Journal of Machine Learning Research 1 (2020) 1-7 Submitted 11/15

Multi-Layer Neural Networks

Jieyi Chen

November 15, 2020

1. Introduction

This project explores the application of multi-layer neural networks on classification and regression datasets. Neural network is inspired by the structure of the neuron in the brain. Each neuron is composed by dendrites, cell body and an axon. The neuron receives inputs from dendrites (attributes), performs the computation (weighted sum and sigmoid activation) in the cell body and send the output to other neurons via axon (weights). The structure of the neural network enables the algorithm to optimize the relationship between the attributes and the target value. Neural network has gained popularity in recent years due to the advance in computational power. Its ability to learn by itself without formal statistical training, detect complex nonlinear relationship and discover interactions between the attributes [1] draw a lot of attention in the machine learning field. Some common applications of neural networks are speech recognition, real-time translation and text classification. In this project, we implement three neural network types (zero hidden layer, one hidden layer, two hidden layers). The neural network with no hidden layer is identical to the logistic regression model and the networks with hidden layers require backpropagation to update the weights. Specifically, we use stochastic gradient descent. We hypothesize that the neural network with two hidden layers outperform the ones with no and one hidden layer because deeper network can better detect the relationship between the input and output values. To test this hypothesis, we compare the performance of the neural network on the classification and regression datasets from the UCI repository [2].

The rest of the paper is organized as follows. Section 2 describes the algorithms that are implemented. Section 3 discusses the experimental approach. Section 4 presents the results of the experiments. Section 5 discusses the results. Section 6 concludes.

2. Algorithm Implementation:

The multi-layer neural networks contain two important computations: Forward propagation and backpropagation. The forward propagation takes the inputs, computes the weighted sum, processes it via the activation function and passes the output to the next layer. The backpropagation calculates the error between the network output and the target value, computes how the weight change affects the error (the partial derivative of the error with respect to the weight) using the chain rule and updates the weights with the gradient.

2.1.1 Forward Propagation:

Figure 1, 2 and 3 visualize the forward propagation for neural network with no hidden layer, one hidden layer and two hidden layers.

