|  |  |  |
| --- | --- | --- |
| Course: | **Artificial Intelligence** | Date: 20/01/2024 |
| Course Code: | CSC-325 | Session: I |
| Faculty’s Name: | Mr. Usama Imtiaz & Mr. Israr Akhter | Max Marks: 50 |
| Time Allowed: | 2.5 Hours | Total Pages: 14 |

* This is a closed-book exam. Communication devices and any written material is strictly prohibited.
* All questions are compulsory.
* Comprehension of questions is part of the exam. Use blue/black pen only.
* ***The answer sheet is not required****; solve your exam on the question paper.*
* Avoid attaching any extra sheets; use the last page for rough work.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **CLO 3** | **CLO 2** | **CLO 4** | **CLO 3** | **CLO 3** | **CLO 1** | **Total** |
| Question No # | **Q.1** | **Q.2** | **Q.3** | **Q.4** | **Q.5** | **Q.6** |
| Total Marks | 8 Marks | 5 Marks | 15 Marks | 6 Marks | 8 Marks | 8 Marks | **50 Marks** |
| Obtained Marks |  |  |  |  |  |  |  |

Student’s Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Enrollment No: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Invigilator's Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Question # 1 (4+4 Marks)** **………….....………….....……….....………….....…..…..……(CLO-3)**

The Sudoku puzzle (an example is shown below) consists of a 9 x 9 grid with 3 x 3 blocks for a total of the 81 cells. Each puzzle, which has a unique solution, has some cells that have already been filled in. The objective of the puzzle is to fill in the remaining cells with the numbers 1 through 9 so that the following rules are satisfied:



* Each horizontal row must contain the numbers 1-9, without repeating any.
* Each vertical column must contain the numbers 1-9, without repeating any.
* Each 3 x 3 block must contain the numbers 1-9, without repeating any.

1. **How would you represent the problem while Applying GA?**

|  |
| --- |
| In a Genetic Algorithm (GA), the problem representation involves encoding potential solutions to the problem in a format that the algorithm can manipulate. For the Sudoku problem, you can represent a solution using a 2D array or matrix, where each element corresponds to a cell in the Sudoku grid. The following components are involved in the representation:   * **Chromosome/Genotype**: The chromosome represents a potential solution to the Sudoku puzzle. It's a 2D array where each element (gene) corresponds to a cell in the 9x9 grid. The initial values of the filled-in cells from the puzzle are included in the chromosome.    **Population**: The population is a collection of these chromosomes. Each chromosome represents a potential solution to the Sudoku puzzle.   **Fitness Function**: The fitness function evaluates how close a given solution is to a valid Sudoku solution. It can calculate the number of conflicts or violations of the rules. A lower fitness value indicates a better solution.   **Crossover (Recombination)**: The crossover operation combines two parent solutions to produce offspring solutions. In the context of Sudoku, this could involve exchanging rows, columns, or 3x3 blocks between two parent chromosomes.   **Mutation**: The mutation operation introduces small changes in a solution. For Sudoku, this could involve swapping numbers within rows, columns, or 3x3 blocks.   **Selection**: The selection mechanism chooses parent solutions for reproduction based on their fitness. Solutions with lower fitness values have a higher chance of being selected.   **Termination Condition**: Define a termination condition, such as reaching a target fitness or a maximum number of generations, to stop the algorithm. |

1. **How would you write the fitness function?**

|  |
| --- |
| The fitness function for a Sudoku Genetic Algorithm evaluates how close a given solution is to a valid Sudoku solution. It assign a higher fitness score to solutions that adhere to the Sudoku rules (no repeated numbers in rows, columns, or 3x3 blocks) and a lower fitness score to solutions with violations. |
| This fitness function will check each row, column, and 3x3 block in the Sudoku grid for violations. The fitness score is increased for each repeated number in these regions. The goal is to minimize the fitness score, so a perfect solution (without any violations) would have a fitness score of 0. |

**Question # 2 (5 Marks)………….....………….....………….....………….....…..…..……(CLO-2)**

Apply naïve bayes algorithm to predict the class of the given test data. The test data is

**Green Large Rectangle Yes ( Class= ?)**

* Calculations are a must. Any solution without calculation will not be graded.

