Name: Prathamesh Arvind Jadhav

Deep Learning

1.Find out derivative for the sigmoid/ logistic activation function and tanh activation function. (Consider Mean squared error loss function and sigmoid function as activation function to find out derivative).

Derivatives of Sigmoid and Tanh Activation Functions (Considering MSE Loss Function)

1. Derivative of Sigmoid Activation Function

The **sigmoid function** is given by:

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

Step 1: Derivative of Sigmoid

Differentiating $\sigma(x)$:

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

Step 2: Derivative of MSE Loss with Sigmoid

The **Mean Squared Error (MSE) loss** is:

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

where $\hat{y} = \sigma(x)$. Differentiating w.r.t. x:

$$\frac{dL}{d\hat{y}} = \hat{y} - y$$

Applying chain rule:

$$\frac{dL}{dx} = (\hat{y}-y)\cdot\sigma(x)(1-\sigma(x))$$

Name: Prathamesh Arvind Jadhav

2. Derivative of tanh Activation Function

The **tanh function** is given by:

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

Step 1: Derivative of tanh

Differentiating $\tanh(x)$:

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$

Step 2: Derivative of MSE Loss with tanh

Using MSE loss:

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

where $\hat{y} = \tanh(x)$, differentiating w.r.t. x:

$$\frac{dL}{dx} = (\hat{y} - y) \cdot (1 - \tanh^2(x))$$

Name: Prathamesh Arvind Jadhav

Final Answers

1. Sigmoid gradient with MSE Loss:

$$\frac{dL}{dx} = (\sigma(x) - y) \cdot \sigma(x)(1 - \sigma(x))$$

2. Tanh gradient with MSE Loss:

$$\frac{dL}{dx} = (\tanh(x) - y) \cdot (1 - \tanh^2(x))$$

2.Explain backprapogation algorithm in detail.

Backpropagation Algorithm - Detailed Explanation

Backpropagation (Backprop) is a supervised learning algorithm used in training artificial neural networks. It efficiently updates the weights of the network by minimizing the error using **gradient descent**.

---

1. Steps of the Backpropagation Algorithm

Step 1: Forward Propagation

1. Inputs are passed to the network.
2. Each neuron computes a weighted sum of its inputs and applies an activation function.
3. The output of each layer is passed to the next layer until the final output is obtained.

Name: Prathamesh Arvind Jadhav

Mathematically, for a neuron:

$$z = W \cdot X + b$$

where:

- W = weights,
- X = input,
- b = bias,
- $a = f(z)$ (activation function applied).

Step 2: Compute the Loss Function

The **error/loss** is computed using a loss function (e.g., Mean Squared Error for regression, Cross-Entropy for classification).

For **Mean Squared Error (MSE)**:

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

where:

- y = actual output,
- $\hat{y}$ = predicted output.

Name: Prathamesh Arvind Jadhav

Step 3: Backward Propagation (Gradient Calculation)

To minimize the loss, we compute the **gradients** using **chain rule of differentiation**:

(a) Compute Error at the Output Layer

$$\frac{dL}{d\hat{y}} = \hat{y} - y$$

(b) Compute Gradient for Weights

Using chain rule:

$$\frac{dL}{dW} = \frac{dL}{d\hat{y}} \cdot \frac{d\hat{y}}{dz} \cdot \frac{dz}{dW}$$

For **sigmoid activation** $\sigma(x)$:

$$\frac{d\hat{y}}{dz} = \sigma(x)(1 - \sigma(x))$$

For **tanh activation**:

$$\frac{d\hat{y}}{dz} = 1 - \tanh^2(x)$$

Step 4: Weight Update (Gradient Descent)

Weights are updated using **gradient descent**:

$$W = W - \eta \cdot \frac{dL}{dW}$$

where η (learning rate) controls the step size.

3. Summary of Backpropagation Steps

1. **Forward Pass:** Compute output using weights and activation functions.
2. **Compute Loss:** Calculate error using a loss function.
3. **Backward Pass:**
    - Compute error at the output layer.
    - Propagate errors back through the layers using gradients.
4. **Weight Update:** Adjust weights using gradient descent.

---

4. Importance of Backpropagation

- **Efficient Training:** Reduces computation by reusing gradients.
- **Optimization:** Helps in minimizing loss function efficiently.
- **Foundation of Deep Learning:** Used in training deep neural networks.

3.Explain loss function used for classification and regression problems.

Loss Functions for Classification and Regression Problems

Loss functions measure the difference between the actual and predicted values in machine learning models. They guide the optimization process by minimizing errors during training.

---

1. Loss Functions for Regression Problems

Regression involves predicting continuous values (e.g., house prices, temperature). The commonly used loss functions are:

(a) Mean Squared Error (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Measures the average squared difference between actual (y) and predicted ($\hat{y}$) values.
- Penalizes large errors more than small errors.
- Used in linear regression and deep learning.

(b) Mean Absolute Error (MAE)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- Measures the average absolute difference.
- Less sensitive to outliers than MSE.
- Used when minimizing large deviations is important.

(c) Huber Loss

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2, & \text{for } |a| \le \delta \\ \delta(|a| - \frac{1}{2}\delta), & \text{for } |a| > \delta \end{cases}$$

where $a = y - \hat{y}$.

- A combination of MSE (for small errors) and MAE (for large errors).
- Robust to outliers.

(d) Log-Cosh Loss

$$L = \sum \log(\cosh(\hat{y} - y))$$

- Similar to MSE but less sensitive to outliers.
- Used in robust regression problems.

2. Loss Functions for Classification Problems

Classification involves predicting categorical labels (e.g., spam or not spam). The common loss functions are:

Name: Prathamesh Arvind Jadhav

(a) Binary Cross-Entropy (Log Loss)

$$L = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

- Used for **binary classification** (e.g., Yes/No).
- Penalizes incorrect predictions more when confidence is high.
- Used in logistic regression and neural networks.

