especially useful for object recognition.
  - Translating images
  - Rotating
  - Scaling
- Donot apply transformations that change the correct label
- Flipping or rotating letters could change 'b' to 'd' '6' to 9 which alters the class label
- flips and 180° rotations are not appropriate

Shot on moto g54 5G
28 Jan 2026, 3:04 pm

- Injecting noise into inputs of neural n/w can be seen as a form of data augmentation. (injecting → randomness)
- Neural n/w trained with noisy i/p improves robustness. (model able to see corrupted data during training)
- Noise injection — denoising autoencoders
- Injecting noise into hidden units can be viewed as data augmentation.
- augmentation can significantly reduce generalization error.
- application specific operations are treated as preprocessing step.

Original pixel = 120
Input gaussian noise mean 0 σ 10
120 + N (0, 10)
= 127

Semisupervised Learning

- Uses labeled and unlabeled data to improve model performance.
- We have unlabeled data from the distribution $P(x)$ tells what i/p looks like.
- We have labelled data from the distribution $P(x, y)$ how i/p relates to o/p
- Estimate $(y|x)$ directly predict $y$ from $x$. Conditional distribution
- The goal is to learn a representation $h = f(x)$ for the i/p data $x$.
- The idea is that examples belonging to the same class should have similar representations in the feature space

Shot on moto g54 5G
28 Jan 2026, 3:05 pm

# Challenges in Optimization

- Optimization is an extremely difficult task.
- Traditional ML : careful design of objective function and constraints to ensure convex optimization.

① Local Minima

In convex optimization, problem is one of finding a local minimum.

Some convex fn have a flat region rather than a global minimum pt.

Any pt within the flat region is acceptable.

With non convexity of neural nets many local minima are possible.

Many deep models are guaranteed to have an extremely large no. of local minima.

This is not necessarily a major problem.

② Ill conditioning of the Hessian

The Hessian matrix $H = \nabla^2 f(x)$ captures the curvature of the fn.

The hessian of a scalar fn $f(x)$ is the matrix of second partial derivatives.

$$H = \nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots \\ \frac{\partial^2 f}{\cdots} & & \end{bmatrix}$$

Shot on moto g54 5G

28 Jan 2026, 3:05 pm

Condition number of Hessian

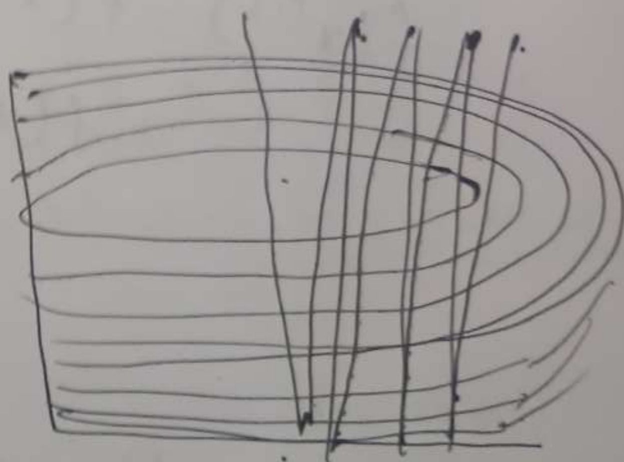
$$k = \frac{\lambda_{max}}{\lambda_{min}}$$

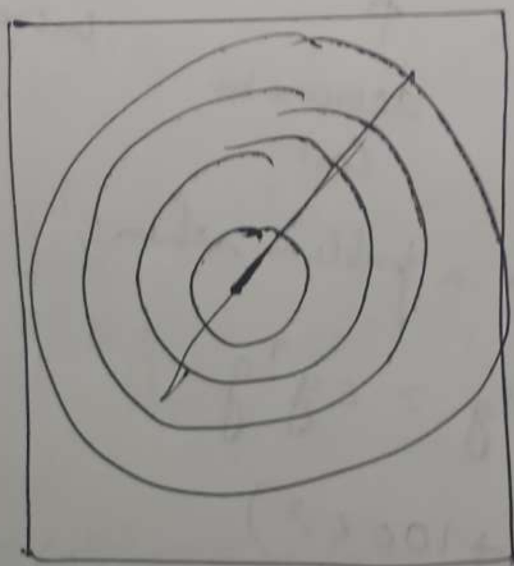
$\lambda_{max}$ : largest eigen value of H.

$\lambda_{min}$ : smallest eigen value of H

If k is large, the hessian is ill conditioned.

This means the fn's contours are very elongated (like thin ellipses) so gradient descent takes zig zag paths and converges very slowly.





Well Conditioned case.
Contours are round.
eigenvalues are equal.

Well conditioned – round contours – fast convergence

Ill conditioned stretched " – slow "

Shot on moto g54 5G
28 Jan 2026, 3:06 pm

Scaling cancels out

## Plateaus, Saddle Points and other flat Regions

A saddle point is a critical point where the gradient is zero but the Hessian has both positive and negative eigen values.

$$f(x,y) = x^2 - y^2$$

Gradient $\nabla f = (2x, -2y)$

At $(0,0)$ gradient is 0.

$$H = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$$

eigen values $+2, -2 \rightarrow$ mixed signs saddle.

Along the $x$ direction the function curves up along the $y$ direction it curves down.

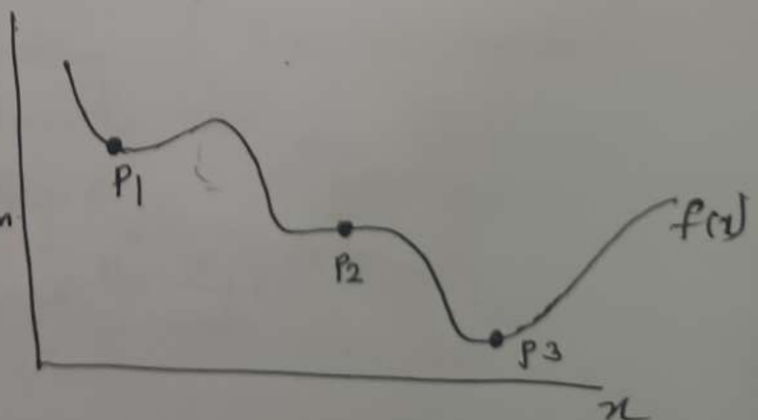
So the point is neither a pure min nor max.

In high dimensions saddle points become very common and can dominate training behavior.

$P_1 \rightarrow$ local minimum

$P_2$ : Saddle point

$P_3$ : Global minimum



Shot on moto g54 5G

28 Jan 2026, 3:06 pm

## Plateau

→ is a region of the loss surface where the fn is almost flat.

