QUE 1-

In ML, optimizers play an important role in training models. They act like guides, helping the model adjust its internal settings (weights and biases) to make fewer mistakes and perform better.

To understand, Imagine a hilly landscape where you want to find the lowest valley (minimum). The loss function tells the model how far the landscape is from the valley (good performance). The optimizer uses this information to adjust the weights and biases (like moving your feet on the landscape) to reach the valley efficiently.

Working: During training, the model makes predictions based on its current settings. The optimizer calculates the difference between these predicted values and the actual values (the loss). It then analyses the gradients (steepness of the loss landscape) to determine how much and in which direction to adjust the weights and biases. This iterative process continues until the model reaches a minimum loss, signifying good performance.

Example:

1- **Stochastic Gradient Descent (SGD)**: A fundamental optimizer that takes small steps based on the current loss gradient. It's simple but can be slow for complex models.

2-**Momentum**: Imagine rolling a ball down the landscape. Momentum considers the direction of previous adjustments, helping the model overcome shallow valleys and reach the minimum faster.

3-**RMSprop (Root Mean Square Prop)**: Addresses a limitation of SGD where gradients can oscillate wildly. RMSprop considers past gradients, leading to smoother convergence.

4-**Adam (Adaptive Moment Estimation**): An advanced optimizer that combines Momentum and RMSprop. It adapts the learning rate for each weight based on past gradients, making it efficient for various problems.

QUE-2-

Imagine one is lost in a maze and want to find the exit (minimum loss) as fast as possible. All three methods (Gradient Descent, Stochastic Gradient Descent, Mini-Batch Gradient Descent) are ways to navigate this maze:

1-**Gradient Descent (GD):** One carefully examine the entire maze (whole dataset) at each step. This ensures always heading in the generally right direction (following the steepest downhill path) but takes a long time (slow to converge).

2-**Stochastic Gradient Descent (SGD):** One pick a random spot in the maze (single data point) and blindly move in the direction that seems most promising (based on the local gradient). This is much faster (makes many updates) but can be noisy (zig-zags a lot) as one might miss the best path.

3-**Mini-Batch Gradient Descent (Mini-batch GD)**: one grab a small group of friends (mini-batch of data) and explore the maze together. One get a more informed

direction than SGD (less noisy) while still making faster progress than GD (updates more frequently). It's a good balance between speed and accuracy.
In essence, they differ in how much data they use for each update:

1-GD - All data (slow, accurate)
2-SGD - Single data point (fast, noisy)
3-Mini-batch GD - Small subset of data (medium speed, medium accuracy)

The best choice depends on your maze (dataset) size and how quickly you need to escape (training time).

QUE 3-

Adam: The Adaptive Optimizer for Efficient Learning

Adam, the Adaptive Moment Estimation optimizer, works similarly in machine learning. It dynamically adjusts the learning rate for each parameter (like treats for the dog) based on their recent behaviour (gradients). This leads to faster and more stable training, especially for complex models.

Working

1-Keeps Track of Gradients: Adam monitors the average historical gradients (first moment) and their squared variations (second moment) for each parameter. These act like a memory of how each parameter has been affecting the loss function.
2-Adapts Learning Rates: Unlike SGD with a fixed learning rate, Adam adjusts the learning rate for each parameter based on its historical gradients. Noisy or oscillating gradients get smaller adjustments, while consistent gradients receive larger ones, ensuring all parameters move at an appropriate pace.
3-Momentum with Bias Correction: Similar to Momentum, Adam incorporates the direction of past updates to accelerate learning. However, it also corrects for an initial bias in the momentum estimates, leading to smoother convergence.

Benefits:

1-Faster Convergence: By adapting learning rates, Adam can navigate the loss landscape more efficiently, often reaching the minimum faster than SGD variants.
2-Less Tuning: Adam has fewer hyperparameters (control knobs) to tune compared to other optimizers, making it more user-friendly.
3-Works Well on Complex Problems: Adam's ability to handle noisy gradients makes it a good choice for challenging datasets and deep learning models.

