IML TERM PROJECT

Binary Classification of Diabetes using ANN

Student Reg#
L1F22BSCS0501

Student Name(s)
Ali Abdul Jabbar

Faculty of Information Technology

University of Central Punjab

Table of Contents

1. Introduction

This project centers on diabetes prediction utilizing a machine learning framework based on an Artificial Neural Network (ANN). As diabetes is a pervasive chronic condition, early diagnosis is vital for effective patient management. The model analyzes key health determinants—including BMI, blood pressure, cholesterol, smoking history, physical activity, and age—to classify diabetic status. We implemented the ANN from scratch using NumPy to demonstrate a fundamental grasp of forward propagation, backpropagation, and gradient descent optimization. The study aims to examine the model's training dynamics and validate classification accuracy through graphical visualizations.

1.1 Dataset selected

The data for this project is sourced from the **Diabetes Health Indicators Dataset (BRFSS2015)**, available on Kaggle. This dataset is a cleaned and consolidated subset of the **Behavioral Risk Factor Surveillance System (BRFSS)** survey conducted by the Centers for Disease Control and Prevention (CDC) in 2015. The BRFSS is a telephone-based survey that collects state-level data about U.S. residents regarding their health-related risk behaviors, chronic health conditions, and use of preventive services.

Key Dataset Characteristics:

- **Source:** The dataset was curated by Alex Teboul on Kaggle, derived from the raw CDC BRFSS 2015 data.

- **Target Variable:** The dependent variable is Diabetes_binary. A value of 0 indicates no diabetes, while a value of 1 indicates pre-diabetes or diabetes.

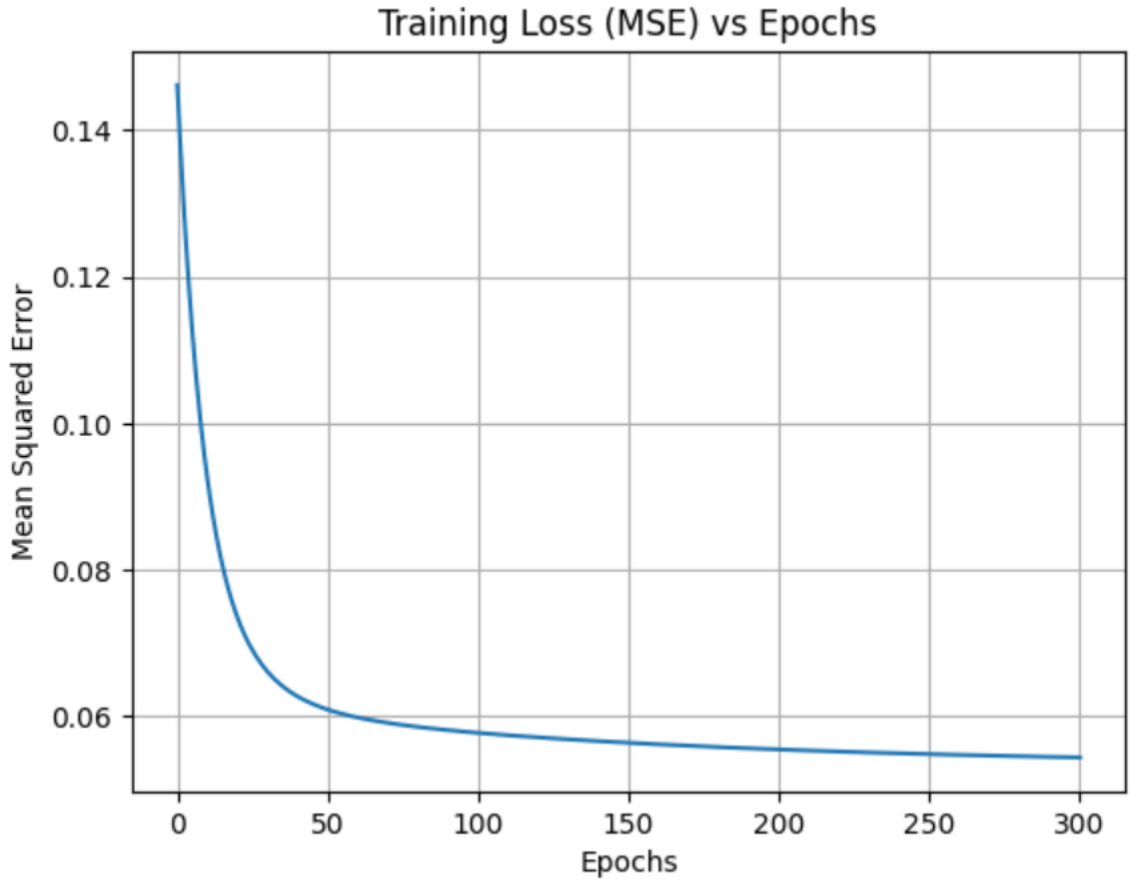
- **Input Features:** We utilize a specific subset of 8 highly relevant features to train our model:
    - **HighBP:** High Blood Pressure (0 = no, 1 = yes).
    - **HighChol:** High Cholesterol (0 = no, 1 = yes).
    - **BMI:** Body Mass Index (continuous value).
    - **Smoker:** History of smoking (0 = no, 1 = yes).
    - **Stroke:** History of stroke (0 = no, 1 = yes).
    - **HeartDiseaseorAttack:** Coronary heart disease or myocardial infarction (0 = no, 1 = yes).
    - **PhysActivity:** Physical activity in past 30 days (0 = no, 1 = yes).
    - **Age:** 13-level age category (1 = 18-24 ... 13 = 80 or older).