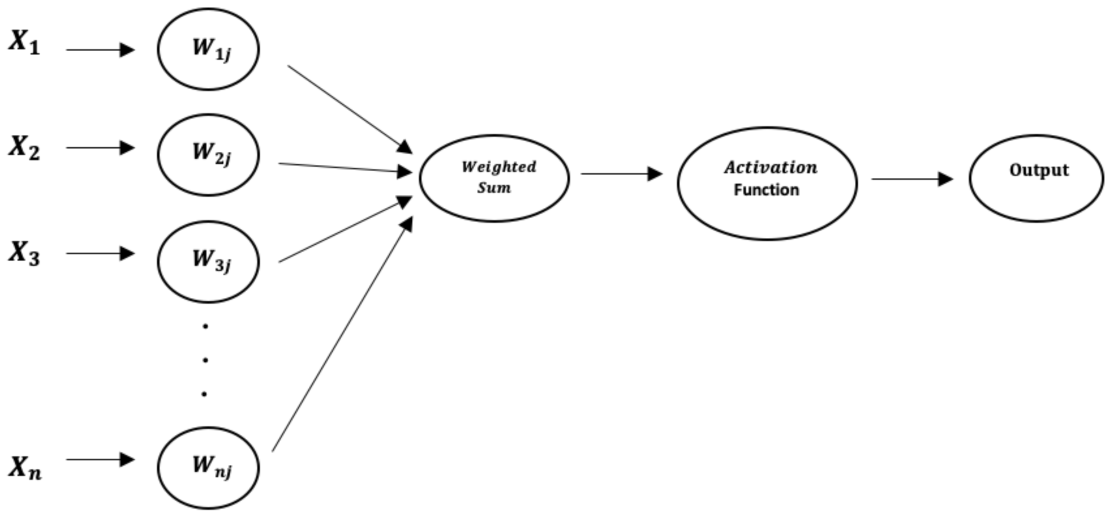

Figure 1: Neural network with no hidden layer

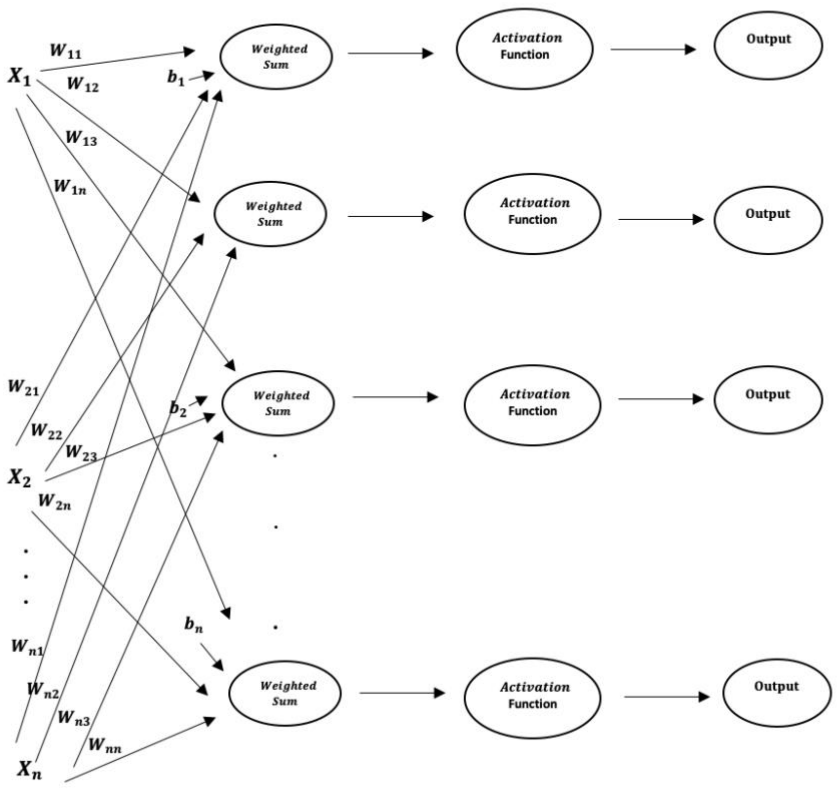

Figure 2: Neural network with one hidden layer

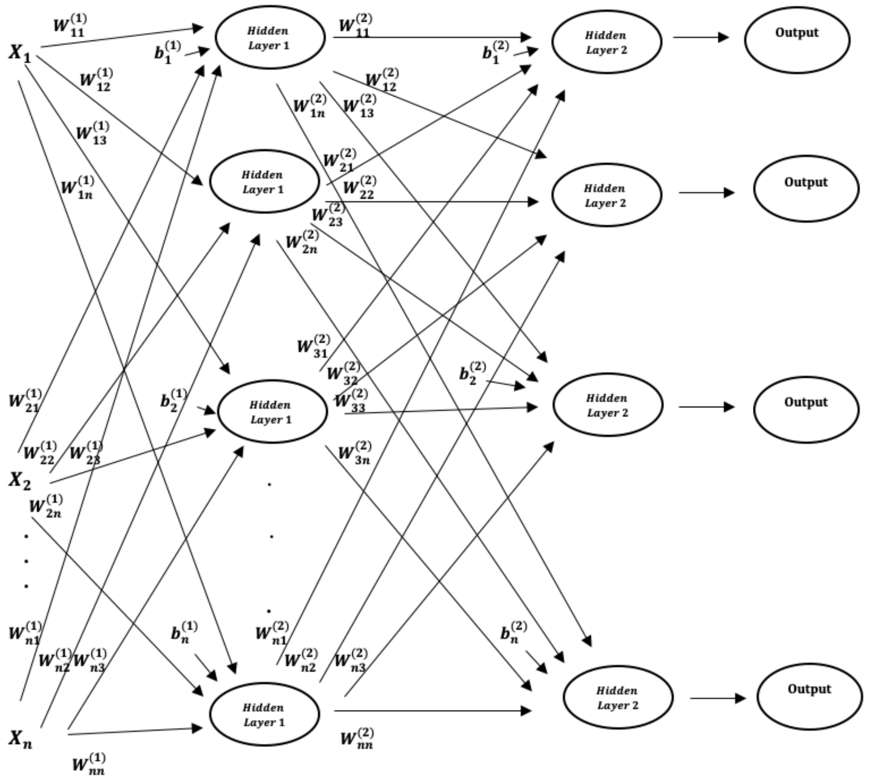

Figure 3: Neural network with two hidden layers

<u>2.1.2 Backpropagation:</u>

In 1986, Rumelhart, Hinton and Williams discussed backpropagation in their paper and published in the neural net literature [3]. The goal of backpropagation is to optimize the weights in the neural network so that it can correctly map the inputs to the outputs. Figure 4 illustrates the backpropagation process.

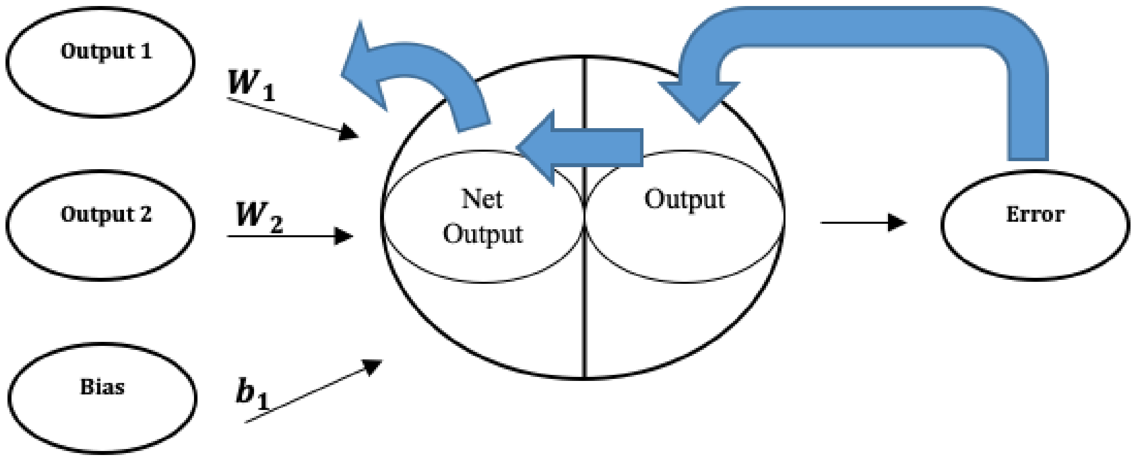

Figure 4: Backpropagation

Notation:

X$_{ji} = i^{th}\ input\ to\ unit\ j$

$W_{ji} = weight\ from\ i\ to\ j$

$net_j = \sum_j W_{ji} X_{ji}$

$O_j = network\ output\ of\ unit\ j$

$d_j = desired\ output\ for\ unit\ j$

$\sigma() = sigmoid\ function$

$output = units\ at\ the\ last\ layer$

$downstream(j) = set\ of\ units\ whose\ immediate\ inputs\ include\ the\ output\ of\ unit\ j$

$$\Delta W_{ji} = -\eta \frac{\partial Err_s}{\partial W_{ji}}$$

$$Err_s(w) = \frac{1}{2} \sum_{k\ \epsilon\ outputs} (d_k - O_k)^2$$

$$d_k = desired\ output$$

$$O_k = network\ output$$

We need to apply the chain rule of differentiation to get the following.

