Comparing the Performance of Single Layer Feedforward Neural Networks Versus Young Adults in Binary Classification Tasks

Norman Kong
Dawson College, Montreal, Canada

Abstract

In the past decade, progress within the field of artificial intelligence has been remarkable. Modern machine learning algorithms are already surpassing human ability in various tasks. This study explores how the performance of one of the simplest neural networks, the perceptron model, fares against young adults in binary classification tasks. In a small study (n=30), young adults between the ages of 18-19, split evenly between scientific and non-scientific domains, score about 8% less on average than the perceptron model when classifying linearly separable data points. Future studies should be conducted with a greater and more varied sample size and should investigate the underlying causes of the disparity in accuracy scores. The code is available here, on GitHub.

Keywords: perceptron model; binary classification;

1. Introduction

The human brain is one of the most advanced processors of information in the universe. In the everlasting search to create more and more proficient algorithms, humans have speculated whether it was possible to somehow apply structures of the brain to hardware. This was compelling to researchers because if we could create a brain equivalent on a machine, we could then amplify it by giving it virtually unlimited resources of computing power. Thus, the idea of an artificial neural networks was born. In 1957, the perceptron algorithm was invented by Frank Rosenblatt (Shiffman). Although completely outdated now, the perceptron model lead the way to much more intricate and intelligent neural networks such as convolutional neural networks (CNN), recurrent neural networks (RNN), etc., which can complete sophisticated actions such as object tracking and sentiment analysis (Held, Thrun, & Savarese, 2016; Zhang, Wang, & Liu, 2018). Many of the models of today can actually outperform humans in complex tasks. For example, researchers from Oxford and Google collaborated to create *LipNet*, an algorithm that greatly surpasses humans in lipreading accuracy (Assael, Shillingford, Whiteson & de Freitas, 2016). However, can one of the more basic neural networks, more specifically, the perceptron, a single layer feedforward neural network, outperform

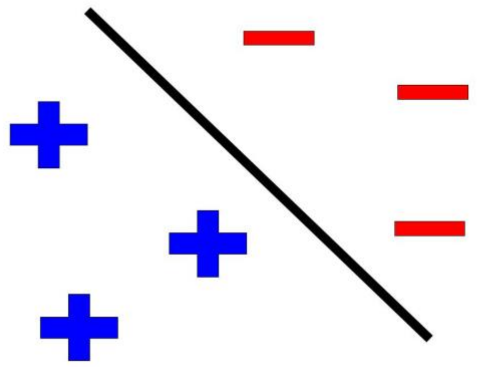

Figure 1. A basic diagram of a set of points to illustrate linear separability. For completeness, the definition of linear separability is as follows: in Euclidean geometry, if there exists a line (or hyperplane if in higher-dimensional Euclidean spaces) that can separate a set of data points that consists of two distinct groups, then the set is linearly separable (Gutiérrez, Pérez-Lantero, & Torres, 2016).

humans in a simple binary classification task? It has been proven that if the data is linearly separable (Figure 1), the perceptron algorithm has a finite limit to the number of mistakes it will make in its predictions, so if it is trained over a sufficient amount of iterations, it will eventually classify the testing set correctly (Freund & Schapire, 1999). For clarity, "humans" in this paper will refer to young adults in CEGEP between the ages of 18-19.

2. Methods

2.1 Perceptron Model and the Math Theory

The perceptron model is one of the simplest neural networks possible because it is a single layer feedforward neural network. The perceptron interprets a data point in the format of an input vector $\vec{A}$ with independent components (or *features*) $\overrightarrow{x_1}$, $\overrightarrow{x_2}$, …, $\overrightarrow{x_n}$, and computes a sum of these features in order to predict the class (or *label*) of that data point to be either 1 or 0. It should be noted that these features all affect the output of the perceptron to varying degrees, and so the sum computed by the perceptron is calculated by a dot product (Minsky & Papert, 1988; Nielsen, 2018):

$$z(x_i, w_i) = \sum_{i=1}^{n} x_i * w_i \qquad (1)$$

In other words, each component $\overrightarrow{x_n}$ is multiplied by what is called a *weight*, denoted by w_i, which is a real number that can be interpreted as the significance of a given component to the final output.

