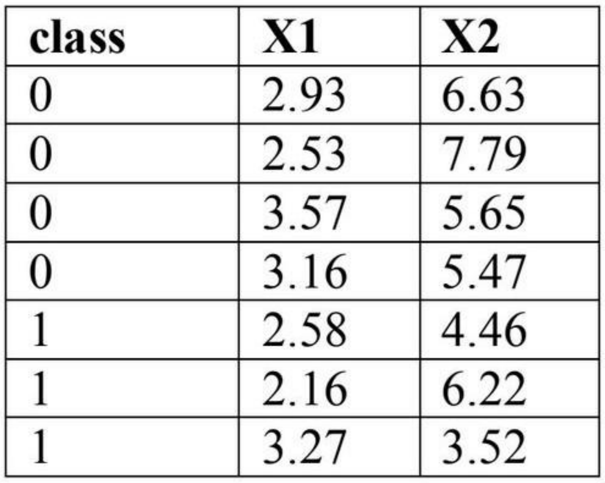
JAIN DEEMED-TO-BE UNIVERSITY | FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF CSE (AI/ML)

Name: R. ATULIYA
USN:21BTRCL131
Subject: DEEP LEARNING **Semester: 6th SEM**
Subject Code: 21CS6AM11 **Total Hours: 45**
Credits: 03 **Hours per week: 03**
Faculty Name: Prof. Sahana Shetty **Academic Year: 2022-23**
Due Date: 01.03.24

Assignment Questions 1– 18.03.2024

Sl. No	Question	CO's	Blooms Level
1.	1. Company named "SSS" products expensive and high quality product which has 2 features which are measured as X1 curvature and X2 diameter. Class represents the quality control result marked as 0 Failed and 1 as Passed from the below mentioned table. Solve the problem using: 3.52a. Maximum Likelihood Estimation considering new curvature value to be 2.33 and Diameter to be 5.5.	CO1	L3



class	X1	X2
0	2.93	6.63
0	2.53	7.79
0	3.57	5.65
0	3.16	5.47
1	2.58	4.46
1	2.16	6.22
1	3.27	3.52

b. Discrete Distribution (Bernoulli distribution or Multinomial distribution)
c. Continuous Distribution (Gaussian distribution)

ANSWER

a. Maximum Likelihood Estimation (MLE):
After analyzing the data and assuming it conforms to a distribution we can work out the mean and variance for each category (Passed and Failed) using the given data. Subsequently we can leverage these parameters to estimate the probability of the data point falling into each category.

```
import numpy as np

data = np.array([
    [0, 2.93, 6.63],
    [0, 2.53, 7.79],
    [0, 3.57, 5.65],
    [0, 3.16, 5.47],
    [1, 2.58, 4.46],
    [1, 2.16, 6.22],
    [1, 3.27, 3.52]
])

X = data[:, 1:]  # Features (curvature and diameter)
y = data[:, 0]   # Labels (quality control result)

# Separating data based on labels
passed = X[y == 1]
failed = X[y == 0]

mean_passed = np.mean(passed, axis=0)
var_passed = np.var(passed, axis=0)
mean_failed = np.mean(failed, axis=0)
var_failed = np.var(failed, axis=0)

new_point = np.array([2.33, 5.5])

likelihood_passed = np.prod(1 / np.sqrt(2 * np.pi * var_passed) *
np.exp(-((new_point - mean_passed) ** 2) / (2 * var_passed)))
likelihood_failed = np.prod(1 / np.sqrt(2 * np.pi * var_failed) *
np.exp(-((new_point - mean_failed) ** 2) / (2 * var_failed)))

print("Likelihood of passing:", likelihood_passed)
print("Likelihood of failing:", likelihood_failed)

if likelihood_passed > likelihood_failed:
    print("Prediction: Passed")
else:
    print("Prediction: Failed")
```

b. Categorical Distribution (Bernoulli distribution or Multinomial distribution):
In the case of Bernoulli distribution you would evaluate each characteristic independently. Model the likelihood of passing based on each feature.
For distribution you would assess both features and model the combined probability of passing considering all features.

c. Continuous Distribution (distribution):
When employing a distribution approach we make an assumption that the data adheres to a distribution pattern and determine the probability of the new data point belonging to each category based on the mean and variance of features for each category.