Gradient = 0

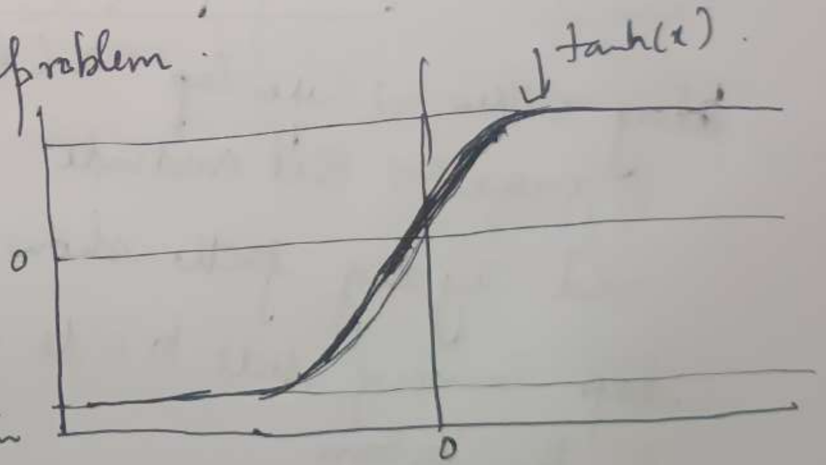
Loss doesn't change much no matter which direction you move.

→ gradients are tiny, updates to weights are super small.

→ optimizer gets stuck.

→ for fn like tanh or sigmoid, when i/p are very large or small derivatives ≈ 0

→ Vanishing gradients problem.




$y = +1$

$y = -1$

tanh saturates

$x > 2$ → positive plateau

$x = -2$ → negative plateau.

Curve is almost flat — gradients vanish.

## Ravine

→ is a very narrow, steep valley in the loss surface.

→ It happens when the loss fn has

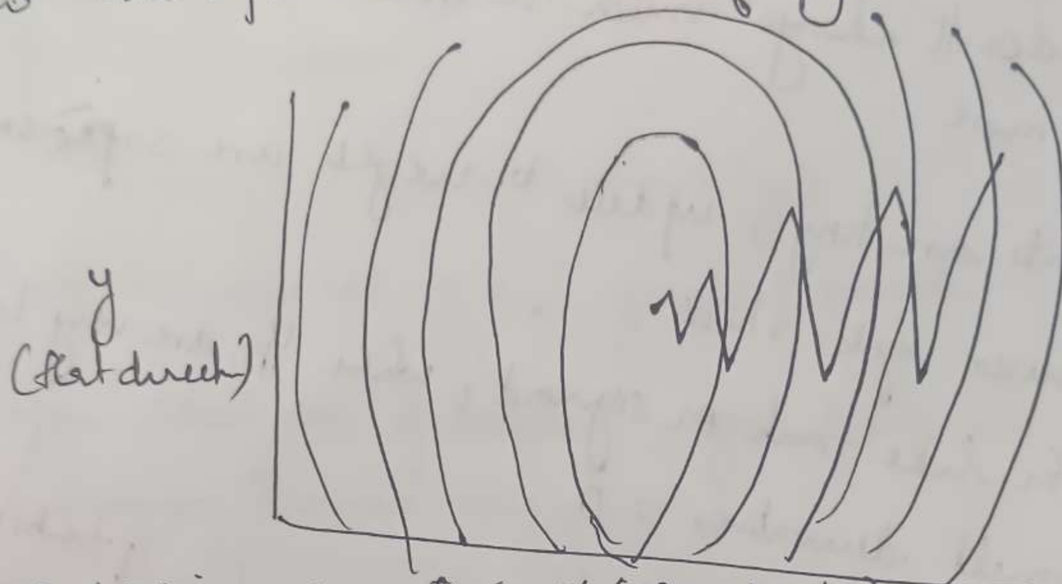
one direction with very steep slope.

Another direction with very gentle slope.

Shot on moto g54 5G
28 Jan 2026, 3:06 pm

## Zig-zag path in gradient descent

Gradient points steeply down the ravine walls instead of smoothly along the valley floor.
So the optimizer bounces left right left.




Along x axis → super steep     x (steep direction)
y axis → flat and wide.

red zig zag path shows how gradient descent keeps bouncing side to side instead of going straight to the bottom.

## Cliffs and exploding gradients

→ In deep N/w multiplying many large weights creates steep regions in the loss surface like a cliff edge.
→ when gradient descent takes a step it can fall off the cliff.
→ Update is too large
→ Optimizer skips [illegible] minima.

Shot on moto g54 5G
28 Jan 2026, 3:06 pm

## Long Term dependencies

→ In deep computational graphs, gradients are multiplied repeatedly.

→ Vanishing gradients → signal shrinks, early layer's can't learn.

→ Exploding gradients – signal grows uncontrollably.

→ In RNN applying the same weights at each time step makes it worse – n/w struggles to remember information far in the past.

⇒ N/w becomes good at remembering recent information but struggles with long term dependencies → RNN forgets the thing happened many steps ago.

LSTM, GRU

Shot on moto g54 5G

28 Jan 2026, 3:06 pm

Module 4

## Convolution N/w

- are also known as Convolutional neural n/w or CNN
- CNN are a specialized kind of neural n/w for processing data that has a known, grid like topology
- 1D grid time series data.

  2D grid - image data (2D grid of pixels)
- network employs a mathematical operation convolution
- special kind of linear function.
- Convolutional n/w are simply neural n/w that use convolution in place of general matrix multiplication in atleast one of their layers.
- Convolution is an operation on 2 fns of a real valued argument.

$$s(t) = (x * w)(t)$$

$s(t)$: O/p → feature map; $x$: I/p; $w$: kernel.

Shot on moto g54 5G
28 Jan 2026, 3:06 pm

In machine learning,

Input = a tensor (multidimensional array of nos. ie image pixels)

Kernel = a tensor of learnable parameters

O/p = another tensor, a smaller or same size depending on padding / stride.

For 2D Convolution

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-$$

$$S(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n)$$

$S(i,j)$ = O/p pixel at row i, col j

$I(m,n)$ pixel value at row m col n

$K(i-m, j-n)$ – kernel value

↓ reverse (flip) the kernel before sliding it across the image

CNN skip the flip, just do the same

Size of the O/p feature map is given by,

$$\left\lfloor \frac{H - k_h + 2p}{s} \right\rfloor + 1$$

I/p image = 32 × 32

Shot on moto g54 5G

Kernel = 5 × 5

Stride = 1

feature map = $\left\lfloor \frac{32 - 5 + 0}{1} \right\rfloor + 1$

28 Jan 2026, 3:06 pm

# The Convolution Operation

Input

| a | b | c | d |
|---|---|---|---|
| e | f | g | h |
| i | j | k | l |

Kernel

| w | x |
|---|---|
| y | z |

$aw + bx + ey + fz$

$bw + cx + fy + gz$

$cw + dx + gy + hz$

$ew + fx + iy + jz$

$fw + gx + jy + kz$

$gw + hx + ky + lz$

Convolve image with a filter (Kernel) matrix.

| 3 | 0 | 1 | 2 | 7 | 4 |
|---|---|---|---|---|---|
| 1 | 5 | 8 | 9 | 3 | 1 |
| 2 | 7 | 2 | 5 | 1 | 3 |
| 0 | 1 | 3 | 1 | 7 | 8 |
| 4 | 2 | 1 | 6 | 2 | 8 |
| 2 | 4 | 5 | 2 | 3 | 9 |

6×6

$*$

| 1 | 0 | -1 |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |

3×3

=

| -5 | -4 | 0 | 8 |
|---|---|---|---|
| -10 | -2 | 2 | 3 |
| 0 | -2 | -4 | -7 |
| -3 | -2 | -3 | -16 |

4×4

$$\frac{6-3}{1} + 1$$

Shot on moto g54 5G

28 Jan 2026, 3:06 pm

## Pooling

→ summarizes nearby o/ps to make representation smaller and more stable to small translation.

