ANN Model Using Gradient Descent to Train Weight Vectors

Xiaoyu He
School of Physical Science
University of California - Irvine
Irvine, U.S.
xiaoyuh8@uci.edu

***Abstract*— Over the past decades, Artificial Neural Network (ANN) has been widely and effectively applied in various learning tasks such as image processing and emotion analysis. A main advantage of ANN is that the performance of the ANN model is not affected by the input variables of the model. Besides, certain invisible but important correlations could be recognized by the ANN model. In this paper, we would like to find out how each layer transfers data to the next layer and how it uses the data and loss to train its weight vectors in every epoch using gradient descent. The experimental results demonstrated that the proposed model achieved promising performance with the increase of epochs. Furthermore, we note that various decay of loss can be observed by verifying different learning rates.**

Keywords-ANN model, Prediction task, Independent performance, Layer transformation, Loss function, Train weight vectors, Gradient descent.

I. INTRODUCTION

Over the past, some ideas of the artificial neural network have been proposed and further studied. found [1]. For instance, McCulloch-Pitts claimed the back of binary threshold neurons. And Rosenblatt introduced the use of perceptrons [2], a specific type of neurons to perform a variety of tasks flexibly. However, after Minsky and Papert, a series of studies mathematically proved that perceptrons could not do very much.

Recently, the neural network has been popular again [3]. Increasing research has emerged in the area. Some capable learners, breaking technical development level results on a wide variety of tasks [4], have been active in the study of neural networks. And this improvement is helped by technology's progress, such as computing hardware, allowing us to train large complex networks in reasonable time [5]. With the ANN model's continuous application, ANN has achieved promising performance in various tasks [6].

The idea of developing an ANN mimicking the biological neural network is very intriguing [7]. It was first started as an attempt to figure out what and how human brains are composed of. It is soon reoriented and aims to improve the outcome or result of computations based on past experiences [8]. We are interested in how ANN calculation works in each step and how the convergence error of the product generated by the network when compared to the outcome that we should get. We would like to implement an ANN from scratch using python for classification problems [9]. In our design, there are three layers in the model: the input layer, the hidden layer, and the output layer. We would like to find out how each layer transfers data to the next layer and uses the data and loss to train its weight vectors in every epoch using gradient descent [10].

The main contributions of this work can be summarized as follows:

1. The Artificial Neural Network's calculations demonstrate how the convergence error of the product generates by the network.
2. The Artificial Neural Network is applied in classification problems.
3. We verify how each layer transfers data to the next layer and how the proposed model achieved promising performance with epochs' increase.

The rest of this paper is organized as follows. In Section 2, the structure of ANN is introduced in detail. The experimental results are reported and analyzed in Section 3. Finally, Section 4 concludes this paper.

II. MODEL FORMULATION

A. Structure of ANN

In this section, we introduced the main structure of ANN consisting of the input layer, hidden layer, and output layer, illustrated in Fig.1.

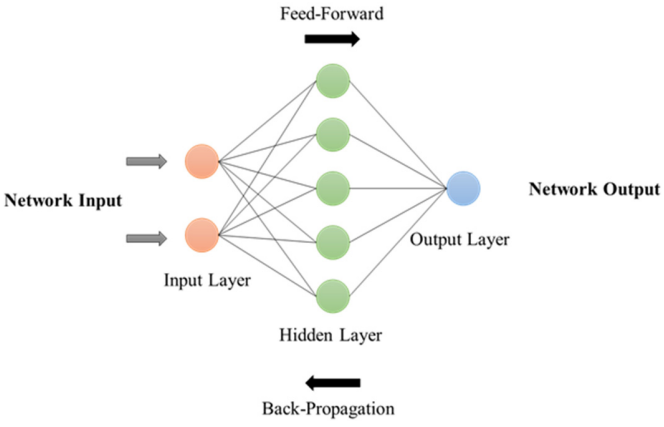


Figure 1. Introduction to Artificial Neural Networks.

978-0-7381-1215-2/21/$31.00 ©2021 IEEE

2021 International Conference on Communications, Information System and Computer Engineering (CISCE) | 978-1-6654-0352-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/CISCE52179.2021.9445953

To begin with the implementation of ANN, we initialize the weight vectors: w1 with a shape of (input dim, hidden dim) and w2 with a shape of (hidden dim, output dim), where the input dimension is the number of features in the data fed in the model. The parameters in the hidden dimension are used to tune for better training results. And output dimension is the number of distinct classes in the data. To avoid the overfitting problem, we introduce a bias vector in the manipulation. Specifically, we construct our ANN in this manner:

$$\begin{aligned} & data, x \rightarrow input\_layer, z_1 = w_1^T x + b_1 \\ & \quad \rightarrow activation\_function(relu / sigmoid), a_1 = f(z_1) \\ & \quad \rightarrow output\_layer, z_2 = w_2^T x + b_2 \\ & \quad \rightarrow activation\_function(soft\max), a_2 = soft\max(z_2) \end{aligned} \tag{1}$$