A screenshot of a table

Description automatically generated with low confidence

|  |
| --- |
| **. Calculate Prior Probabilities:**   * P(A) = 4/9 * P(B) = 3/9 * P(C) = 2/9   **2. Calculate Likelihood Probabilities:**   * P(Green | A) = 2/4 * P(Large | A) = 0/4 * P(Rectangle | A) = 0/4 * P(Yes | A) = 3/4 * P(Green | B) = 0/3 * P(Large | B) = 2/3 * P(Rectangle | B) = 2/3 * P(Yes | B) = 2/3 * P(Green | C) = 2/2 * P(Large | C) = 0/2 * P(Rectangle | C) = 0/2 * P(Yes | C) = 1/2   **3. Apply Bayes' Theorem:**   * P(A | Green, Large, Rectangle, Yes) = P(Green | A) \* P(Large | A) \* P(Rectangle | A) \* P(Yes | A) \* P(A) = 0 * P(B | Green, Large, Rectangle, Yes) = P(Green | B) \* P(Large | B) \* P(Rectangle | B) \* P(Yes | B) \* P(B) = 0 * P(C | Green, Large, Rectangle, Yes) = P(Green | C) \* P(Large | C) \* P(Rectangle | C) \* P(Yes | C) \* P(C) = 0   **4. Select the Class with the Highest Posterior Probability:**   * All posterior probabilities are 0 due to feature combinations not present in the training data.   **(If someone attempted)** |
| **Applying Laplace Smoothing: (Optional)**  **1. Adjust Likelihood Probabilities:**   * P(Green | A) = (2 + 1) / (4 + 3) = 3/7 * P(Large | A) = (0 + 1) / (4 + 3) = 1/7 * P(Rectangle | A) = (0 + 1) / (4 + 3) = 1/7 * P(Yes | A) = (3 + 1) / (4 + 3) = 4/7 * ... (Apply smoothing to all likelihood probabilities)   **2. Recalculate Posterior Probabilities:**   * P(A | Green, Large, Rectangle, Yes) = (3/7) \* (1/7) \* (1/7) \* (4/7) \* (4/9) ≈ 0.0054 * P(B | Green, Large, Rectangle, Yes) = (1/6) \* (2/4) \* (2/4) \* (2/3) \* (3/9) ≈ 0.0167 * P(C | Green, Large, Rectangle, Yes) = (1) \* (1/3) \* (1/3) \* (1/2) \* (2/9) ≈ 0.0119   **3. Select the Class with the Highest Posterior Probability:**   * Class B has the highest posterior probability (0.0167).   **Prediction:**   * The predicted class for the test data (Green, Large, Rectangle, Yes) is **B**. |

**Question # 3 (15 Marks) .………….....…… .…… …… …….....…… .………….....…… (CLO-4)**

Consider the neural network below. Perform one forward pass and fix the error and update the weight for the given inputs and outputs. You need to be accurate in your calculations. Please perform calculations up to 4 decimal places. This shows the complete working of the algorithm; **fill in the table on the right**, no marks will be awarded without that. Direct answers will not get any marks.

Inputs: X1 = 2, X2 = 3,

Output: Y5 = 1, Y6 = 0,

w1 = 0.3, w2 = 0.4, w3 = 0.9, w4 = 1.0, w5 = -1, w6 = 1.1, w7 = 0.7 and w8 = 0.6  
Alpha = 0.2

Activation function in the neurons is sigmoid function.

|  |  |
| --- | --- |
|  | **Value** |
| **Y3** | 0.9002 |
| **Y4** | 0.9900 |
| **Y5** | 0.4228 |
| **Y6** | 0.7357 |
| **E5** | 0.5771 |
| **E6** | -0.7357 |
| **ΔW** | Null |
| **ΔW5** | 0.0253 |
| **ΔW6** | -0.0257 |
| **ΔW7** | 0.0278 |
| **ΔW8** | -0.02832 |

A picture containing diagram, line, circle, technical drawing

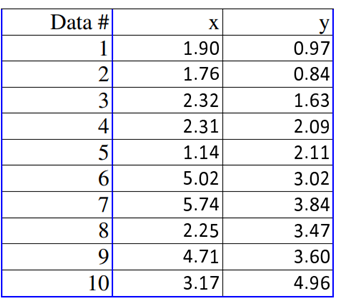
Description automatically generated

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

**Question # 4 (6 Marks)** **………………………………………………………………………….(CLO-3)**

Suppose you are given the following data pairs. You will simulate the k-means algorithm to identify TWO clusters in the data.