→ Reduce Size (Dimensionality Redu

Feature maps from convolution are often big.

Pooling makes them small

faster training, fewer parameters, less memory

### Max Pooling

- Keeps the max. in each window.

### Avg Pooling

- Keeps the avg.

$$\begin{bmatrix} 1 & 1 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 3 & 2 & 1 & 0 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

2x2 Max Pooling with stride 2

{1, 1, 5, 6} max = 6

{2, 4, 7, 8} max = 8

{3, 2, 1, 2} max = 3

{1, 0, 3, 4} max = 4.

$$\begin{bmatrix} 6 & 8 \\ 3 & 4 \end{bmatrix}$$

Avg Pooling

Shot on moto g54 5G

28 Jan 2026, 3:07 pm

## Pooling

→ A typical layer of Convolutional n/w consists of 3 stages

1. Convolution
2. Detector stage → each linear activation is run through a nonlinear activation fn such as relu.
3. Pooling function to modify the O/P of the layer.

- A pooling fn replaces the O/P of the net at a certain location with a summary statistic of the nearby O/P (max pooling or avg pooling)

A pooling independently operates on each feature map to produce another feature map.

Operation of pooling doesnt change the no. of feature maps.

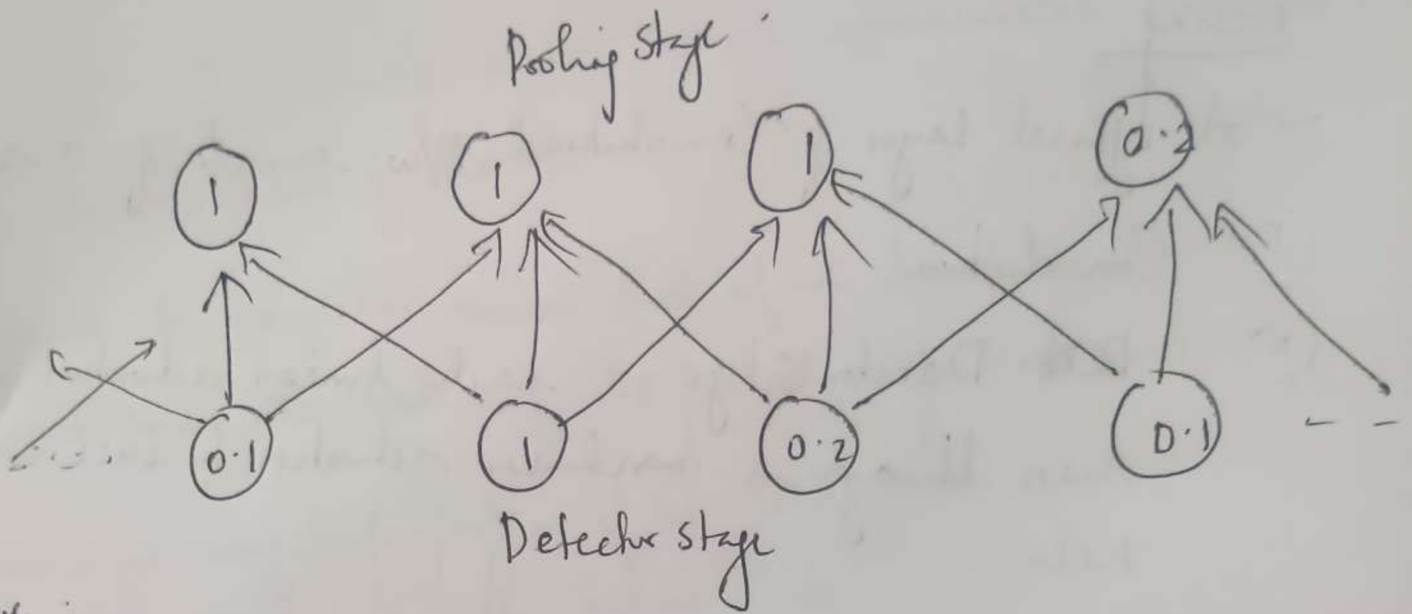
An earliest convolutional n/w LeNet-5 → avg pooling

Max Pooling is more popular than avg. pooling

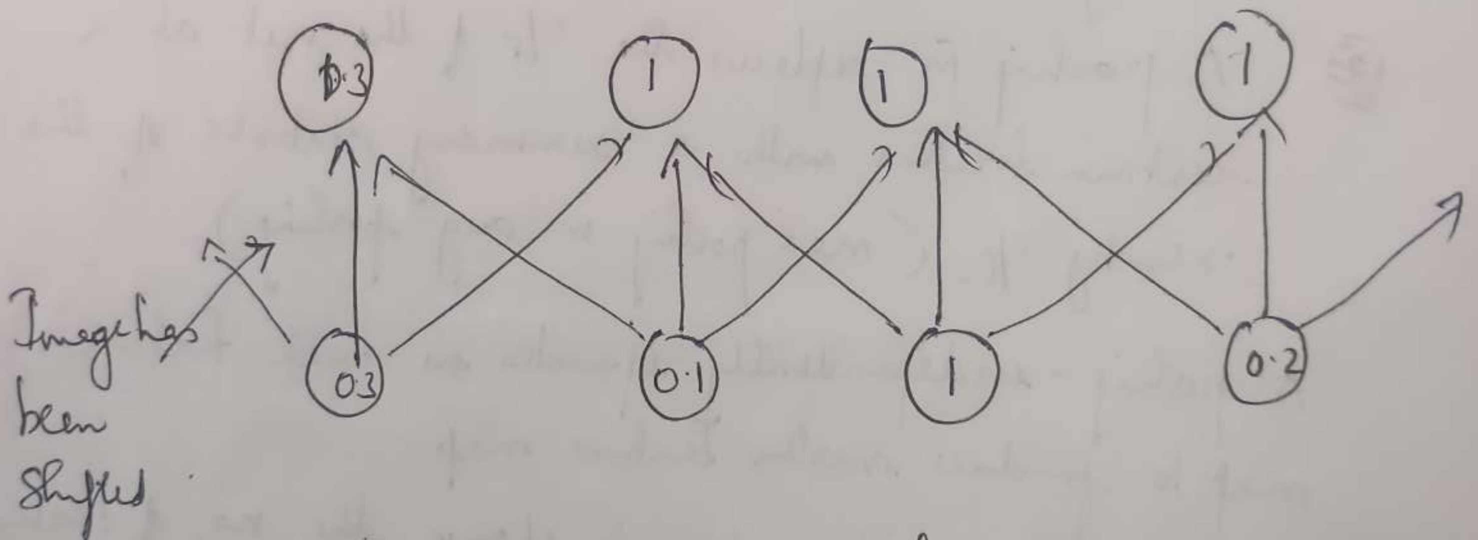
Pooling reduces the spatial size of the feature and only few operations are required.