(b) Categorical Cross-Entropy

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

- Used for **multi-class classification** (e.g., classifying images into multiple categories).
- Requires softmax activation in the output layer.

4.With the help of neat sketch illustrate feed forward neural network.

Feedforward Neural Network (FNN)

A **Feedforward Neural Network (FNN)** is the simplest type of artificial neural network where information moves in **one direction**—from the **input layer** to the **output layer** through **hidden layers** (if any), without any cycles or loops.

Structure of Feedforward Neural Network

A Feedforward Neural Network consists of the following components:

1. **Input Layer**:
   - Takes input features x1,x2,…,xn.
   - Passes them to the next layer through weighted connections.
2. **Hidden Layer(s)**:

- Consists of neurons that apply weights and activation functions.
- Transforms input into a more complex representation.

3. **Output Layer**:
   - Produces the final prediction (classification or regression).
   - Uses activation functions like **Softmax** (for classification) or **Linear** (for regression).

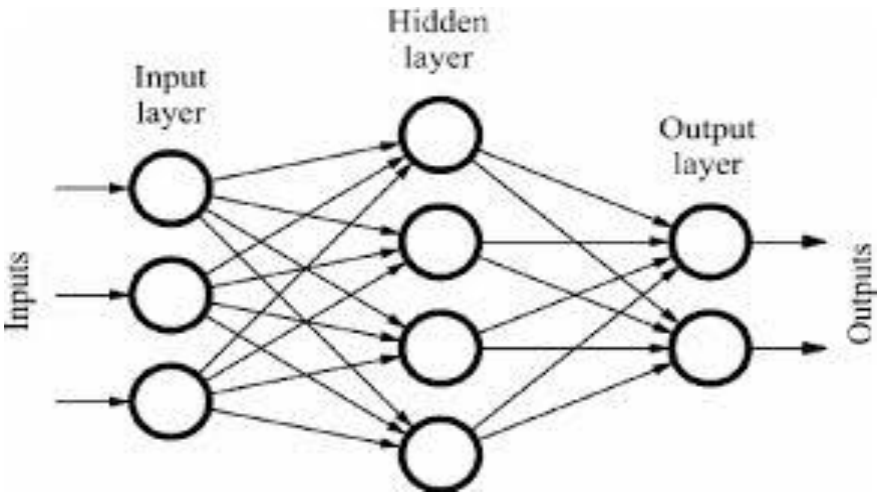
Mathematical Representation

Each neuron computes:

$$z = W \cdot X + b$$

where:

- W = Weights,
- X = Inputs,
- b = Bias,
- $a = f(z)$ (activation function).



Working of Feedforward Neural Network

1. **Forward Propagation**: Inputs are multiplied with weights, activation functions are applied, and results are passed to the next layer.
2. **Prediction**: The final output is generated at the output layer.

3. **Training (Backpropagation)**: The error is calculated using a loss function, and weights are updated using **gradient descent**.

Key Features of Feedforward Neural Network

- No loops or cycles (unlike recurrent networks).
- Used in image recognition, text classification, and more.
- Can be single-layer or multi-layer (MLP - Multi-Layer Perceptron).

5.List and explain any 4 activation functions in deep learning.

Activation Functions in Deep Learning

Activation functions introduce **non-linearity** in neural networks, allowing them to learn complex patterns. They decide whether a neuron should be activated based on the weighted sum of inputs.

1. Sigmoid Activation Function

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

Properties:

- **Range**: $(0, 1)$
- **Use Case**: Binary classification
- **Advantage**: Smooth gradient, easy to differentiate
- **Disadvantage**: Causes **vanishing gradient problem**, slow learning for large/small values

Graph:

- S-shaped curve
- Converts any input into a probability-like value between 0 and 1.

Name: Prathamesh Arvind Jadhav

2. Tanh (Hyperbolic Tangent) Activation Function

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

Properties:

- **Range**: $(-1, 1)$
- **Use Case**: Used in hidden layers of neural networks
- **Advantage**: Centered around 0, better than Sigmoid
- **Disadvantage**: Still suffers from vanishing gradient problem

Graph:

- S-shaped curve like Sigmoid, but symmetric around 0.

3. ReLU (Rectified Linear Unit) Activation Function

$$f(x) = \max(0, x)$$

Properties:

- **Range**: $[0, \infty)$
- **Use Case**: Most commonly used in hidden layers of deep networks
- **Advantage**: Avoids vanishing gradient, computationally efficient
- **Disadvantage**: **Dying ReLU problem** (neurons stuck at 0 for negative inputs)

Graph:

- Linear for $x > 0$, zero for $x \leq 0$.

Name: Prathamesh Arvind Jadhav

4. Softmax Activation Function

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Properties:

- **Range:** $(0, 1)$
- **Use Case:** Multi-class classification (output layer)
- **Advantage:** Converts logits into probabilities
- **Disadvantage:** Computationally expensive

Graph:

- Outputs a probability distribution summing to 1.

6.Explain Gradient descent algorithm in detail. Also derive expression for the ∇w and ∇b for the gradient descent algorithm for regression.

Gradient Descent Algorithm

Gradient Descent is an **optimization algorithm** used to minimize the loss function by iteratively updating the model parameters (**weights w and bias b**). It helps machine learning models learn by reducing the error between predicted and actual values.

1. Working of Gradient Descent

1. **Initialize weights** (w) **and bias** (b) **randomly.**
2. **Compute the loss function** $J(w, b)$.
3. **Calculate gradients** ∇w and ∇b (partial derivatives of loss function).
4. **Update the parameters** using the learning rate α:

$$w = w - \alpha \cdot \nabla w$$

$$b = b - \alpha \cdot \nabla b$$

5. **Repeat until convergence** (i.e., loss is minimized).

Name: Prathamesh Arvind Jadhav

2. Types of Gradient Descent

- **Batch Gradient Descent**: Uses the entire dataset for one update.
- **Stochastic Gradient Descent (SGD)**: Updates using one sample at a time.
- **Mini-Batch Gradient Descent**: Uses a small batch of data for updates.

