Predict Diabetes using Perceptron

A. Introduction

Diabetes is a chronic medical condition that affects millions of people worldwide, leading to serious health complications if not properly managed. Early and accurate diagnosis of diabetes is crucial for timely intervention and effective treatment. This project aims to describe, implement, and test the Perceptron algorithm, which can be used for predicting diabetes in individuals.

A.1. Problem Statement

The objective of the project is to develop a predictive model that can diagnostically predict whether or not a patient has diabetes. To achieve this, we will utilize the Pima Indians Diabetes dataset, originally sourced from the National Institute of Diabetes and Digestive and Kidney Diseases. This dataset contains diagnostic measurements of females, at least 21 years old, of Pima Indian heritage, and is well-suited for binary classification tasks, specifically for identifying diabetes cases.

A.2. Comparison with Leading Competitors

While the Perceptron algorithm is a foundational concept in deep learning, it is essential to recognize its limitations in comparison to leading competitors. For instance [6], more advanced neural network architectures, such as deep neural networks and convolutional neural networks, have achieved remarkable success in diverse medical diagnostic tasks, thanks to their ability to capture complex patterns in data.

However, for the Pima Indians Diabetes dataset consisting of only 768 samples, each containing 8 features, the perceptron algorithm is more than sufficient for binary classification tasks of this scale. This project implements the Perceptron algorithm as a starting point and evaluate its performance.

B. Method Description

Artificial Neural Networks (ANNs) are a computational framework inspired by the organization of biological brains. They are composed of interconnected nodes called artificial neurons, which aim to mimic the behavior of neurons in the human brain. In this system, information is transmitted between neurons through connections, with each connection carrying a real-numbered signal. The output of each neuron is determined by a non-linear function applied to the sum of its inputs X. These connections are known as edges and are associated with weights W and bias b that adjust as the network learns. These weights and bias control the strength of the signal transmitted along each connection and get the final result y.

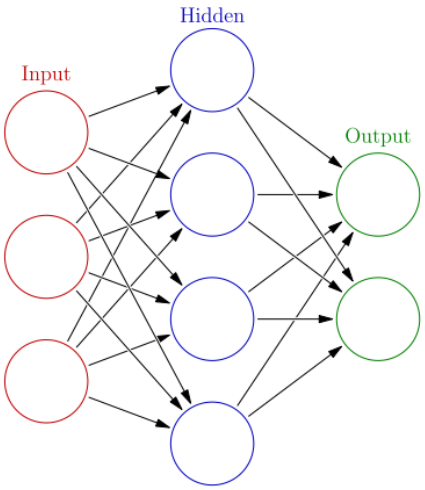

Figure 1. MLP with One Hidden Layer [7]

$$y = W * X + b \tag{1}$$

Neurons may also have a threshold, and a signal is sent only if the cumulative signal surpasses this threshold. Signals propagate through the network, starting from the input layer and progressing towards the output layer, possibly traversing intermediate layers multiple times.

A Single Layer Perceptron (SLP) represents the simplest form of an ANN, utilizing a threshold transfer function. SLPs are limited to solving problems where the classes to be classified are linearly separable and have binary target values (usually 0 and 1).

In contrast, a Multi-Layer Perceptron (MLP) shares a similar structure to the SLP but includes one or more hidden layers, as illustrated in Fig. 1. The training of MLPs involves the backpropagation algorithm, which consists of two main phases. First, during the forward phase, activations are computed and propagated from the input layer through the hidden layers to the output layer. Then, during the backward phase, the error between the actual output and the desired target values is propagated backward through the network. This backward propagation is used to adjust the weights and bias values in order to minimize the error. This process of weight updates is repeated iteratively until termination conditions are met.