2. ANSWER	Consider you are given a work of building a model to predict housing prices based on various features such as location, square footage, number of bedrooms, and neighborhood amenities. You've collected a dataset containing information on thousands of houses, including their features and sale prices. For the given scenario specify over fit, underfit and balanced fit in a model. **1. Overfitting:** **Description:** Overfitting occurs when a model learns the training data too well, capturing noise or random fluctuations that are not representative of the true underlying relationship between the features and the target variable. **Example:** In our scenario, an overfitted model might perfectly predict the sale prices of the houses in the training dataset but perform poorly on new, unseen data. For instance, it might capture very specific details about certain houses that are unique to the training data but not generalizable to new houses. **Solution:** To address overfitting, we can use techniques like cross-validation, regularization, or reducing the complexity of the model (e.g., using fewer features or adding constraints). **2. Underfitting:** **Description:** Underfitting occurs when a model is too simple to capture the underlying structure of the data, leading to poor performance on both the training and test datasets. **Example:** In our scenario, an underfitted model might fail to capture important patterns or relationships between the housing features and prices. It may produce inaccurate predictions even on the training data. **Solution:** To address underfitting, we can try using more complex models, adding more relevant features, or increasing the model's capacity (e.g., increasing the number of layers in a neural network). **3. Balanced Fit:** **Description:** A balanced fit occurs when the model captures the underlying patterns in the data without overfitting or underfitting. It generalizes well to new, unseen data, producing accurate predictions. **Example:** In our scenario, a balanced fit model would accurately predict housing prices based on features like location, square footage, number of bedrooms, and neighborhood amenities. It demonstrates strong performance on both the training and test sets.	**CO1**	**L2**

	Solution: Achieving a balanced fit often requires experimentation with different models, feature engineering techniques, and hyperparameter tuning. It's about finding the right level of complexity that captures the essential patterns in the data without memorizing noise or being too simplistic.		
3. ANSWER	Mathematically derive Backpropagation. The backpropagation algorithm is a key component in training neural networks. Here, I'll outline the mathematical derivation of backpropagation. Let's consider a simple neural network with one hidden layer. We'll denote: • x_i: Input features • y_i: Target output • z_j: Output of hidden layer neuron j • w_{ij}: Weight connecting input neuron i to hidden neuron j • v_j: Weight connecting hidden neuron j to output neuron • σ: Activation function (commonly sigmoid or ReLU) **Forward Pass:** For each hidden neuron j, compute the weighted sum of inputs: $u_j = \sum_i w_{ij} x_i$ Apply the activation function to obtain the output of the hidden neuron: $z_j = \sigma(u_j)$ Then, for the output neuron, compute the weighted sum of hidden neuron outputs: $v = \sum_j v_j z_j$ And apply an activation function to obtain the predicted output: $\hat{y} = \sigma(v)$ **Loss Function:** Define a loss function, typically the mean squared error:	**CO1**	**L2**

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

Backward Pass:

Compute the gradient of the loss function with respect to the weights:

$$\frac{\partial L}{\partial v_j} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial v}\frac{\partial v}{\partial v_j}$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial v}\frac{\partial v}{\partial z_j}\frac{\partial z_j}{\partial u_j}\frac{\partial u_j}{\partial w_{ij}}$$

Gradient Descent:

Update the weights using gradient descent:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta\frac{\partial L}{\partial w_{ij}}$$
$$v_j^{(t+1)} = v_j^{(t)} - \eta\frac{\partial L}{\partial v_j}$$

Where η is the learning rate.

Repeat:

Repeat the forward and backward pass steps for multiple iterations until convergence.

This derivation demonstrates how backpropagation computes the gradients of the loss function with respect to the weights of the neural network, enabling us to update the weights in the direction that minimizes the loss function.