3. Derivation of Gradient Descent for Regression

Consider **Linear Regression**:

$$\hat{y} = wX + b$$

Loss Function (Mean Squared Error - MSE)

$$J(w, b) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where:

- y_i = actual value,
- $\hat{y}_i = wX_i + b$ = predicted value,
- n = number of samples.

To update w **and** b, we compute the gradients:

Gradient w.r.t w

$$\nabla w = \frac{\partial J}{\partial w} = \frac{2}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)(-X_i)$$

$$\nabla w = -\frac{2}{n} \sum (y - (wX + b))X$$

Name: Prathamesh Arvind Jadhav

Gradient w.r.t b

$$\nabla b = \frac{\partial J}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)(-1)$$

$$\nabla b = -\frac{2}{n}\sum(y - (wX + b))$$

4. Update Equations

Using **learning rate** α, update the parameters:

$$w = w - \alpha \cdot \left(-\frac{2}{n}\sum(y - (wX + b))X\right)$$

$$b = b - \alpha \cdot \left(-\frac{2}{n}\sum(y - (wX + b))\right)$$

7.With the help of proper example and neat sketch elaborate loss function for regression problem and loss function for classification problem. (For regression consider movie recommendation system example and for classification problem consider image classification example)

Loss Functions for Regression and Classification

Loss functions measure the difference between the **actual output** and the **predicted output** of a machine learning model. They guide optimization algorithms like **Gradient Descent** to minimize errors.

Name: Prathamesh Arvind Jadhav

1. Loss Functions for Regression (Movie Recommendation Example)

Regression problems predict **continuous values**. Example: Predicting a **movie rating** on a scale of 1 to 5.

Example Scenario (Movie Recommendation System)

- Suppose a movie recommendation system predicts the rating a user might give to a movie.
- **Actual rating** (y) = 4.5
- **Predicted rating** ($\hat{y}$) = 3.8
- The loss function calculates the error between **4.5** and **3.8**.

Common Regression Loss Functions

1. **Mean Squared Error (MSE)**

$$J(w,b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

   - **Penalizes larger errors more** (since squared term increases error effect).
   - **Smooth and differentiable**, making it ideal for gradient descent.

2. **Mean Absolute Error (MAE)**

$$J(w,b) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

   - **Treats all errors equally** by taking the absolute difference.
   - More **robust to outliers** than MSE.

- **MSE** is a quadratic curve (increases rapidly for larger errors).
- **MAE** is a linear function (constant slope).

---

2. Loss Functions for Classification (Image Classification Example)

Classification problems predict **discrete categories**. Example: Identifying whether an image contains a **cat or dog**.

Example Scenario (Image Classification System)

- Given an image, a model predicts:
  - Cat (Class 1, Probability = 0.8)
  - Dog (Class 2, Probability = 0.2)
- The true label is **Cat (1)**, but the model assigns a probability of **0.8**.
- The loss function calculates how far the predicted probability is from 1.

Name: Prathamesh Arvind Jadhav

Common Classification Loss Functions

1. **Binary Cross-Entropy (for two classes, e.g., Cat vs. Dog)**

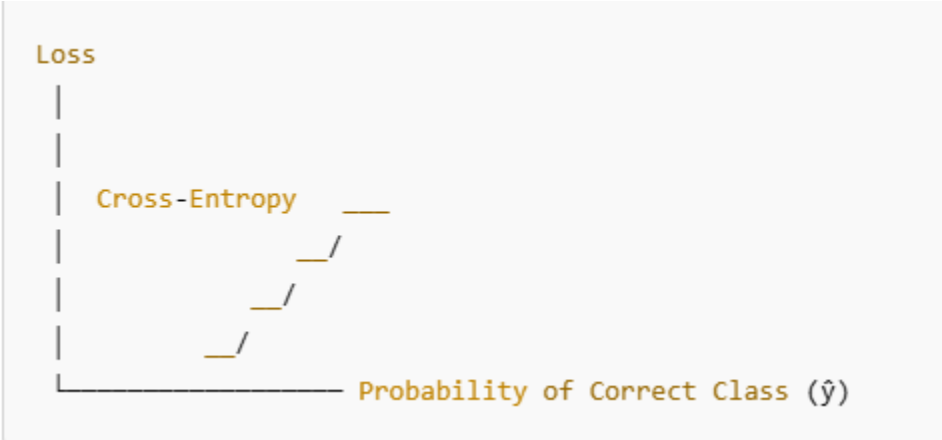
$$J = -\frac{1}{n}\sum [y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$$

- **Used for binary classification** (0 or 1).
- **Higher confidence correct predictions = lower loss.**

2. **Categorical Cross-Entropy (for multiple classes, e.g., Dog, Cat, Bird, etc.)**

$$J = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

- **Used for multi-class classification** (Softmax output).
- Encourages higher probability for the correct class.

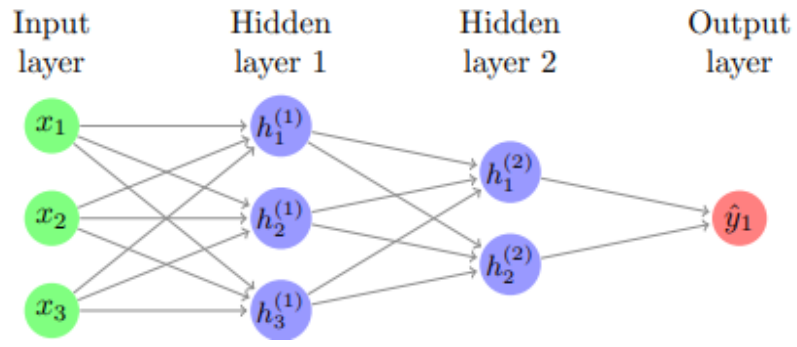


- **Loss is high when the predicted probability is low**.
- **Loss is low when the predicted probability is high**.