Limitations

1-May Not Always Be the Best: While Adam is powerful, it might not be the optimal choice for every situation.
2-Memory Usage: Storing historical gradients can consume more memory compared to SGD.

QUE 4-

Rmsprop and Adam are both optimizers in machine learning that address the issue of vanishing/exploding gradients during training, but they differ in their approach:

1-Approach used:

i-Rmsprop: Focuses on the squared gradients to get an idea of the variance of past updates. This helps it avoid large swings in parameter values.
ii-Adam: Tracks both the average gradients (momentum) and the uncentered variance (squared gradients) for a more comprehensive picture.

2-Updation in bias:

i-Rmsprop: Does not account for an initial bias in the momentum estimates, which can slow down convergence initially.
ii-Adam: Includes a bias correction term, leading to smoother convergence from the start.

3-Complexity:

i-Rmsprop: Simpler with fewer hyperparameters to tune.
ii-Adam: More complex with additional hyperparameters for the momentum and bias correction terms.

Choosing between them:

i-Rmsprop: Good especially for simpler models or when memory is a concern.
ii-Adam: better choice for complex models due to its adaptive nature and bias correction, but might require more hyperparameter tuning.

Que 5-

Based on uses for different purposes, there is no best optimizer. Here's a breakdown of some popular optimizers and their strengths and weaknesses:

1-**Stochastic Gradient Descent (SGD)**:

Strength: Simple, works well with large datasets, easy to understand and implement.

Weakness: Slow convergence, requires careful tuning of the learning rate, can get stuck in local minima.

Application: Good baseline for many problems, can be effective with momentum or learning rate scheduling.

2. **Mini-batch Gradient Descent:**

Strength: Faster than SGD, balances speed and accuracy.

Weakness: Choice of mini-batch size can affect performance.

Application: Often the default choice for many machine learning tasks.

3. **RMSprop:**

Strength: Adapts learning rates, addresses vanishing/exploding gradients better than SGD, less complex than Adam.

Weakness: Can still be slow to converge, may not be suitable for all problems.

Application: Good choice for problems with noisy or sparse gradients, often used for recurrent neural networks (RNNs).

4. **Adam:**

Strength: Highly adaptive, often converges faster than SGD or RMSprop, generally works well with complex models and deep learning.

Weakness: More complex, requires some hyperparameter tuning, can be computationally expensive for large models.

Application: good for deep learning tasks.

QUE 6-

Overfitting and underfitting are two common challenges faced in machine learning when training models. They represent the model's ability to generalize, meaning how well it performs on unseen data. Here's a detailed explanation:

Overfitting:

Imagine a student studying for an exam by memorizing every single practice question and answer. This student might ace the practice tests but struggle with entirely new questions on the actual exam. This is analogous to overfitting in machine learning.

1-Effect: The model becomes overly focused on capturing every detail and random noise in the training data. It learns the "tricks" specific to that data set instead of the underlying patterns.

2-Symptoms: The model performs very well on the training data but poorly on unseen data (test data). This is a significant performance gap.

3-Causes:

i-Too complex model: A model with too many parameters (like weights and biases) can easily overfit, especially for small datasets.

ii-Training for too long: Continuing to train the model even after it has learned the training data well can lead to overfitting.

iii-Noisy data: Training data with a lot of noise can be memorized by the model, leading to overfitting.

Underfitting:

Think of a student who skims the textbook chapters without really understanding the concepts. This student might perform poorly on all assessments, both practice and actual exams. This is similar to underfitting in machine learning.

1-Effect: The model fails to capture the underlying patterns in the training data itself. It's too simple to learn the complexities of the data.

2-Symptoms: The model performs poorly on both the training data and unseen data. It generally has high bias and low variance (meaning high average error and low sensitivity to changes in the training data).

3-Causes:

i-Too simple model: A model with too few parameters lacks the capacity to learn the complexities of the data.

ii-Not enough training data: The model doesn't have enough information to learn the relationships within the data.

iii-Features not representative: The features used to train the model might not be capturing the important factors influencing the target variable.