|  |  |  |
| --- | --- | --- |
| Data | 1st | 2nd |
| 1 | C1 | C1 |
| 2 | C1 | C1 |
| 3 | C1 | C1 |
| 4 | C1 | C1 |
| 5 | C1 | C1 |
| 6 | C2 | C2 |



Suppose you are given initial assignment cluster center as {cluster1: #1}, {cluster2: #6} – the first data point is used as the first cluster center and the 10-th as the second cluster center.

Please simulate the k-means (k=2) algorithm for TWO iteration. What are cluster assignments after TWO iteration? (Fill in the table below)

Assume k-means uses Euclidean distance. Calculate distance value up to 4 decimal places. In case of tie, you can assign the data point to cluster #1.

|  |
| --- |
|  |
| Initial cluster centers: {cluster1: #1}, {cluster2: #6}  Let's calculate the distances and assign the points to clusters for the first iteration:  Iteration 1: Cluster 1 center: (1.9, 0.97) Cluster 2 center: (5.02, 3.02)  For each data point, calculate the Euclidean distance to both cluster centers:  Data 1: Distance to Cluster 1 center: sqrt((1.9 - 1.9)^2 + (0.97 - 0.97)^2) = 0 Distance to Cluster 2 center: sqrt((1.9 - 5.02)^2 + (0.97 - 3.02)^2) = 3.6056  Since the distance to Cluster 1 is smaller, Data 1 is assigned to Cluster 1.  Data 2: Distance to Cluster 1 center: sqrt((1.76 - 1.9)^2 + (0.84 - 0.97)^2) = 0.1633 Distance to Cluster 2 center: sqrt((1.76 - 5.02)^2 + (0.84 - 3.02)^2) = 3.6703  Since the distance to Cluster 1 is smaller, Data 2 is assigned to Cluster 1.  Data 3: Distance to Cluster 1 center: sqrt((2.32 - 1.9)^2 + (1.63 - 0.97)^2) = 0.9759 Distance to Cluster 2 center: sqrt((2.32 - 5.02)^2 + (1.63 - 3.02)^2) = 2.9884  Since the distance to Cluster 1 is smaller, Data 3 is assigned to Cluster 1.  Data 4: Distance to Cluster 1 center: sqrt((2.31 - 1.9)^2 + (2.09 - 0.97)^2) = 1.2163 Distance to Cluster 2 center: sqrt((2.31 - 5.02)^2 + (2.09 - 3.02)^2) = 2.5166  Since the distance to Cluster 1 is smaller, Data 4 is assigned to Cluster 1.  Data 5: Distance to Cluster 1 center: sqrt((1.41 - 1.9)^2 + (2.11 - 0.97)^2) = 1.2703 Distance to Cluster 2 center: sqrt((1.41 - 5.02)^2 + (2.11 - 3.02)^2) = 3.8467  Since the distance to Cluster 1 is smaller, Data 5 is assigned to Cluster 1.  Data 6: Distance to Cluster 1 center: sqrt((5.02 - 1.9)^2 + (3.02 - 0.97)^2) = 4.4788 Distance to Cluster 2 center: sqrt((5.02 - 5.02)^2 + (3.02 - 3.02)^2) = 0  Since the distance to Cluster 2 is smaller, Data 6 is assigned to Cluster 2.  Now let's update the cluster centers based on the assigned points:  Cluster 1: (1, 2, 3, 4, 5) Cluster 2: (6)  Iteration 2: Cluster 1 center: (1.938, 1.328) Cluster 2 center: (5.02, 3.02)  For each data point, calculate the Euclidean distance to both cluster centers:  Data 1: Distance to Cluster 1 center: sqrt((1.938 - 1.9)^2 + (1.328 - 0.97)^2) = 0.3653 Distance to Cluster 2 center: sqrt((1.938 - 5.02)^2 + (1.328 - 3.02)^2) = 3.8767  Since the distance to Cluster 1 is smaller, Data 1 is assigned to Cluster 1.  Data 2: Distance to Cluster 1 center: sqrt((1.938 - 1.76)^2 + (1.328 - 0.84)^2) = 0.2082 Distance to Cluster 2 center: sqrt((1.938 - 5.02)^2 + (1.328 - 3.02)^2) = 3.8767  Since the distance to Cluster 1 is smaller, Data 2 is assigned to Cluster 1.  Data 3: Distance to Cluster 1 center: sqrt((1.938 - 2.32)^2 + (1.328 - 1.63)^2) = 0.4207 Distance to Cluster 2 center: sqrt((1.938 - 5.02)^2 + (1.328 - 3.02)^2) = 3.8767  Since the distance to Cluster 1 is smaller, Data 3 is assigned to Cluster 1.  Data 4: Distance to Cluster 1 center: sqrt((1.938 - 2.31)^2 + (1.328 - 2.09)^2) = 0.8664 Distance to Cluster 2 center: sqrt((1.938 - 5.02)^2 + (1.328 - 3.02)^2) = 3.8767  Since the distance to Cluster 1 is smaller, Data 4 is assigned to Cluster 1.  Data 5: Distance to Cluster 1 center: sqrt((1.938 - 1.41)^2 + (1.328 - 2.11)^2) = 1.0924 Distance to Cluster 2 center: sqrt((1.938 - 5.02)^2 + (1.328 - 3.02)^2) = 3.8767  Since the distance to Cluster 1 is smaller, Data 5 is assigned to Cluster 1.  Data 6: Distance to Cluster 1 center: sqrt((1.938 - 5.02)^2 + (1.328 - 3.02)^2) = 3.1127 Distance to Cluster 2 center: sqrt((1.938 - 5.02)^2 + (1.328 - 3.02)^2) = 3.1127  Since the distance to Cluster 2 is smaller, Data 6 is assigned to Cluster 2.  After the second iteration, the cluster assignments are as follows:  Data# Cluster Assignment after 1st Iteration Cluster Assignment after 2nd Iteration 1 Cluster 1 Cluster 1 2 Cluster 1 Cluster 1 3 Cluster 1 Cluster 1 4 Cluster 1 Cluster 1 5 Cluster 1 Cluster 1 6 Cluster 2 Cluster 2  After the second iteration, Data 6 is correctly assigned to Cluster 2. |
|  |