Name: Prathamesh Arvind Jadhav

8.Use the following data to answer the questions below:

The following diagram represents a neural network containing two hidden layers and one output layer. The input to the network is a column vector x € R3 . The activation function used in hidden layers is sigmoid. The output layer doesn't contain any activation function and the loss used is squared error loss (predy − truey) 2 .



The following network doesn't contain any biases and the weights of the network are given below:

$$\mathbf{W1}=\begin{bmatrix}1 & 1 & 2\\ 3 & 1 & 1\\ 1 & 2 & 3\end{bmatrix}\quad \mathbf{W2}=\begin{bmatrix}1 & 1 & 2\\ 3 & 1 & 1\end{bmatrix}\quad \mathbf{W3}=\begin{bmatrix}2 & 5\end{bmatrix}$$

The input to the network is: $\mathbf{x}=\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}$

The target value y is: $\mathbf{y}=10$

a. What is the total number of parameters in the following network?
b. What is the predicted output for the given input x1 after doing the forward pass?
c. Compute and enter the loss between the output generated by input x and the true output y.
d. If we call the predicted y as ˆy (y hat) then what is the gradient dL/dyˆ? (L is the loss function).
e. What is the sum of elements of ∇w3?

Name: Prathamesh Arvind Jadhav

Step 1: Total Number of Parameters

The total number of parameters in a neural network is given by the total number of weights (since there are no biases in this case).

- Weight matrix W_1 (Input to Hidden Layer 1)
  W_1 has 3 rows and 3 columns → $3 \times 3 = 9$ parameters
- Weight matrix W_2 (Hidden Layer 1 to Hidden Layer 2)
  W_2 has 2 rows and 3 columns → $2 \times 3 = 6$ parameters
- Weight matrix W_3 (Hidden Layer 2 to Output Layer)
  W_3 has 1 row and 2 columns → $1 \times 2 = 2$ parameters

Total Parameters

$$9 + 6 + 2 = 17$$

Step 2: Forward Pass (Computing $\hat{y}$)

Given:

$$W_1 = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 1 & 1 \end{bmatrix}, \quad W_3 = \begin{bmatrix} 2 & 5 \end{bmatrix}$$

$$x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Computing Hidden Layer 1 Output

$$z^{(1)} = W_1 x = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} (1 \cdot 1 + 1 \cdot 1 + 2 \cdot 1) \\ (3 \cdot 1 + 1 \cdot 1 + 1 \cdot 1) \\ (1 \cdot 1 + 2 \cdot 1 + 3 \cdot 1) \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

Name: Prathamesh Arvind Jadhav

Applying **sigmoid activation function:**

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

$$h^{(1)} = \begin{bmatrix} \sigma(4) \\ \sigma(5) \\ \sigma(6) \end{bmatrix} = \begin{bmatrix} \frac{1}{1+e^{-4}} \\ \frac{1}{1+e^{-5}} \\ \frac{1}{1+e^{-6}} \end{bmatrix}$$

Approximating sigmoid values:

$$\sigma(4) \approx 0.982, \quad \sigma(5) \approx 0.993, \quad \sigma(6) \approx 0.997$$

$$h^{(1)} \approx \begin{bmatrix} 0.982 \\ 0.993 \\ 0.997 \end{bmatrix}$$

Computing Hidden Layer 2 Output

$$z^{(2)} = W_2 h^{(1)} = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0.982 \\ 0.993 \\ 0.997 \end{bmatrix}$$

$$= \begin{bmatrix} (1 \times 0.982 + 1 \times 0.993 + 2 \times 0.997) \\ (3 \times 0.982 + 1 \times 0.993 + 1 \times 0.997) \end{bmatrix}$$

$$= \begin{bmatrix} 0.982 + 0.993 + 1.994 \\ 2.946 + 0.993 + 0.997 \end{bmatrix} = \begin{bmatrix} 3.969 \\ 4.936 \end{bmatrix}$$

Applying **sigmoid activation:**

$$h^{(2)} = \sigma(z^{(2)}) = \begin{bmatrix} \sigma(3.969) \\ \sigma(4.936) \end{bmatrix}$$

Name: Prathamesh Arvind Jadhav

Approximating sigmoid:

$$\sigma(3.969) \approx 0.981, \quad \sigma(4.936) \approx 0.992$$

$$h^{(2)} \approx \begin{bmatrix} 0.981 \\ 0.992 \end{bmatrix}$$

Computing Final Output (No Activation Function on Output Layer)

$$\hat{y} = W_3 h^{(2)} = \begin{bmatrix} 2 & 5 \end{bmatrix} \times \begin{bmatrix} 0.981 \\ 0.992 \end{bmatrix}$$

$$= (2 \times 0.981) + (5 \times 0.992)$$

$$= 1.962 + 4.960 = 6.922$$

Step 3: Compute Loss

Loss function = Squared Error Loss:

$$L = (y - \hat{y})^2 = (10 - 6.922)^2$$

$$= (3.078)^2 = 9.474$$

Step 4: Compute $\frac{dL}{d\hat{y}}$

$$\frac{dL}{d\hat{y}} = 2(\hat{y} - y)$$

$$= 2(6.922 - 10) = 2 \times (-3.078) = -6.156$$

Step 5: Compute Sum of Elements of ∇W_3

The gradient of loss with respect to W_3 is:

$$\nabla W_3 = \frac{dL}{d\hat{y}} \cdot h^{(2)}$$

$$= (-6.156) \times \begin{bmatrix} 0.981 \\ 0.992 \end{bmatrix}$$

$$= \begin{bmatrix} -6.038 \\ -6.107 \end{bmatrix}$$

Sum of elements:

$$-6.038 + (-6.107) = -12.145$$

9.Use Multilayer perceptron to solve XOR gate problem which is non linearly separable. Also explain in detail multilayer perceptron.