QUE 7-

Vanishing Gradients:

Vanishing gradient problem is a phenomenon that occurs during the training of deep neural networks, where the gradients that are used to update the network become extremely small or "vanish" as they are backpropogated from the output layers to the earlier layers.
During the training process of the neural network, the goal is to minimize a loss function by adjusting the weights of the network.

The backpropogation algorithm calculates these gradients by propogating the error from the output layer to the input layer.

In machine learning, backpropagation is an effective algorithm used to train artificial neural networks, especially in feed-forward neural networks.
Backpropagation is an iterative algorithm, that helps to minimize the cost function by determining which weights and biases should be adjusted. During every epoch, the model learns by adapting the weights and biases to minimize the loss by moving down toward the gradient of the error. Thus, it involves the two most popular optimization algorithms, such as gradient descent or stochastic gradient descent.

After understanding what is vanishing gradient problem, here's how it impacts Deep Learning models:

1. Limited learning capacity
2. Difficulty in capturing the long-term dependencies
3. Slow convergence & training instability
4. Preferential learning in shallow layers
5. Architectural design considerations

Exploding Gradients:

The exploding gradient problem is a challenge encountered during the training of deep neural networks, particularly in the context of gradient-based optimization methods such as backpropagation. This issue occurs when the gradients of the network's loss with respect to the parameters (weights) become excessively large. The explosion of the gradient can lead to numerical instability and the inability of the network to converge to a suitable solution.

Causes:

The root cause of exploding gradients can often be traced back to the network architecture and the choice of activation functions. In deep networks, when multiple layers have weights greater than 1, the gradients can grow exponentially as they propagate back through the network during training. This is exacerbated when using activation functions with outputs that are not bounded, such as the hyperbolic tangent or the sigmoid function.

Another contributing factor is the initialization of the network's weights. If the initial weights are too large, even a small gradient can be amplified through the layers, leading to very large updates during training.

Consequences:

When gradients explode, the weight updates during training can become so large that they cause the learning algorithm to overshoot the minima of the loss function. This can result in model parameters diverging to infinity, causing the learning process to fail. The model may exhibit erratic behaviour, with the loss becoming NaN (not a number) or Inf (infinity), and the model's predictions becoming meaningless.

QUE 8-

Batch Normalization:

Normalization is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.

Generally, when we input the data to a machine or deep learning algorithm we tend to change the values to a balanced scale. The reason we normalize is partly to ensure that our model can generalize appropriately.
Now coming back to **Batch normalization**, it is a process to make neural networks faster and more stable through adding extra layers in a deep neural network. The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer.98700

But what is the reason behind the term “Batch” in batch normalization? A typical neural network is trained using a collected set of input data called batch. Similarly, the normalizing process in batch normalization takes place in batches, not as a single input.

Layer Normalization:

In layer normalization, all neurons in a particular layer effectively have the same distribution across all features for a given input.

For example, if each input has d features, it's a d-dimensional vector. If there are B elements in a batch, the normalization is done along the length of the d-dimensional vector and not across the batch of size B.
Normalizing across all features but for each of the inputs to a specific layer removes the dependence on batches. This makes layer normalization well suited for sequence models such as transformers and recurrent neural networks(RNN) that were popular in the pre-transformer era.

QUE 9-

Regularization is a technique used to prevent overfitting by adding a penalty term to the model's objective function during training. The objective is to discourage the model from fitting the training data too closely and promote simpler models that generalize better to unseen data. Regularization methods control the complexity of models by penalizing large coefficients or by selecting a subset of features, thus helping to strike the right balance between bias and variance.

L1 Regularization (Lasso):

L1 regularization, a popular regularization technique in machine learning, offers a powerful approach to mitigate overfitting and perform feature selection in regression modeling. Traditional regression models may struggle when dealing with high-dimensional datasets containing many irrelevant features, leading to poor predictive performance and model interpretability. Lasso regression addresses these challenges by introducing a penalty term to the loss function, encouraging sparsity in the model and selecting only the most relevant predictors for the target variable.