B.1. Activation Function

Activation functions introduce non-linear network and empower the network to effectively model and approximate complex functions. These functions enable neural networks

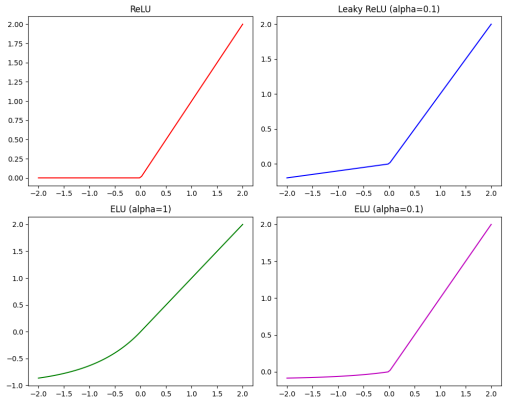

Figure 2. Activation Functions

to learn and adapt representations of input data across different layers. This adaptability allows the network to capture hierarchical features and abstract patterns from raw data.

There are several common activation functions, including the Sigmoid Function, Hyperbolic Tangent Function (tanh), Rectified Linear Unit (ReLU), Leaky ReLU, and Exponential Linear Unit (ELU). Among these following as Fig. 2, ReLU stands out as one of the most prevalent choices. It behaves by returning the input as-is when it's positive and zero when it's negative. Leaky ReLU, a variant of ReLU, addresses the vanishing gradient issue by permitting a small gradient for negative inputs. On the other hand, ELU, similar to Leaky ReLU, provides a smoother transition for negative input values. All of these activation functions are computationally efficient and contribute to mitigating the vanishing gradient problem.

Softmax is often used in the output layer for multiclass classification problems. It converts a vector of raw scores into a probability distribution, making it suitable for classification tasks.

B.2. Forward Propagation

In the forward propagation, input data is combined with weights and biases at each layer, passed through an activation function, and progressively transmitted through layers to ultimately produce the network's prediction or output, as shown in Fig. 3.

$$y = ActivationFunction(W * X + b) \tag{2}$$

This process continues layer by layer until reaching the output layer, thus achieving data processing and prediction generation.

B.3. Backward Propagation

Backward Propagation trains neural networks by updating W and b, in a way that minimizes the error between predicted and actual outputs. The loss function is used to evaluate the error. The common loss function is Cross-Entropy Loss which can be utilized in Binary Classification (3) and Multi-classification (4)

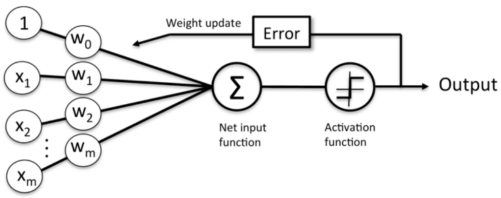

Figure 3. $y = ActivationFunction(W * X + b)$

$$Loss = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \tag{3}$$

$$Loss = -\sum_{k=1}^{K} y_k \log(\hat{y}_k) \tag{4}$$

After calculating the error, it will propagate the error backward through the network, layer by layer, using the chain rule to calculate the gradients that represent how much each parameter contributed to the error.

B.4. Parameter Update

Once the gradients are computed, the model parameters are updated using Gradient Descent (5)(6), where α is learning rate

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w_0, w_1 \ldots\ldots w_n) \tag{5}$$

$$b := b - \alpha \frac{\partial}{\partial w_j} J(w_0, w_1 \ldots\ldots w_n) \tag{6}$$

The update rule typically involves subtracting a fraction of the gradient from each parameter, aiming to minimize the error in the next forward pass.

B.5. Mini-batch Stochastic Gradient Descent

This project uses Mini-batch Stochastic Gradient Descent (Mini-batch SGD) as an optimization algorithm for training the SLPs and MLPs, as following:

Step 1 Initialization:

Determine the number of layers and the number of neuron in each layers. Initialize weight W and bias b. Define hyper-parameters learning rate α, batch size, and the number of training epochs.

Step 2 Training Loop:

The training process consists of multiple epochs. In each epoch, the following steps are repeated until the entire training dataset has been processed.

Step 3 Mini-batch Sampling:

Each mini-batch typically contains a subset of training examples allowing for parallelization and faster computation.