Multilayer Perceptron (MLP) and XOR Problem Solution

Introduction to Multilayer Perceptron (MLP)

A **Multilayer Perceptron (MLP)** is a type of feedforward artificial neural network (ANN) that consists of three types of layers:

1. **Input Layer** – Receives the input features.
2. **Hidden Layer(s)** – Contains neurons that apply activation functions to learn complex patterns.
3. **Output Layer** – Produces the final result of the network.

Each neuron in an MLP is connected to neurons in the next layer using **weights**, and each neuron applies an **activation function** (like sigmoid, ReLU, or tanh). MLPs are trained using **backpropagation** and **gradient descent** to minimize the loss function.

Why MLP is Needed for XOR Problem?

Name: Prathamesh Arvind Jadhav

- The **XOR gate** (Exclusive OR) is a **non-linearly separable** problem, meaning it **cannot** be solved using a single-layer perceptron.
- A **single-layer perceptron** can only solve linearly separable problems (like AND and OR gates), but XOR requires at least one hidden layer to learn a **nonlinear decision boundary**.
- **MLP with one hidden layer** can solve the XOR problem by learning a transformation that makes XOR linearly separable.

XOR Truth Table

Input x_1	Input x_2	XOR Output y
0	0	0
0	1	1
1	0	1
1	1	0

MLP Architecture for XOR

The MLP consists of:

- **Input layer** with 2 neurons (for x_1 and x_2).
- **One hidden layer** with 2 neurons (to learn nonlinear transformation).
- **Output layer** with 1 neuron (final XOR output).
- **Activation function:** Sigmoid or ReLU.

Step-by-Step Solution Using MLP

Name: Prathamesh Arvind Jadhav

Step 1: Define Weights and Biases

Assume random weights and biases:

Hidden Layer Weights and Biases

$$W_1 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

$$b_1 = \begin{bmatrix} -0.5 \\ 1.5 \end{bmatrix}$$

Output Layer Weights and Bias

$$W_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad b_2 = -0.5$$

Step 2: Compute Hidden Layer Outputs

Using sigmoid activation function:

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

For each input (x_1, x_2), compute:

$$h = \sigma(W_1 \cdot x + b_1)$$

For example, when $x = (0, 0)$:

$$h_1 = \sigma(0.5(0) + 0.5(0) - 0.5) = \sigma(-0.5) \approx 0.377$$

$$h_2 = \sigma(0.5(0) + 0.5(0) + 1.5) = \sigma(1.5) \approx 0.817$$

Name: Prathamesh Arvind Jadhav

Step 3: Compute Output Layer

$$\hat{y} = \sigma(W_2 \cdot h + b_2)$$

For $x = (0, 0)$:

$$\hat{y} = \sigma(1(0.377) + (-1)(0.817) - 0.5)$$

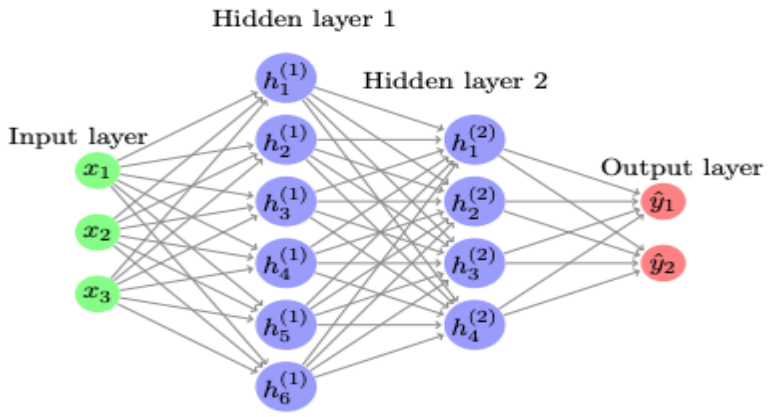
$$= \sigma(0.377 - 0.817 - 0.5) = \sigma(-0.94) \approx 0.28$$

We apply similar calculations for all inputs.

Step 4: Training the MLP

1. **Initialize weights randomly**.
2. **Perform forward pass**: Compute outputs using weighted sums and activation functions.
3. **Compute loss** using Mean Squared Error (MSE) or Cross-Entropy Loss.
4. **Backpropagate errors**: Compute gradients using chain rule.
5. **Update weights** using **Gradient Descent**.
6. Repeat until error is minimized.

10. The diagram below shows a neural network. The network contains two hidden layers and one output layer. The input to the network is a column vector x∈R3. The first hidden layer contains 6 neurons, the second hidden layer contains 4 neurons and the output layer contains 2 neurons. Each neuron in the lth layer is connected to all the neurons in the (l+1)th layer. Each neuron has a bias connected to it (not explicitly shown in the figure).



Name: Prathamesh Arvind Jadhav

In the diagram, W1 is a matrix and x,a1,h1 and O are all column vectors. The notation Wi[j,:] denotes the jth row of the matrix Wi, Wi[:,j] denotes the jth column of the matrix Wi and Wkij denotes an element at ith row and jth column of the matrix Wk

1) Choose the correct dimensions of W1 and a1

2) How many learnable parameters(including bias) are there in the network?

3) Which of the loss functions can be used for the classification problem?

1) Choose the correct dimensions of W_1 and a_1

- The **input layer** has **3 neurons.**
- The **first hidden layer** has **6 neurons.**
- The **weight matrix** W_1 connects the **input layer** to the **first hidden layer.**
  - Each of the **6 neurons** in the first hidden layer receives input from all **3 neurons** in the input layer.
  - Thus, W_1 **has dimensions:**

$$W_1 \in \mathbb{R}^{6\times 3}$$

  - The output after applying activation is a_1, which contains activations of 6 neurons.
  - So, the **dimension of** a_1 is:

$$a_1 \in \mathbb{R}^{6\times 1}$$

Name: Prathamesh Arvind Jadhav

2) Total number of learnable parameters (including bias)

A learnable parameter includes **weights and biases** in all layers.