This regularization technique is particularly beneficial when dealing with high-dimensional datasets, where it helps simplify the model and improve interpretability by focusing on the most influential predictors.

L2 Regularization (Ridge):

Ridge Regression, a powerful regularization technique in the realm of machine learning, offers a robust solution to mitigate overfitting and improve model generalization. Traditional regression models, such as linear regression, may struggle when dealing with datasets containing multicollinear features, where predictors are highly correlated. This phenomenon often leads to unstable coefficient estimates and poor predictive performance. Ridge regression addresses these challenges by introducing a penalty term to the loss function, encouraging smaller coefficient magnitudes and promoting model simplicity.

The core principle behind Ridge regression lies in its ability to balance the trade-off between bias and variance. By adding a penalty term proportional to the square of the coefficients to the loss function, Ridge regression effectively shrinks the

coefficient estimates towards zero while still allowing them to be non-zero. This regularization technique is particularly beneficial when dealing with multicollinear datasets, where it helps stabilize the model by reducing the sensitivity of coefficient estimates to small changes in the training data.

Elastic Net Regularization:

Elastic net regularization, a hybrid approach combining Ridge and Lasso regression techniques, offers a versatile solution to mitigate overfitting and perform feature selection in regression modeling. Traditional regression models may face challenges when dealing with datasets containing multicollinear features and high dimensionality, where balancing model complexity and sparsity is crucial for optimal performance. Elastic Net regularization addresses these challenges by adding both L1 and L2 penalty terms to the loss function, providing a flexible framework to control the trade-off between coefficient shrinkage and feature selection.

The key principle behind Elastic Net regularization is to strike a balance between Ridge and Lasso regression techniques. By adding both L1 and L2 penalty terms to the loss function, Elastic Net regularization combines the strengths of both approaches while mitigating their individual limitations.One of the key advantages of Elastic Net regularization is its ability to handle datasets with complex structures efficiently.

Que 10-
Dropout:

It is another regularization technique that prevents neural networks from overfitting. Regularization methods like L1 and L2 reduce overfitting by modifying the cost function but on the contrary, the Dropout technique modifies the network itself to prevent the network from overfitting.

Working Principle:

It randomly drops some neurons except for the output layer from the neural network during training in each iteration or we can assign a probability p to all the neurons in a network so that they are temporarily ignored from calculations.
where p is known as the Dropout Rate and is usually instantiated to 0.5.
Then, as each iteration is going on, the neurons in each layer with the highest probability get dropped. This results in creating a smaller network with each pass on the training dataset(epoch). Since in each iteration, a random input value can be eliminated, the network tries to balance the risk and not to favor any of the features and reduces bias and noise.
Sometimes, this technique is also known as the Ensemble Technique for Neural Networks as when we drop different sets of neurons, it’s equivalent to training different neural networks. So, in this technique, the different networks will overfit in different ways, so the net effect of dropout will be to reduce overfitting.

QUE-11

L1 Regularization (Lasso):

L1 regularization, a popular regularization technique in machine learning, offers a powerful approach to mitigate overfitting and perform feature selection in regression modeling. Traditional regression models may struggle when dealing with high-dimensional datasets containing many irrelevant features, leading to poor predictive performance and model interpretability. Lasso regression addresses these challenges by introducing a penalty term to the loss function, encouraging sparsity in the model and selecting only the most relevant predictors for the target variable.

This regularization technique is particularly beneficial when dealing with high-dimensional datasets, where it helps simplify the model and improve interpretability by focusing on the most influential predictors.

L2 Regularization (Ridge):

Ridge Regression, a powerful regularization technique in the realm of machine learning, offers a robust solution to mitigate overfitting and improve model generalization. Traditional regression models, such as linear regression, may struggle when dealing with datasets containing multicollinear features, where predictors are highly correlated. This phenomenon often leads to unstable coefficient estimates and poor predictive performance. Ridge regression addresses these challenges by introducing a penalty term to the loss function, encouraging smaller coefficient magnitudes and promoting model simplicity.

