Deep learning Fundamentals
Assignment 1
Predict diabetes using Perceptron

Ujjwal Bhardwaj
The University of Adelaide
A1881450
ujjwal.bhardwaj@student.adelaide.edu.au

Abstract

This research project delves into applying a perceptron-based model, for the critical task of diabetes classification. The study carefully examines the model's ability to correctly predict the possibility of diabetes by utilizing a large dataset with various patient attributes. Our results demonstrate the compelling potential of this machine-learning approach in healthcare diagnostics through methodical experimentation and thorough analysis. The importance of early diabetes detection is underlined in this study, which also provides profound insights into the capabilities and nuances of deep neural networks in the context of medical risk assessment.

1. Introduction

One of the earliest algorithms in the field of artificial intelligence was the Perceptron. The most significant effect of it was that it increased enthusiasm and anticipation for the field of neural networks. The attempt to develop a perceptron succeeded in modeling linear decision boundaries by drawing inspiration from brain neurons. The output of a perceptron, a type of artificial neuron, is typically used for binary classification by determining whether the output is above or below a threshold. It receives a weighted sum of input features, applies an activation function, and produces an output. Midway through the 1950s, Frank Rosenblatt created the perceptron, which was based on the McCulloch-Pitts model. [2] Warren Sturgis McCulloch and Walter Pitts, a legendary team, put forward the McCulloch-Pitts model. These models established the way for research for many years even though they are no longer in utilization.

In our research project, we leverage a multiple-layer Perceptron algorithm for diabetes prediction. Employing the 'Pima Indians Diabetes' which includes features like Pregnancies, Glucose levels, Blood Pressure, Skin Thickness, Insulin levels, BMI, Diabetes Pedigree Function, and Age, our model performs binary classification—categorizing individuals as Diabetic or Non-Diabetic based on binary labels. Our utilization of a multi-layer perceptron (MLP) in diabetes prediction demonstrates the practical applicability of this neural network architecture in healthcare diagnostics. By analyzing a diverse set of patient attributes, we aim to showcase the MLP's potential in accurately classifying medical outcomes. This research explores the capabilities of deep learning in medical risk assessment and underscores the importance of early detection in diabetes management.

2. Methodology

The perceptron's methodology, shown in Figure 1, is based on the idea of an artificial neuron that closely resembles the way biological neurons work. This artificial neuron receives a linear combination of input features, multiplies each by its corresponding weight, and then applies an activation function to the weighted sum.

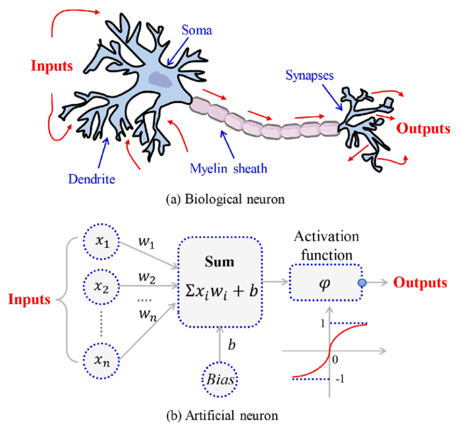

Figure 1. Figure 1- A brief comparison between biological neuron and artificial neuron, as a similar structure used in Perceptron algorithm. [5]

Based on supervised learning principles, the perceptron modifies its weights iteratively during training by comparing its predictions to the actual target values. The perceptron learning rule, a technique used in this learning process, incrementally updates the weights to reduce prediction errors. The perceptron, which is essentially a binary classifier, is extremely useful for tasks requiring binary distinctions because it divides data points into two categories by comparing its output to a predetermined threshold. This fundamental unit, on which neural networks are built, has historically been used to solve a variety of pattern recognition problems and linearly separable issues.

2.1. Structure of a Neuron

A perceptron is made up of 4 components.

1. An Input Vector, denoted by X_i
2. Weights, denoted by w_i
3. The Bias (or the threshold), denoted by θ
4. An Activation function, denoted by Y

Combining these parameters altogether, we can derive the below pivotal formula that embodies these parameters, thereby mathematically framing our perceptron model.

$$Y = F\left(X_i \cdot W_i + \theta\right) \tag{1}$$

We will now examine the meaning and operation of the given equation in order to clarify the practical implications of these elements, using Figure 2.