**Question # 5 (2+2+2+2 Marks)** **……………………………….…….…….…….……..……(CLO-3)**

A KNN classifier assigns a test instance the majority class associated with its K nearest training instances. Distance between instances is measured using Euclidean distance. Suppose we have the following training set of positive (+) and negative (-) instances and a single test instance (o). All instances are projected onto a vector space of two real-valued features (X and Y). Answer the following questions. Assume “unweighted” KNN (every nearest neighbor contributes equally to the final vote).



1. How is training performed in KNN?

|  |
| --- |
| **In k-nearest neighbors (KNN) classification, the training is relatively simple compared to other machine learning algorithms. The algorithm essentially memorizes the entire training dataset, as it doesn't build a specific model during the training phase. Here are the steps involved in training a KNN classifier:**  **Store the Training Data:**  **The training phase in KNN involves storing the entire training dataset. Each data point in the training set consists of features (X, Y in your case) and the corresponding class label (positive or negative).**  **No Explicit Training:**  **Unlike other algorithms (e.g., decision trees, neural networks) that learn a model during training, KNN doesn't explicitly learn a model. Instead, it stores the entire training dataset.**  **Data Normalization (Optional):**  **Normalization of features is often performed to ensure that all features contribute equally to the distance calculations. This step is optional, and its necessity depends on the nature of the features and the distance metric used.**  **The training process in KNN is essentially the preparation phase. Once the training set is stored, the algorithm is ready to classify new, unseen instances based on their proximity to the instances in the training set.**  **During classification (testing or prediction), the algorithm calculates the distance between the test instance and all instances in the training set. It then selects the k-nearest neighbors based on the calculated distances and assigns the majority class among these neighbors to the test instance.**  **The training phase in KNN is minimalistic, involving the storage of the training dataset. The majority of the work occurs during the testing phase when making predictions for new instances.** |