Step 4 Forward Propagation:
Mentioned in B.2

Step 5 Back Propagation:
Mentioned in B.3

Step 6 Parameter Update:
Mentioned in B.4

Step7 Repeat:
Repeat steps 3 to 6 for the next mini-batch. Continue this process until all mini-batches have been processed once (completing one epoch).

Step 8 Epochs:
After processing all mini-batches in one epoch, the training process may repeat for a predefined number of epochs or until some convergence criterion is met.

C. Experimental Analysis

The experiments conducted to evaluate the performance of SLP and MLP models with varying numbers of layers and hyperparameter configurations and drew conclusions to gain insights into the capabilities and limitations of these neural network architectures. To build an optimized model to predict the diabetes based on some diagnostic measurements.

C.1. Experimental Setup and Motivation

The original dataset contains 768 observations with 8 features. It will be splited into three subsets. Training Datasetwas used for training the neural network models. It comprises 70% of the total data. Validation Dataset was created to tune hyperparameters and monitor the models' performance during training. It constitutes 15% of the data. Testing Dataset, the remaining 15% of the data, was reserved for evaluating the models' final performance.

The experiment trained and tested both SLP and MLP models with different layer configurations, including single-layer, two-layer, three-layer and four-layer architectures. It also varied hyperparameters such as the learning rate, batch size, activation functions and weight initialization. Loss, training accuracy, and validating accuracy, as metrics, are used for evaluating the performance of models.

Through the experiment, the primary objective was to gain a deeper understanding of how the depth and complexity of neural network architectures influence their learning capacity and generalization capabilities within the critical context of medical diagnosis.

C.2. Layers

The experiment involved initializing four perceptrons with varying numbers of layers, ranging from a single layer

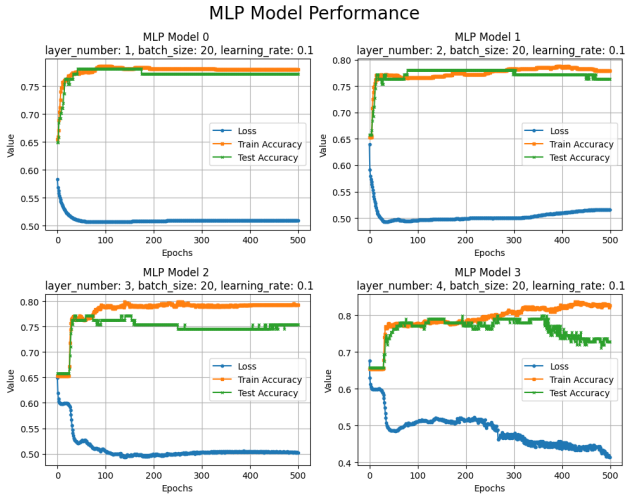

Figure 4. MLP Performance

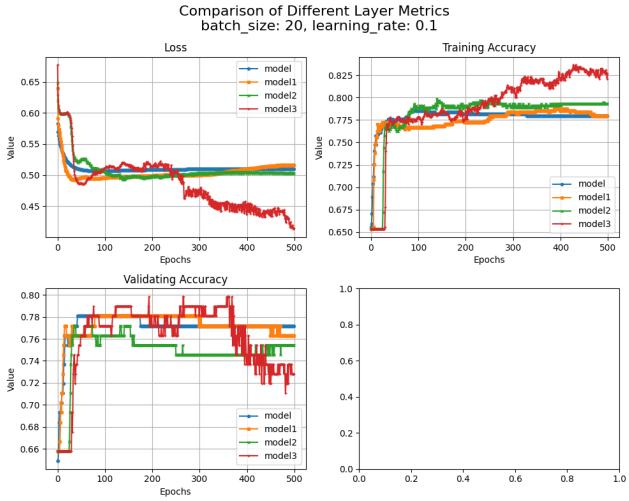

Figure 5. Comparison of Metrics with Different Layers

to a four-layer structure, corresponding to model through model3.

