

**Artificial Intelligence**

**Module 1 and 2 Assignment**

[support@intellipaat.com](mailto:support@intellipaat.com)

+91-7022374614

US: 1-800-216-8930 (Toll-Free)

**Assignment Questions**

.

**Tasks to be performed:**

1. Define a use case of Deep Learning in your current domain (if fresher, pick any of your favorite) and propose a solution through DL.

*Consider the Domain accounting.Deep Learning can be used in the domain of accounting to improve various processes, such as invoice processing, financial statement analysis, and fraud detection. One specific use case of DL in accounting is the prediction of cash flows.*

*Solution: One solution using deep learning could be to use a recurrent neural network (RNN) to predict cash flows based on historical data. An RNN is a type of neural network that is well-suited for processing sequential data such as time series data. In this case, the input data would be historical cash flow data, and the RNN would be trained to predict future cash flows.*

*To improve the accuracy of the model, it can be trained on a large dataset of historical cash flow data. Additionally, other relevant data such as macroeconomic indicators, industry trends, and company-specific data can be included to improve the accuracy of the predictions.*

*Once the model is trained, it can be used to predict future cash flows for budgeting and forecasting purposes. This can help businesses to plan for future expenses and investments more accurately and efficiently. Furthermore, the model can be updated periodically to incorporate new data and adjust the predictions as necessary.*

*Overall, using deep learning to predict cash flows can help businesses to make more informed financial decisions and improve their financial performance.*

1. A steel manufacturing plant is spending many hours in manual inspection of their steel product(flat 2\*2 ft steel plates) The inspection is a crucial part of their delivery cycle.
2. Design and detail out a proposal to automate the process using DL
3. Please, explain what will be your major questions to client regarding the process & data

*Design proposal for automating steel plate inspection using Deep Learning:*

*Objective: To develop a DL model that can automate the inspection of steel plates in order to improve the efficiency of the inspection process, reduce manual errors, and ensure consistent quality of the final product.*

*Major questions to ask the client:*

* 1. *What are the key features or defects that need to be identified in the steel plates during inspection?*
  2. *How is the current inspection process carried out? Is it performed manually by human inspectors or using automated systems?*
  3. *What is the volume of steel plates that needs to be inspected daily or weekly?*
  4. *Is there a labeled dataset available for training the DL model, or does it need to be created?*
  5. *What are the constraints on the inspection system such as lighting conditions, positioning of the steel plates, or type of camera that can be used?*
  6. *Are there any specific quality control standards or regulations that the inspection process must comply with?*

*Proposal:*

* 1. *Data Collection: Collect a large dataset of images of steel plates with defects labeled according to the client's requirements.*
  2. *Data Preprocessing: Preprocess the data to standardize the images, resize them, and convert them to a suitable format for input to the DL model.*
  3. *DL Model Selection: Select a DL model suitable for image recognition tasks, such as a convolutional neural network (CNN).*
  4. *Model Training: Train the DL model using the labeled dataset of steel plate images to learn the features and defects that need to be identified during inspection.*
  5. *Model Testing: Test the trained DL model on a separate set of labeled images to evaluate its performance in identifying defects accurately.*
  6. *System Integration: Integrate the DL model into the existing inspection process using an automated camera system to capture images of the steel plates, which will be analyzed by the DL model.*
  7. *Deployment and Maintenance: Deploy the automated inspection system and provide ongoing maintenance and support to ensure its continued effectiveness.*

*The proposed solution will help the steel manufacturing plant to automate the inspection process, reducing manual labor and improving the accuracy and consistency of the inspection results.*

1. A particular linear process has an input (x=5) and output (y=10)
2. Create a model using numpy to solve for “y” and record your best error.

import numpy as np

# Define input x and output y

x = np.array([5])

y = np.array([10])

# Define the initial weight and bias

w = np.random.randn()

b = np.random.randn()

# Define the learning rate

lr = 0.01

# Define the number of epochs

epochs = 1000

# Training loop

for i in range(epochs):

    # Forward pass

    y\_pred = x \* w + b

    # Compute the mean squared error

    mse = np.mean((y - y\_pred)\*\*2)