The core principle behind Ridge regression lies in its ability to balance the trade-off between bias and variance. By adding a penalty term proportional to the square of the coefficients to the loss function, Ridge regression effectively shrinks the coefficient estimates towards zero while still allowing them to be non-zero. This regularization technique is particularly beneficial when dealing with multicollinear datasets, where it helps stabilize the model by reducing the sensitivity of coefficient estimates to small changes in the training data.

QUE 12-

Validation Accuracy

Validation accuracy is a crucial metric in machine learning used to gauge a model's ability to perform well on unseen data. It helps prevent a common pitfall called **overfitting**. Here's a breakdown of why validation accuracy is important:

To prevent overfitting, we split the available data into three sets:

1. **Training Set:** Used to train the model. The model learns patterns and relationships from this data.

2. **Validation Set:** Used to assess the model's performance on unseen data during training. The model doesn't "see" this data while training, but it shares similar characteristics with the training data.
3. **Test Set:** Used for final evaluation after training is complete. This data is completely unseen by the model and represents real-world scenarios.

Validation accuracy is the percentage of predictions the model makes correctly on the validation set. By monitoring this metric during training, we can:

- **Identify Overfitting:** If the training accuracy is significantly higher than the validation accuracy, it's a red flag for overfitting.
- **Fine-tune the Model:** We can adjust hyperparameters (like model complexity or training time) to improve the model's ability to generalize based on validation accuracy.

QUE 13-

Data Augmentation:

Data augmentation is the process of artificially generating new data from existing data, primarily to train new machine learning (ML) models. ML models require large and varied datasets for initial training, but sourcing sufficiently diverse real-world datasets can be challenging because of data silos, regulations, and other limitations. Data augmentation artificially increases the dataset by making small changes to the original data.

Advantages:

1. Increased Data Diversity:
Data augmentation introduces variety into the training data, which can help models better generalize to unseen data. It exposes the model to a broader range of scenarios and conditions, making it more robust. For instance, in image classification, augmenting images with rotations, flips, and color variations can help the model recognize objects from various angles and lighting conditions.

2. Improved Model Performance:
Augmenting data can enhance a model's ability to learn and recognize patterns. This often results in improved accuracy, as the model becomes more capable of handling real-world variations. For example, in natural language processing, augmenting text data with synonym substitutions and paraphrasing can make the model more resilient to slight phrasing changes.

3. Reduced Overfitting:
By expanding the training dataset, data augmentation helps prevent overfitting, where a model memorizes the training data rather than learning its underlying patterns. This is especially important in situations with limited training data.

Augmentation can make the model generalize better to new examples, reducing the risk of poor performance on unseen data.

Disadvantages:

1.Risk of Overfitting Augmented Data:
While data augmentation can reduce overfitting on the original data, it's possible to overfit the augmented data if transformations are not applied judiciously. Extreme augmentations can introduce noise and result in poor model generalization. Care must be taken to strike a balance between diversity and realism in augmented data.

2. Computation Complexity:
Applying data augmentation techniques increases the computational requirements during training, as each data point is transformed multiple times. This can be a concern in resource-constrained environments, as it may lead to longer training times and increased hardware requirements. Balancing the benefits of augmentation with the computational costs is essential.

QUE 14-

Transfer Learning:

Transfer learning is a popular method in computer vision because it allows us to build accurate models in a timesaving way . With transfer learning, instead of starting the learning process from scratch, you start from patterns that have been learned when solving a different problem.

In computer vision, transfer learning is usually expressed through the use of pre-trained models. A pre-trained model is a model that was trained on a large benchmark dataset to solve a problem similar to the one that we want to solve. Accordingly, due to the computational cost of training such models, it is common practice to import and use models from published literature.

Pre Trained Models:

There are perhaps a dozen or more top-performing models for image recognition that can be downloaded and used as the basis for image recognition and related computer vision tasks.
Perhaps three of the more popular models are as follows:

- VGG (e.g. VGG16 or VGG19).
- GoogLeNet (e.g. InceptionV3).
- Residual Network (e.g. ResNet50).