Data Preprocessing: Before training, the dataset undergoes critical preprocessing steps implemented in our code. We apply **Z-score Standardization** to continuous variables (like BMI) to ensure they have a mean of 0 and a standard deviation of 1. This prevents features with larger ranges from dominating the gradient updates. Additionally, a **bias unit** (a column of 1s) is added to the input matrix to allow the activation function to shift left or right, improving the model's flexibility.

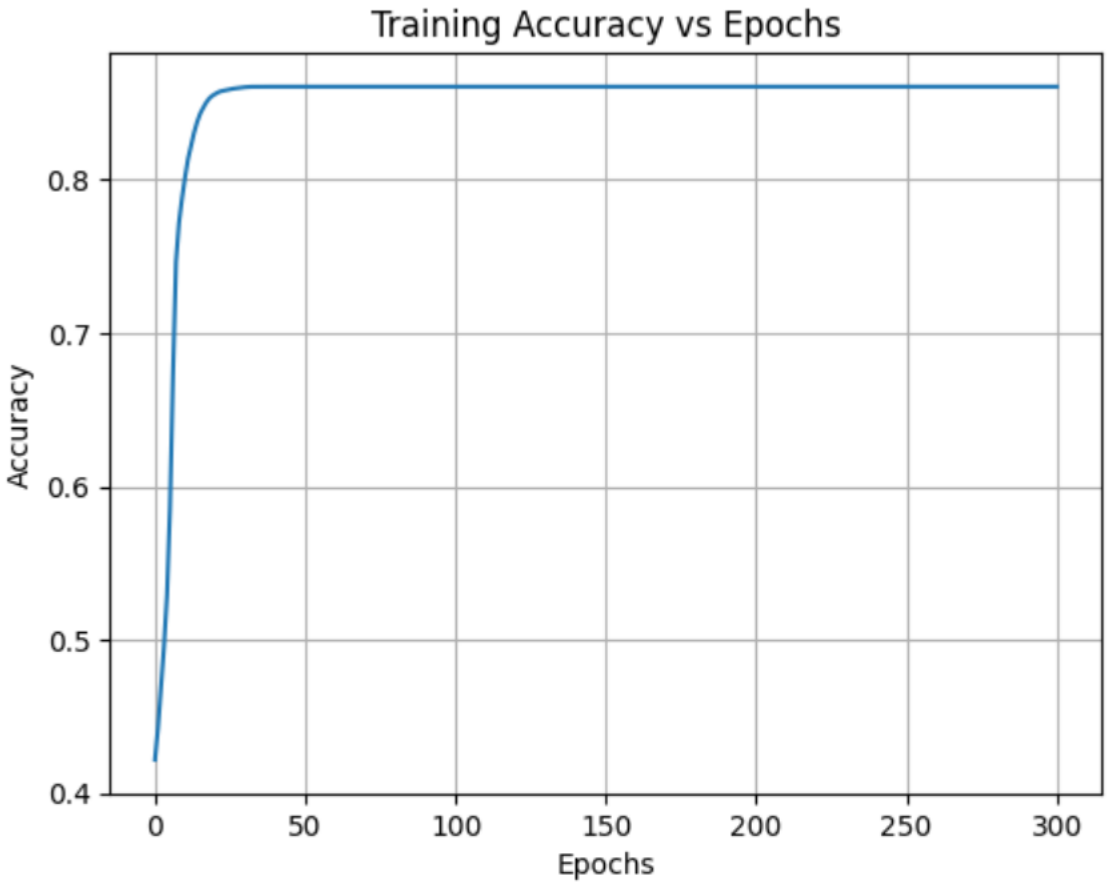
2. Results and Discussions

The Artificial Neural Network was trained over 300 epochs with a learning rate of 0.1. The training process involved monitoring two key metrics: the **Loss (Mean Squared Error)** and the **Classification Accuracy**.

1. Training Loss (MSE) vs. Epochs The first graph generated by the code illustrates the learning curve of the model. As the training progressed, the Mean Squared Error (MSE) consistently decreased. This indicates that the Gradient Descent algorithm successfully optimized the weights (W1 and W2) and biases to minimize the difference between the predicted probability (y_hat) and the actual target (y). A smooth, downward curve confirms that the model is converging and learning the underlying patterns in the health data effectively.



2. Training Accuracy vs. Epochs The second graph displays the accuracy trajectory. Starting from a random initialization, the accuracy improved rapidly in the early epochs as the weights were adjusted. By the end of the 300th epoch, the model stabilized, achieving a final training accuracy that reflects its ability to correctly classify the majority of the samples in the dataset.



