Perceptron

Dhasarath Dhayanidhi
The University of Adelaide
a1906632@adelaide.edu.au

Abstract

Perceptron, a Biomimetic replication of the primary functional unit of the human brain known as neuron. In this paper we aim to explain the architecture, working and test the perceptron's ability on machine learning tasks. We then further make any possible improvements and report the results accordingly.

1. Introduction

Perceptron is an algorithm which mimics the working of a neuron and are the building blocks of a Neural Network. We will see the structure, workings and comparison of perceptron with a neuron. We will then test the perceptron model with a prediction problem specifically the presence or absence of diabetes in a patient using PIMA diabetes dataset. We will show any improvements and their respective results and conclude the paper with a future scope or improvements.

1.1. History

Perceptron architecture was invented in 1943 by Warren McCulloch and Walter Pitts and the first implementation was executed in 1957 by Frank Rosenblatt. In his paper published in 1958 Rosenblatt explained about the 3 cell units from his implementation which perform the functions of projection, association and response.

Advancements were stagnant for quite a while due to two major reasons. One being that a single perceptron is a binary classifier, and researchers believed that it will not be able to handle multiple classes until the finding of multi-layer perceptron which showed that had a greater ability to do that. Even the addition of one more single layer allowed the model to learn more complex relationships which aren't linearly separable. Other being an incorrectly reported research results which showed (incorrectly) that the perceptron can't learn the XOR relationship which being cited by researchers over a period, led to significant loss of interest and funding into research. All the errors were corrected in 1987 when a second version of this paper was released with the corrected results and explanations.

1.2. Architecture

Billions of neurons connected in a highly complex ways forms the thinking part of the brain, they receive inputs as chemical and electrical signals, which then they process and transmit to other neurons and parts of the brain, which then in turn sends messages to other parts of the body to control them. When taken at a single neuron level, the scope of this paper, it has mainly 3 parts, dendrites, axon and axon terminals as seen in Fig 1. Dendrites receive the input and passes it to axon, which then process it, and sends it to the axon terminal to be sent as input to multiple other neurons.

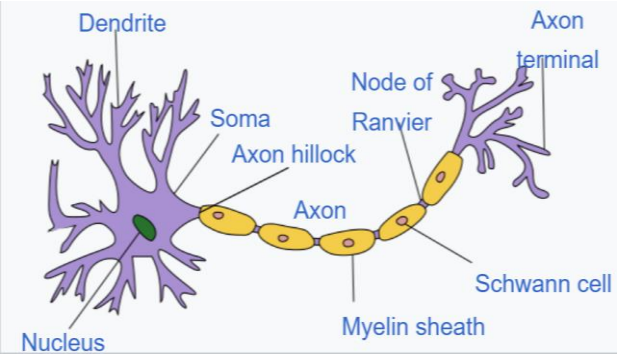

Figure 1: A Single Neuron

Perceptron is a biomimicry of a single neuron. (Biomimicry is "translating nature's strategies into design", that is taking nature's designs and workings as inspiration to create solutions. E.g. Tokyo's railway planning using slime). like Neuron, Perceptron has Input nodes, internal layers and output nodes as seen in Fig 2. where the data flows into the input node, processed in the internal layer and given out through the output node. An activation function can be used in the output node to introduce non-linearity. The interconnected layers are just a combination of connections from the input nodes to the center of the perceptron. The inputs are then multiplied with their respective weights and added bias to form the net value which will then be passed to activation function before being sent as output.