4×4 → 2×2

Shot on moto g54 5G
28 Jan 2026, 3:07 pm




Image



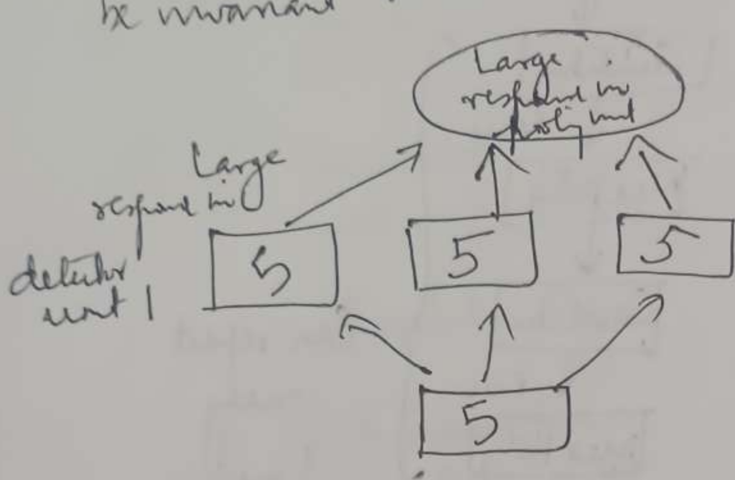

Every value in the bottom row has changed, but only half of the values in the top row have changed.

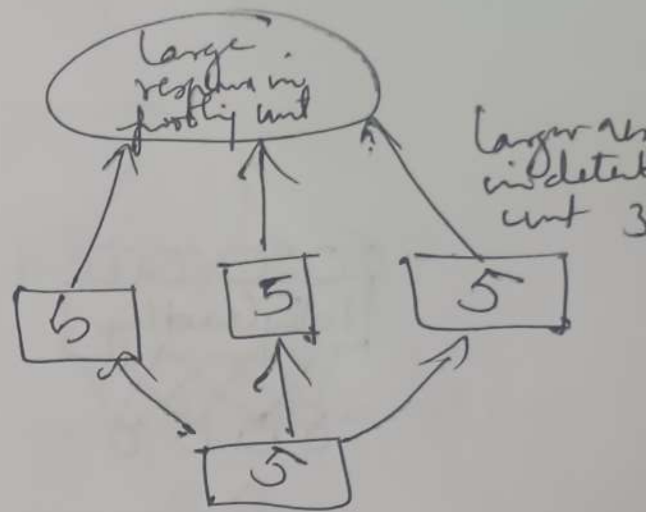
Max pooling introduces invariance.

Shot on moto g54 5G

28 Jan 2026, 3:07 pm

The use of pooling can be viewed as adding an infinitely strong prior that the function the layer learns must be invariant to small translations.




A pooling unit that pools over multiple features that are learned with seperate parameters can learn to be invariant to transformations of the i/p
3 filters are intended to detect a handwritten image 5

Pooling is essential for handling i/p of varying size. for example if we want to classify images of variable size the i/p to the classification layer must have a fixed size.

Varying the size of an offset b/w pooling regions so that classification layer receives the same no. of summary statistics.

Shot on moto g54 5G
28 Jan 2026, 3:07 pm

## Variants of the Basic Convolution

→ when we refer to convolution in the context of neural n/w we actually mean an operation that consists of many applications of convolutions in parallel.

→ Convolution with single kernel can only extract one kind of feature.

→ We use many filters at once.

→ Each filter extracts a diff. kind of feature (edges, colors, textures).

→ CNN perform many convolutions together — thats gives multiple feature maps.

→ Each layer of network extract many kinds of features at many locations — many kernels per layer.

→ images are not single value grids. Each pixel has multiple values (R, G, B).

→ Each pixel is like a vector.

→ Each pixel = R, G, B.

→ input as a 3D tensor w × h × 3 channels.

→ In a multilayer CN, i/p to the second layer is the o/p of first layer. Each layer receives multiple i/p and applies new filters.

Shot on moto g54 5G

28 Jan 2026, 3:08 pm

CNN works on 3D data. One dimension R, G

## Multichannel Convolution

CNN → deal with multichannel data. Grayscale RGB

i/p → several channels

kernels → handle all these channels.

operations are not guaranteed to be commutative even if kernel flipping is used

a+b = b+a ×

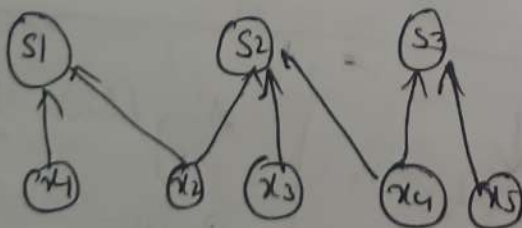
Multichannel operations are commutative if

no. of i/p channels = no. of o/p channels.

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,\, j+m-1,\, k+n-1} \, K_{l,i,m,n}$$

We may want to skip over some positions of the kernel in order to reduce the computation cost.

Stride




## Zero Padding

One essential feature.

When we do convolution the image becomes smaller after every layer.

To prevent it shrinking too fast we add extra zeros around the borders of the image.

i/p 5×5 kernel 3×3 o/p 3×3.

Without zero padding n/w will shrink the image size.

3 types

1. No zero padding – Valid Convolution.
   We don't add any zeros.
   o/p becomes smaller than i/p.
   Valid convolution – because it only uses valid pixels inside the image.

2. Enough zero padding – Same convolution
   We add zeros to keep
   o/p = i/p size
   5×5 – 3×3 kernel + padding = 1
   o/p 5×5

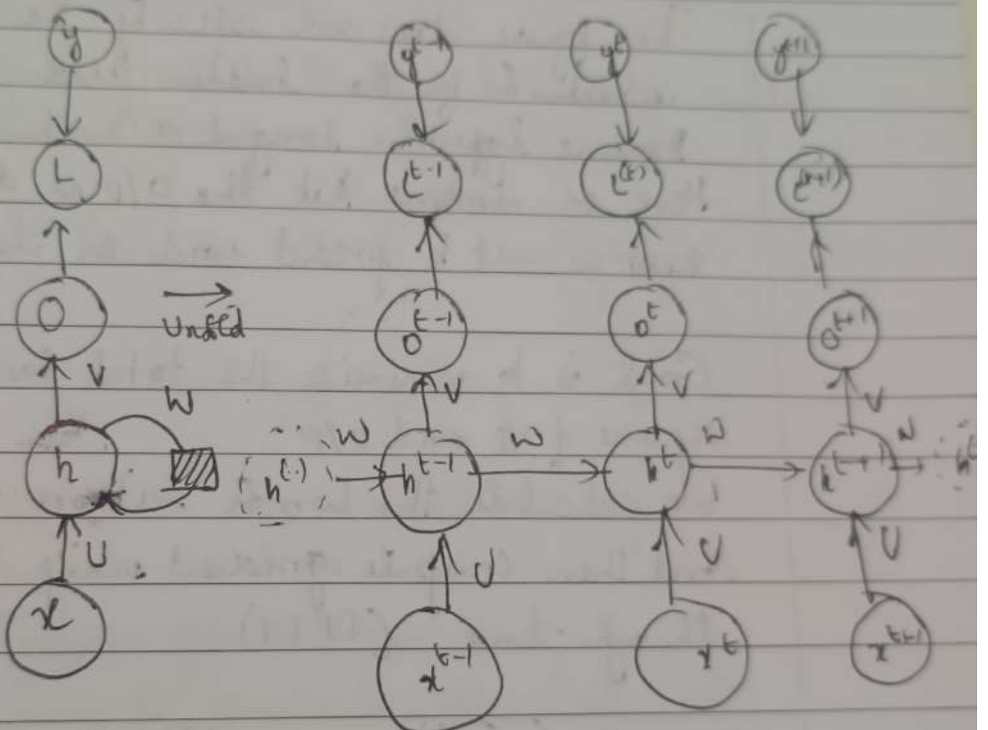
3. Full Convolution
   We add enough zeros multiple times
   larger o/p than i/p
   o/p width = m + k – 1

