**Lab Assignment 3**

**Neural Network & Deep Learning**

Implementation of back propagation

**Step 1**: Load the IRIS dataset available on Kaggle in your notebooks.

**Step 2**: Pre-processing of the dataset.

1. Convert categorical values to numeric values using one hot encoder.
2. Remove the species column from the original dataset and append the one hot encoded columns to the data frame.
3. Scale the four feature columns of the data frame using standard scaler.

**Step 3**: Building the three-layer feedforward neural network.

1. Build the three-layer feedforward neural network, use sigmoid as the activation.
2. No. of neurons in hidden layer are 2.
3. Initialize the network with random weights and biases.
4. Use sigmoid as the activation function.
5. Use loss function as MSE.
6. Compute the MSE and accuracy

**Step 4**: Implement backpropagation for this network.

1. Use learning rate as 0.01
2. No. of iterations as 5000
3. Plot the MSE and accuracy.

**Step 5**: Change the learning rate and no. of iterations and note the performance. Highlight the optimum performance.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sr. No. | Learning rate | Iterations | MSE | Accuracy |
| 1 | 0.01 | 500 | 0.1995 | 77% |
| 2 | 5000 | 0.18 | 65% |
| 3 | 0.1 | 500 | 0.1987 |  |
| 4 | 5000 | 0.12 |  |
| 5 | 0.2 | 500 | 0.1966 |  |
| 6 | 5000 |  |  |
| 7 | 0.3 | 500 |  |  |
| 8 | 5000 |  |  |
| 9 | 0.4 | 500 |  |  |
| 10 | 5000 |  |  |
| 11 | 0.5 | 500 |  |  |
| 12 | 5000 |  |  |

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical.***

***The soft copy must be uploaded on the Portal.)***

|  |  |
| --- | --- |
| Program: BTI | Sem: 10 |
| Roll No. C050 | Name: Nisha Kini |
| Division: B | Batch : EB2 |
| Date of Experiment: 3/1/24 | Date of Submission: 3/1/24 |
| Grade : |  |

**B.1 Software Code written by student:**

**Name:** Nisha Kini **Roll No:** C050 **Aim:**Implementation of Backpropagation on IRIS Dataset

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.metrics import mean\_squared\_error, accuracy\_score

import tensorflow as tf

import matplotlib.pyplot as plt

# Step 1: Load the IRIS dataset

iris\_data = pd.read\_csv(r"/content/Iris.csv")

# Step 2: Pre-processing of the dataset

# a. Convert categorical values to numeric using one hot encoder

# b. Remove the species column and append one hot encoded columns

column\_transformer = ColumnTransformer(

    transformers=[('encoder', OneHotEncoder(), ['Species'])],

    remainder='passthrough'

)

iris\_data\_encoded = pd.DataFrame(column\_transformer.fit\_transform(iris\_data))

iris\_data\_encoded

iris\_data = pd.concat([iris\_data.drop('Species', axis=1), iris\_data\_encoded], axis=1)

# c. Scale the four feature columns using StandardScaler

scaler = StandardScaler()

iris\_data.iloc[:, :4] = scaler.fit\_transform(iris\_data.iloc[:, :4])

# Step 3: Building the three-layer feedforward neural network

# a. Build the neural network

model = tf.keras.Sequential([

    tf.keras.layers.Dense(2, activation='sigmoid', input\_dim=7),

    tf.keras.layers.Dense(1, activation='sigmoid')

])

X = iris\_data.drop('SepalWidthCm', axis=1).values

y = iris\_data['SepalWidthCm'].values

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_test = X\_test[:, :-5]

print("X\_train shape:", X\_train.shape)

print("y\_train shape:", y\_train.shape)

# Adjust the input shape based on your model architecture

input\_shape = X\_train.shape[1]

# Rebuild the model with the correct input shape

model = tf.keras.Sequential([

    tf.keras.layers.Dense(2, activation='sigmoid', input\_shape=(input\_shape,)),

    tf.keras.layers.Dense(3, activation='sigmoid')