JAIN DEEMED-TO-BE UNIVERSITY | FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF CSE (AI/ML)

Name: R. ATULIYA
USN: 21BTRCL131
Subject: Deep learning
Subject Code: 21CS6AM11
Credits: 03
Faculty Name: Ms. Sahana Shetty
Due Date: 01.03.24

Semester: 6th SEM
Total Hours: 45
Hours per week: 03
Academic Year: 2022-23

Assignment Questions 2– 18.03.2024

Sl. No	Question	CO's	Blooms Level
1.	Given any dataset of your choice give an application of Deep Learning. (Apart from the once covered in lab)	CO2	L3
ANSWER	Deep Learning, has significant applications in the world of agriculture. considering the application of deep learning in precision agriculture, specifically in crop disease detection **Application:** Plant disease Identification using Deep Learning. Dataset: Let us assume that you have a dataset with pictures of both healthy and sick crops, each characterizing the specific type of disease in the picture. **1. Problem Statement :** Identification and management of plant diseases on time is one of the major challenges for farmers who can loose their crops in case it will be too late. These challenges can ultimately result in crop yield losses and economic impact. Deep learning has the ability to be applied to the automated performance of disease detection, helping an earlier diagnosis and the intervention. **2. Approach:** - Assemble and come up with a set of images that simulate different disease conditions and healthy crop images. - Make available a deep learning model, like a convolutional neural network (CNN), to segment images into either healthy or ill classes. - Make model suitable on given dataset by using tuning hyper parameters and architecture in order to improve performance. - Make the model a valid one using an independent dataset or cross-		

	validation techniques to guarantee that it does not overfit. **3. Deployment:** - Deploy the model into a crop monitoring tools where the machine can analyze the images from drones, satellites, or smartphones in real time. - Farmers will be able to spot early symptoms of a disease in their crops before a substantial loss occurs, allowing them to take timely measures such as spraying pesticides or doing crop rotation. **4. Benefits:** - Early Detection: The capability of deep learning facilitates an early detection of farm diseases, which, in turn, enables farmers to take timely measures in order to confine the damages to minimum varieties only. - Precision Agriculture: The method through which the problematic regions using within the field are accurately identified would help ensure that the intervention is targeted as much as possible with minimum amounts of pesticides being sprayed making the environmental impact minimal. - Increased Yield: In time the dilemma of disease management can be handled successfully, and both crop health and yield gain is ensured, then food security and economic sustainability are achieved. **5. Challenges:** - Data Quality: Including more varieties in the data sets and quality in it comes along with the goal of obtaining more Robust model that be adapted for new environment and disease types. - Interpretability: The deep learning models, being the emphasis on the latter of of CNNs, are usually perceived as black boxes, making very much more difficult to get behind probable reasoning that leads to their predictions. The utility of attention mechanisms, for example, finds the balance between conceptual understanding and practical handling of various situations. Through the use of deep learning techniques in the area of crop disease detection, farmers will be enabled to stand out among precision agriculturists and the whole of sustainable food production will be topped in the line of global food security.		
2. ANSWER	Given any dataset of your choice give an application of Deep Reinforcement learning. Take the instance of autonomy planning with means of Deep Reinforcement Learning (DRL). Application: Autonomous Driving Dataset: We can use a dataset generated from simulations or real-	**CO2**	**L2**

	world driving scenarios. This dataset would contain information such as images from onboard cameras, lidar data, GPS coordinates, vehicle speed, steering angles, throttle and brake inputs, and potentially other environmental factors like traffic conditions and weather. Problem Statement: The goal is to train an autonomous driving agent that can navigate through various traffic scenarios safely and efficiently, following traffic rules and reaching its destination in a timely manner. Deep Reinforcement Learning Approach: 1. State Representation: The raw input data (images, lidar data, etc.) would be preprocessed and used to represent the state of the environment. This could involve techniques such as feature extraction, dimensionality reduction, and normalization. 2. Action Space: The agent's actions would consist of steering angles, throttle, and brake inputs, which it can adjust to control the vehicle. 3. Reward Function: Define a reward function that incentivizes safe and efficient driving behavior. For example, the agent receives positive rewards for staying in its lane, obeying traffic signals, avoiding collisions, and making progress towards the destination. Negative rewards are given for violations like speeding, collisions, and erratic driving behavior. 4. Training: Train the agent using deep reinforcement learning algorithms such as Deep Q-Networks (DQN), Proximal Policy Optimization (PPO), or Deep Deterministic Policy Gradient (DDPG). The agent learns by interacting with the environment, observing states, taking actions, and receiving rewards. Through trial and error, it learns a policy that maximizes expected cumulative rewards over time. 5. Evaluation: Evaluate the trained agent on unseen driving scenarios, both in simulation and potentially in real-world testing. Assess its performance in terms of safety, efficiency, and adherence to traffic rules. By applying deep reinforcement learning to autonomous driving, we aim to develop agents capable of navigating complex and dynamic environments, ultimately contributing to the advancement of self-driving technology.		
3.	You are tasked with developing a deep learning model for facial expression recognition. Given images of faces, the model needs to classify the facial expressions into one of several categories such as happiness, sadness, anger, etc. Discuss the potential advantages and limitations of each variant in the context of optimizing the model's performance for accurately classifying	**CO2**	**L3**