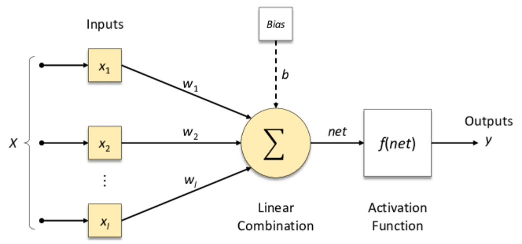

Figure 2. Figure 2- The structural working of perceptron [3]

The concept is pretty straightforward. Given the weights and inputs' numerical values, a function within the neuron will process these and this will result in an output. Delving deeper into this function, we can understand our function by the below equation:

$$y = x_1 w_1 + x_2 w_2 + x_3 w_3 \tag{2}$$

In the context of a perceptron, this equation signifies the weighted sum of input values x_1, x_2, x_3, which is an intermediate step before applying an activation function to make a binary classification decision.

2.2. Major types of Perceptrons

There exist various types of perceptions, but the major two types are listed below:

1. **Single-Layer Perceptron (Standard Perceptron):** The basic perceptron model with a single layer of input nodes and no hidden layers. This model is primarily used for linearly separable binary classifications.

$$Y = \begin{cases} 1, & \text{if } \sum_{i=1}^{n}(X_i \cdot W_i) + \theta > 0 \\ 0, & \text{otherwise} \end{cases}$$

Here,

$$\begin{aligned} Y &: \text{Output} \\ X_i &: \text{Input features} \\ W_i &: \text{Weights} \\ \theta &: \text{Bias} \end{aligned}$$

2. **Multi-Layer Perceptron (MLP):** It is a type of feed-forward neural network with multiple layers, including input, hidden, and output layers, capable of approximating complex, non-linear functions and is used in various machine learning programs.

$$Y = f^{(2)}(W^{(2)} \cdot f^{(1)}(W^{(1)} \cdot X + b^{(1)}) + b^{(2)}) \tag{4}$$

Here,

$$\begin{aligned} Y &: \text{Output} \\ X &: \text{Input vector} \\ W^{(1)} &: \text{Weight matrix for input-to-hidden connections} \\ b^{(1)} &: \text{Bias vector for the hidden layer} \\ f^{(1)} &: \text{Activation function for the hidden layer} \\ W^{(2)} &: \text{Weight matrix for hidden-to-output connections} \\ b^{(2)} &: \text{Bias vector for the output layer} \\ f^{(2)} &: \text{Activation function for the output layer} \end{aligned}$$

2.3. The Algorithm

The perceptron learning algorithm aims to adjust the weights (w_i) and the bias (θ) of a perceptron to correctly classify input samples. It employs the following equation to update the weights at every iteration:

$$w_i \leftarrow w_i + \alpha \cdot (target - output) \cdot x_i \tag{5}$$

Here's the explanation of each parameter:

$$
\begin{aligned}
w_i &: \text{Weight associated with input feature} x_i. \\
\alpha &: \text{Learning rate, to control the step size} \\
target &: \text{The desired target output} \\
output &: \text{The current/actual output} \\
x_i &: \text{The value of the input feature } x_i \text{ for the sample.}
\end{aligned}
$$

The perceptron learning algorithm iterates through the training samples, calculates the result using the current weights and bias, and then modifies the weights depending on whether the result matches what is expected. The objective is to modify the weights so that classification errors are minimized and that they eventually come together to a decision boundary that distinguishes between the classes, or ”classification”.

3. Data pre-processing

The data preprocessing in the provided code involves two key steps: normalization and standardization. Normalization scales the data to a range between 0 and 1, while standardization centers the data around a mean of 0 with a standard deviation of 1. [4] Mathematically, normalization for a feature x can be represented as:

$$x_{\text{normalized}} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{6}$$

And standardization as:

$$x_{\text{standardized}} = \frac{x - \mu}{\sigma} \tag{7}$$

Here,

- x represents the feature values.
- $\min(x)$ is the minimum value.
- $\max(x)$ is the maximum value.
- μ is the mean.
- σ is the standard deviation.

In the data pre-processing steps, the standard deviation (σ) is calculated as a measure of the spread or variability of the feature data. It quantifies how much the individual data points deviate from the mean (μ).

The formula for calculating the standard deviation (σ) is as follows:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{8}$$

Here,

- σ represents the standard deviation.
- N is the number of data points.
- x_i denotes each individual data point.
- μ is the mean of the data.