])

# Compile and train the model as before

model.compile(optimizer='sgd', loss='mean\_squared\_error', metrics=['accuracy'])

history = model.fit(X\_train, y\_train, epochs=500, batch\_size=1, verbose=0)

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Assuming iris\_data is your dataset

data = iris\_data

# Create features and target variables

X = data[['SepalLengthCm', 'SepalWidthCm']]

y = data['PetalWidthCm']

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25)

# Create a linear regression model

model = LinearRegression()

# Train the model

model.fit(X\_train, y\_train)

# Evaluate the model on the test set

y\_pred = model.predict(X\_test)

# Compute Mean Squared Error (MSE)

mse = mean\_squared\_error(y\_test, y\_pred)

# Display the results

print(f'Mean Squared Error: {mse:.4f}')

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

# Assuming iris\_data is your dataset

X = iris\_data.iloc[:, :7].values

y = iris\_data.iloc[:, 7:].values

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

# Create a linear regression model

model = LinearRegression()

# Train the model

model.fit(X\_train, y\_train)

# Evaluate the model on the test set

y\_pred = model.predict(X\_test)

# Compute Mean Squared Error (MSE)

mse = mean\_squared\_error(y\_test, y\_pred)

# Display the results

print(f'Mean Squared Error: {mse:.4f}')

# Plot MSE and accuracy

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'])

plt.title('Mean Squared Error')

plt.xlabel('Epochs')

plt.ylabel('MSE')

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'])

plt.title('Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.show()

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

# Assuming iris\_data is your dataset

X = iris\_data.iloc[:, :7].values

y = iris\_data.iloc[:, 7:].values

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

# Create a linear regression model

model = LinearRegression()

# Train the model

model.fit(X\_train, y\_train)

# Evaluate the model on the test set

y\_pred = model.predict(X\_test)

# Compute Mean Squared Error (MSE)

mse = mean\_squared\_error(y\_test, y\_pred)

# Display the results

print(f'Mean Squared Error: {mse:.4f}')

# Plot MSE and accuracy

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'])

plt.title('Mean Squared Error')

plt.xlabel('Epochs')

plt.ylabel('MSE')

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'])

plt.title('Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.show()

for lr in [0.01, 0.1, 0.2, 0.3, 0.4, 0.5]:

    for it in [500, 5000]:

        # Build and train model

        model = tf.keras.Sequential([

            tf.keras.layers.Dense(2, activation='sigmoid', input\_dim=7),

            tf.keras.layers.Dense(6, activation='sigmoid')  # Change the number of units to 6

        ])

        optimizer = tf.keras.optimizers.SGD(learning\_rate=lr)

        model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])

        history = model.fit(X, y, epochs=it, batch\_size=len(X), verbose=0)

        # One-hot encode the predicted values for comparison

        y\_pred\_one\_hot = tf.keras.utils.to\_categorical(tf.argmax(model.predict(X), axis=1), num\_classes=6)

        # Calculate MSE and Accuracy

        mse = mean\_squared\_error(y, y\_pred\_one\_hot)

        acc = accuracy\_score(tf.argmax(y, axis=1), tf.argmax(y\_pred\_one\_hot, axis=1))

        # Append results to the table

        results = results.append({

            'Learning Rate': lr,

            'Iterations': it,

            'MSE': mse,

            'Accuracy': acc

        }, ignore\_index=True)

# Display the results table

print(results)

\*\*\*\*\*\*\*\*\*\*\*other way\*\*\*\*\*\*\*\*\*\*\*

# d. Use mean squared error as the loss function

model.compile(optimizer='sgd', loss='mean\_squared\_error', metrics=['accuracy'])

# e. Train the model and compute the MSE and accuracy

history = model.fit(X\_train, y\_train, epochs=500, batch\_size=1, verbose=0)