$$\frac{\partial Err_s}{\partial W_{ji}} = \frac{\partial Err_s}{\partial net_j} * \frac{\partial net_j}{\partial W_{ji}} = \frac{\partial Err_s}{\partial net_j} * X_{ji}$$

Output Nodes:

$$\frac{\partial Err_s}{\partial net_j} = \frac{\partial Err_j}{\partial O_j} * \frac{\partial O_j}{\partial net_j}$$

$$\frac{\partial Err_s}{\partial O_j} = -(d_j - O_j)$$

$$\frac{\partial O_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\sigma(net_j)} = O_J(1 - O_j)$$

$$\frac{\partial Err_s}{\partial net_j} = -(d_j - O_j)O_J(1 - O_j)$$

$$\Delta W_{ji} = -\eta \frac{\partial Err_s}{\partial W_{ji}} = \eta(d_j - O_j)O_J(1 - O_j)X_{ji}$$

Hidden Nodes:

$Let\ \delta_i = \frac{-\partial Err_s}{\partial net_i}$

$$\frac{\partial Err_s}{\partial net_j} = \sum_{k\,\epsilon\, downstream(j)} \frac{\partial Err_s}{\partial net_k} * \frac{\partial net_k}{\partial net_j}$$
$$= \sum_{k\,\epsilon\, downstream(j)} \delta_k * \frac{\partial net_k}{\partial net_j}$$
$$= \sum_{k\,\epsilon\, downstream(j)} \delta_k * \frac{\partial net_k}{\partial O_j} * \frac{\partial O_j}{\partial net_j}$$
$$= \sum_{k\,\epsilon\, downstream(j)} \delta_k * W_{kj} * \frac{\partial O_j}{\partial net_j}$$
$$= \sum_{k\,\epsilon\, downstream(j)} \delta_k * W_{kj} * O_j(1 - O_j)$$

$$\delta_j = O_j(1 - O_j) \sum_{k\,\epsilon\, downstream(j)} \delta_k * W_{kj}$$

$$\Delta W_{ji} = \eta \delta_j X_{ji}$$

<u>**2.2.1 Algorithm:**</u>

<u>Parameters Definition:</u>
W: weight vector, X: input vector, O: network output, d: desired output, η: learning rate in the range (0, 1)

1. Set the number of nodes, hidden layers for the network
2. Initialize the weights and bias for the entire network and their values range from -1 and 1
3. Repeat the following step a-h for k epochs
    a. Receive the attributes of an instance in the training set as the input values for the network
    b. Calculate the weighted sum of the input values and their initialized weights
    $$\mathrm{u} = \sum_{\mathrm{i}=0}^{\mathrm{n}} \mathrm{w_i x_i} + \mathrm{b_i} \qquad n = total\ number\ of\ attributes$$
    c. Send the weighted sum through the sigmoid activation function
    $$\mathrm{d} = \frac{1}{1 + \mathrm{e}^{-\mathrm{u}}}$$
    d. Set the result from c as the input values for the next layer, repeat step b and c until we reach the last layer of the network
    e. Calculate the error for each output node using the squared error function
    $$Err = \sum \frac{1}{2}(d - O)^2$$
    f. Compute the partial derivative of the error with respect to the weight, update the weight with the gradient value and the learning rate η
    $$w_{(t+1)} \leftarrow w_t + \eta \frac{\partial Err}{\partial w_t}$$
    g. Repeat step f for all the weights in the network

h. Use the updated weights as the initialized weights for the next instance in the dataset and repeat step a-g until all the instances are used for training. This iterative weight updating method is called stochastic gradient descent, a strategy for optimizing an objective function with an estimate gradient calculated from a random sample in the training set.

3. Experimental Approach:

3.1 Datasets:

The datasets are from UCI Machine learning Repository [1]. Table 1 summarizes the properties of the six datasets.