In binary classification tasks, the value provided by this dot product is wrapped within a sigmoid function in order to assure all outputs will be between 0 and 1. At its most basic, the sigmoid function is defined as (Han & Moraga, 1995):

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad (2)$$

Additionally, the derivative of the sigmoid function is simple to calculate, which is useful for calculating gradient descent (Han & Moraga, 1995). In this study, it is convenient to designate a threshold of 0.5, so that the prediction of the perceptron can easily be translated to a label prediction. That is, if $\sigma(z)$ is greater than 0.5, the perceptron will predict a 1, and if $\sigma(z)$ is less than 0.5, it will predict a 0.

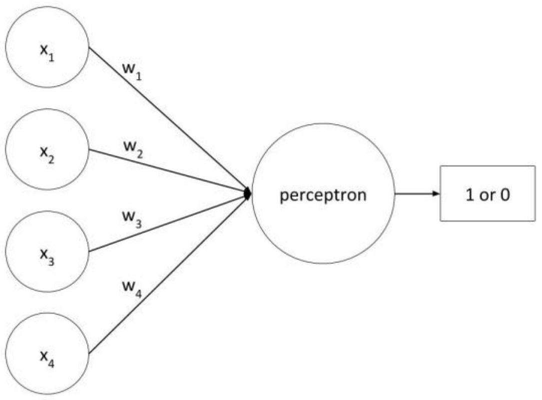

Figure 2. A visual representation of the perceptron model. Each component of the input vector is on the left, multiplied by its corresponding weight, to give a prediction.

Initially, all that is known is the data given by the training set and the testing set. The relation between the features and the label is unknown. Consequently, the weights must be initialized randomly when the perceptron is created. Randomized weights lead to poor predictions, and so the model must have some sort of learning process. This self-correction is accomplished through back-propagation. First, a cost/loss function must be defined. It is a quantitative measure of how far off the prediction, $\sigma(z)$, is from the correct value, a. In this study, the squared loss (also known as L_2 loss) function is used:

$$C(w_1, w_2, w_3, w_4) = (\sigma(z) - a)^2 \qquad (3)$$

Note that C is a function of σ, which is a function of z, which is in turn a function of w_1, w_2, w_3 and w_4, so C is in fact a function w_1, w_2, w_3 and w_4. Squared loss was chosen because for several reasons: firstly, both positive and negative error would be valued equally; second, greater error would entail a greater cost; and third, the derivative of such a function is relatively simple to compute. In order to ameliorate future predictions, it would logically follow to try to minimize this cost function. This can be accomplished through *gradient descent*. The gradient is defined to be (Stewart, 2016):

$$\nabla C(w_1, w_2, w_3, w_4) = \langle C_{w_1}, C_{w_2}, C_{w_3}, C_{w_4} \rangle \; (4)$$

Gradient descent is appropriate because the gradient vector provides the direction of fastest increase of $C(w_1, w_2, w_3, w_4)$ (Stewart, 2016). Thus, by adjusting the weights in the opposite direction of the gradient, that is, subtracting the components of the gradient from the corresponding weights, we can find specific values for each weight so as to minimize the cost function, which in turn leads to more accurate results.