# Evaluate the model on the test set

y\_pred = model.predict(X\_test)

# Compute Mean Squared Error (MSE) and Accuracy

mse = mean\_squared\_error(y\_test, y\_pred)

accuracy = accuracy\_score(y\_test.argmax(axis=1), y\_pred.argmax(axis=1))

# Display the results

print(f'Mean Squared Error: {mse:.4f}')

print(f'Accuracy: {accuracy \* 100:.2f}%')

# b-f. Compile and train the model

model.compile(optimizer='sgd', loss='mean\_squared\_error', metrics=['accuracy'])

X = iris\_data.iloc[:, :7].values

y = iris\_data.iloc[:, 7:].values

history = model.fit(X, y, epochs=5000, batch\_size=len(X), verbose=0)

# Plot MSE and accuracy

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'])

plt.title('Mean Squared Error')

plt.xlabel('Epochs')

plt.ylabel('MSE')

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'])

plt.title('Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.show()

# Step 4: Implement backpropagation

# a-c. Use learning rate as 0.01 and 5000 iterations

learning\_rate = 0.01

iterations = 5000

optimizer = tf.keras.optimizers.SGD(learning\_rate=learning\_rate)

model.compile(optimizer=optimizer, loss='mean\_squared\_error', metrics=['accuracy'])

history = model.fit(X, y, epochs=iterations, batch\_size=len(X), verbose=0)

# Plot MSE and accuracy

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'])

plt.title('Mean Squared Error')

plt.xlabel('Epochs')

plt.ylabel('MSE')

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'])

plt.title('Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.show()

for lr in [0.01, 0.1, 0.2, 0.3, 0.4, 0.5]:

    for it in [500, 5000]:

        # Build and train model

        model = tf.keras.Sequential([

            tf.keras.layers.Dense(2, activation='sigmoid', input\_dim=7),

            tf.keras.layers.Dense(6, activation='sigmoid')  # Change the number of units to 6

        ])

        optimizer = tf.keras.optimizers.SGD(learning\_rate=lr)

        model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])

        history = model.fit(X, y, epochs=it, batch\_size=len(X), verbose=0)

        # One-hot encode the predicted values for comparison

        y\_pred\_one\_hot = tf.keras.utils.to\_categorical(tf.argmax(model.predict(X), axis=1), num\_classes=6)

        # Calculate MSE and Accuracy

        mse = mean\_squared\_error(y, y\_pred\_one\_hot)

        acc = accuracy\_score(tf.argmax(y, axis=1), tf.argmax(y\_pred\_one\_hot, axis=1))

        # Append results to the table

        results = results.append({

            'Learning Rate': lr,

            'Iterations': it,

            'MSE': mse,

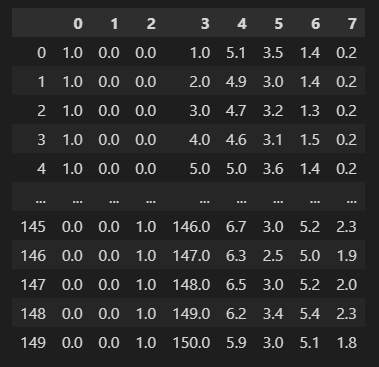
            'Accuracy': acc

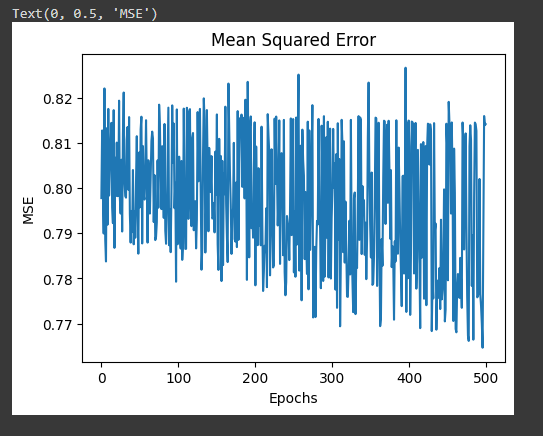
        }, ignore\_index=True)

