A Brief Discussion on Artificial Neural Networks

Research Project Progress Report 1

Submitted

by

Umasankar Nayak

(Roll Number : 417MA5017)

under the supervision of

Prof. Snehashish Chakraverty

September, 2021

Department of Mathematics

National Institute of Technology Rourkela

A Brief Discussion on Artificial Neural Networks

1 Introduction

Neural Networks have been widely implemented in several disciplines of engineering and science, with a significant number of successes in solving issues such as predicting, machine translation, images or voice recognition, and any other variety of recognition of pattern , which is complicated task for us.

2 Study of Neural Networks

McCulloch and Pitts developed Artificial Neural Networks (ANN) in 1947, which capture complicated input-output correlations from data by artificially mimicking the human brain. Nervous System of humans is made up of a large number of interconnected processing units known as Neurons. Dendrites are a group of thin fibres which help a biological neuron collect the information from the surroundings. Soma: This is where the collected information is totalled up ,then passed through a long cylindrical rod known as an axon. Synapse: This is where information is transferred from one neuron to another.Pratihar [1]).

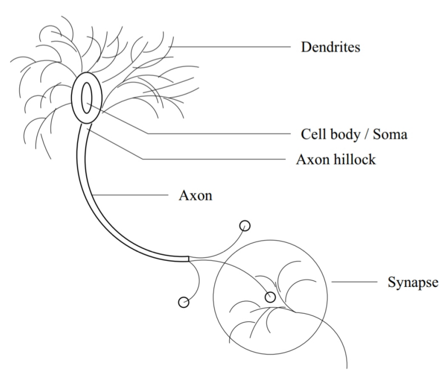

Figure 1: Human Nervous System

Let's use n as an example $[I] = [I_1, I_2, ..., I_n] = f^{[0]}$ and $[W]^{[1]T} = [w_1^{[1]}, w_2^{[1]}, ..., w_n^{[1]}]^T$ are weight for 1st hidden layer of Neural Network. One input and output layer, as well as 1 or more hidden layers, comprise an ANN. The weighted inputs are added with bias (Comparable to dendrites and soma) and sent through an a Transfer function $u_[1]$ (Comparable to with axon and synapse) for the determining the output of neural networks.RELU, Sigmoid, Tanh etc. are various Transfer function in ANN. (Sharma and Sharma [2])

$$\hat{o} = g(u_[1]) = g([I] * [W]^{[1]T} + [b]^{[1]})$$

or

$$\hat{o} = g(u_[1]) = g(f^{[0]} * [W]^{[1]T} + [b]^{[1]})$$

Where $[b]^{[1]}$is just the bias in 1st hidden layer.

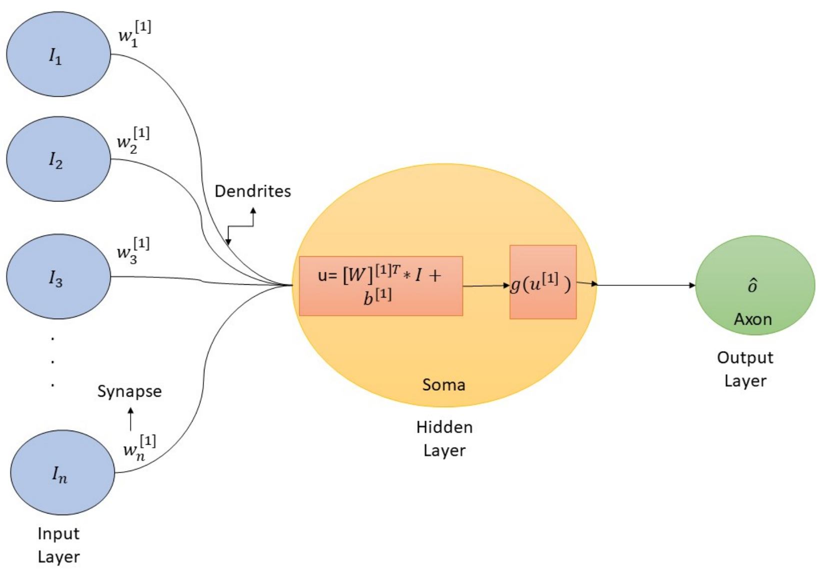

Figure 2: A diagram of a Single Layer Artificial Neural Network

2.1 Sigmoid Transfer Function

$$g(u) = \frac{1}{1 + e^{-u}}$$

Advantages: Normalize the output in the scale 0,1 .
Disadvantages:Support Vanishing Problem Gradient (i.e., past and present weights are the same in back propagation in weight updating) .