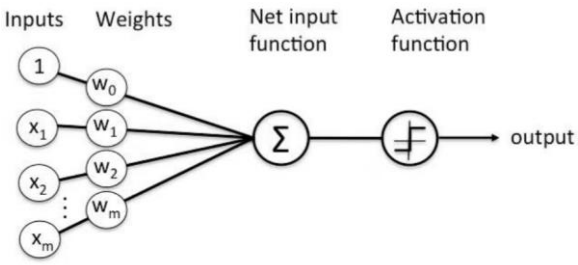

Figure 2: A Perceptron

2. Description

As we have already introduced how the data flows through the perceptron to the output layer, this is called as a feed forward run. Let's assume that the bias is b, and the weights are $w_1,\ldots,w_n$ and inputs are $x_1,\ldots,x_n$, then the Summation i.e. the net input function S, is calculated by

$$S = b + \sum_{i=1}^{n} w_i.x_i \quad (1)$$

The output layer consists of the activation function and the output node. The purpose of activation is to provide the ability of learning non-linear data for neural networks by deciding whether to allow the data to be passed onto to the next node or not. Since a perceptron has only a single layer it wouldn't be able to learn non-linearly separable data, so activation function might not improve the model much. But we are going to test with the sigmoid activation function which gives the result between -1,1. I feel that using sigmoid activation we can push the model to reach better scores that is make the model predict values much closer to the actual value, if not penalize it more. Step and Sigmoid functions,

$$H(x) = \begin{cases} 1 & ;x > 0 \\ 0 & ;x < 0 \end{cases} \quad (2)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (3)$$

If a perceptron just had feedforward run, it will calculate the output and find the loss and change the weights using Gradient Descent. Gradient Descent is an optimization algorithm which uses the gradients to minimize the error between predicted and actual value by modifying the learnable parameters. But this would mean that all the weights would be penalized with the same amount, irrespective of their respective input's importance.

Backpropagation, with it's first applied introduction in 1989 by Yann LeCun at bell labs for digit recognition, lead to huge jump in the performance and progress of perceptron and neural networks. Backpropagation is an iterative process, which determines how much of the net loss between the actual and predicted should be passed onto layer after layer from last layer and how much should the weights and biases of a particular layer should be penalized before passing on the remaining loss to the previous layer. So, the process of training a neural network follows, generating the predictions in forward pass, calculating the loss, back propagating respective losses/gradients to each bias and weight, and using gradient descent to find minimize the losses for further training runs. But since we are using a single layer perceptron, backpropagation will not be as useful as using it for multi -layer perceptrons or deep neural networks (complex models made of multiple connections of perceptrons). So, the update of the weights and bias is happening for a single layer. The weights will now be update using the following,

$$W_{t_i} = W_{t_{(i-1)}} + lr * \big(Pred(x_j) - y_j\big)x_j \quad (4)$$

To summarize, the predictions will be made using the existing weights as seen in formulae (1) and using activation function as seen in formulae (2,3) and the error is calculated, then the weights will be adjusted using the formulae (4), then the above process will be re-iterating till it reaching convergence.

3. Experimental Analysis

3.1. Dataset

The Dataset is collected by "National Institute of Diabetes and Digestive and Kidney Diseases", the intent being to develop a model which can predict the presence or absence of diabetes. The current dataset is a filtered from a larger database, using multiple constraints, one being that the dataset consists of the data collected from only females of the Pima Indian heritage who are at least 21 years old.

The data is loaded from the provided *link*, the unscaled version of it was selected as scaled one wasn't split into train, test and validation set before scaled or preprocessed so data leakage can happen leading to better performance with data from this dataset but might perform worse in unseen dataset. Even if it was split, since now it's a single dataset, and we don't have enough information to replicate the earlier split.

The file format was a svm, so the data is first parsed and saved in a local file, then read through a sklearn's function. There are 9 columns in total, "Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree, Age and target". There are 768 rows of data with no missing values. Some of the variables were distributed normally (Glucose, Blood Pressure, BMI), whereas the other variables were Positive (right) skewed. The data was unbalanced based on the target, consisting of 500 of the entry from patient having diabetes and 268 from the patients who doesn't have diabetes. All the above insights were inferred from the plotted graph in Appendix.