Fig. 4 and Fig. 5 are evident that the loss consistently decreases, and the training accuracy steadily improves as the number of layers increases. This indicates that these models are adept at fitting the training dataset.

However, there is a noteworthy observation: the testing accuracy of the 4-layer MLP is marginally lower than that of the 2-layer MLP. In other words, as we increase the number of layers, the model starts to exhibit signs of overfitting, which can lead to suboptimal generalization performance.

In situations where computational resources are limited, adding more layers may not improve the performance, but only slow down the training process, even render it impractical to complete.

Interestingly, the dataset in this project doesn't appear

to be particularly complex. A single or 2-layer neural network already yields accurate predictions. Consequently, this project opted for the 2-layer MLP as the model for subsequent experiment.

C.3. Learning Rate

Fig. 6 illustrates that a smaller learning rate, saying lr =0.005, leads to slow training progress, often requiring more iterations to converge. Initially, the training accuracy may improve, but the rate of improvement gradually decreases, eventually stabilizing. This is because weight updates are small, resulting in slower model convergence. The loss function decreases slowly during training, and it may eventually reach a higher stable value.

A larger learning rate, saying lr = 2.5, can lead to an unstable training process, where the model may perform exceptionally well on the training data but exhibit significant fluctuations, making it challenging to converge to the optimal solution. Larger learning rates typically result in leading to poorer performance on the testing dataset. The loss function decreases rapidly during the early stages of training but may exhibit fluctuations or oscillations, making it difficult to stabilize.

A moderate learning rate, lr = 0.05 or lr = 0.1, typically fosters a more stable training process where training accuracy gradually improves and converges to a higher level within a reasonable time frame. Both moderate learning rate allows the model to achieve good performance on both the training and testing datasets because the model can effectively learn the features during training. The loss function steadily decreases and converges within a reasonable range without excessive fluctuations. So in this project, 0.05 is set as learning rate.

C.4. Batch Size

Due to the absence of GPU capabilities in our experimental equipment, Fig. 7 indicates that larger batch sizes result in slower convergence. In reality, if we were to introduce parallel computing, larger batch sizes would lead to faster convergence.

C.5. Different Activation Function

Rectified Linear Unit(ReLU) is simple and computationally efficient. It only involves a thresholding operation and helps mitigate the vanishing gradient problem. It may suffer from the ”dying ReLU” problem, where some neurons may become inactive during training and never activate again (always output zero) [3].

Leaky ReLUL addresses the ”dying ReLU” problem by allowing a small gradient for negative values, preventing neurons from becoming completely inactive [5].

Exponential Linear Unit(ELU) also have negative values, making the activation function centered around zero,