1. What would be the class assigned to this test instance for K=1

|  |
| --- |
| KNN assigns a test instance the target class associated with the majority of the test instance’s K nearest neighbors. For K=1, this test instance would be predicted negative because it’s single nearest neighbor is negative |

1. What would be the class assigned to this test instance for K=5

|  |
| --- |
| KNN assigns a test instance the target class associated with the majority of the test instance’s K nearest neighbors. For K=5, this test instance would be predicted positive. Out of its five nearest neighbors, two are negative and three are positive |

1. Setting K to a large value seems like a good idea. We get more votes! Given this particular training set, would you recommend setting K = 11? Why or why not?

|  |
| --- |
| 1. There are only 5 negative instances in the training set. Therefore, any value of K > 10 would have a majority of positive instances. |

**Question # 6 (2+2+2+2 Marks) ……………………………….…….…….…….……..……(CLO-1)**

1. The Back-Propagation learning algorithm for training feed-forward neural networks requires the activation and error functions to be differentiable. Explain what that means and why it is true ?

|  |
| --- |
| The Back-Propagation (BP) learning algorithm is a widely used method for training feed-forward neural networks. In the context of BP, the requirement for activation and error functions to be differentiable is crucial. Let's break down what this means and why it is true:  Activation Function:  Neural networks consist of interconnected nodes organized in layers. Each node applies an activation function to its input to produce an output. Common activation functions include sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU).  Differentiability refers to the ability to calculate the derivative of the activation function at any given point. The derivative represents the rate of change of the function at that point.  In the context of BP, derivatives are essential for updating the weights of the network during the training process. The gradient of the activation function helps in determining the direction and magnitude of the weight adjustments that minimize the error.  Error Function:  The error function (also known as the loss or cost function) quantifies the difference between the predicted outputs of the neural network and the actual target values.  During the training process, the goal is to minimize this error by adjusting the weights of the network using optimization techniques like gradient descent.  Differentiability of the error function is crucial because BP relies on the chain rule of calculus to propagate errors backward through the network. The chain rule breaks down the total error with respect to the network's output into contributions from each layer, allowing for the calculation of gradients.  Why it is true:  The BP algorithm is based on the gradient descent optimization method. Gradient descent involves computing the gradient (derivative) of the error with respect to the network's weights. This gradient is used to update the weights in the opposite direction of the gradient to reduce the error.  If the activation or error functions were not differentiable, the gradient descent algorithm would not have a clear direction for weight adjustments, making it challenging to minimize the error systematically.  Differentiability enables the use of calculus and optimization techniques to find the minimum of the error function, leading to effective training of neural networks. |

1. Describe what is likely to happen when a learning rate is used that is too large, and when one is used that is too small. How can one optimize the learning rate?

|  |
| --- |
| **The learning rate is a hyperparameter in machine learning and neural network training that determines the size of the steps taken during the optimization process, such as gradient descent. Choosing an appropriate learning rate is crucial for achieving effective and efficient training. Here's what is likely to happen when a learning rate is too large or too small, and how one can optimize the learning rate:**  **Learning Rate Too Large:**  **Issues:**  **Overshooting:**  **With a large learning rate, the model might take excessively large steps during optimization, overshooting the minimum of the loss function.**  **This can lead to divergence, where the optimization process fails to converge to a minimum, and the parameters oscillate or explode.**  **Instability:**  **Large learning rates can make the optimization process highly sensitive to the initial conditions, leading to instability.**  **Consequences:**  **The model may fail to converge, leading to poor performance.**  **Oscillations or divergence might occur, preventing the model from learning effectively.**  **Learning Rate Too Small:**  **Issues:**  **Slow Convergence:**  **With a very small learning rate, the optimization process takes tiny steps, which can lead to slow convergence.**  **It might take a long time for the model to reach a satisfactory level of performance.**  **Getting Stuck in Local Minima:**  **A very small learning rate might cause the model to get stuck in local minima, preventing it from finding the global minimum of the loss function.**  **Consequences:**  **Slow training times, especially for large datasets or complex models.**  **Increased vulnerability to getting stuck in suboptimal solutions.**  **Optimizing the Learning Rate:**  **Grid Search:**  **Perform a grid search over a range of learning rates and select the one that results in the best performance on a validation set.**  **Learning Rate Schedulers:**  **Use learning rate schedules that dynamically adjust the learning rate during training. For example, decrease the learning rate over time to allow for more significant steps in the beginning and smaller steps as the optimization process progresses.**  **Use Adaptive Learning Rate Methods:**  **Algorithms like Adagrad, RMSprop, and Adam dynamically adjust the learning rate based on the past gradients, which can be more effective in different scenarios.**  **Early Stopping:**  **Monitor the performance on a validation set and stop training when the performance plateaus or starts to degrade. This can help avoid overfitting and prevent unnecessary updates with a potentially inappropriate learning rate.**  **Visualize Loss Curves:**  **Plot the loss curve during training to observe its behavior. If the loss is fluctuating wildly, it might indicate that the learning rate is too high.** |