2.2 RELU Function

$$g(u) = \begin{cases} 0 & ; u < 0 \\ u & ; otherwise \end{cases}$$

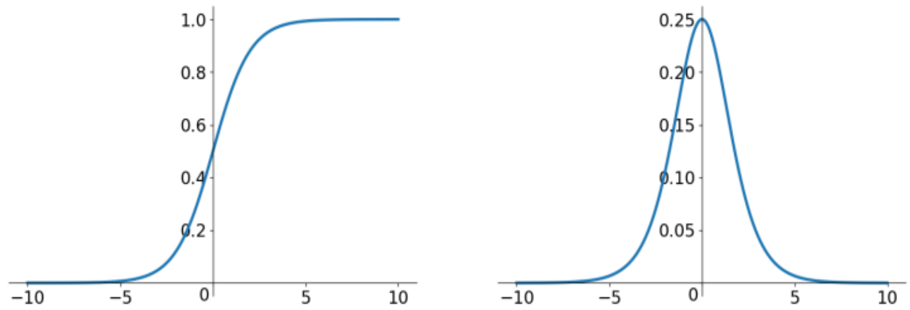

Figure 3: Sigmoid Transfer Function and The derivative of the sigmoid function

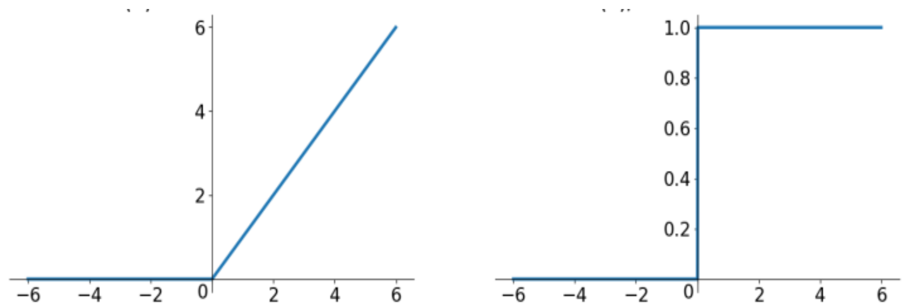

Figure 4: Relu Transfer Function and The derivative of the RELU function

3 Neural Network with Multiple Layer and its Training

Suppose a Neural Network with layers m, input layer is layer 0, the output layer is layer m, and the hidden layers are layers 1 until m - 1. $[x_1, x_2, x_3, ..., x_n] = f^{[0]}$ (say) are the inputs multiplied by $w^{[1]T}$ is the weights and add $b^{[1]}$ is the bias for 1st layer. That will send through a Transfer Function $f^{[1]}$ for the 1st layer. The results of layer 1 is used as the input in the next layer. A similar procedure will be followed, then finally $f^{[m]} = \hat{o}$ is expected output. i.e. for layer l :

$$u_i^{[m]} = w_i^{[m]T} * f_i^{[m-1]} + b_{[}i]^{[m]}$$

$$f_i^{[m]} = g(u_i^{[m]})$$

This Process will be known as **feed forward Neural Network**. The original output o is compared to expected $\hat{o}$ and loss(L) is to determined.

$$L = \sum_{i=1}^{n} (o - \hat{o})^2$$

As a result, our goal is to minimise loss (i.e., modify the weight so that the expected value matches the original value). so use optimizer for such a reason. There are a variety of optimizer available, such as Adam, Gradient Descent,Adagrad , Adadelta, Mini batch Stochastic Gradient Descent, etc. However, Adam is the best in the end.(Oppermann [3]).

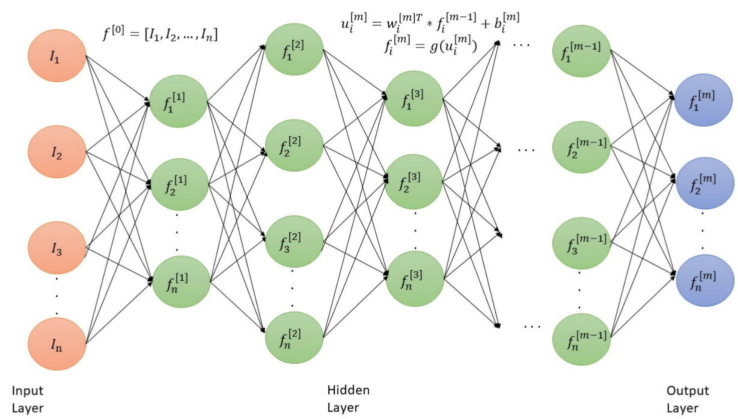

Figure 5: Multiple Layer of Neural Network

3.1 Gradient descent

Gradient descent attempts to minimize the loss of training a neural network by updating weights .

$$L = \sum_{i=1}^{n} (o - \hat{o})^2$$

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$$

Where η is the learning rate.
Disadvantage: Computationally Expensive and that's why low rate of convergence.

3.2 Mini batch Stochastic Gradient descent

Instead of taking all the n data point for calculation of loss use k number of data point (k<n).

$$L = \sum_{i=1}^{k} (o - \hat{o})^2$$

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$$

Disadvantage: It Doesn't guarantee good convergence, in this process noise was also generated so we add exponential weighted average in SGD to avoid noise and this is called SGD with momentum.