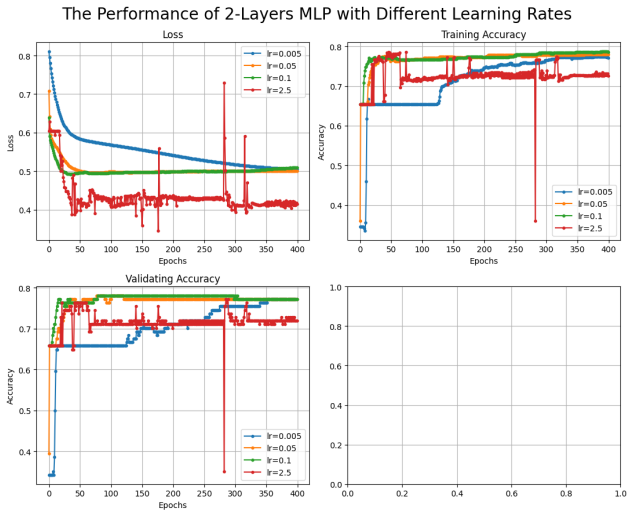

Figure 6. The Performance of 2-Layers MLP with Different Learning Rates

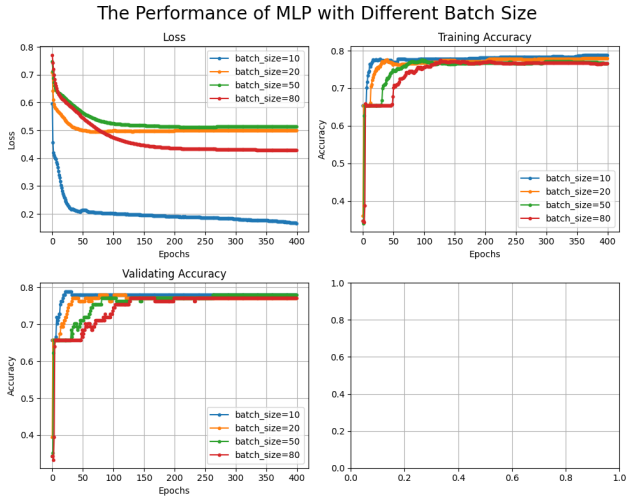

Figure 7. The Performance of 2-Layers MLP with Different Batch Size

which helps with optimization. Generally it outperforms ReLU and Leaky ReLU in deep networks, but more computationally expensive due to the exponential function [1].

Fig. 8 draws that Leaky ReLU and ELU may converge faster than regular ReLU, especially in deep neural networks. Due to its smoother gradient, ELU can sometimes converge to local optima more rapidly. Leaky ReLU and ELU can lead to lower training loss because they address the vanishing gradient problem more effectively. Although all three performed similarly training accuracy, Leaky ReLU and ELU generally performs well in terms of test accuracy since they contributes to better generalization. Therefore, taking into account the computational costs, Leaky ReLU

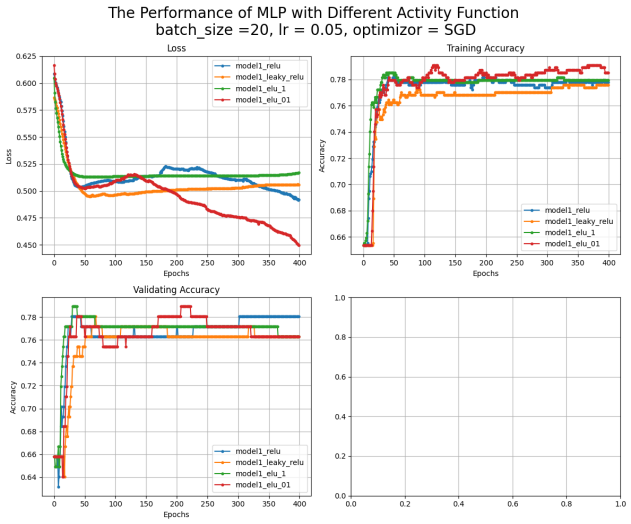

Figure 8. The Performance of 2-Layers MLP with Different Activation Function

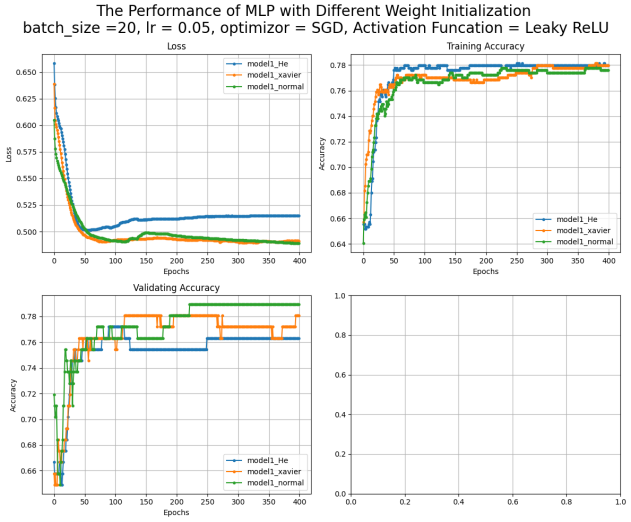

Figure 9. The Performance of MLP with Different Weight Initialization

is a better choice for the model.