1. Explain the main reasons why a Back-Propagation training algorithm might not find a set of weights which minimizes the training error for a given feed-forward neural network ?

|  |
| --- |
| Several reasons can contribute to the Back-Propagation training algorithm not finding a set of weights that minimizes the training error for a given feed-forward neural network. Here are some of the main reasons:   1. **Local Minima:**    * The optimization landscape of the error surface may contain multiple local minima, where the gradient is zero but the error is not at its global minimum. Back-Propagation can get stuck in these local minima and fail to find the global minimum. 2. **Vanishing or Exploding Gradients:**    * In deep neural networks, gradients can become very small (vanishing gradients) or very large (exploding gradients) as they are propagated backward through the layers during training. This can make it difficult to update the weights effectively, especially in deep architectures. 3. **Poor Initialization:**    * The initial weights of the neural network play a crucial role in determining the convergence of the training algorithm. Poor initialization can lead to the network getting stuck in a suboptimal solution or converging very slowly. 4. **Saddle Points:**    * In addition to local minima, saddle points in the error surface can also pose challenges. At saddle points, the gradient is zero, but the point is not a true minimum. Back-Propagation may stall at saddle points, affecting convergence. 5. **Overfitting:**    * If the neural network is too complex or if the training dataset is too small, the model may overfit the training data, capturing noise rather than learning the underlying patterns. In such cases, the trained model may not generalize well to new, unseen data. 6. **Hyperparameter Settings:**    * Poor choices of hyperparameters, such as the learning rate, batch size, or regularization strength, can hinder the convergence of the Back-Propagation algorithm. Optimizing hyperparameters is a crucial part of achieving successful training. |

1. What are local minima, and why are they a problem? How might one improve the chances of finding the global minimum?

|  |
| --- |
| In the context of optimization problems, a local minimum is a point in the parameter space where the objective function (such as the error or loss function in the case of neural network training) has a lower value than its nearby points. At a local minimum, the derivative of the objective function is zero, indicating that there is no direction to move where the function decreases.  **Why They Are a Problem:** Local minima pose a challenge for optimization algorithms, such as Back-Propagation, because these algorithms rely on the gradient (derivative) of the objective function to update the parameters iteratively. If the optimization algorithm gets trapped in a local minimum, it may converge to a suboptimal solution and fail to find the global minimum, where the objective function has its lowest value.  **Improving the Chances of Finding the Global Minimum:**   1. **Initialization:**    * Use careful weight initialization techniques to start the optimization process closer to a region with a high probability of finding a good solution. Techniques like Xavier/Glorot initialization help prevent vanishing or exploding gradients. 2. **Learning Rate Scheduling:**    * Implement learning rate schedules that dynamically adjust the learning rate during training. This can help the algorithm take larger steps initially to escape shallow local minima and smaller steps later for fine-tuning. 3. **Momentum and Adaptive Learning Rate Methods:**    * Incorporate momentum or adaptive learning rate methods (e.g., Adam, RMSprop) to navigate through regions with varying curvature in the optimization landscape. These methods can help the optimization process escape from saddle points and navigate around local minima. 4. **Stochasticity:**    * Introduce stochasticity into the optimization process by using stochastic gradient descent (SGD) or mini-batch gradient descent. This can help the algorithm escape from local minima by introducing randomness in the parameter updates. |

|  |
| --- |
| **Rough Work** |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Best of Luck**