These models are both widely used for transfer learning both because of their performance, but also because they were examples that introduced specific architectural innovations, namely consistent and repeating structures (VGG), inception modules (GoogLeNet), and residual modules (ResNet).

Keras provides access to a number of top-performing pre-trained models that were developed for image recognition tasks.

QUE 15-

There is a tradeoff between a model's ability to minimize bias and variance which is referred to as the best solution for selecting a value of Regularization constant.

Bias

The bias is known as the difference between the prediction of the values by the Machine Learning model and the correct value. Being high in biasing gives a large error in training as well as testing data. It recommended that an algorithm should always be low-biased to avoid the problem of underfitting. By high bias, the data predicted is in a straight line format, thus not fitting accurately in the data in the data set. Such fitting is known as the Underfitting of Data. This happens when the hypothesis is too simple or linear in nature.

Variance

The variability of model prediction for a given data point which tells us the spread of our data is called the variance of the model. The model with high variance has a very complex fit to the training data and thus is not able to fit accurately on the data which it hasn't seen before. As a result, such models perform very well on training data but have high error rates on test data. When a model is high on variance, it is then said to as Overfitting of Data. Overfitting is fitting the training set accurately via complex curve and high order hypothesis but is not the solution as the error with unseen data is high. While training a data model variance should be kept low.

Bias-Variance Tradeoff

If the algorithm is too simple (hypothesis with linear equation) then it may be on high bias and low variance condition and thus is error-prone. If algorithms fit too complex (hypothesis with high degree equation) then it may be on high variance and low bias. In the latter condition, the new entries will not perform well. Well, there is something between both of these conditions, known as a Trade-off or Bias Variance Trade-off. This tradeoff in complexity is why there is a tradeoff between bias and variance. An algorithm can't be more complex and less complex at the same time.

We try to optimize the value of the total error for the model by using the Bias_Variance Tradeoff.

The best fit will be given by the hypothesis on the tradeoff point.

QUE 16-

In this scenario, training a single neural network with one output neuron for each disease (multi-class classification) is likely the better approach. Because:

Advantages of a Single Neural Network:

- Efficiency: Training a single network is generally more efficient than training four separate ones. It requires less computational power and memory, especially for large datasets.

- Shared Features: Many diseases might share underlying physiological factors reflected in the features (e.g., blood work). A single network can learn these shared features and then specialize for each disease in the final layer.
- Regularization: A single network often benefits from regularization techniques that prevent overfitting. These techniques can be more effective when applied to a single, larger model compared to four smaller ones.

Dealing with its disadvantages:

- Multi-class vs. Binary Networks: While some argue that binary one-vs-rest networks (training separate networks for each disease vs. all others) might be more accurate for imbalanced datasets (unequal distribution of disease types), this can be mitigated by using techniques like class weighting during training.
- Interpretability: A single network might be slightly less interpretable than four separate ones in terms of understanding which features influence each disease prediction. However, techniques like feature attribution can still provide insights into the single network's decision-making process.

QUE 17-

Part (a): Number and Shapes of Weight and Bias Matrices

In a neural network with two hidden layers, the following weight (W) and bias (b) matrices are required:

- Weight Matrix W1 (shape: n1 x (n0 + 1))
    - n1: Number of neurons in the first hidden layer (A1).
    - n0: Number of input features.
        - 1: Bias term for each neuron in A1.
    - This matrix stores the weights for connections between each input feature and every neuron in the first hidden layer.
- Bias Vector b1 (shape: n1 x 1)
    - n1: Number of neurons in the first hidden layer (A1).
    - 1: Represents the bias terms for each neuron in A1.
    - This vector contains the bias values added to the weighted sum of inputs in each neuron of A1.
- Weight Matrix W2 (shape: n2 x (n1 + 1))
    - n2: Number of neurons in the second hidden layer (A2).
    - n1: Number of neurons in the first hidden layer (A1).
        - 1: Bias term for each neuron in A2.
    - This matrix stores the weights for connections between each neuron in the first hidden layer and every neuron in the second hidden layer.
- Bias Vector b2 (shape: n2 x 1)
    - n2: Number of neurons in the second hidden layer (A2).
    - 1: Represents the bias terms for each neuron in A2.
    - This vector contains the bias values added to the weighted sum of inputs in each neuron of A2.