C.6. Weight Initialization

Common weight initialization methods include Xavier/Glorot initialization [2] and He initialization [4]. Xavier/Glorot Initialization is designed to balance the scale of gradients and activations and He Initialization initializes weights with larger values. Both methods result in faster convergence and lower training loss. They often result in good training accuracy and competitive test accuracy.

Normal Initialization initializes weights with random values sampled from a normal distribution. The convergence and loss can vary widely depending on the specific initialization values. Normal initialization can work but is less predictable and may require more careful tuning of other hyperparameters for optimal performance. Therefore, Xavier/Glorot Initialization was set as weight initialization in this project.

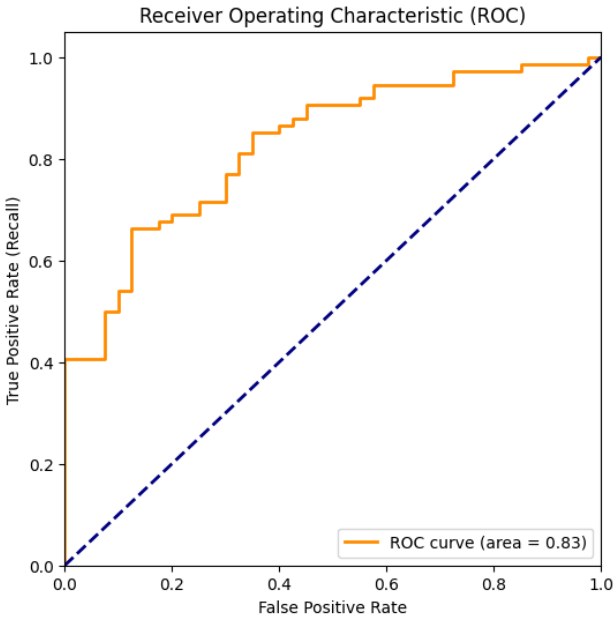

Figure 10. ROC Curve

C.7. Test the Model

Based on the conducted experiments, the optimized parameter configuration for a 2-layer MLP includes a learning rate of 0.05, a batch size of 20, the Leaky ReLU activation function, and Xavier weight initialization. This configuration yielded promising results in the test phase, with an accuracy of 77.19%, precision of 77.9%, recall of 90.54%, and an ROC score of 83% as shown in Fig. 10. These outcomes demonstrate the model's effectiveness in accurately classifying and identifying diabetic cases.

D. Conclusion

In conclusion, this project focused on predicting diabetes using neural network models, specifically the SLP and MLP.

The depth and complexity of neural network architectures, such as the number of layers, significantly impact the model's ability to fit training data. However, increasing model complexity can lead to overfitting on the training data, which may not generalize well to unseen data. Smaller learning rates result in slow convergence, while larger ones may lead to unstable training. Choosing a moderate learn-

ing rate is essential for achieving both convergence and generalization.

Activation functions play a vital role in neural networks. While ReLU is a popular choice, Leaky ReLU and ELU provide smoother gradients, mitigating the vanishing gradient problem and aiding convergence. Leaky ReLU, in particular, balances performance and computational efficiency.

Weight initialization methods, such as Xavier/Glorot and He initialization, can significantly impact training speed and convergence. Xavier/Glorot initialization helps stabilize training and is a suitable choice for many applications.

Finally the test metrics demonstrate the model's effectiveness in diagnosing diabetes. The ROC curve further illustrates its performance across different thresholds.

To further enhance the method, future work could involve experimenting with more advanced neural network architectures, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), to capture more complex patterns in the data. Exploring techniques for handling class imbalance, as medical datasets often exhibit imbalanced distributions. Performing more detailed data preprocessing and EDA, which will result in a more accurate and generalised model that is trained. Conducting more extensive hyperparameter tuning to optimize model performance further.

References

[1] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015. 4

[2] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010. 5

[3] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011. 4

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 5

[5] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013. 4

[6] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014. 1

[7] Wikipedia contributors. Artificial neural network — Wikipedia, the free encyclopedia, 2023. [Online; accessed 27-September-2023]. 1