    # Backward pass

    grad\_w = -2 \* np.mean((y - y\_pred) \* x)

    grad\_b = -2 \* np.mean(y - y\_pred)

    # Update the weights and bias

    w -= lr \* grad\_w

    b -= lr \* grad\_b

# Compute the final prediction and error

y\_pred = x \* w + b

mse = np.mean((y - y\_pred)\*\*2)

rmse = np.sqrt(mse)

print("Input x: ", x)

print("Output y: ", y)

print("Predicted y: ", y\_pred)

print("Root Mean Squared Error: ", rmse)

print("Weight: ", w)

print("bias: ", b)

1. Repeat the above experiment with different hyper parameters like Learning Rate & activation function.
2. Record the various hyperparameters used in experiment and submit the table with data (x | y | y\_pred | error (RMSE) | LR | activation used).

import numpy as np

# Define input x and output y

x = np.array([5])

y = np.array([10])

# Define the learning rates and activation functions to try

learning\_rates = [0.1, 0.01, 0.001]

activations = ['relu', 'sigmoid']

# Define the table headers

headers = ['x', 'y', 'y\_pred', 'error (RMSE)', 'LR', 'Activation']

# Define the table rows

rows = []

# Try all combinations of learning rates and activation functions

for lr in learning\_rates:

    for activation in activations:

        # Define the initial weight and bias

        w = np.random.randn()

        b = np.random.randn()

        # Define the number of epochs

        epochs = 1000

        # Training loop

        for i in range(epochs):

            # Forward pass with activation function

            if activation == 'relu':

                y\_pred = np.maximum(0, x \* w + b)

            elif activation == 'sigmoid':

                y\_pred = 1 / (1 + np.exp(-(x \* w + b)))

            else:

                raise ValueError("Invalid activation function")

            # Compute the mean squared error

            mse = np.mean((y - y\_pred)\*\*2)

            # Backward pass

            if activation == 'relu':

                grad\_w = -2 \* np.mean((y - y\_pred) \* x \* (y\_pred > 0))

            elif activation == 'sigmoid':

                grad\_w = -2 \* np.mean((y - y\_pred) \* x \* y\_pred \* (1 - y\_pred))

            else:

                raise ValueError("Invalid activation function")

            grad\_b = -2 \* np.mean((y - y\_pred))

            # Update the weights and bias

            w -= lr \* grad\_w

            b -= lr \* grad\_b

        # Compute the final prediction and error

        y\_pred = x \* w + b

        mse = np.mean((y - y\_pred)\*\*2)

        rmse = np.sqrt(mse)

        # Add the row to the table

        rows.append([x[0], y[0], y\_pred[0], rmse, lr, activation])

# Print the table

print("{:<10} {:<10} {:<10} {:<15} {:<10} {:<10}".format(\*headers))

for row in rows:

    print("{:<10} {:<10} {:<10} {:<15} {:<10} {:<10}".format(\*row))

1. Define the role of Learning Rate in Neural Networks and why they are important using an experiment?
2. Use an experiment like above one with LR and one without LR.#

*The learning rate is a hyperparameter that controls how much the weights and biases are updated in each iteration of the training process in a neural network. A larger learning rate can result in faster convergence, but it can also cause the optimization to overshoot the minimum and result in unstable training. A smaller learning rate can make the training more stable but it can take longer to converge. Therefore, it is important to choose an appropriate learning rate for the specific problem.*

*Here is an experiment to demonstrate the effect of learning rate on the performance of a neural network. We will use the same linear regression problem as in the previous example, but we will compare the performance of two models: one with a fixed learning rate and one without a learning rate.*

*First, let's define the model with a fixed learning rate:*

import numpy as np

import warnings

# suppress warnings

warnings.filterwarnings('ignore')

# Define input x and output y

x = np.array([5])

y = np.array([10])

# Define the initial weight and bias

w = np.random.randn()

b = np.random.randn()

# Define the number of epochs

epochs = 1000

# Training loop with learning rate

lr = 0.01

for i in range(epochs):