B. Input Dimension-Number of Features

The function of the input layer is to receive as input the values of the explanatory attributes for each observation. Inputs can be loaded from an external source such as a web service or a csv file. In general, the number of input nodes is equal to the number of explanatory variables. “input layer” presents the patterns to the network, which communicates to one or more “hidden layers”. The data is hardly changed by the fixed nodes in the input layer. A single value can be regarded as input and duplicate the value to the corresponding outputs. From the input layer, it duplicates each value and sends it to all the hidden nodes.

The input layer of a neural network comprises artificial input neurons and brings the initial data into the system for further processing by subsequent layers of artificial neurons. The input layer is the first step of the workflow for the artificial neural network. The number of neurons in an input layer is dependent on the shape of your training data. Traditionally, the number of neurons is the number of training data feature + 1 (one additional node is to capture the bias term.)

C. Hidden Dimension-Parameter to Tune

In neural networks, a hidden layer is located between the input and output of the algorithm. The function applies weights to the inputs and directs them through an activation function as the output. Namely, the hidden layers perform nonlinear transformations of the inputs entered into the network. Following this, incoming arcs connect hidden nodes and the corresponding input nodes. It connects with outgoing arcs to output nodes or other hidden nodes. In the hidden layer, the actual processing is done via a system of weighted connections. There may be one or more hidden layers regardless of the number of the input layers. The values were entering a hidden node multiplied by weights, which presents a set of predetermined numbers stored in the program. The weighted inputs are then added to produce a single number.

There could be zero or more hidden layers in a neural network regardless of the input layers' number. One hidden layer is sufficient for the large majority of problems. Generally, each hidden layer contains the same number of neurons. If the number of hidden layers in a neural network is large, then the time of producing the output is longer so that the more complex neural network problems need to be addressed.

D. Output Dimension-Number of Distinct Classes

The output layer receives connections from hidden layers or the input layer. The output layer returns an output value that corresponds to the prediction of the response variable. There is usually only one output node in classification problems. The active nodes of the output layer combine and change the data to produce the output values.

E. Implementation of ANN

The implementation of an ANN includes mainly two parts: the forward and backward steps. The forward step is merely to feed the data in the model and get the data's predicted labels. Since the dataset's actual labels in our training process, the Cross-Entropy loss function can be used to compute the loss between the actual labels and predict labels. The loss function is formulated in the following equation:

$$Loss = -\sum_{c=i}^{M} y_{a,i} \log(p_i) \tag{2}$$

where is an indicator of whether the current class i is the actual class of the training sample. If so, the value of is 1, otherwise its value is 0; is the probability we got from the output of the softmax function. There are several reasons why we choose the Cross-Entropy loss function rather than the mean-squared error (MSE). MSE is well advanced at measuring problems such as regression and prediction. However, the classification problems should be valued as a great decision boundary. Therefore, this work utilizes Cross-entropy to better measure the classification performance. The backward method is used to update the weight vectors. The following rules can update the corresponding vector:

$$w_i = w_i - learning\_rate \times \frac{dLoss}{dw_i} \tag{3}$$

$$b_i = b_i - learning\_rate \times \frac{dLoss}{db_i} \tag{4}$$

We use the chain rule to compute these gradients. The calculation of gradient in this work can be modeled by:

$$\begin{aligned} & \frac{dLoss}{dw_2} = \frac{dLoss}{da_2} \times \frac{da_2}{dz_2} \times \frac{dz_2}{dw_2} \\ & \frac{dLoss}{db_2} = \frac{dLoss}{da_2} \times \frac{da_2}{dz_2} \times \frac{dz_2}{db_2} \quad where \frac{dz_2}{db_2} = 1 \\ & \frac{dLoss}{dw_1} = \frac{dLoss}{da_1} \times \frac{da_1}{dz_1} \times \frac{dz_1}{dw_1} \quad where \frac{dLoss}{da_1} = \frac{dLoss}{dz_2} \times \frac{dz_2}{da_1} \\ & \frac{dLoss}{db_1} = \frac{dLoss}{da_1} \times \frac{da_1}{dz_1} \times \frac{dz_1}{db_1} \quad where \frac{dz_1}{db_1} = 1 \end{aligned} \tag{5}$$

In the entire training process, we employ gradient descent update to train our weight and bias vectors and observe the change of losses as more iterations occur. Moreover, we construct a neural network with the same structure using the PyTorch package, a python library. Using the same dataset and value of parameters (learning rate, hidden dimension, and epochs), a similar result of accuracy in the test dataset is obtained.