First Hidden Layer:

- Weight matrix W_1:

$$W_1 \in \mathbb{R}^{6\times 3} \Rightarrow 6 \times 3 = 18 \text{ parameters}$$

- Bias vector b_1 (one bias per neuron in hidden layer 1):

$$b_1 \in \mathbb{R}^{6\times 1} \Rightarrow 6 \text{ parameters}$$

- Total for first hidden layer:

$$18 + 6 = 24$$

Second Hidden Layer:

- Weight matrix W_2:

$$W_2 \in \mathbb{R}^{4\times 6} \Rightarrow 4 \times 6 = 24 \text{ parameters}$$

- Bias vector b_2:

$$b_2 \in \mathbb{R}^{4\times 1} \Rightarrow 4 \text{ parameters}$$

- Total for second hidden layer:

$$24 + 4 = 28$$

Name: Prathamesh Arvind Jadhav

Output Layer:

- Weight matrix W_3:

$$W_3 \in \mathbb{R}^{2\times 4} \Rightarrow 2 \times 4 = 8 \text{ parameters}$$

- Bias vector b_3:

$$b_3 \in \mathbb{R}^{2\times 1} \Rightarrow 2 \text{ parameters}$$

- Total for output layer:

$$8 + 2 = 10$$

Final Total Learnable Parameters:

$$24 + 28 + 10 = 62$$

3) Loss functions for classification problems

For classification, the commonly used loss functions are:

1. **Cross-Entropy Loss (Categorical Cross-Entropy or Binary Cross-Entropy)**
   - Used when the outputs represent probabilities (e.g., softmax or sigmoid activation in the output layer).
   - Formula for **Binary Cross-Entropy** (for two classes):

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

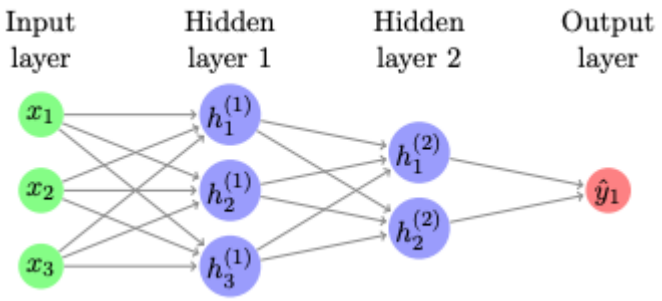
   - Formula for **Categorical Cross-Entropy** (for multiple classes):

$$L = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}\log(\hat{y}_{i,j})$$

11.The following diagram represents a neural network containing two hidden layers and one output layer. The input to the network is a column vector x∈R3. The activation function used in hidden layers is sigmoid. The

Name: Prathamesh Arvind Jadhav

output layer doesn't contain any activation function and the loss used is squared errorloss (predy−truey)2.



1. The following network doesn't contain any biases and the weights of the network are given below:

$$\mathbf{W_1} = \begin{bmatrix} 1 & 1 & 3 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \quad \mathbf{W_2} = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix} \quad \mathbf{W_3} = \begin{bmatrix} 2 & 3 \end{bmatrix}$$

The input to the network is: $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

The target value y is: y=8

What is the total number of parameters in the given network?

What is the predicted output for the given input x after doing the forward pass?

Also Compute and enter the loss between the output generated by input x and the true output y.

Step 1: Compute the Total Number of Parameters

Since there are no biases in the network, the total number of parameters is the sum of all weight elements.

- W1 is a 3×3 matrix → 9 parameters
- W2 is a 2×3 matrix → 6 parameters
- W3 is a 1×2 matrix → 2 parameters

$$\text{Total parameters} = 9 + 6 + 2 = 17$$

Name: Prathamesh Arvind Jadhav

Step 2: Compute the Forward Pass

Compute First Hidden Layer Activations

The input vector x is:

$$x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

The first hidden layer activations (before applying sigmoid):

$$h_1^{(1)} = W_1 \cdot x$$

$$= \begin{bmatrix} 1 & 1 & 3 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} (1 \times 1 + 1 \times 1 + 3 \times 1) \\ (2 \times 1 + 1 \times 1 + 1 \times 1) \\ (1 \times 1 + 2 \times 1 + 3 \times 1) \end{bmatrix}$$

$$= \begin{bmatrix} 5 \\ 4 \\ 6 \end{bmatrix}$$

Applying the **sigmoid activation function**:

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

$$h_1^{(1)} = \sigma\left(\begin{bmatrix} 5 \\ 4 \\ 6 \end{bmatrix}\right)$$

$$= \begin{bmatrix} \frac{1}{1+e^{-5}} \\ \frac{1}{1+e^{-4}} \\ \frac{1}{1+e^{-6}} \end{bmatrix}$$

Approximating sigmoid values:

$$\approx \begin{bmatrix} 0.9933 \\ 0.9820 \\ 0.9975 \end{bmatrix}$$

Name: Prathamesh Arvind Jadhav

Compute Second Hidden Layer Activations

$$h_1^{(2)} = W_2 \cdot h_1^{(1)}$$

$$= \begin{bmatrix} 1 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.9933 \\ 0.9820 \\ 0.9975 \end{bmatrix}$$

$$= \begin{bmatrix} (1 \times 0.9933 + 1 \times 0.9820 + 2 \times 0.9975) \\ (3 \times 0.9933 + 2 \times 0.9820 + 1 \times 0.9975) \end{bmatrix}$$

$$= \begin{bmatrix} (0.9933 + 0.9820 + 1.9950) \\ (2.9799 + 1.9640 + 0.9975) \end{bmatrix}$$

$$= \begin{bmatrix} 3.9703 \\ 5.9414 \end{bmatrix}$$

Applying **sigmoid activation**:

$$h_1^{(2)} = \sigma\left(\begin{bmatrix} 3.9703 \\ 5.9414 \end{bmatrix}\right)$$

