Project 3

Khairul Islam
CPSC 444

Neural Network Implementation for Diabetes Prediction

1. Introduction

This project involves implementing an Artificial Neural Network (ANN) from scratch in Java to predict diabetes based on the CDC health indicators dataset. The primary goal was to build a functional neural network without using any machine learning libraries, train it on health indicator data, and evaluate its performance with different architectural configurations.

Neural networks are computational models inspired by the human brain, composed of interconnected nodes (neurons) organized in layers. Each connection has a weight, and the network learns by adjusting these weights through a process called backpropagation. ANNs have been widely applied in healthcare for disease prediction and diagnosis due to their ability to capture complex non-linear relationships [1].

In this project, the network was tasked with binary classification - predicting whether an individual has diabetes (or prediabetes) based on various health indicators such as BMI, blood pressure, cholesterol levels, and lifestyle factors.

2. Tutorial and Implementation

2.1 Neural Network Architecture

The implementation follows the feedforward neural network architecture with backpropagation as presented in the "Neural Network Tutorial - Developing a Neural Network from Scratch" series [3]. The key components include:

The neural network consists of:

- An input layer with 21 neurons (corresponding to the 21 features)
- Variable hidden layers (experiments with 2, 4, and 6 layers)
- Variable neurons per hidden layer (experiments with 5, 50, and 100 neurons)
- An output layer with 1 neuron (for binary classification)
- ReLU activation for hidden layers and Sigmoid activation for the output layer

2.2 Implementation Details

The implementation was done in Java, following object-oriented principles with distinct classes for different components:

```
// Matrix class for neural network operations
class Matrix {
    double[][] data;
    int rows, cols;

    // Matrix operations like multiply, add, subtract, transpose, etc.
    // ...
```

```
}

// Neural Network class
class NeuralNetwork {
    private Matrix[] weights;
    private Matrix[] biases;
    private Matrix[] layerOutputs;
    private Matrix[] layerInputs;

    // Feedforward and backpropagation methods
    // ...
}
```

The key algorithms implemented include:

1. **Forward Propagation**: For making predictions by passing inputs through the network
2. **Backpropagation**: For calculating gradients and updating weights
3. **Stochastic Gradient Descent**: For optimizing the network parameters

2.3 Matrix Operations

The neural network heavily relies on matrix operations, which were implemented from scratch:

- Matrix multiplication: Used for weighted sums in forward propagation
- Element-wise multiplication (Hadamard product): Used in gradient calculations
- Matrix addition and subtraction: Used for weight updates
- Matrix transposition: Used in backpropagation

2.4 Activation Functions

Two activation functions were implemented:

ReLU (Rectified Linear Unit): Used in hidden layers

```
 // ReLU activation function
@Override
public double activate(double x) {
    return Math.max(0, x);
}

@Override
public double derivative(double x) {
    return x > 0 ? 1 : 0;
}
```

1.

Sigmoid: Used in the output layer for binary classification

```
 // Sigmoid activation function
@Override
public double activate(double x) {
```

```
    return 1.0 / (1.0 + Math.exp(-x));
}

@Override
public double derivative(double x) {
    double sigmoid = activate(x);
    return sigmoid * (1 - sigmoid);
}
```

2.

3. Dataset Analysis

3.1 Dataset Description

The CDC diabetes health indicator dataset contains records from the Behavioral Risk Factor Surveillance System (BRFSS) 2015 survey [2]. The dataset has 253,680 records with 22 features:

- **Target variable**: Diabetes_012 (0: No diabetes, 1: Prediabetes, 2: Diabetes)
- **Features**: Health indicators like BMI, blood pressure, cholesterol, physical activity, etc.

For this project, the target variable was converted to binary (0: No diabetes, 1: Diabetes/Prediabetes).

3.2 Data Preprocessing

The data preprocessing steps included:

Feature normalization: All features were normalized to the range [0,1] using min-max scaling:

```
features[j-1] = (val - minValues[j-1]) / (maxValues[j-1] - minValues[j-1]);
```

1.

Binary target conversion: The original 3-class problem was converted to binary:

```
double diabetesValue = Double.parseDouble(values[0]);

double target = diabetesValue > 0 ? 1.0 : 0.0; // Convert to binary target
```

2.
3. **Train-test split**: The dataset was split into 70% training (177,576 samples) and 30% testing (76,104 samples)

3.3 Class Distribution

After analysis, the dataset showed significant class imbalance:

- Class 0 (No diabetes): 84.2% of samples
- Class 1 (Diabetes/Prediabetes): 15.8% of samples

This imbalance has significant implications for model training and evaluation.

4. Experimental Setup and Results

4.1 Experiment 1: Varying Hidden Layer Size

This experiment examined the effect of neuron count per layer while keeping the number of layers fixed at 2.

Hidden Layer Size	Training Accuracy	Testing Accuracy	Training Time (s)
5-5	84.20%	84.33%	1,863.6
50-50	84.20%	84.33%	12,797.8
100-100	84.20%	84.33%	29,436.8

4.2 Experiment 2: Varying Hidden Layer Count

This experiment varied the network depth while keeping each layer fixed at 50 neurons.

Hidden Layers	Configuration	Testing Accuracy	Training Time (s)
2	50-50	84.33%	12,797.8
4	50-50-50-50	84.33%	21,532.0
6	50-50-50-50-50-50	84.33%	47,623.8

4.3 Experiment 3: Varying Learning Rate

This experiment tested different learning rates using the 4-layer network with 50 neurons per layer.