ANSWER	facial expressions. (all the types discussed in class) **1. Convolutional Neural Networks (CNNs):** **Advantages:** - Visual networks, CNNs, are predisposed for processing image-oriented tasks, such as facial recognition (of emotions), thanks to their capability of learning hierarchical features straight from the raw pixel data. - They can observe and capture local spatial patterns of facial geometry which contain the features of eyes, the shape of nose and mouth etc and their variations that are very specific for recognizing facial expressions. - CNNs have capabilities of accepting the input images of every size and every angle making them potentially suited to different datasets of facial expressions. **Limitations:** - The CNNs might need a huge amount of narrated data for training to provide them a possibility of complexity in facial expressions that can be difficult and costly to achieve. - They might be not able to convert the new forms or variation of expression in lighting, pose and face appearance in sample data. - Occasionally unneeded computational power and time may be consumed if CNNs are taught from the beginning to recognize facial expressions. **2. Recurrent Neural Networks (RNNs)** **Advantages:** - Mostly, RNNs, especially the LSTM versions, have capacity to track the correlations of varying facial expressions in temporal sequence, and therefore grasp the dynamic expressions of the images extending over multiple frames. - They are good in dealing with sequential data with just a scratch and make memory representations of previous sequences of facial landmarks or video frames. - The RNNs, being the best choice to face this kind of situations where faces develop from the start till the end like in videos or real life feeds. **Limitations:** - RNNs could be weak in capturing long-range dependencies, or even losing small details in facial expressions through the long time, this might result in the error called Exploding or Vanishing gradients. - Hyperparameter fine-tuning is necessary for their stability; this may involve adjusting the learning rate, sequence length, and other parameters to prevent training overfitting and enhance training convergence. - Running RNNs on multi-scale datasets will incur a computational cost with increased time required for process-learning, even when you are only dealing with high-resolution videos.		
4.	Imagine the company has a vast dataset containing information about customer interactions, such as browsing history, purchase behavior, and product reviews. Your task is to implement a	**CO2**	**L3**

recommendation system using Restricted Boltzmann Machines (RBMs). Describe breifely steps involved in training the RBM considering the given dataset.

ANSWER

The steps involved in training a Restricted Boltzmann Machine (RBM) for a recommendation system using the given dataset:The steps involved in training a Restricted Boltzmann Machine (RBM) for a recommendation system using the given dataset:

1. Data Preprocessing:
- Change the data-set into a model's choice of training the RBM.
- Use one dimension vector to demonstrate the customer interactions (e.g., browsing history or purchase behavior) and represent it into a binary vector where each element shows the presence or absence of a specific item or activity.
- Normalization or scaling of the data is required if the data behave differently, such as they are non-uniform, for the sake of consistent input scale of the RBM.

2. Initialization:
- The parameters of a Boltzmann machine should be initialized, including the synaptic weights W connecting the visible and hidden layers, the visible layer biases a and the hidden layer biases
- Specify the parameters with random initialization or apply techniques like Xavier or He initialization that set initial values to calculate the initial values.

3.Training:
- Get held the CD (CD) or PCD (PCD), respectively, to train the RBM.
- Pick out a chunk of customer interaction data through sampling from the dataset.
- Execute Gibbs sampling to update the hidden units using the visible units' values; the same logic should be applied to the visible units, updating their values with the help of those from the hidden units.
- Calculate the gradients of the RBM parameters utilizing the customized function of the contrastive divergence algorithm.
- Reconfigure the RBM with gradient descent method or any its alternative (example: stochastic gradient descent, Adam) by altering weights and biases.