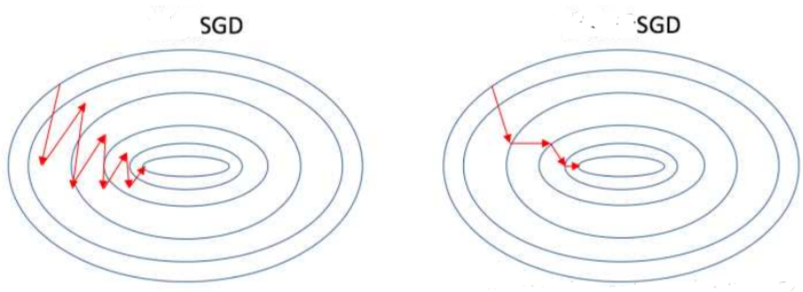

Figure 6: SGD without Momentum SGD without Momentum

3.3 Adagrad

We use different learning rate η for each iteration and for all parameters.

$$w_t = w_{t-1} - \eta_t' \frac{\partial L}{\partial w_{t-1}}$$

$$\eta_t' = \frac{\eta}{\sqrt{\alpha_t - \epsilon}}$$

$$\alpha_t = \sum_{i=1}^{m} (\frac{\partial L}{\partial w_i})^2$$

Disadvantage: Sometime as increase in number of iteration α_t increases(i.e. vary vary high) so that η decreases and we get w_t= w_{t-1} with the help of next optimizer Adadelta we can fix the problem.

3.4 Adadelta and RMSProp

Adadelta used to reduce the learning rate in Adagrad.

$$w_t = w_{t-1} - \eta_t{}' \frac{\partial L}{\partial w_{t-1}}$$

$$\eta_t' = \frac{\eta}{\sqrt{WeightedAverage - \epsilon}}$$

$$WeightedAverage_t = \gamma * WeightedAverage_{t-1} + (1-\gamma)(\frac{\partial L}{\partial w_{t-1}})^2; \gamma = 0.95$$

3.4.1 Adam

It is a combination of Momentum (to get smooth noise) and RMSProb (to reduce Learning rate).

4 Data Normalization

In order to improve, faster and accurate of training we transfer the data in standard scale.

4.1 MinMaxScalar

(Martin, Yohai, and Zamar [4])

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}(max - min) + min$$

Where max,min are feature range.

5 Strategy to Design a Neural Network

- Find out independent (output) and dependent (input) features and normalize them.
- Set Number of hidden layer,appropriate activation function, number of neuron, weights (normalized scale) and bias (randomly) in each layer.
- Update weights through Back propagation and conitnue the iteration till stoping criteria reached.

6 Progress

(Q).Is it possible for a neural network to approximate $x^2$for using neural network?

Yes, Since we've discussed above, that Neural Networks may easily approximate the function by collecting input-output data. So we developed it in Python using the Keras Library with a 3 (keras [5]) neural network using 10,000 random data as input in the range [50,50], and the results are shown in the graph below.

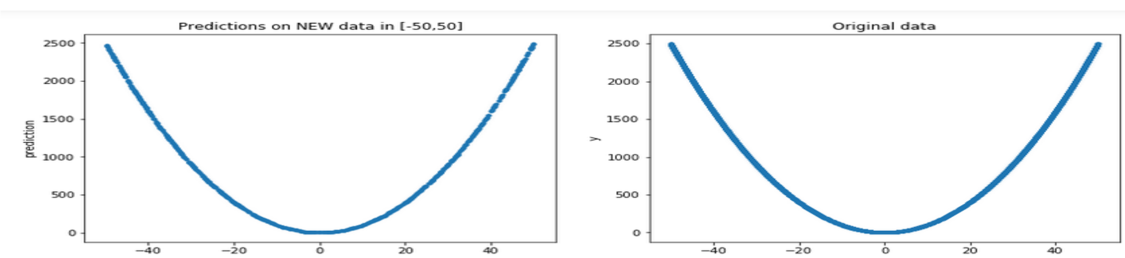

Figure 7: Approximated value vs Original Value

References

[1] D. K. Pratihar, *Soft computing: fundamentals and applications*. Alpha Science International, Ltd, 2013.

[2] S. Sharma and S. Sharma, "Activation functions in neural networks," *Towards Data Science*, vol. 6, no. 12, pp. 310–316, 2017.

[3] A. Oppermann. (Oct. 20, 2019). "Optimization algorithms in deep learning," [Online]. Available: https://medium.com/mindboard/lstm-vs-gru-experimental-comparison-955820c21e8b.

[4] R. D. Martin, V. J. Yohai, and R. H. Zamar, "Min-max bias robust regression," *The Annals of Statistics*, vol. 17, no. 4, pp. 1608–1630, 1989.

[5] keras. (). "Keras documentation," [Online]. Available: https://keras.io/.