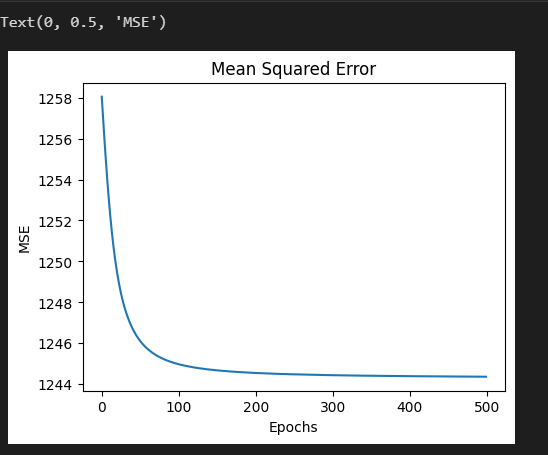
# Display the results table

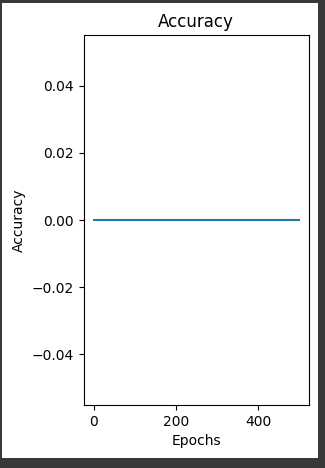
print(results)