    # Compute the prediction

    y\_pred = x \* w + b

    # Compute the mean squared error

    mse = np.mean((y - y\_pred)\*\*2)

    # Compute the gradients

    grad\_w = -2 \* np.mean((y - y\_pred) \* x)

    grad\_b = -2 \* np.mean((y - y\_pred))

    # Update the weights and bias

    w -= lr \* grad\_w

    b -= lr \* grad\_b

# Compute the final prediction and error

y\_pred\_lr = x \* w + b

mse\_lr = np.mean((y - y\_pred\_lr)\*\*2)

rmse\_lr = np.sqrt(mse\_lr)

# Define the initial weight and bias

w = np.random.randn()

b = np.random.randn()

# Define the number of epochs

epochs = 1000

for i in range(epochs):

    # Compute the prediction

    y\_pred = x \* w + b

    # Compute the mean squared error

    mse = np.mean((y - y\_pred)\*\*2)

    # Compute the gradients

    grad\_w = -2 \* np.mean((y - y\_pred) \* x)

    grad\_b = -2 \* np.mean((y - y\_pred))

    # Update the weights and bias

    w -= grad\_w

    b -= grad\_b

# Compute the final prediction and error

y\_pred\_no\_lr = x \* w + b

mse\_no\_lr = np.mean((y - y\_pred\_no\_lr)\*\*2)

rmse\_no\_lr = np.sqrt(mse\_no\_lr)

# Print the results

print(f"LR: y\_pred={y\_pred\_lr}, RMSE={rmse\_lr}")

print(f"No LR: y\_pred={y\_pred\_no\_lr}, RMSE={rmse\_no\_lr}")

1. Explain the 3 step life cycle of Deep Learning projects with an example use case.

The 3-step life cycle of a deep learning project includes:

* + 1. *Data preparation and preprocessing*
    2. *Model building and training*
    3. *Model evaluation and deployment*

*Let's take an example of a deep learning project to classify handwritten digits using the MNIST dataset, and see how these three steps are involved in the project life cycle.*

*Data preparation and preprocessing This step involves acquiring the necessary data and cleaning, processing, and preparing it for use in the model. In this example, we will use the MNIST dataset that contains 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9. We will use the TensorFlow library to load and preprocess the data, including normalizing the pixel values to a range of 0 to 1, converting the class labels to one-hot encoding, and splitting the data into training and validation sets.*

*Model building and training This step involves selecting an appropriate deep learning architecture, defining the model parameters and hyperparameters, and training the model on the prepared data. In this example, we will use a convolutional neural network (CNN) with several layers of convolution, pooling, and fully connected layers. We will define the model architecture using the Keras API and compile it with a categorical cross-entropy loss function and an optimizer such as Adam. We will then fit the model to the training data and evaluate its performance on the validation set using metrics such as accuracy, precision, and recall. We may also perform hyperparameter tuning by adjusting the learning rate, batch size, number of epochs, and other parameters to optimize the model's performance.*

*Model evaluation and deployment This step involves evaluating the trained model's performance on a separate testing dataset and deploying the model for use in real-world applications. In this example, we will evaluate the model's performance on the testing dataset and compare it to the validation set's performance. We may also visualize the model's predictions and performance metrics using various tools and techniques. Once the model's performance is satisfactory, we can deploy it in various ways, such as as a web service, mobile application, or embedded system, depending on the project's requirements.*

*In summary, the 3-step life cycle of a deep learning project involves preparing and preprocessing the data, building and training the model, and evaluating and deploying the model for use in real-world applications. Each of these steps requires careful planning, experimentation, and validation to ensure that the model performs well and meets the project's goals and requirements.*

1. Mathematically, derive the complete process of a single perceptron from an input to output and perform the below task:
2. Explain the above in 3 phases of Feed Forward, Error Calculation, Back Propagation mathematically.
3. Explain the role and importance of each and every parameter other than x and y.