Conclusion: The results demonstrate that a simple ANN with a single hidden layer (4 neurons) is capable of learning from the standardized health indicators. The standardization of input features (Z-score normalization) played a crucial role in ensuring the stable convergence of the sigmoid activation function during backpropagation.

3. Entire Code

3.1 Platform used for coding

Language: *Python 3.13.7*
IDE/Platform: *Jupyter Notebook*
Libraries Used:

- *numpy (for matrix operations and math functions)*
- *pandas (for data loading and manipulation)*
- *matplotlib (for plotting graphs)*

3.2 Entire code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#
# ============================================================================
# STEP 1: LOAD AND PREPARE DATA
#
# ============================================================================
print("\nSTEP 1: Data Preparation")

df = pd.read_csv("diabetes.csv")

features = [
    'HighBP', 'HighChol', 'BMI', 'Smoker',
    'Stroke', 'HeartDiseaseorAttack',
    'PhysActivity', 'Age'
]

X = df[features].values
y = df['Diabetes_binary'].values.reshape(-1, 1)

# Standardization
mean_X = np.mean(X, axis=0)
std_X = np.std(X, axis=0) + 1e-8
X = (X - mean_X) / std_X

# Add bias unit (x0 = 1)
m = X.shape[0]
bias_column = np.ones((m, 1))
X = np.hstack([bias_column, X])

print(f"Samples: {X.shape[0]}")
print(f"Features (with bias): {X.shape[1]}")