To find the gradient vector of the cost function, it is necessary to use the chain rule from multivariable calculus (Figure 3). The partial derivative of the cost function with respect to w_i is computed as follows:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \sigma} * \frac{\partial \sigma}{\partial z} * \frac{\partial z}{\partial w_i} \qquad (5)$$

where,

$$\frac{\partial C}{\partial \sigma} = 2 * (\sigma(z) - a) * \frac{\partial \sigma}{\partial z} \qquad (6)$$

$$\frac{\partial \sigma}{\partial z} = \sigma(z) * (1 - \sigma(z)) \qquad (7)$$

$$\frac{\partial z}{\partial w_i} = x_i \qquad (8)$$

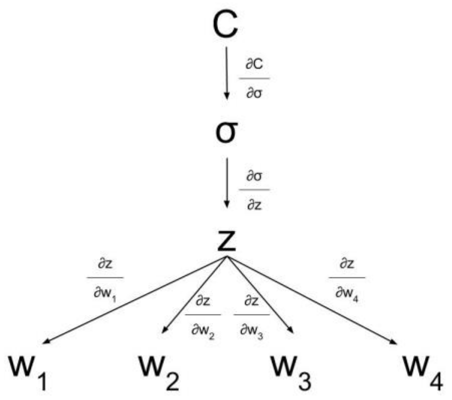

Figure 3. A visual representation of the chain rule. In order to find the partial derivative of C, it is necessary to follow each "branch" of the tree (Stewart, 2016).

2.2 Hyperparameters of the Model

This model is defined by several hyperparameters. First is the batch size; that is, the number of data points, out of the training data set, that are used to calculate the gradient in a single iteration. This model will utilize stochastic gradient descent, that is, the batch size will only include one data point from the training set (Goodfellow et al., 2016). This method is commonly used to save computational resources, especially when dealing with extremely large datasets, because computing the gradient over becomes too computationally taxing (although that is not necessarily the case here).

Another hyperparameter for this model is the number of epochs, which is the total number of times the model will traverse the entire set of training data (Goodfellow et al., 2016).

One last element involved in gradient descent is the learning rate, denoted by η. The learning rate is a constant that represents how quickly the model will correct itself by dictating how large the magnitude of each "step" in the direction of the gradient vector, will be. Therefore, the weights are updated over many iterations as follows:

$$\overrightarrow{w_{k+1}} = \overrightarrow{w_k} - \nabla C(w_1, w_2, w_3, w_4) * \eta \quad (9)$$

where $\overrightarrow{w_{k+1}}$ is the updated weight, $\overrightarrow{w_k}$ is the previous weight, $\nabla C(w_1, w_2, w_3, w_4)$ is the gradient vector and η is the learning rate (Nielsen, 2018).

The implementation of this model in this study was done in the Java programming language but could easily be reproduced in similar object-oriented languages such as Python. The specific values for the hyperparameters of this implementation are:

- batch size: 1
- learning rate η: 0.5
- number of epochs: 3

It should be noted that due to the simplicity of the relationship between features and label, similar results can be found even when these parameters are dramatically modified.

2.3 Testing and Training Data
The training and testing data sets are sufficiently small to include here, shown in Table 1 and Table 2.

Training Data					
Data Point	Features				Label
A	0	0	1	0	0
B	1	1	1	0	1
C	1	0	1	1	1
D	0	1	1	1	0
E	0	1	0	1	0
F	1	1	1	1	1
G	0	0	0	0	0

Table 1. The training data. Each data point, denoted arbitrarily by A, B, and so on, has 4 features and 1 label. The pattern, described qualitatively, is that the label will take on the value of the number of the first feature (from the left).

Testing Data					
Data Point	Features				Label
A	0	1	1	0	0
B	0	0	0	1	0
C	0	1	0	0	0
D	1	0	0	1	1
E	1	0	0	0	1
F	1	1	0	0	1
G	1	0	1	0	1

Table 2. The testing data. Each point, arbitrarily denoted by A, B and so on (but not to be mistaken with the training data as they are all distinct points forming two distinct groups), contains 4 features and one label. The perceptron model is capable of achieving a score of 100% accuracy when tested on this data set and after having been trained on the training set.

2.4 Human Component
30 CEGEP students participated in this study. They were evenly split between science and non-science programs and all were between the ages of 18-19. Through online media, they were asked to study the training set, both features and labels, for 5 minutes. Then, they were explained that the training and testing set were homogeneous in nature (i.e. contained the same patterns) and so were asked to predict what they thought the label was for each data point in the testing set.