Shot on moto g54 5G

28 Jan 2026, 3:08 pm

## Recurrent Neural n/w

Recurrent n/w that produce an o/p at each time step and have recurrent connections b/w hidden units.




I/p sequence of x values are mapped to the corresponding sequence of o/p o values.

- Loss L is calculated from o and y.

$\hat{y}$ = softmax(o)

y = target.

U, V, W → weight matrix

Shot on moto g54 5G
28 Jan 2026, 3:10 pm

Shot on moto g54 5G

Forward Propagation begins with a specified initial state $h^{(0)}$. Time step $t=1$ to $t=\tau$ __/__/__

we apply,

$$a^{(t)} = b + W \cdot h^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = \tanh(a^{(t)})$$
$$o^{(t)} = c + Vh^{(t)}$$
$$y^{(t)} = \text{softmax}(o^{(t)})$$

b, c → bias. U, V, W → weight vectors.

The figure does not specify the choice of activation fn for the hidden units.
Assume hyperbolic tangent activation fn.
Here we assume that the O/P is discrete as if the RNN is used to predict words or characters.

Goal is to minimize the total loss over a sequence of I/P and O/P.
We calculate the loss fn across time steps and then compute gradient using Backpropagation through time (BPTT).

$$L(\{x^{(1)}, \ldots x^{(\tau)}\}\{y^{(1)} \ldots y^{(t)}\})$$
$$= \sum_t L^{(t)}$$

$L^{(t)}$ loss at time step t

t : total no of time steps

28 Jan 2026, 3:10 pm

Model Prediction

$p_{model}(y^{(t)} \mid \{x^{(1)} \dots x^{(t)}\})$

$= \text{softmax}(o^{(t)})$.

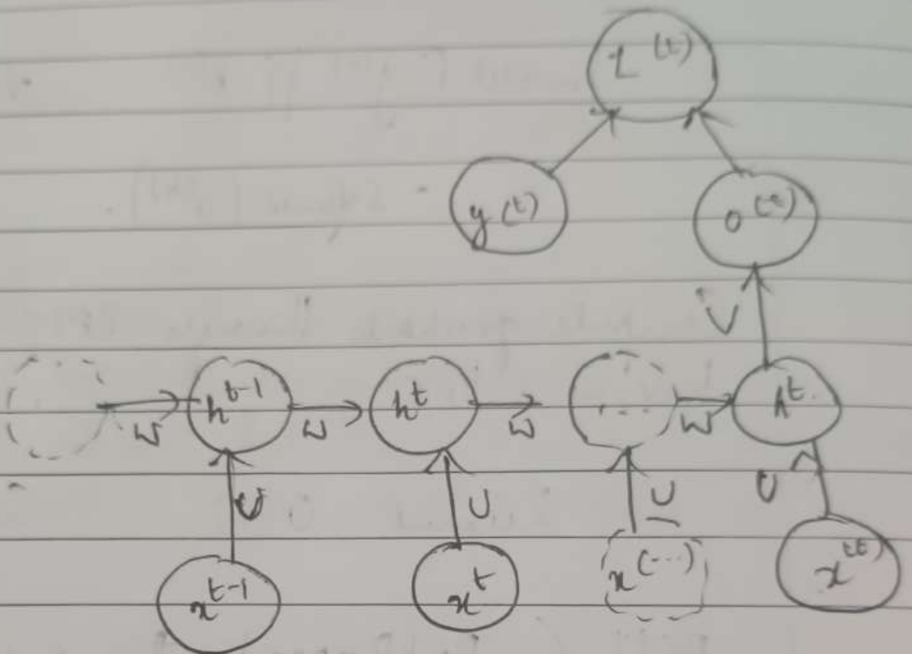
Compute gradients through BPTT and update $U, V, W, b, c$.

Runtime $O(t)$ Sequential - cannot parallel[illegible]

BPTT ( BackPropogation through time)

- applied to the unrolled RNN n/w.
- Compute total loss $L = \sum_t L^{(t)}$
- Perform forward pass through time.
- Compute gradients $\frac{\partial L}{\partial W}, \frac{\partial L}{\partial U}, \frac{\partial L}{\partial V}$
- update parameters using gradient descent.

Shot on moto g54 5G
28 Jan 2026, 3:10 pm




Time Unfolded recurrent neural n/w with a single o/p at the end of the sequence.
Used to summarize a sequence and produce a fixed size representation
Used as i/p for further processing.

Shot on moto g54 5G
28 Jan 2026, 3:11 pm

__/__/__

## Feedforward vs CNN vs RNN

**FF**

Learns separately for each i/p.

**CNN**

Shares parameters locally.

**RNN**

Shares parameters through time
Each o/p depends on the former o/p.

## Unfolding Computational graphs:

- A computational graph is a visual representation of how computations are performed step by step
- It helps us formalize and visualize how i/p are transformed into o/p.

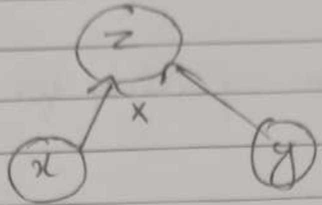
- Each node in the graph represents a variable. (scalar, vector, matrix or tensor)
- Each edge represents an operation applied b/w variables.

$x \rightarrow y$

y is computed from x.

Shot on moto g54 5G
28 Jan 2026, 3:11 pm

$z = x \times y$




Recurrent Computation is defined as

$$s^{(t)} = f(s^{(t-1)}; \theta)$$

- $s^{(t)}$ ↑ state a time step t
- $s^{(t-1)}$ ↑ previous state
- $\theta$ → parameters: weights shared across all time steps

Unfolding means expanding the recurrence to show how each state depends on earlier ones

$$\begin{aligned} s^{(t)} &= f(s^{(t-1)}; \theta) \\ &= f(f(s^{(t-2)}; \theta); \theta) \\ &= f(f(f(s^{(t-3)}; \theta); \theta); \theta) \\ &= f(f(f(\ldots f(s^{(1)}; \theta) \ldots))) \end{aligned}$$

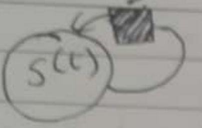
info flows through many layers over time creating a deep chain of computation

$\theta \rightarrow$ parameters.