III. Experiments

A. Datasets

The dataset utilized in the experiment: mnist_test, mnist_train. All experiments are conducted on a computer with 2.60GHz Intel(R) Core (TM) i5-7300U processor and 8GB RAM under the 64-bit operating system.

B. Evaluation Metrics

$$Pred = wx + b$$

where Pred indicates prediction value, w indicates weight, x indicates input, and b indicates the bias term.

TABLE I. Random Integers Test Example

Epoch	Loss Function Value	Train Accuracy	Validation Accuracy
10	0.4225	0.7587	0.2314
20	0.3397	0.8213	0.3946
30	0.2855	0.8576	0.4955
40	0.2472	0.8882	0.5786
50	0.2188	0.9017	0.6351
60	0.1967	0.9145	0.6795
70	0.1791	0.9238	0.7359
80	0.1645	0.9338	0.7893
90	0.1523	0.9381	0.8308
100	0.1421	0.9409	0.8694

C. Experimental Results Analysis

To test whether the implementation works correctly, we use a vector of random oats as our data and a vector of random integers, ranging from 1 to 10, to denote which class those ten numbers belong to. Furthermore, we split the data into three groups: training, validation, and test, in the ratio of 6:1:1 (approximately). We set the learning rate to be 0.001 and 0.01 and the activation function to be relu. We set the learning rate to be 0.001 and 0.01, and the activation function to be relu. The input dimension and the number of features in the training samples are set to 10. The hidden dimension is set to 5. Of course, it is worth remarking that the output dimension, which represents the total number of classes, is 10 in this case.

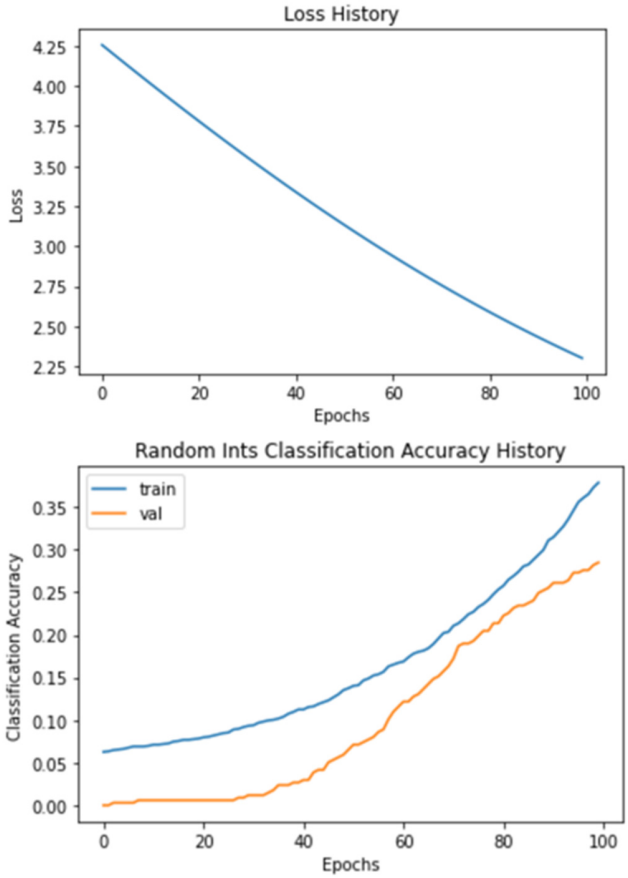
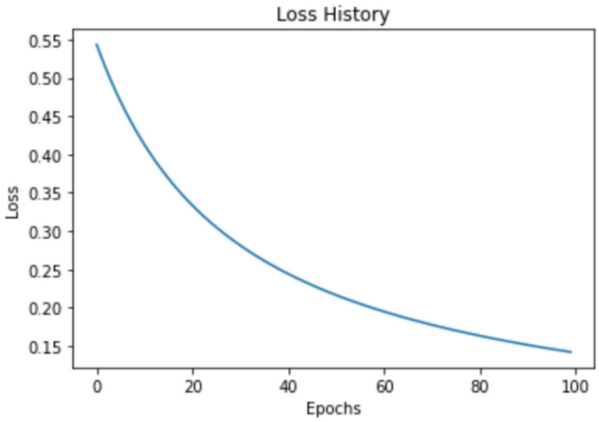


Figure 2. The correlation between loss history and epochs when Learning rate = 0.001.