$$\approx \begin{bmatrix} 0.9815 \\ 0.9974 \end{bmatrix}$$

Compute Output Layer

$$\hat{y} = W_3 \cdot h_1^{(2)}$$

$$= \begin{bmatrix} 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 0.9815 \\ 0.9974 \end{bmatrix}$$

$$= (2 \times 0.9815 + 3 \times 0.9974)$$

$$= 1.9630 + 2.9922 = 4.9552$$

So, the **predicted output**:

$$\hat{y} \approx 4.96$$

Name: Prathamesh Arvind Jadhav

Step 3: Compute the Loss

The loss function used is **squared error loss**:

$$L = (y - \hat{y})^2$$
$$= (8 - 4.96)^2$$
$$= (3.04)^2$$
$$= 9.2416$$

So, the **loss is**:

$$L \approx 9.24$$

12.With the help of proper example, elaborate basic structure of convolutional neural network with respect to Padding, strides, the Relu layer and pooling.

Basic Structure of a Convolutional Neural Network (CNN)

A **Convolutional Neural Network (CNN)** consists of multiple layers that process and extract important features from an image. The fundamental components of a CNN include **Convolutional layers, Activation functions (ReLU), Padding, Strides, and Pooling layers**. Below is a structured explanation with examples.

---

1. Convolutional Layer

This layer applies **filters (kernels)** to extract features such as edges, textures, and shapes from an input image.

Name: Prathamesh Arvind Jadhav

Example:

Consider a **5×5 input image** and a **3×3 filter** (kernel). The filter slides over the image, performing an element-wise multiplication and summing up the values.

$$\begin{bmatrix} 1 & 2 & 3 & 0 & 1 \\ 4 & 5 & 6 & 1 & 0 \\ 7 & 8 & 9 & 2 & 1 \\ 1 & 2 & 3 & 0 & 1 \\ 4 & 5 & 6 & 1 & 0 \end{bmatrix}$$

$*$

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

The result is a **feature map** (smaller matrix) that highlights important features.

2. Padding

Padding adds extra rows/columns around the input image to maintain the spatial size after convolution.

Types of Padding:

1. **Valid Padding (No Padding, Output size decreases)**
2. **Same Padding (Zero Padding, Output size remains the same)**

Example:

For a **5×5 input** with a **3×3 filter**:

- **Without Padding (Valid Padding) → Output: 3×3**
- **With Padding (Same Padding, 1-pixel border) → Output: 5×5**

3. Strides

Stride refers to how much the filter moves at each step.

- **Stride = 1** (default) → Moves **one pixel** at a time.
- **Stride = 2** → Moves **two pixels**, reducing output size.

Example:

For a **5×5 image** with a **3×3 filter**:

- **Stride = 1** → Output: **3×3**
- **Stride = 2** → Output: **2×2**

4. ReLU (Rectified Linear Unit) Activation Function

ReLU applies **non-linearity** by converting negative values to **zero** and keeping positive values unchanged.

Example:

$$\begin{bmatrix} -1 & 2 & -3 \\ 4 & -5 & 6 \\ -7 & 8 & -9 \end{bmatrix}$$

After ReLU:

$$\begin{bmatrix} 0 & 2 & 0 \\ 4 & 0 & 6 \\ 0 & 8 & 0 \end{bmatrix}$$

5. Pooling Layer

Pooling reduces the **dimensions** of the feature map while retaining important information.

Types of Pooling:

1. **Max Pooling** → Takes the maximum value in a window.

2. **Average Pooling** → Takes the average value.

Example (Max Pooling 2×2 with Stride 2):

Input:

$$\begin{bmatrix} 1 & 3 & 2 & 4 \\ 5 & 6 & 8 & 7 \\ 3 & 2 & 0 & 1 \\ 9 & 7 & 5 & 4 \end{bmatrix}$$

After 2×2 Max Pooling:

$$\begin{bmatrix} 6 & 8 \\ 9 & 5 \end{bmatrix}$$

13 Illustrate the steps in the training the convolutional neural network with the help of image classification example.

Training a Convolutional Neural Network (CNN) for Image Classification

Training a CNN involves several steps, from preprocessing the dataset to making predictions. Let's illustrate these steps using an **image classification** example where we classify images of **cats and dogs**.

1. Data Collection and Preprocessing

Before training, we collect a dataset (e.g., **Cats vs. Dogs dataset**) and preprocess it.

- **Dataset:** A collection of labeled images (e.g., `cat1.jpg`, `dog1.jpg`).
- **Preprocessing:**
  - Resize images to a fixed size (e.g., **64×64 pixels**).
  - Normalize pixel values (scale them between **0 and 1**).
  - Convert images into numerical arrays.

2. Model Architecture (CNN Design)

A CNN consists of convolutional layers, activation functions, pooling layers, and fully connected layers.

- **Convolution Layer:** Extracts features.
- **ReLU Activation:** Introduces non-linearity.
- **MaxPooling:** Reduces dimensions.
- **Flatten Layer:** Converts features into a 1D vector.
- **Fully Connected Layer:** Classifies the image.

3. Training the CNN Model

The model is trained using labeled images.

- **Epochs:** Number of training iterations.
- **Loss Function:** Measures classification error.
- **Optimizer (Adam):** Adjusts weights using backpropagation.

4. Model Evaluation

After training, we evaluate the CNN using validation data.

5. Making Predictions

The trained model classifies new images.

6. Model Optimization

To improve accuracy:

- **Data Augmentation** (Rotation, flipping, zooming).
- **Dropout Layers** (Reduce overfitting).
- **Tuning Hyperparameters** (Learning rate, batch size).