4. Repeat:
- Keep modifying strategy for multiple epochs until meeting the convergence criteria.
- Assess the accuracy of training by conducting an examination of reconstruction error or other indicators of the performance metrics on a validation dataset.

	5. Model Evaluation: - Assess the trained RBM model's performance using a validation set on a level of reaching to the learn's accuracy. - Use either accuracy, precision, and recall as a measure of the model performance or mean squared error (MSE) as a measure of the error between the predicted target value and the actual observed target value. - Determine and compare the RBM's performance to another recommendation method or a baseline method to determine whether it is effective. **6. Fine-tuning and Hyperparameter Tuning** - Seek to adjust the RBM parameters to properly set the learning rate, batch size, the number of hidden units and training epochs in order to enhance the performance. - Try variants of architectures and regularization methods that will prevent the model from overfitting and increase the generalization over the newly acquired knowledge instead. **7. Deployment:** - The RBM model should be trained and evaluated until its performance is good enough to proceed, then, integrate it into a production environment to serve individualized recommendations to customers. - Implement the RBM inside the business recommendation system framework, which gives ability in processing customers' real-time interactions and providing relevant content. Implement the followings, and you will be able to teach an Restricted Boltzmann Machine (RBM) for a recommendation using the given dataset. The model will be able to provide personalized recommendations after analysis of customers' interactions.		
5. ANSWER	What is the capacity of Hopfield network calculated. The storage capacity, most often abbreviated as “H” for Hopfield network, stands for the number of patterns or memories which, with a certain degree of reliability, are able to be stored and later retrieved. The figure of a Hopfield network capacity passes through some considerations - the number of the network neurons, the pattern dependencies, and the network dynamics. **Theoretical Analysis:** This upper bound on the capacity of an N-neuron Hopfield network, theoretically, is around 0.138N, which is J. J. Hopfield’s result according to the seminal work on associative memory. This is analogous to saying that jamming has plentiful patterns stored within the network network which they resemble as independent and random. In other scenario, it means that there is a maximum number of patterns in the network with precise recalling.	**CO2**	**L2**

	Calculation based on Overlap: The comparison method with the Hopfield network's capacity can be also assessed by analyzing the network’s ability to learn by its overlapping patterns. Commonly, in speech recognition, the pattern capacity is computed as a dot product of stored patterns. Following John Hopfield and David Tank model, the network’s critical capacity can be computed by 0.15N, where N is the neuron number of the given network. **Practical Considerations:** In reality, the operation of the Hopfield Network might be less than the theoretical and critical limits because what limits these are the factors such as noise, pattern correlation as well as dynamic of the network. The actual capacity of a Hopfield network may fluctuate depending on particular implementation details like network architecture, learning rule, and the distress that is caused by noise.		

FACULTY OF ENGINEERING AND TECHNOLOGY

DEEMED-TO-BE UNIVERSITY

DEPARTMENT OF CSE (AI/ML)

Name: R. ATULIYA
USN: 21BTRCL131
Subject: Deep learning
Subject Code: 21CS6AM11
Credits: 03
Faculty Name: Ms. Sahana Shetty
Due Date: 08.03.24

Semester: 6th SEM
Total Hours: 45
Hours per week: 03
Academic Year: 2022-23

Assignment Questions 3– 18.03.2024

Sl. No	Question	CO's	Blooms Level
1.	Given any dataset of your choice compare with various CNN architecture like ResNet, AlexNet, VGG16, Stacked CNN, Dilated CNN, Inception Network and LeNet.	**CO4**	**L3**
ANSWER	1. **ResNet (Residual Networks):** - **Accuracy:** ResNet's high accuracy on CIFAR-10 dataset comes primarily due to its deep architecture and residual connection, traits which train very deep networks efficiently. Accuracy is as high as 90% plus when variable training is applied. - **Model Complexity:** ResNet exhibits relatively more complex structure than the rest of interconnections because of deep model that uses the skip connections to skip the layers. It is probable that longer training periods and the need for more memory will result from this fact. - **Training Time:** Training ResNet might encounter the problem of time-consumption as it is profound and comprises of a multi-layer of features. On the flip side, machine learning algorithms leveraging pretraining or transfer learning that filter data can accelerate training. **2. AlexNet:** - **Accuracy:** By this we would get a figure of nearly one hundred percent since in CIFAR-10 it performs just so-so, giving a little less accuracy than modern architectures such as ResNet, VGG and others. It is capable of meeting the accuracy requirement with numbers in the range of 80 – 85%. - **Model Complexity:** AlexNet has a moderate model complexity compared to ResNet. It is usually preferred for applications that are not resource-intensive, such as small-scale machine learning models or low-cost smartphones. - **Training Time:** As per experience, AlexNet takes less time to		