3. Results

After training, the perceptron managed to score an accuracy of 100% when tested on the testing dataset. Additionally, its weights adjusted themselves towards values that follow the logic of the pattern of the training set. More specifically, the weights strongly favour the first component of the input vector (Table 3). This is to be expected because it is indeed solely the first component of the input vector that affects the value of label. It should be noted that since the weights are initialized with pseudorandom numbers, there is variation from one training session to the next. However, each run produced weights similar in format and magnitude and scored an accuracy of 100% on the testing set. Results from student testing can be found in Table 4 and Table 5.

Initial Weights	Final Weights
0.6651506768208026	4.28727366812632
0.11960869227004922	-0.6814778089534985
0.6513523764658022	-1.1052446205477703
0.2722507058738065	-0.6772754550786568

Table 3. Weights before and after processing the training set over 3000 iterations. Each weight corresponds to a different component of the input vector $\vec{A}$.

	Science	Non-Science	Total
Number	15	15	30
Average Score (%)	92.3809524	91.4285714	91.9047619
Standard Deviation	17.7992258	18.5478657	17.8678129

Table 4. The scores of two groups of students, divided by their general field of study. Science students average 1% more above non-science students. Both groups combined score about 8% less accurate than the perceptron model.

Program	Score (on 7)	Answers
Science	6	0, 0, 0, 1, 0, 1, 1
Science	3	0, 0, 0, 0, 0, 0, 0
Science	4	0, 0, 1, 1, 0, 1, 0
Non-Science	5	0, 0, 0, 1, 0, 0, 1
Non-Science	4	0, 0, 0, 0, 0, 0, 1
Non-Science	3	0, 0, 0, 0, 0, 0, 0

Table 5. Incorrect answers from student responses. For completeness, the correct sequence of labels is: 0, 0, 0, 1, 1, 1, 1.

4. Conclusion

Both average score and standard deviation varied little between science and non-science groups. When human results are compared to the perceptron model, we see that the perceptron scores 8% higher than humans. This evidence supports the fact that even simple models like the perceptron algorithm can outperform humans in certain tasks. Limitations of this study include a small sample size (n=30) as well as the fact that participants were chosen based on my personal social network, which almost certainly introduces bias. Limitations of this model include the fact that it is only capable of learning linearly separable patterns. Further studies should be done using more complex patterns within the data, on a larger and more diverse sample size, as well as with a deeper analysis of potential causes for differences in human classification.

Acknowledgements

I thank Sameer Bhatnagar, Jean-François Brière and Jonathan Sumner for their help and guidance throughout.

References

Assael, Y. M., Shillingford, B., Whiteson, S., and de Freitas, N. (2016). Lipnet: Sentence-level lipreading. CoRR, abs/1611.01599.

Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning - The Eleventh Annual Conference on Computational Learning Theory,37*(3), 277-296. doi:10.1023/A:1007662407062

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning. MIT Press.

Gutiérrez, G., Pérez-Lantero, P., and Torres, C. (2016). Linear separability in spatial databases. CoRR, abs/1602.04399.

Han, J. and Moraga, C. (1995). The influence of the sigmoid function parameters on the speed of backpropagation learning. volume 930, pages 195–201.

Held, D., Thrun, S., and Savarese, S. (2016). Learning to track at 100 FPS with deep regression networks. CoRR, abs/1604.01802.

Minsky, M. L., & Papert, S. A. (1988). *Perceptrons* (Expanded ed.). Halliday Lithograph.

Nielsen, M. A. Neural Networks and Deep Learning. Retrieved May 19, 2019, from http://neuralnetworksanddeeplearning.com/chap1.html

Shiffman, D. (2012). The Nature of Code. Retrieved May 18, 2019, from https://natureofcode.com/book/

Stewart, J. (2016). Calculus (8th ed.). Cengage Learning

Zhang, L., Wang, S., and Liu, B. (2018). Deep learning for sentiment analysis: A survey. CoRR, abs/1801.07883.