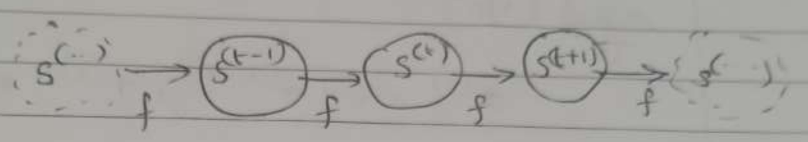
Shot on moto g54 5G
28 Jan 2026, 3:11 pm

Shot on moto g54 5G

time delay → fn or operator $f(\,;\theta)$ that track one state to the next



$\longrightarrow$ Unfold



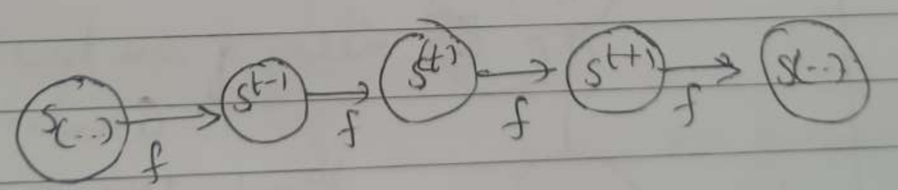

A classical form of a dynamical system

$$s^{(t)} = f(s^{(t-1)};\theta).$$

Unfold 3 times

$$s^{(t)} = f(f(f(s^{(t-3)};\theta);\theta);\theta)$$

Recurrent models reuse parameters (same θ) over many time steps.




28 Jan 2026, 3:11 pm

## Bidirectional RNNs

In many applications we want to O/P a prediction of $y^{(t)}$ which may depend on the whole I/P sequence.
Speech recognition – linguistic dependencies b/w nearby words. Bidirectional RNN were implemented to address this need.
Application such as handwriting recognition, speech recognition and bioinformatics.
Bidirectional RNN combine an RNN that moves forward through time beginning from the start of the sequence with another RNN that moves backward through time beginning from the end of the sequence.

Shot on moto g54 5G
28 Jan 2026, 3:11 pm

$y^{t-1}$ $y^{t}$ $y^{t+1}$

$L^{t-1}$ $L^{t}$ $L^{t+1}$

$o^{t-1}$ $o^{t}$ $o^{t+1}$

$g^{t-1}$ $g^{t}$ $g^{t+1}$

$h^{t-1}$ $h^{t}$ $h^{t+1}$

$x^{t-1}$ $x^{t}$ $x^{t+1}$

Shot on moto g54 5G

28 Jan 2026, 3:11 pm

... is of ... hidden state -
i/p from hidden
state at prev. timest.
t - 1

## Encoder - Decoder Sequence to sequence architecture

Discuss how an RNN can be trained to map an i/p sequence to an o/p sequence which is not necessarily of the same length.

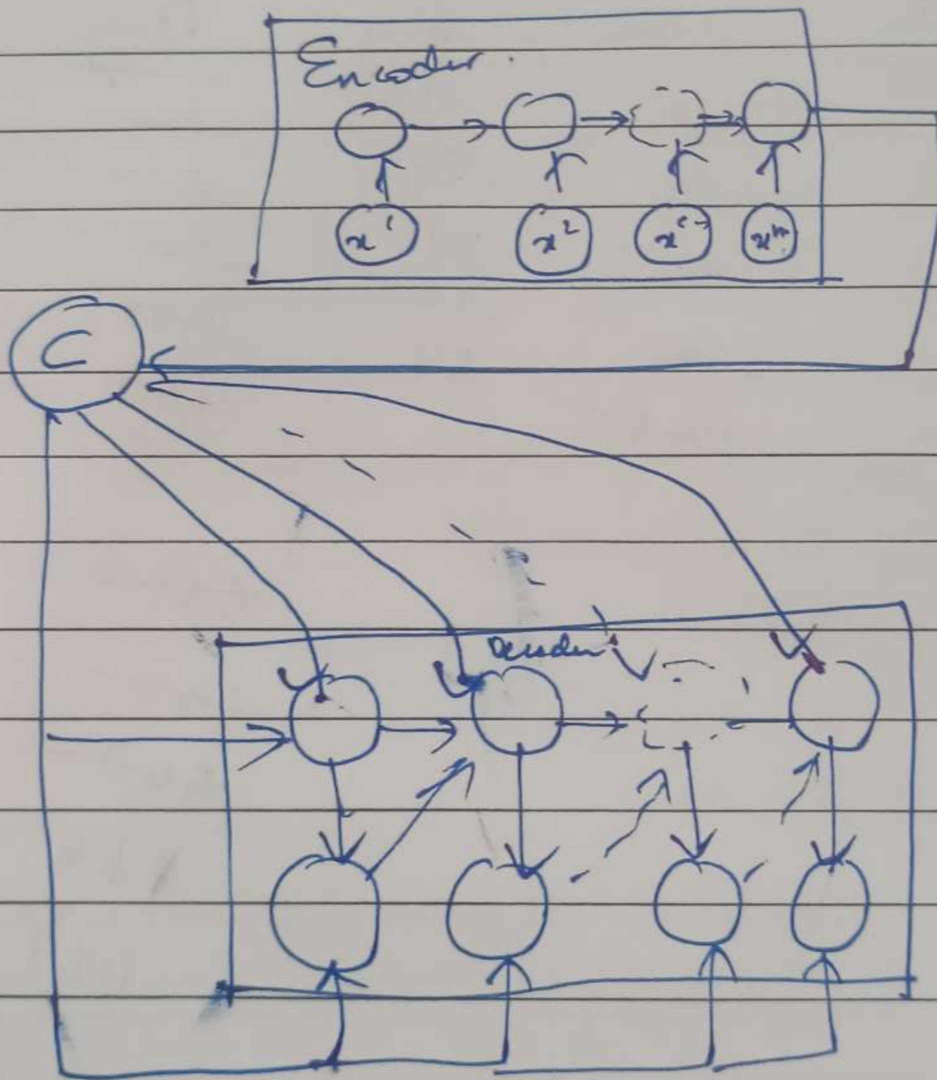
- In speech recog, machine translation, i/p o/p sequence in the training set are not of same length.
- Representation of context $C$.
- The Context $C$ might be a vector or sequence of vectors that summarize the initial sequence $X = x^{(1)} \ldots x^{(n_x)}$.
- An encoder or reader or i/p RNN processes the i/p sequences. The encoder emits the context $C$, usually as a simple fn of its final hidden state.