Dataset Property Summary Table						
	Breast Cancer Dataset	**Glass Dataset**	**Soybean Dataset**	**Abalone Dataset**	**Computer Hardware Dataset**	**Forest Fires Dataset**
# Attributes	9	9	35	8	7	12
# Examples	699	214	47	4,177	209	517
Type of dataset	Classification	Classification	Classification	Regression	Regression	Regression

Table 1 - Dataset Property Summary Table

3.2 Data Preprocessing Steps:

The breast cancer datasets have missing values and we use mean imputation and the conditional probability of the attribute given the class to handle them. We separate the observations with missing values based on their classes and replace those values with the average of the attribute in the class. In addition, we normalize the numerical attributes so that their distribution is in the range of -1 to 1.

3.3 Hyperparameter Tuning Process:

In order to find the optimal hyperparameters, we reserve 10% of the data for tuning, and 2/3 of the remaining 90% for training and 1/3 for testing. The hyperparameters for the neural network algorithms are the learning rate η {0.1, 0.01, 0.001}, number of iterations {10, 30, 50} and the number of nodes in the hidden layer {5, 8, 10} for classification and {1, 3, 5} for regression.

4. Result

Neural Network Classification Algorithm - Accuracy									
	Breast Cancer Dataset			Glass Dataset			Soybean Dataset		
	Number of Hidden Layer			Number of Hidden Layer			Number of Hidden Layer		
	0	1	2	0	1	2	0	1	2
Fold 1	0.961	0.984	0.945	0.61	0.439	0.341	1.0	0.778	0.333
Fold 2	0.961	0.976	0.961	0.488	0.415	0.341	1.0	0.444	0.333
Fold 3	0.976	0.921	0.984	0.537	0.463	0.341	0.889	0.778	0.333
Fold 4	0.953	0.969	0.984	0.475	0.35	0.35	1.0	1.0	0.333
Fold 5	0.975	0.967	0.967	0.571	0.536	0.429	1.0	0.833	0.5
Average Accuracy	0.965	0.963	0.968	0.536	0.441	0.361	0.978	0.767	0.367

Table 2 - Algorithm Accuracy Summary Table

Neural Network Regression Algorithm - Mean Square Error									
	Abalone Dataset			Computer Hardware Dataset			Forestfire Dataset		
	Number of Hidden Layer			Number of Hidden Layer			Number of Hidden Layer		
	0	1	2	0	1	2	0	1	2
Fold 1	5.683	4.777	5.913	4,500	43,461	9,967	776	13,601	12,700
Fold 2	16.498	7.834	4.989	2,735	21,285	13,533	863	1,269	1,419
Fold 3	7.843	3.999	4.741	15,939	4,549	6,056	18,763	7,355	6,563
Fold 4	5.892	6.814	5.526	7,226	14,810	10,868	664	963	1,250
Fold 5	19.863	5.927	5.504	26,134	11,362	75,690	1,343	621	452
Average Mean Square Error	11.156	5.870	5.335	11,307	19,093	23,223	4,482	4,761	4,477

Table 3 - Algorithm Mean Square Error Summary Table

Hyperparameters Summary Table								
			Classification			Regression		
Number of Hidden Layer	Number of Fold	Hyperparameter	Breast Cancer Dataset	Glass Dataset	Soybean Dataset	Abalone Dataset	Computer Hardware Dataset	Forestfire Dataset
0	1	η (Learning rate)	0.1	0.1	0.1	0.1	0.1	0.001
		# of iterations	10	30	10	50	10	30
		# of Nodes	0	0	0	0	0	0
	2	η (Learning rate)	0.1	0.1	0.1	0.1	0.01	0.001
		# of iterations	10	30	10	10	50	10
		# of Nodes	0	0	0	0	0	0
	3	η (Learning rate)	0.1	0.1	0.1	0.01	0.1	0.001
		# of iterations	30	10	10	50	10	10
		# of Nodes	0	0	0	0	0	0
	4	η (Learning rate)	0.1	0.1	0.1	0.01	0.1	0.001
		# of iterations	30	10	10	10	10	10
		# of Nodes	0	0	0	0	0	0
	5	η (Learning rate)	0.1	0.01	0.1	0.01	0.1	0.001
		# of iterations	30	30	10	30	10	10
		# of Nodes	0	0	0	0	0	0
1	1	η (Learning rate)	0.1	0.1	0.1	0.001	0.01	0.1
		# of iterations	10	30	30	30	30	10
		# of Nodes	8	8	8	5	5	1
	2	η (Learning rate)	0.1	0.1	0.1	0.001	0.01	0.1
		# of iterations	10	50	10	50	30	10
		# of Nodes	8	8	10	5	5	5
	3	η (Learning rate)	0.1	0.1	0.1	0.001	0.01	0.1
		# of iterations	10	50	30	30	50	30