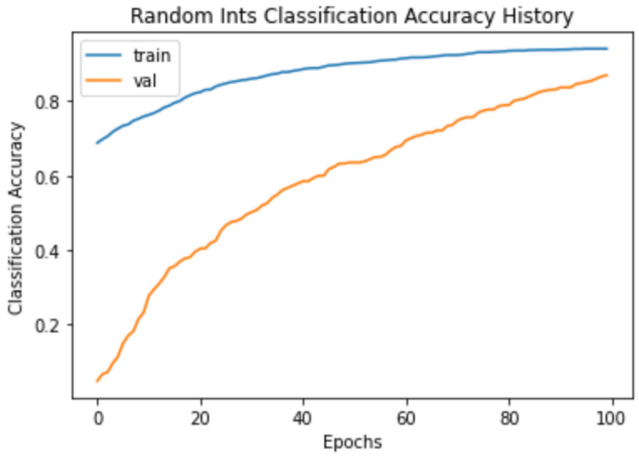


Figure 3. The correlation between loss history and epochs when Learning rate = 0.01

From an analysis of Figs. 2-3, it can be found decay of loss along with the increase of epochs. And the decay of loss indicates that the difference between predictions of the proposed model and the actual labels decreases. This is nicely demonstrated on the increasing trends of the accuracy plot. Furthermore, the two sets of graphs show the impact of the learning rate in the model. As the learning rate increasing from 0.001 to 0.01, it can be observed that the loss and accuracy graph converges in a shorter time. By contrast, we note that there are chances when the step of the learning rate is greater, it will miss the convergence point. In this case, the ANN model cannot be enhanced by the learning rate.

IV. Conclusion

This paper proposed how Artificial Neural Network apply to classification problems. The ANN model explains how those three layers (input layer, hidden layer, and output layer) transform data from the first layer to the next layer. Using gradient descent, the ANN model also uses the data and loss to train the weight vectors in every epoch. Trying different values of the learning rate, we can observe various rates of the decay of loss and also realize some characteristics of ANN. Unlike using packages like PyTorch, sklearn directly, we better understand the model and neural network as a whole: how it proceeds and updates the parameters using back-propagation.

In the future, the weakness of neural networks could easily be solved. For example, we can integrate neural nets with complementary technology. We expect to use the technique such as symbolic functions to make systems work together to produce a common goal. Moreover, new applications can be applied in neural nets by faster processing power and more sheer complexity. Furthermore, neural networks might allow the improvement of stock prediction.

References

[1] R. Gates, Myung Choi, S. K. Biswas and J. J. Helferty, "Stabilization of flexible structures using artificial neural networks," Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan), vol. 2, no. 4, pp. 1817-1820, 1993.

[2] G. P. J. Schmitz, C. Aldrich and F. S. Gouws, "ANN-DT: an algorithm for extraction of decision trees from artificial neural networks," IEEE Transactions on Neural Networks, vol. 10, no. 6, pp. 1392-1401, 1999.

[3] V. Z. Manusov, I. S. Makarov, S. A. Dmitriev and S. A. Eroshenko, "Training sample dimensions impact on artificial neural network optimal structure," 2013 12th International Conference on Environment and Electrical Engineering, vol. 12, no. 3, pp. 156-159, 2013.

[4] S. Wang, X. Wang, P. Ye, Y. Yuan, S. Liu, and F.-Y. Wang, "Parallel crime scene analysis based on acp approach," IEEE Transactions on Computational Social Systems, vol. 5, no. 1, pp. 244–255, 2018.

[5] S. Husain and M. Bober, "Improving large-scale image retrieval through robust aggregation of local descriptors," IEEE Transactions on Pattern Analysis and Machine Intelligence., vol. 39, no. 9, pp. 1783–1796, 2017.

[6] Yu-Xue Sun and Guang-Hui Guo, "Application of artificial neural network on prediction reservoir sensitivity," IEEE 2005 International Conference on Machine Learning and Cybernetics, vol.8, no.2, pp. 4770-4773, 2005.

[7] M. de Francesco and C. Pellegrini, "A functional framework for the specification of artificial neural networks," IEEE International Joint Conference on Neural Networks, vol. 2, no. 4, pp. 1290-1295, 1911.

[8] M. Mishra and M. Srivastava, "A view of Artificial Neural Network," 2014 International Conference on Advances in Engineering & Technology Research (ICAETR - 2014), vol.1, no. 1, pp. 1-3, 2014.

[9] L. Wang, "Dynamical models of stock prices based on technical trading rules—part II: analysis of the model," IEEE Transactions on Fuzzy Systems, vol. 23, no.4, pp. 1127-1141 2015.

[10] N. A. Al-Sammarraie, Y. M. H. Al-Mayali and Y. A. Baker El-Ebiary, "Classification anddiagnosis using back propagation Artificial Neural Networks (ANN)," 2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE), pp. 1-5. 2018.