Shot on moto g54 5G

28 Jan 2026, 3:12 pm

- A decoder or writer or O/P of RNN is fixed length to generate the O/P sequence
  $Y = y^{(1)} \dots y^{(n_y)}$

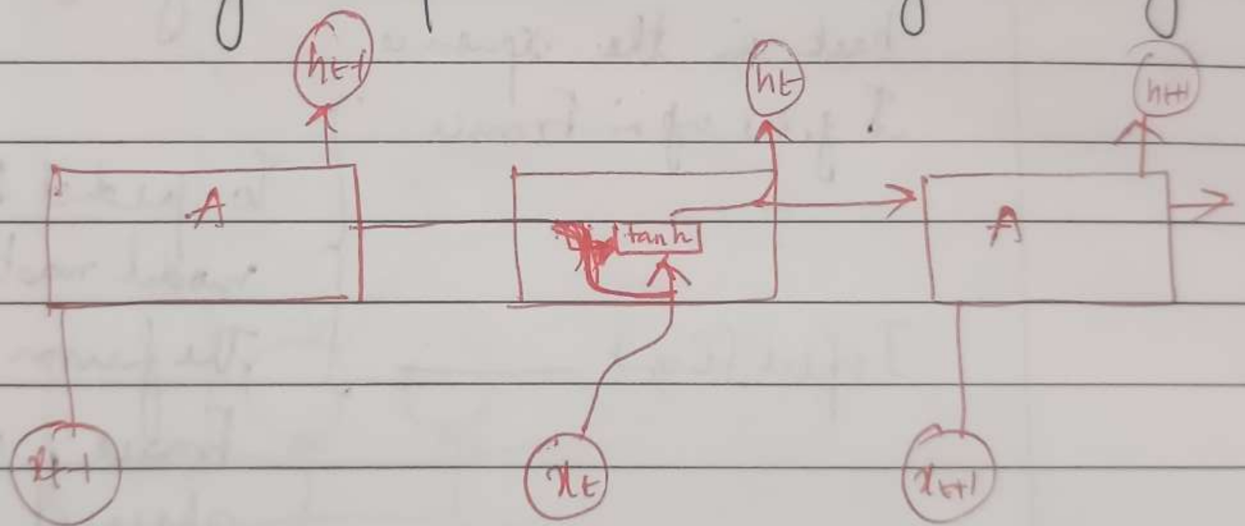
Limitation: – Dimension of context C can be too small to summarize a long sequence.



Shot on moto g54 5G
28 Jan 2026, 3:12 pm

# LSTM (Long Short Term Memory)

- special kind of RNN.
- made to remember info for long time periods and avoid the long term dependency problem.
- it can keep both short-term and long term memories easily.
- All recurrent neural n/w have the form of a chain of repeating modules of neural n/w.
- In std RNN this repeating module will have very simple structure i.e single tanh layer.




Each green box A is one cell or timestep in the n/w

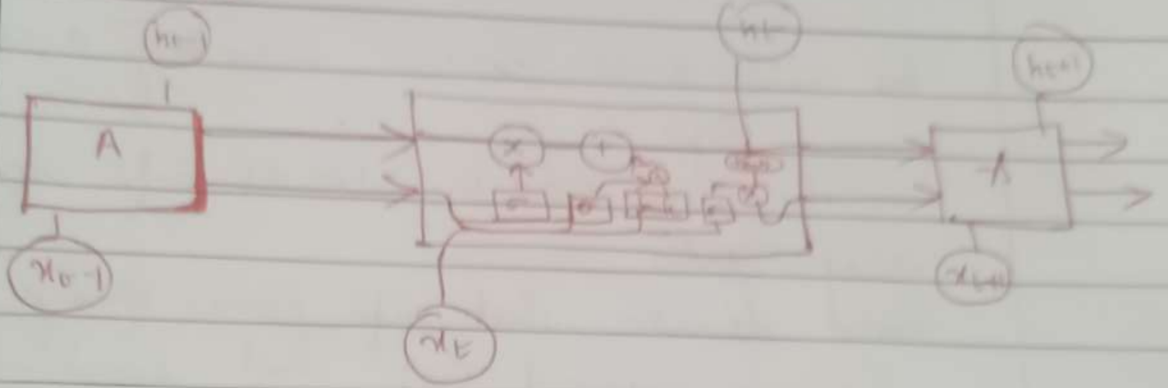
$x_t$ → i/p at each time step

$h_t$ → o/p or hidden states passed to the next step.

Simple RNN → box – one operation – usually tanh layer that updates the hidden state.


Shot on moto g54 5G
28 Jan 2026, 3:12 pm

LSTM Cell




+ × → math operations

σ tanh → neural n/w layers

black lines info flows

Takes input $x_t$ and previous hidden state $h_{t-1}$

Uses gate to decide,

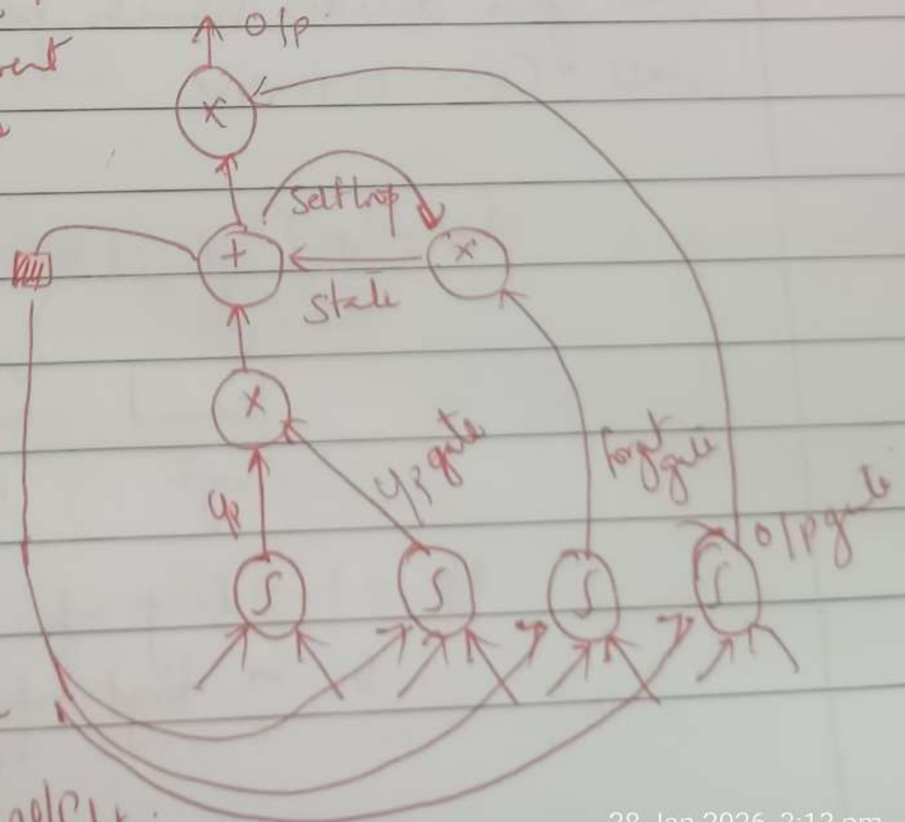
- what to forget (forget gate)
- what new info to add (i/p gate)
- what to o/p (o/p gate)

$C_t$ → updated, new $h_t$ is send to next st

i/p gate → controls how much new info from current i/p should enter the cell state.

Forget gate → decides what part of old info to erase from the cell state.

o/p gate → decides what part of the cell state should be sent as o/p $h_t$




Shot on moto g54 5G
28 Jan 2026, 3:13 pm

The cell state carries memory across time steps.