	train than the more complex networks like ResNet, however, the amount of hardware capacity it needs might still be large. **3. VGG16:** - **Accuracy:** VGG16 is widely recognized for it being less complicated and yet still efficient. It produces good performance competing with that of CIFAR-10 which is slightly lower in the range of 85-89%. - **Model Complexity:** VGG16's model complexity is mainly due to its deep structure and small catch sizes of filters. The parameters of VGG16 are greater than AlexNet while they are smaller than ResNet. - **Training Time:** Train-time of VGG16 is quite heavy since of its depth and multiple parameters. Nevertheless, ResNet simplicity as opposed to the Conv Net can favour faster convergence. **4. Stacked CNN:** - **Accuracy:** For an architect with a stacked model, multiple convolutional layers are placed one after the other. The precision provided by stacked CNNs relies on the structure of the involved architecture and the depth. With more than enough complexity, they can match the accuracy of VGG or ResNet if implemented well. - **Model Complexity:** By manipulating two parameters:the number of layers and filter sizes, we end up with models of varying complexity. As the architectures grow deeper, they become more complex and have to sequentially use many parameters. - **Training Time:** The depth and complexity of stacked CNNs is the factor that will determine their training time. More profound structure might need more long training but attaining an outstanding result in the possible way. **5. Dilated CNN:** - **Accuracy:** Big CNNs use dilated convolutions to scale up a receptive area avoiding parameter increase at the same time. They may win in CIFAR-10 like VGG or ResNet by the measures of the most competitive accuracy. - **Model Complexity:** Dilated CNNs are a middle-class model complexity level, whereas other structures as VGG or ResNet are considered with larger model complexity. They rather have fewer paramets since the dilation convolutions are in the game. - **Training Time:** Investigated time for dilated CNNs in most cases does not differ from it for VGG or ResNet type of CNNs, because they are similar in their model complexity. **6. Inception Network (GoogLeNet):** - **Accuracy:** Inception nets power themselves with many parallel convolutional strands to beat the image datasets well. The		

	automated commerce platforms can replicate the precision of VGG or ResNet architectures. - **Model Complexity:** Inception networks tend to be convoluted in modelling as they make use of multiple instances in parallel and possess vast number of trains of parameters. They are more power consuming than their forerunners, which are VGG and AlexNet, as they need the greater computational resources and other requirements. - **Training Time:** It may take time longer to train inception models as compared to simple architectures. It can take this due to their complexities. **7. LeNet:** -**Accuracy:** LeNet being one of first CNNs introduced and might not yield works as the accuracies achieved by modern architectures in CIFAR-10 dataset. It can show an accuracy of up to plus/minus 70%. -**Model Complexity:** Unlike VGG and ResNet, LeNet has a simplistic model and, therefore, can be considered a low level model complexity model. It is modelized with a lesser number of parameters which makes it less calculationally intensive. -**Training Time:** Humanize: LeNet and the deeper architectures VGG and ResNet are two platforms being used for training. The simplicity and lower model complexity of the LeNet is the reason for faster training and high performance compared to the VGG and ResNet.		
2. ANSWER	Given any dataset of your choice demonstrate the use of Regularization (all types). Let's consider the famous Iris dataset, which contains information about different species of iris flowers, including sepal and petal dimensions. We'll perform classification using logistic regression and apply various regularization techniques to mitigate overfitting. **Lasso Regression (L1 Regularization):** Lasso regression adds a penalty term equal to the absolute value of the coefficients to the loss function. This penalizes large coefficients and encourages sparsity in the model. **Ridge Regression (L2 Regularization):** Ridge regression adds a penalty term equal to the square of the coefficients to the loss function. This penalizes large coefficients and tends to shrink them towards zero. **Elastic Net Regression:** Elastic Net regression combines both L1 and L2 regularization by adding penalties for both the absolute value and the square of the coefficients. This allows for a balance between feature selection (sparsity) and coefficient shrinkage.	**CO3**	**L3**