In conclusion, the data preprocessing steps, which include feature standardization and scaling, ensure that all input features have consistent scales and a zero mean. By facilitating efficient training, this key preprocessing improves the model’s accuracy and performance as well as lays the groundwork for an accurate diabetes classification, which is the ultimate objective of our analysis.

4. Experimentation

In the experimental analysis, a neural network for diabetes classification is improved. In order to improve model convergence, it starts with careful dataset preprocessing. Multiple hidden layers are added to the neural network architecture to improve its refinement, and non-linear activation functions are used to capture complex data patterns. Dropout and weight decay are two methods that are effectively employed to reduce overfitting and improve generalization to new data. Early stopping is used to avoid overtraining by monitoring validation loss and stopping training when performance reaches a plateau. [1] The accuracy achieved is a result of these methodical improvements, demonstrating the value of extensive experimentation in neural network development. The code demonstrates a methodical approach to model optimization, leading to a more reliable and potent classification system for diabetes.

We take an 80-20 split between the train and the test data, and then go ahead and experiment on other parameters like batch size, learning rate, and hidden dimensions. The number of Epochs is limited to 100 in every scenario.

First, we begin by experimenting with batch size, keeping the learning rate at 0.001 and hidden dimensions at 128. Below are our observations:

Batch size	Accuracy on Test	Accuracy on Train
32	93.97	94.81
64	92.35	96.10
128	**95.44**	**97.4**
256	93.00	94.81

Table 1. Experimenting with batch size, keeping the learning rate at 0.001 and hidden dimensions at 128.

As we can see, we achieve the best possible results at batch size 128, with the best accuracy over both train and test data sets.

Proceeding to the second phase of our tests, we will conduct tests on our learning rate while keeping our batch size fixed at 128, and hidden dimensions at 128.

Learning Rate	Accuracy on Test	Accuracy on Train
0.001	**94.79**	**96.10**
0.005	93.16	96.10
0.01	94.30	91.56
0.05	88.11	90.26
0.1	65.31	64.29
0.5	65.31	64.29

Table 2. Experimenting with learning rate, keeping the batch size at 128 and hidden dimensions at 128.

It can be clearly seen that the efficiency decreases with an increase in the learning rate. Therefore we can conclude that our accuracy is best achieved at a learning rate of 0.001.

Moving on to our last phase of the experiment, we have already come to the conclusion that a learning rate of 0.001 and a batch size of 128 gives the most optimum results, so we will try to find out the optimum value of hidden dimensions as well. Below are our observations:

Batch size	Accuracy on Test	Accuracy on Train
32	84.20	85.06
64	90.39	88.31
128	**93.00**	**96.1**
256	**95.11**	**94.81**

Table 3. Experimenting with hidden dimensions, keeping the learning rate at 0.001 and batch size at 128.

There seem to be two promising results with the best possible outcomes in terms of accuracy. The number of hidden dimensions in a perceptron can significantly impact the entire model's capacity to understand complex patterns and influence both underfitting and overfitting tendencies.

5. Code

The perceptron for the prediction of diabetes on the data set "Pima Indian Dataset" available at https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database has been implemented and uploaded to GitHub and can be accessed at https://github.com/ujjwalb18/Predicting_Diabetes_Using_Perceptron_UofA.git

6. Conclusion

With all the rigorous and vast testing conducted over our data in this project, we have gained knowledge on how the accuracy of a perceptron can depend on various factors like learning rate, batch size, and hidden dimensions. Our rigorous parameter tuning and experimentation underscore the importance of comprehensive optimization in harnessing the full potential of neural network models for accurate diabetes classification. We were able to achieve an accuracy of 94.30

With this, we can conclude our project can accurately detect diabetes in the given data, using a multi-layered perception.

7. Future Work

We can further carry forward our study and focus on considering multiple parameters while experimenting. We can make a different split ratio between the test and the train data, and consider using a different number of layers as well. We could delve into optimizing their capacity to handle non-linearly separable data by exploring different activation functions, network depth, and different training algorithms.

References

[1] Algorithm - a hands on introduction.

[2] Perceptron algorithm - a hands on introduction, november 2020.

[3] Anjali Bhardwaj. What is a perceptron? – basics of neural networks, Oct 2020.

[4] Peshawa Muhammad Ali and Rezhna Faraj. Data normalization and standardization: A technical report, 01 2014.

[5] Xianlin Wang, Yuqing Liu, and Haohui Xin. Bond strength prediction of concrete-encased steel structures using hybrid machine learning method. *Structures*, 32:2279–2292, 09 2021.