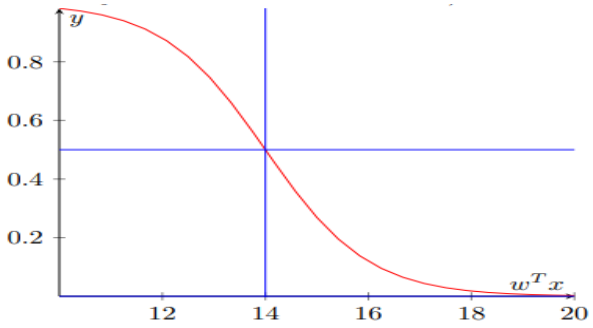
14. Solve following questions:

a) We have a classification problem with labels 0 and 1. We train a logistic model and find out that w 0

learned by our model is -17. What is the value of predicted label of a new test point x using this trained model. If w T x = 1.

Name: Prathamesh Arvind Jadhav

b) The diagram given below is a section of the sigmoid function given by y=1 /1+e −(ω0+ωTx) . What is the value of w 0 with respect to the given diagram? If y=0.5 and ω T x=14.



(a) Predicting the Label for a New Test Point

We have a **logistic regression model** where the output y is given by the **sigmoid function**:

$$y = \frac{1}{1 + e^{-(w_0 + w^T x)}}$$

Given:

- $w_0 = -17$
- $w^T x = 1$

Name: Prathamesh Arvind Jadhav

Substituting these values into the sigmoid function:

$$y = \frac{1}{1 + e^{-(-17+1)}}$$

$$y = \frac{1}{1 + e^{-(-16)}}$$

$$y = \frac{1}{1 + e^{16}}$$

Since e^{16} is a very large number, the denominator becomes very large, making y very small.

$$y \approx 0$$

Since logistic regression predicts **1 if $y \geq 0.5$ and 0 otherwise**, the predicted label is **0**.

(b) Finding w_0 from the Given Diagram

From the **sigmoid function**:

$$y = \frac{1}{1 + e^{-(w_0 + w^T x)}}$$

Given:

- $y = 0.5$
- $w^T x = 14$

Setting $y = 0.5$:

$$0.5 = \frac{1}{1 + e^{-(w_0 + 14)}}$$

Since $\frac{1}{1+e^0} = 0.5$, we must have:

$$w_0 + 14 = 0$$

Solving for w_0:

$$w_0 = -14$$

Thus, the value of w_0 is **-14**.

15. Solve following Problems with repect to Convolutional Neural Networks:

a) Consider a convolution operation with an input image of size 256x256x3 and 63 filters of size 11x11x3, using a stride of 4 and a padding of 2. What is the depth (h in w x b x h) of the output image?.

b) What is the result of convolving a 3x3 matrix with a 2x2 filter with stride 1 and no padding? What is the purpose of using a stride larger than 1 in convolutional neural networks?

(a) Calculating the Depth of the Output Image

We have an **input image** of size **256 × 256 × 3**.

- Number of filters = 63
- Filter size = 11 × 11 × 3
- Stride = 4
- Padding = 2

Name: Prathamesh Arvind Jadhav

The formula to calculate the output **width/height** (W_{out}) of a convolutional layer is:

$$W_{\text{out}} = \frac{W_{\text{in}} - F + 2P}{S} + 1$$

Where:

- W_{in} = Input size **(256)**
- F = Filter size **(11)**
- P = Padding **(2)**
- S = Stride **(4)**

Substituting the values:

$$\begin{aligned} W_{\text{out}} &= \frac{256 - 11 + 2(2)}{4} + 1 \\ &= \frac{256 - 11 + 4}{4} + 1 \\ &= \frac{249}{4} + 1 \\ &= 62.25 + 1 = 63.25 \end{aligned}$$

Since the output size must be an integer, it should be **63 × 63** (rounded down).

The **depth** of the output image is equal to the number of filters used, which is **63**.

Final Answer: The depth of the output image is **63**.

Name: Prathamesh Arvind Jadhav

(b) Convolution of a 3×3 Matrix with a 2×2 Filter (Stride = 1, No Padding)

Given:

- **Input matrix:** 3×3
- **Filter size:** 2×2
- **Stride:** 1
- **Padding:** 0

Using the convolution formula:

$$W_{\text{out}} = \frac{W_{\text{in}} - F + 2P}{S} + 1$$

Substituting the values:

$$W_{\text{out}} = \frac{3 - 2 + 2(0)}{1} + 1$$

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

Thus, the output matrix will have a size of **2 × 2.**

Name: Prathamesh Arvind Jadhav

Example Calculation

If the **input matrix** is:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

And the **filter** is:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The **convolution output** is computed as:

$$\begin{bmatrix} (1 \times 1 + 2 \times 0 + 4 \times 0 + 5 \times 1) & (2 \times 1 + 3 \times 0 + 5 \times 0 + 6 \times 1) \\ (4 \times 1 + 5 \times 0 + 7 \times 0 + 8 \times 1) & (5 \times 1 + 6 \times 0 + 8 \times 0 + 9 \times 1) \end{bmatrix}$$

$$\begin{bmatrix} 1 + 0 + 0 + 5 & 2 + 0 + 0 + 6 \\ 4 + 0 + 0 + 8 & 5 + 0 + 0 + 9 \end{bmatrix}$$

$$\begin{bmatrix} 6 & 8 \\ 12 & 14 \end{bmatrix}$$

Thus, the output matrix is **2 × 2**.

Purpose of Using a Larger Stride in CNNs

Using a stride **larger than 1** serves the following purposes:

1. **Reduces Computational Cost:**
   - A larger stride reduces the output feature map size, decreasing the number of computations.
2. **Downsamples the Input:**
   - Instead of using a separate **pooling layer**, a larger stride itself reduces the spatial dimensions.
3. **Increases Receptive Field:**

    - A higher stride makes each convolutional filter cover a larger portion of the input image.
4. **Faster Processing:**
    - Useful for real-time applications like **object detection** where speed is crucial.

Example:

- A **stride of 1** keeps most spatial details (dense feature map).
- A **stride of 2** halves the output size, improving efficiency.