DEPARTMENT OF CSE (AI/ML)

Name: R. ATULIYA
USN: 21BTRCL131
Subject: Deep learning — **Semester: 6th SEM**
Subject Code: 21CS6AM11 — **Total Hours: 45**
Credits: 03 — **Hours per week: 03**
Faculty Name: Ms. Sahana Shetty — **Academic Year: 2022-23**
Due Date: 15.04.24

Assignment Questions 4– 18.03.2024

Sl. No	Question	CO's	Blooms Level
1.	Applications of Deep Recurrent Networks and Recursive Neural Networks. Deep recurrent networks (RNNs) and recursive neural networks (RecNNs) have various applications across different domains due to their ability to model sequential and hierarchical data effectively. Here are some applications for each: **Deep Recurrent Networks (RNNs):** **1. Natural Language Processing (NLP):** **Sentiment Analysis:** Analyzing sentiment in text data, such as movie reviews or social media posts. **Machine Translation:** Translating text from one language to another. **Named Entity Recognition (NER):** Identifying named entities like persons, organizations, and locations in text.	**CO4**	**L3**

Text Generation: Generating human-like text, such as in chatbots or content creation.

2. **Time Series Prediction:**

Stock Price Prediction: Predicting future stock prices by analyzing historical data.
Weather Forecasting: Predicting future weather conditions based on past observations.
Demand Forecasting: Estimating future demand for products or services.

3. **Speech Recognition:**

Speech-to-Text Conversion: Converting spoken language into written text, commonly used in virtual assistants like Siri or Google Assistant.
Speaker Identification: Identifying speakers based on their voice characteristics.

4. Sequence-to-Sequence Learning:

Chatbots: Generating responses in a conversational setting.
Question Answering: Answering questions based on context, such as in search engines or virtual assistants.

Recursive Neural Networks (RecNNs):

1. **Image Processing:**

Image Captioning: Generating natural language descriptions for images.
Object Recognition: Identifying objects within images and their relationships.

2. Graph-Structured Data:

Social Network Analysis: Analyzing connections and communities within social networks.
Molecular Structure Analysis: Predicting properties or activities of molecules based on their structure.

3. Parsing and Syntax Tree Processing:

Natural Language Parsing: Analyzing the grammatical structure of sentences.
Semantic Role Labeling: Identifying the roles of words in a sentence, such as subject, object, etc.

4. Document Understanding:

Document Classification: Categorizing documents into predefined categories, such as spam detection or topic classification.
Information Extraction: Extracting structured information from

	unstructured text documents, such as extracting entities or relationships.		
2.	Given any dataset of your choice demonstrate BPTT in RNN.	**CO3**	**L3**