*A single perceptron is a basic unit of a neural network that takes one or more inputs and produces a binary output. Mathematically, the process of a single perceptron can be broken down into three phases: feedforward, error calculation, and backpropagation.*

*a. Phases of a single perceptron:*

*Feedforward: The feedforward phase is the initial step where the perceptron takes the input values and multiplies them by their respective weights. The weighted inputs are then added together along with a bias term to produce a weighted sum. The weighted sum is then passed through an activation function to produce an output value. This can be expressed mathematically as:*

*z = b + ∑(w\*x) y = f(z)*

*where:*

*z is the weighted sum*

*b is the bias term*

*w is the weight vector*

*x is the input vector*

*f() is the activation function*

*y is the output value*

*Error Calculation: The error calculation phase involves comparing the output value of the perceptron with the desired output value to calculate the error. The error is then used to adjust the weights and bias term to minimize the error in the future iterations. The error can be calculated using different methods such as Mean Squared Error (MSE) or Binary Cross-Entropy (BCE), depending on the type of problem being solved. For binary classification problems, the BCE loss function can be used, and the error can be expressed mathematically as:*

*E = -y\*log(y\_pred) - (1-y)\*log(1-y\_pred)*

*where:*

*E is the error*

*y is the true output value (0 or 1)*

*y\_pred is the predicted output value*

*Backpropagation: The backpropagation phase involves propagating the error backward through the network to adjust the weights and bias term. This is done by computing the gradient of the loss function with respect to the weights and bias term using the chain rule of calculus. The weights and bias term are then updated in the opposite direction of the gradient to minimize the error. This process is repeated for multiple iterations until the error is minimized. The weight update rule can be expressed mathematically as:*

*w = w - α∂E/∂w b = b - α∂E/∂b*

*where:*

*α is the learning rate*

*∂E/∂w is the partial derivative of the error with respect to the weights*

*∂E/∂b is the partial derivative of the error with respect to the bias term*

*b. Role and importance of each parameter:*

*x: The input values are the features of the problem being solved and are essential in producing the output value.*

*y: The output value is the result of the perceptron's computation and is compared to the desired output to calculate the error.*

*w: The weight vector determines the importance of each input feature in producing the output value and is adjusted during training to minimize the error.*

*b: The bias term shifts the activation function to the left or right and allows the perceptron to produce different outputs for the same input values.*

*z: The weighted sum of the inputs is the input to the activation function and determines whether the perceptron should produce a positive or negative output.*

*f(): The activation function transforms the input value into a desired output range and introduces nonlinearity into the model, allowing it to model complex patterns in the data.*

*E: The error measures the difference between the predicted output and the true output and is used to adjust the weights and bias term to minimize the error.*

*α: The learning rate determines the step size of the weight and bias updates and is critical in balancing*

1. What are activation functions and why are they required in NN?
2. Create python functions for sigmoid, tanh, relu & softmax.

*Activation functions are mathematical operations applied to the output of each neuron in a neural network. The purpose of activation functions is to introduce non-linearity into the output of the neuron, allowing neural networks to learn complex, non-linear relationships between inputs and outputs.*

*Activation functions determine the output of a neuron given an input or set of inputs. Without activation functions, the output of a neuron would simply be a linear combination of its inputs, and the network would only be able to learn linear relationships between inputs and outputs. The use of activation functions allows neural networks to model more complex, non-linear relationships between inputs and outputs.*

*There are several commonly used activation functions, including the sigmoid function, the hyperbolic tangent function, the rectified linear unit (ReLU) function, and variants of the ReLU function such as leaky ReLU and exponential linear unit (ELU). The choice of activation function depends on the specific problem being solved and can have a significant impact on the performance of the neural network.*

import numpy as np

def sigmoid(x):

    """

    Compute the sigmoid activation function for the input x.

    """

    return 1 / (1 + np.exp(-x))

def tanh(x):

    """

    Compute the hyperbolic tangent activation function for the input x.

    """

    return np.tanh(x)

def relu(x):

    """

    Compute the ReLU (Rectified Linear Unit) activation function for the input x.

    """

    return np.maximum(0, x)

def softmax(x):

    """

    Compute the softmax activation function for the input x.

    """

    exp\_scores = np.exp(x)

    return exp\_scores / np.sum(exp\_scores, axis=1, keepdims=True)