3.2. Experimentations

The base model was trained with sigmoid activation function and the targets were changed to have 0 and 1 values. The base model had no scaling, a split ratio of [0.7,0.15,0.15] for 100 epochs, batch size of 32, learning rate of 1 and no decay. The base model had a validation accuracy of 62.61 and f1 of 71.9. all the below reported metrics are for validation. All the following variables were experimented to arrive at a best model,

1. Scaling
2. Scaler
3. Split Ratio
4. Epochs
5. Batch Size
6. Learning Rate
7. Learning Rate Decay

The better suited parameter or the better reasoned one was chosen before moving on to the next feature.

Values	Accuracy	F1 Score
No	51.3	60.56
Yes	55.65	65.77

Table 1 Scaling Results

Values	Accuracy	F1 Score
Standard	68.7	76.32
MinMax	62.61	74.25

Table 2 Scaler Results

Values	Accuracy	F1 Score
[0.7, 0.15, 0.15]	68.7	76.32
[0.7, 0.2, 0.1]	64.94	73
[0.8, 0.1, 0.1]	74.03	79.17

Table 3 Split Ratio Results

Values	Accuracy	F1 Score
25	72.73	78.35
50	71.43	77.08
100	74.03	79.17

Table 4 Epochs Results

Values	Accuracy	F1 Score
4	62.34	72.9
8	62.34	73.39
16	59.74	70.48
32	74.03	79.17
64	64.94	71.58

Table 5 Batch Size Results

Values	Accuracy	F1 Score
1	74.03	79.17
0.1	72.73	78.79
0.01	72.73	78.35
0.001	64.94	76.11

Table 6 Learning Rate Results

Values (learning rate, decay=0.1)	Accuracy	F1 Score
(1,0.1)	75.32	79.57
(0.1,0.1)	75.32	79.57
(0.011,0.1)	75.32	79.57

Table 6 Best of Learning Rate Decay Results

Since the learning rate decay had 3 best performers, the accuracy graph was looked at, but since that was also not conclusive, the best selected model was (0.1,0.1) as it had good learning not to overshoot the global minima and at the same time doesn't take too many epochs to converge or stuck at local minima. From the above experimentation the model with the following hyperparameters were chosen, scaling as True, scaler as StandardScaler, split ratio as [0.8,0.1,0.1], epochs as 100, learning rate as 0.01, batch size as 32 and learning rate decay as 0.1. This produced a test accuracy of 67.53 and f1 score of 80.

Upon some more manual experimentation with choosing based on the graphs (convergence and fluctuations) a model with the following hyperparameters produced better results so chose to go with it, scaling as True, scaler as StandardScaler, split ratio as [0.8,0.1,0.1], epochs as 100, learning rate as 0.001, batch size as 4 and learning rate decay as 0. This produced a test accuracy of 74.03 and a f1 score of 82.76.

4. Code

GitHub Repo: https://github.com/dhashu0015/Deep-Learning

5. Conclusion

I have learnt how to construct and train a perceptron model from scratch. I have learnt to read svm files, internal architectures of perceptrons and deep neural networks, the theory of backpropagation and gradient descent, activation functions etc. I have also learned how to experiment and choose hyperparameters to arrive at a better model and how each of them influences the model and it's performance. Future ideas would include creating and testing of multi layered perceptrons, deep neural networks which can handle and perform better with lesser sized data, experiment on more hyper parameters such as activation functions, losses etc. We can also try and train the model on a larger dataset or a more evenly balanced dataset or try to up sampling or down sampling. The conclusion is that the final model shows good performance but not enough to be considered for productionizing in the field of medical science as the errors needs to be very minimal, so I would prefer developing a more complex model or if the size were a constraint I would try with better balanced data before productionizing it, adding more data might also help.

6. References

- (n.d.). Retrieved from Biomimicry Institute: https://biomimicry.org/inspiration/what-is-biomimicry/
- (n.d.). Retrieved from Simplilearn: https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron#:~:text=A%20Perceptron%20is%20a%20neural,value%20%E2%80%9Df(x).
- (n.d.). Retrieved from 3blue1brown: https://www.3blue1brown.com/lessons/backpropagation

(n.d.). Retrieved from Kaggle: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

Chandra, A. L. (n.d.). Retrieved from Medium: https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975