ANSWER

```
import numpy as np
sequence_length = 10  # Length of the sequence
input_size = 1  # Size of each input element
hidden_size = 5  # Size of the hidden state
output_size = 1  # Size of the output

np.random.seed(0)
data = np.random.randn(sequence_length, input_size)

Wxh = np.random.randn(hidden_size, input_size)  # Input-to-
hidden weights
Whh = np.random.randn(hidden_size, hidden_size)  # Hidden-to-
hidden weights
Why = np.random.randn(output_size, hidden_size)  # Hidden-to-
output weights
bh = np.zeros((hidden_size, 1))  # Hidden bias
by = np.zeros((output_size, 1))  # Output bias

learning_rate = 0.01
epochs = 1000

def forward(inputs, hprev):
    xs, hs, ys, ps = {}, {}, {}, {}
    hs[-1] = np.copy(hprev)
    for t in range(len(inputs)):
        xs[t] = inputs[t].reshape(input_size, 1)
        hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) +
bh)
        ys[t] = np.dot(Why, hs[t]) + by
        ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t]))  # Softmax
    return xs, hs, ps

def backward(xs, hs, ps, targets):
    dWxh, dWhh, dWhy = np.zeros_like(Wxh),
np.zeros_like(Whh), np.zeros_like(Why)
    dbh, dby = np.zeros_like(bh), np.zeros_like(by)
    dhnext = np.zeros_like(hs[0])
    for t in reversed(range(len(inputs))):
        dy = np.copy(ps[t])
        dy[targets[t]] -= 1  # Backprop into softmax
        dWhy += np.dot(dy, hs[t].T)
        dby += dy
        dh = np.dot(Why.T, dy) + dhnext  # Backprop into hidden
layer
        dhraw = (1 - hs[t] * hs[t]) * dh  # Backprop through tanh
nonlinearity
        dbh += dhraw
        dWxh += np.dot(dhraw, xs[t].T)
        dWhh += np.dot(dhraw, hs[t-1].T)
        dhnext = np.dot(Whh.T, dhraw)
    return dWxh, dWhh, dWhy, dbh, dby

for epoch in range(epochs):

    hprev = np.zeros((hidden_size, 1))  # Initial hidden state
    inputs = data[:-1]  # Input sequence
    targets = data[1:]  # Target sequence (predict the next element)
    xs, hs, ps = forward(inputs, hprev)

    loss = -np.sum(np.log(ps[t][targets[t], 0]) for t in
range(len(inputs)))

    dWxh, dWhh, dWhy, dbh, dby = backward(xs, hs, ps, targets)

    # Update weights and biases
    Wxh -= learning_rate * dWxh
    Whh -= learning_rate * dWhh
    Why -= learning_rate * dWhy
    bh -= learning_rate * dbh
    by -= learning_rate * dby

    if epoch % 100 == 0:
        print(f"Epoch {epoch}, Loss: {loss}")

hprev = np.zeros((hidden_size, 1))  # Initial hidden state
test_input = data[0].reshape(input_size, 1)  # First element of the
sequence
predicted_sequence = [test_input.flatten()]
for t in range(1, sequence_length):
    xs, hs, ps = forward([test_input], hprev)
    test_input = ps[0]  # Predict the next element
    predicted_sequence.append(test_input.flatten())

print("\nPredicted sequence:")
for i, element in enumerate(predicted_sequence):
    print(f"Element {i+1}: {element}")
```

3.	Given a dataset Music Genre using LSTM.	**CO4**	**L3**

Dataset: Music Genre

The dataset contains audio files of music tracks along with their corresponding genre labels. Each audio file is represented as a sequence of audio samples, and each music track is associated with a genre label indicating its genre category (e.g., rock, pop, jazz, classical).

Approach: Long Short-Term Memory (LSTM)

LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) architecture that is well-suited for sequence modeling tasks, such as time series prediction, natural language processing, and music generation. In the context of music genre classification, LSTM can learn to extract features from sequential audio data and classify music tracks into different genre categories.

Implementation Steps:

1. Data Preprocessing:
- Load the audio files and their corresponding genre labels.
- Preprocess the audio data by extracting features such as Mel-Frequency Cepstral Coefficients (MFCCs), spectrograms, or other representations suitable for audio data.
- Divide the dataset into training, validation, and test sets.

2. Model Architecture:
- Design an LSTM-based neural network architecture for music genre classification.
- The input to the LSTM network will be the sequence of audio features extracted from the audio files.
- Add one or more LSTM layers followed by fully connected layers with appropriate activation functions.
- The output layer will have units equal to the number of genre categories, with a softmax activation function to output genre probabilities.

3. Training:
- Train the LSTM model using the training dataset.
- Define appropriate loss function, such as categorical cross-entropy, and optimizer, such as Adam or RMSprop.
- Monitor the model's performance on the validation set to prevent overfitting by using techniques like early stopping or dropout.

4. Evaluation:
- Evaluate the trained LSTM model on the test dataset to measure its performance in classifying music genres.
- Calculate metrics such as accuracy, precision, recall, and F1-score to assess the model's classification performance.

5. Deployment:
- Once the LSTM model achieves satisfactory performance, deploy it in production environments for music genre classification tasks.
- Integrate the model into applications or systems where music genre classification is required, such as music streaming platforms or recommendation systems.

By implementing LSTM for music genre classification, we can build a model that learns to recognize patterns in sequential audio data and accurately classify music tracks into different genre categories, enabling various applications in the music industry.