LSTM can add or remove info from it using gates.

Gates decides what info passes through. They use sigmoid ($\sigma$) and pointwise multiplication ($\times$) operations.

$\sigma$ — O/p value b/w 0 and 1,

0 → block everything

1 → let everything pass.

## Step by step LSTM Walkthrough.

### Forget gate.

This gate decides which part of the old cell state $C_{t-1}$ to remove.

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

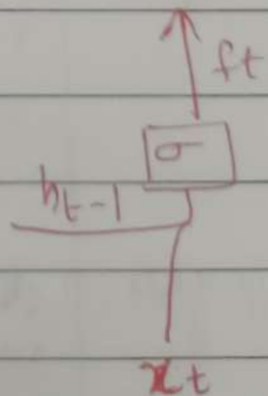
$W_f$ → learned wgts

$[h_{t-1}, x_t]$ → prev. hidden state.

$b_f$ → bias.

O/p = 1 → keep info.

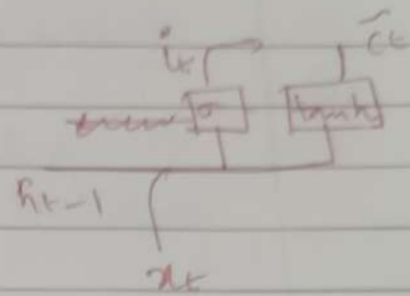
0 → forget info.




Shot on moto g54 5G
28 Jan 2026, 3:13 pm

___/___/___

## Input gate




## Input Gate Eqn

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

Candidate value Eqn.

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

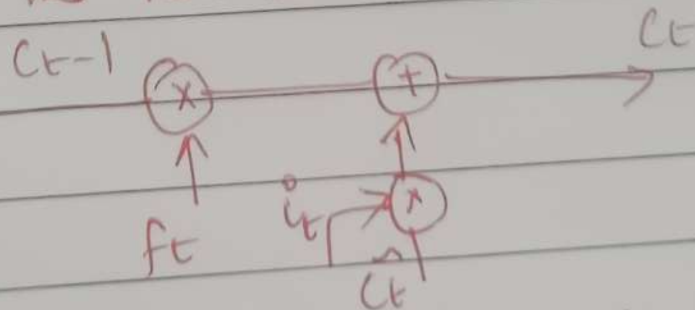
Creates new candidate memory values.

Combine both

$i_t * \tilde{C}_t$ → new update to the cell.

## Update Cell state.

We use the forget gate and i/p gate together to form the new cell state.



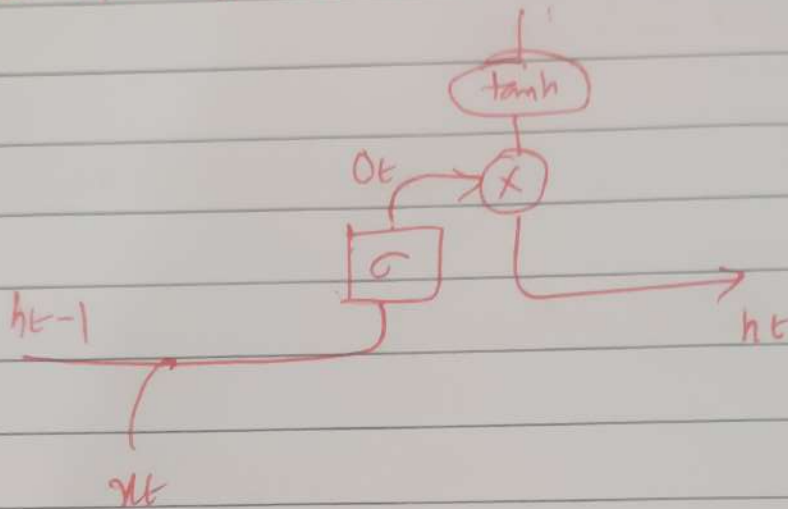

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Shot on moto g54 5G

28 Jan 2026, 3:13 pm

## Output gate

This gate decides what part of the cell state to o/p as hidden state ht.




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

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

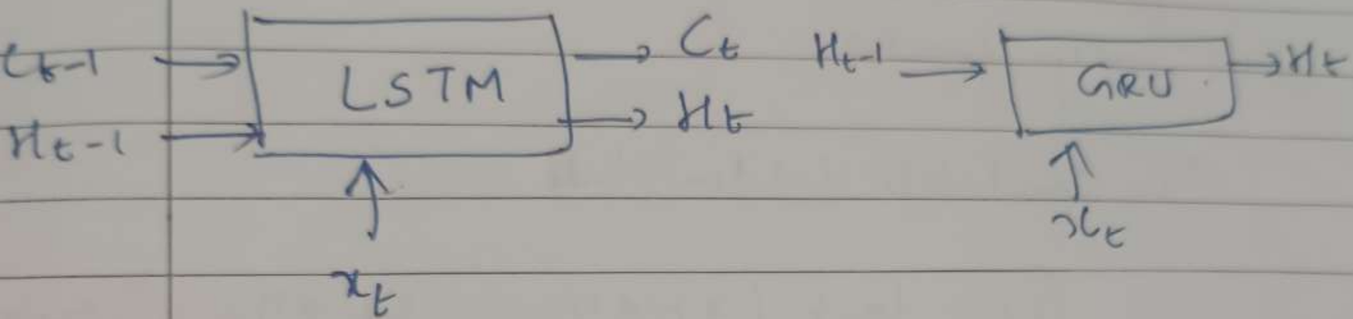
$O_t$ → how much of the memory should be visible outside.

$h_t$ – becomes the o/p of this time step and also goes to the next one.

Shot on moto g54 5G
28 Jan 2026, 3:13 pm

# GRU (Gated Recurrent Unit)

→ Similar to LSTM
- uses gates to control the flow of info.
- Simple architecture.




GRU doesnt have a seperate cell state.
Easier to train.

A timestamp t, it takes an i/p $x_t$ and hidden state $H_{t-1}$ from the previous timestamp. Later it o/p a new hidden state $H_t$ which again passed to the next timestamp.

- GRU has 2 gates Reset gate, Update gate

## Reset gate (Short term memory)

- is responsible for the short term memory of the n/w. (weights for reset gate.)

$$r_t = \sigma(x_t * U_r + H_{t-1} * W_r)$$

Shot on moto g54 5G

28 Jan 2026, 3:13 pm

## Update gate (Long term memory)

$$U_t = \sigma(x_t * U_u + H_{t-1} * W_u)$$

To find the hidden state $H_t$ it is a 2 step process.

① Candidate hidden state

$$\hat{H}_t = \tanh(x_t * U_g + (r_t * H_{t-1}) * W_g)$$

if $r_t = 1$ ⇒ entire info from the previous hidden state $H_{t-1}$ is considered.

$r_t = 0$ → info from the prev. hidden state is completely ignored.

Hidden state $H_t$

$u_t$ → update gate

$$H_t = u_t * H_{t-1} + (1 - u_t) * \hat{H}_t$$

$u_t = 0$ first term vanishes.

new hidden state will not have much

Shot on moto g54 5G
28 Jan 2026, 3:13 pm