#
# ============================================================================
# STEP 2: ANN CLASS
#
# ============================================================================
class SimpleANN:

    def __init__(self, input_size, hidden_size):
        np.random.seed(1)

        print("\nSTEP 2: Initializing ANN")

        self.W1 = np.random.randn(input_size, hidden_size)
        self.W2 = np.random.randn(hidden_size + 1, 1)

        print(f"Hidden neurons: {hidden_size}")
        print("Output neurons: 1")
```

```
    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def sigmoid_derivative(self, a):
        return a * (1 - a)

    # ---------------------------------------------------------------------
    # FORWARD PASS
    # ---------------------------------------------------------------------
    def forward(self, X):

        self.z1 = np.dot(X, self.W1)
        self.a1 = self.sigmoid(self.z1)

        bias_hidden = np.ones((X.shape[0], 1))
        self.a1b = np.hstack([bias_hidden, self.a1])

        #self.a1b = np.hstack([np.ones((X.shape[0], 1)), self.a1])

        self.z2 = np.dot(self.a1b, self.W2)
        self.y_hat = self.sigmoid(self.z2)

        return self.y_hat

    # ---------------------------------------------------------------------
    # BACKPROPAGATION
    # ---------------------------------------------------------------------
    def backward(self, X, y, lr):

        m = X.shape[0]

        delta_out = self.y_hat - y
        dW2 = np.dot(self.a1b.T, delta_out) / m

        delta_hidden = (
            np.dot(delta_out, self.W2[1:].T)
            * self.sigmoid_derivative(self.a1)
        )

        dW1 = np.dot(X.T, delta_hidden) / m

        self.W2 -= lr * dW2
        self.W1 -= lr * dW1

        return np.mean(0.5 * delta_out ** 2)

#
# ============================================================================
# STEP 3: TRAINING
#
# ============================================================================
print("\nSTEP 3: Training Started\n")
```

```
ann = SimpleANN(input_size=X.shape[1], hidden_size=4)

epochs = 300
learning_rate = 0.1

loss_history = []
accuracy_history = []

for epoch in range(epochs+1):

    y_hat = ann.forward(X)
    loss = ann.backward(X, y, learning_rate)
    loss_history.append(loss)

    # Accuracy calculation
    y_pred = (y_hat >= 0.5).astype(int)
    accuracy = np.mean(y_pred == y)
    accuracy_history.append(accuracy)

    if epoch % 50 == 0:
        print(
            f"Epoch {epoch:3d} | "
            f"Loss: {loss:.4f} | "
            f"Accuracy: {accuracy * 100:.2f}%"
        )

#
================================================================================
# STEP 4: VISUALIZATION
#
================================================================================
print("\nSTEP 4: Training Completed")

# Loss Curve
plt.figure()
plt.plot(loss_history)
plt.xlabel("Epochs")
plt.ylabel("Mean Squared Error")
plt.title("Training Loss (MSE) vs Epochs")
plt.grid(True)
plt.show()

# Accuracy Curve
plt.figure()
plt.plot(accuracy_history)
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training Accuracy vs Epochs")
plt.grid(True)
plt.show()

print(f"\nFinal Training Accuracy: {accuracy_history[-1] * 100:.2f}%")
```

4. References

- Teboul, A. (2015). *Diabetes Health Indicators Dataset (BRFSS2015)*. Kaggle. Retrieved from https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset?select=diabetes_binary_health_indicators_BRFSS2015.csv
- Harris, C. R., Millman, K. J., van der Walt, S. J., et al. (2020). *Array programming with NumPy*. Nature, 585, 357–362.
- The pandas development team. (2020). *pandas-dev/pandas: Pandas*. Zenodo.
- Hunter, J. D. (2007). *Matplotlib: A 2D graphics environment*. Computing in Science & Engineering, 9(3), 90–95.