- Weight Matrix W3 (shape: n3 x (n2 + 1)) [Only for Multi-Class Classification]
    - n3: Number of output neurons (depends on the number of classes).
    - n2: Number of neurons in the second hidden layer (A2).
        - 1: Bias term for each output neuron.
    - This matrix stores the weights for connections between each neuron in the second hidden layer and every output neuron. (Not applicable for Binary Classification or Regression.)
- Bias Vector b3 (shape: n3 x 1)) [Only for Multi-Class Classification]
    - n3: Number of output neurons (depends on the number of classes).
    - 1: Represents the bias terms for each output neuron.
    - This vector contains the bias values added to the weighted sum of inputs in each output neuron. (Not applicable for Binary Classification or Regression.)

Matrix Multiplication Verification:

- Forward propagation calculations ensure compatible dimensions:
    - W1 * X (n1 x (n0 + 1)) * (n0 x m) = A1 (n1 x m)
    - W2 * A1 (n2 x (n1 + 1)) * (n1 x m) = A2 (n2 x m)
    - For Multi-Class Classification: W3 * A2 (n3 x (n2 + 1)) * (n2 x m) = Y (n3 x m) (Not applicable for Binary Classification or Regression.)

Part (b): Activation Functions for Binary Classification

(i) Hidden Layers: ReLU (Rectified Linear Unit) is a common choice for hidden layers due to:

- Computational efficiency: Faster than functions like sigmoid or tanh.
- Vanishing gradient problem mitigation: ReLU prevents gradients from becoming very small or zero during backpropagation, allowing the network to learn effectively.

(ii) Output Layer: Sigmoid is often used for binary classification because it outputs a value between 0 and 1, which can be interpreted as the probability of belonging to the positive class:

- Thresholding: A value greater than a threshold (e.g., 0.5) is typically considered positive, while a value below is negative.

Part (c): Activation Functions for Multi-Class Classification

(i) Hidden Layers: Same reasoning as in Binary Classification (ReLU for efficiency and gradient flow).

(ii) Output Layer: Softmax is the preferred activation for multi-class classification because it:

- Normalizes outputs: Values between 0 and 1, representing probabilities for each class.
- Mutually exclusive outputs: The sum of outputs across all classes is always 1, ensuring only one class can be active at a time.

Part (d): Activation Functions for Regression

In regression problems, the goal is to predict continuous numerical values (e.g., house price, temperature). Therefore, the activation function in the output layer should map the activations to the unbounded real number line:

1-Output Layer: Linear Activation

- No transformation applied to the weighted sum of inputs.
- This allows the network to directly output continuous values within the desired range.

Here's why linear activation is suitable for regression:

- Preserves Magnitude: The output retains the scale of the weighted sum, representing the predicted continuous value.
- No Bounding: Unlike sigmoid or ReLU, there's no upper or lower limit on the output, allowing for predictions across the entire real number line.
- Simplicity: Linear activation is computationally inexpensive and interpretable, making it a good choice for regression tasks

2-Hidden layer:

In regression problems, for the hidden layers of your neural network with two hidden layers, you can still use the ReLU (Rectified Linear Unit) activation function. Here's why:

- Non-linearity: While the output layer needs a linear activation to preserve the predicted continuous values, hidden layers benefit from non-linearity. ReLU allows the network to learn complex relationships between the input features and the hidden layer activations.
- Vanishing Gradient Problem Mitigation: As explained in part (b)(i), ReLU helps prevent the vanishing gradient problem during backpropagation, which is crucial for training the network effectively. ReLU allows gradients to flow back through the network during training, enabling the network to learn from errors and adjust its weights and biases.
- Computational Efficiency: Compared to activation functions like sigmoid or tanh, ReLU is computationally faster, making it a practical choice for large neural networks.