		# of Nodes	8	10	10	5	5	3
	4	η (Learning rate)	0.1	0.1	0.1	0.001	0.01	0.1
		# of iterations	10	50	50	50	50	30
		# of Nodes	8	5	5	3	5	1
	5	η (Learning rate)	0.1	0.1	0.1	0.01	0.001	0.1
		# of iterations	10	50	30	10	30	30
		# of Nodes	8	10	5	5	5	5
2	1	η (Learning rate)	0.1	0.1	0.1	0.001	0.01	0.001
		# of iterations	10	10	10	30	10	50
		# of Nodes	5	5	5	3	5	5
	2	η (Learning rate)	0.1	0.1	0.1	0.001	0.01	0.1
		# of iterations	10	10	10	10	50	50
		# of Nodes	5	5	5	5	5	1
	3	η (Learning rate)	0.1	0.1	0.1	0.001	0.001	0.01
		# of iterations	10	10	10	30	50	50
		# of Nodes	5	5	5	5	5	3
	4	η (Learning rate)	0.1	0.1	0.001	0.001	0.01	0.01
		# of iterations	10	10	50	50	30	10
		# of Nodes	5	5	8	3	1	3
	5	η (Learning rate)	0.1	0.1	0.01	0.001	0.001	0.1
		# of iterations	10	10	50	50	30	50
		# of Nodes	5	5	5	5	5	1

Table 4 - Hyperparameters Summary Table – Neural Network

5. Discussion:

Table 2 shows that the neural network with different number of hidden layers have similar accuracy in the breast cancer dataset. The neural network with no hidden layer performs the best in the glass and soybean datasets, followed by the ones with one and two hidden layers. Table 3 shows that the neural network with two hidden layers have the lowest mean square error, followed by zero and one hidden layer for the Abalone dataset. The neural network with no hidden layer performs the best in the computer hardware dataset and the error started to increase as more hidden layers are added. The neural network with two hidden layers outperforms the ones with zero and one hidden layer in the forestfire dataset. We can also see that datasets with more training instances such as breast cancer and abalone datasets tend to have better performance. The soybean dataset has the highest accuracy with neural network that has no hidden layer and the error on the test set started to increase as we add more layers. The result shows that our initial hypothesis is incorrect because adding more layers might cause overfitting, which leads to worse performance on the test set. Table 4 shows the optimal hyperparameters: learning rate, number of iterations and number of nodes per fold in each of the three neural network types.

6. Conclusions:

This project helps us to gain a better understanding of neural network, forward propagation, backpropagation and stochastic gradient descent. This paper can help analysts to implement the neural network for classification and regression datasets. Some future work on how to optimize the neural network for multi-class classification datasets can be done. We can also compare the performance of gradient descent and stochastic gradient descent in different types of datasets to see which strategy performs better.

7. Reference:

[1] Tu, Jack V. "Advantages and Disadvantages of Using Artificial Neural Networks versus Logistic Regression for Predicting Medical Outcomes." Journal of Clinical Epidemiology, Pergamon, 22 Mar. 1999, www.sciencedirect.com/science/article/abs/pii/S0895435696000029.

[2] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[3] Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0