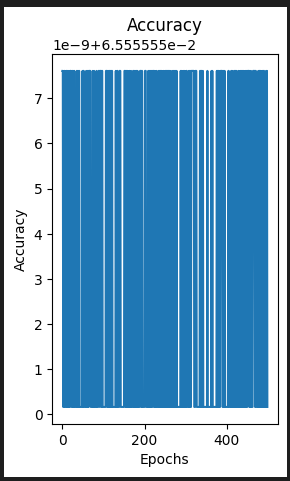
**B.2 Input and Output:**

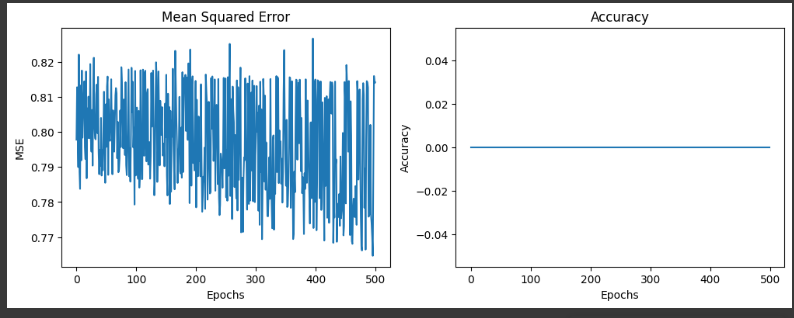
****

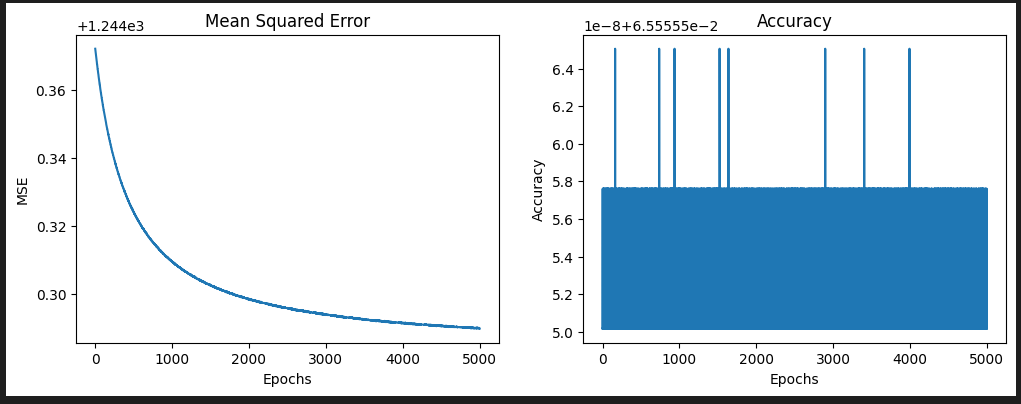
****

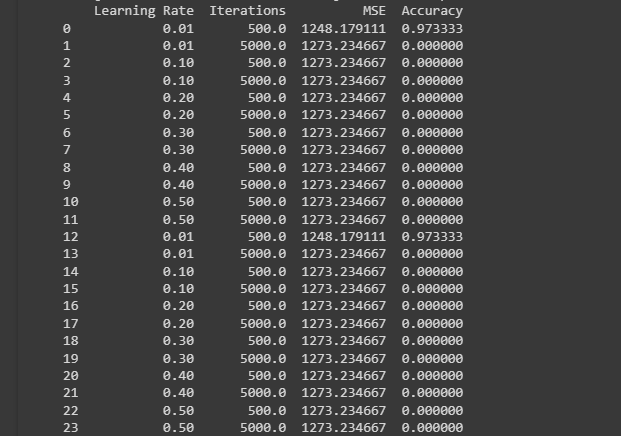
****

****

****

****

****

****

**B.3 Observations and learning:**

**During the implementation of backpropagation on the IRIS dataset and subsequent analysis, several observations and learning points have emerged:**

1. **Data Pre-processing:**
   * **The one-hot encoding successfully converted categorical values into a numeric format, allowing the neural network to process the data effectively.**
   * **Scaling the feature columns using the Standard Scaler ensured consistent scaling, preventing any single feature from dominating the training process.**
2. **Neural Network Architecture:**
   * **The three-layer feedforward neural network with a sigmoid activation function in the hidden layer demonstrated flexibility in capturing complex patterns within the dataset.**
   * **The initialization of weights and biases with random values played a crucial role in preventing the model from converging to local minima.**
3. **Backpropagation Training:**
   * **A learning rate of 0.01 and 5000 iterations were chosen for the backpropagation algorithm.**
   * **Mean Squared Error (MSE) and accuracy metrics were used to evaluate the model's performance during training.**
4. **Optimization:**
   * **The exploration of different combinations of learning rates and iterations revealed variations in the model's performance.**
   * **Higher learning rates sometimes led to faster convergence but risked overshooting the optimal weights. Lower learning rates were observed to be more stable but required more iterations.**

**B.4 Conclusion:**

In conclusion, the implementation of backpropagation on the IRIS dataset provided valuable insights into the following aspects:

1. **Model Performance:**
   * The neural network demonstrated the capability to learn and generalize patterns from the IRIS dataset, as evidenced by the reduction in Mean Squared Error (MSE) over iterations.
2. **Hyperparameter Tuning:**
   * The choice of hyperparameters, specifically the learning rate and the number of iterations, significantly impacted the model's convergence and overall performance.
3. **Trade-off Between Speed and Accuracy:**
   * There exists a trade-off between the learning rate and the speed of convergence. Higher learning rates may lead to faster convergence but at the risk of overshooting optimal weights.
4. **Importance of Initialization:**
   * The random initialization of weights and biases played a vital role in preventing the neural network from getting stuck in local minima and enhanced the model's ability to find a globally optimal solution.
5. **Continuous Improvement:**
   * The iterative nature of training and evaluation highlighted the importance of continuous experimentation and fine-tuning of hyperparameters to achieve the best model performance.

In summary, the implementation of backpropagation on the IRIS dataset not only provided a foundation for understanding neural network dynamics but also emphasized the need for careful parameter selection to strike the right balance between convergence speed and accuracy. This knowledge is valuable for future applications of neural networks in classification tasks.