Learning Rate	Testing Accuracy	Training Time (s)
0.01	84.33%	17,173.1

0.1	84.33%	21,877.9
0.5	84.33%	32,187.4

5. Analysis and Discussion

5.1 Accuracy Analysis

The most striking observation from the experiments is the consistency of the accuracy metrics across all architectural variations. The accuracy remained at approximately 84.2% for training and 84.33% for testing, regardless of:

- Layer size (5, 50, or 100 neurons)
- Network depth (2, 4, or 6 hidden layers)
- Learning rate (0.01, 0.1, or 0.5)

This consistency strongly suggests that the model is not learning meaningful patterns from the training data. Instead, it appears to be defaulting to a strategy of predicting the majority class (Class 0, No diabetes) for all instances, which yields an accuracy of about 84.2% due to the class distribution in the dataset.

5.2 Class Imbalance Impact

The class imbalance in the dataset (84.2% no diabetes vs. 15.8% diabetes/prediabetes) is likely the primary cause of the observed behavior. In classification problems with imbalanced classes, accuracy becomes a misleading metric since a model can achieve high accuracy by simply predicting the majority class.

In our implementation, this manifests as:

1. The model quickly learns to predict "no diabetes" for all cases
2. This yields ~84.2% accuracy from the beginning
3. There's little incentive for the model to improve beyond this baseline
4. The backpropagation updates become minimal, essentially stopping meaningful learning

5.3 Training Time Analysis

While accuracy remained constant, training times varied significantly across experiments:

1. **Layer Size Impact**: Increasing from 5 to 100 neurons per layer increased training time by ~16× (from ~31 minutes to ~8 hours)
2. **Network Depth Impact**: Increasing from 2 to 6 layers increased training time by ~3.7× (from ~3.5 hours to ~13 hours)
3. **Learning Rate Impact**: Higher learning rates required more computation time, possibly due to more erratic weight updates

This exponential increase in computational requirements with model complexity is a well-documented challenge in neural network training.

5.4 Why Architectural Changes Showed No Impact

Several factors might explain why architectural changes did not affect accuracy:

1. **Dominated by Class Imbalance**: The strong class imbalance effect overshadows any potential benefits from architectural changes.

2. **Backpropagation Issues**: The implementation might have issues in the gradient calculation or weight update steps.

3. **Vanishing/Exploding Gradients**: Especially relevant for deeper networks, this common problem can halt learning [5].

4. **Feature Relevance**: The health indicators might have limited predictive power for diabetes in this dataset.

6. Recommended Improvements

6.1 Addressing Class Imbalance

1. **Resampling Techniques**:
    - Undersampling the majority class
    - Oversampling the minority class using techniques like SMOTE (Synthetic Minority Over-sampling Technique) [6]
2. **Cost-Sensitive Learning**:
    - Modify the backpropagation to assign higher costs to misclassifying minority class examples
3. **Alternative Metrics**:
    - Use precision, recall, F1-score, or AUC-ROC instead of accuracy for evaluation
    - Implement a custom loss function that accounts for class imbalance

6.2 Implementation Enhancements

1. **Weight Initialization**:
    - Implement Xavier/He initialization for better gradient flow [7]
2. **Regularization**:
    - Add L1/L2 regularization to prevent overfitting
    - Implement dropout for more robust feature learning
3. **Batch Training**:
    - Implement mini-batch gradient descent instead of stochastic (per-example) updates
    - Add batch normalization between layers
4. **Optimization Algorithms**:
    - Implement momentum to accelerate convergence
    - Add adaptive learning rate methods like Adam or RMSProp [8]

6.3 Debugging and Monitoring

1. **Loss Tracking**:
    - Monitor and log the loss function values during training
    - Implement early stopping based on validation loss
2. **Gradient Checking**:
    - Verify backpropagation implementation using numerical gradient checking
3. **Weight Visualization**:
    - Track the distribution of weights and activations across training
    - Identify layers with potential vanishing/exploding gradient issues

7. Conclusion

This project implemented an artificial neural network from scratch in Java for diabetes prediction using the CDC health indicators dataset. Despite the sophisticated implementation with matrix operations, activation functions, and backpropagation, the model consistently achieved an accuracy of ~84.33% across all architectural variations.

The analysis strongly suggests that class imbalance in the dataset (84.2% no diabetes vs. 15.8% diabetes/prediabetes) is the primary reason for this behavior, with the model defaulting to predicting the majority class. This underscores the importance of addressing class imbalance in healthcare predictive modeling, where disease cases are typically the minority class of interest.

The implementation provides valuable insights into the inner workings of neural networks and the challenges of applying them to real-world healthcare data. Future work should focus on implementing the recommended improvements, particularly those addressing class imbalance and using more appropriate evaluation metrics.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] "CDC Diabetes Health Indicators Dataset," UCI Machine Learning Repository. [Online]. Available: https://archive.ics.uci.edu/dataset/891/cdc+diabetes+health+indicators

[3] "Neural Network Tutorial - Developing a Neural Network from Scratch," YouTube, Coding With John. [Online]. Available: https://www.youtube.com/watch?v=3MMonOWGe0M&list=PLpcNcOt2pg8k_YsrMjSwVdy3GX-rc_ZgN