Journal of Machine Learning Research 1 (2000) 1-48 Submitted 2/23; Published 2/23

Training Artificial Neural Networks for Classification and Regression

Brendan Foley BFOLEY3@JH,EDU
Department of Biotechnology
Johns Hopkins University
Baltimore, MD 21202, USA

Editor: Brendan Foley

Abstract

This paper details the creation, training, and tuning of three Artifical Neural Networks (ANNs). First, we buld a simple linear model capable of handling classification and regression by utilizing the softmax and linear activation functions, respectively. We also define a two-layer multilayer perceptron (MLP) with a varying number of nodes in its hidden layers, depending on the dataset training it. The final model we create is an autoencoder that first memorizes the inputs then, after removing the decoding layer, links up to a predictive hidden layer that is connected to the output layer. After these systems have been made, we begin to train and tune them. By forward propagating, finding the error of the prediction, then backpropagating several times, we are able to achieve high predictive power in most cases. After every training, the accuracy is assessed and the combination of nodes in the hidden layers is modified according to our tuning scheme described in Experimental Approach. This yields our optimal configuration, which can be used to show the accuracy of the general model on the dataset.

Keywords: Assignment 4, Artificial Neural Networks, ANN, Perceptron, Autoencoder, Machine Learning

1. Introduction

Artificial Neural Networks (ANN) were first realized in 1958 by Frank Rosenblatt in the form of the Perceptron. It was designed after the human brain's visual data processing and ability to recognize objects. However, their use was not limited to visual data and were soon adopted into various fields of research and development.

The Perceptron model is a single neuron that receives input, weighs them, and classifies that input through an activation function into one of 2 classes. After assessing the correctness of output, backpropagation occurs and the weights are adjusted accordingly.

This model has been expanded and generalized from a single neuron, binary classifier, into a multi-neuron, multi-layer network that is capable of $K > 2$ class classification, as well as regression. These models are classified as Multi-Layer Perceptrons (MLPs).

Another form of ANNs is the autoencoder, which is a type of MLP. It was introduced in 1986 by Rumelhart et al. as a generalization of principal component analysis (PCA). While traditionally employed as a dimnension reduction or feature learning algorithm, it was later used as a generative model.

In this paper, we create and test three neural networks: a simple linear network, a two layer

MLP, and an Autoencoder, across six datasets consisting of three for classification and three for regression. We predict that the autoencoder will outperform both the two layer MLP and the simple network, with the two layer network also outperforming the simple network.

2. Experimental Approach

In order to facilitate a general approach to creating multilayer perceptrons (MLPs), we defined several classes, inspired by PyTorch's approach. We first define an abstract class, Layer, off of which we create two classes, DenseLayer and ActivationLayer. These two layers are always used in conjunction with one another. The separation eases the implementation of backpropagation and allows for modifications on the fly, if required.

In a forward propagation, the DenseLayers coordinate input values and weights from the previous layer and output the dot product of the inputs, weights, and adds bias. This output is then passed to their respective ActivationLayer, which consists of an activation function, which can be defined as the user, as well as the respective derivative of the activation function. Within this paper, each hidden layer uses the hyperbolic tangent function as an activation function, while output layers use the linear activation function for regression and the softmax function for classification.

On backpropagation, we must traverse the Layers in reverse order. We first go through the final activation layer, calculating the result of the activation function's derivative on the error presented. We then progress to the DenseLayer associated with the ActivationLayer and calculate the error of the weights and update them, while passing the error of the input out of our backpropagation function to facilitate the update of the weights in the preceeding layer. By the end of the backpropagation we have adjusted all of the weights according to the gradient of the error.

The final class of the implementation is the Network. This provides the structure onto which we can build our Layers. Network consists of a list of layers that we fill with a series of DenseLayers and ActivationLayers. It also includes a user-defined loss function and its derivative, for forward and backward propagation, respectively. Network includes a train function that handles forward propagation through each layer, sequentially, then proceeds to backpropagation in a reverse-layer order. This occurs on the total dataset as determined by the user defined integer parameter, epochs. Each epoch trains the model on the entirety of the training data in a forward pass, then backpropagates to correct error. The predict function is used after training and returns the results of forward propagation, which is the predicted values. Lastly, we implement a function to remove the most recently added layer to facilitate the hybrid autoencoder-prediction network where we need to remove the decoding phase of the autoencoder.

While not a class, we additionally define a Driver file that is the main controller of the entire MLP implementation. Within it we define the loss and activation functions, as well as several helper functions to create normalized features and to help assess functionality of various network implementations.

We also implemented tuning for the nodes in the hidden layers. For each hidden layer, we selected a range of 4 to number of inputs in the previous layer - 1 and tested the combinations of these in the two layer and autoencoder models, selecting the one with the best accuracy or MSE, depending on the dataset. I.e. in a dataset with j features, we test

all permutations of Layer1 having [4, j-1] nodes and layer 2 having [4, # of nodes in Layer1] nodes. This is not an exhaustive list, but we progressed with the idea that we should be attempting to reduce the dimensions the further we get into the network, this lead us to the upper limit of our range being the number of nodes in the previous layer - 1. We performed this on each model for each dataset, yielding 18 optimized models.

3. Results

The optimal layers for each model and dataset pair are shown in tables 1 and 2. These were obtained by the previously described layer tuning method in the Experimental Approach. Tables 3 and 4, below, are summaries of 5x2 cross validation of each model's (Simple linear network, Two-Layer MLP, and Autoencoder) performance. For individual run performance, tables of the accuracies and MSEs are tables 3-8 in the Supplemental Tables section of the Appendix. Each case used 30 epochs for training. The learning rate for classification and regression were 0.1 and 0.01, respectively.

	Breast Cancer	Car Evaluation	Congress
Two Layer	Input-8-4-Output	Input-5-5-Output	Input-6-4-Output
Autoencoder	Input-6-6-Output	Input-5-5-Output	Input-5-5-Output

Table 1: The optimal number of nodes present in each layer of the two layer and autoencoder models for each classification dataset.

	Abalone	Computer	Forest Fires
Two Layer	Input-7-4-Output	Input-5-4-Output	Input-8-5-Output
Autoencoder	Input-5-5-Output	Input-5-5-Output	Input-7-7-Output

Table 2: The optimal number of nodes present in each layer of the two layer and autoencoder models for each regression dataset.

	Breast Cancer	Car	Congress
Simple Network	0.9778	0.90784	0.97702
Two Layer	0.9849	0.66396	0.84254
Autoencoder	0.9864	0.90784	0.64368

Table 3: The average accuracy of each model on the classification datasets.

	Abalone	Computer	Forest Fires
Simple Network	3.840	3222.1	1.18E+28
Two Layer	4.924	8912.74	2.10E+01
Autoencoder	4.269	4290.74	7.62E+00

Table 4: The average MSE of each model on the regression datasets.

4. Discussion

Based on the results we obtained, the two layer model struggled to get accurate results the most. This was surprising, as the simple linear model would often be thought to have a more simplistic output, as it cannot approximate nonlinear functions. The two layer model had 66% accuracy on the car evaluation data set, although it had nearly equal the predictive capabilities in the breast cancer dataset.
When looking at the autoencoder model, we see it struggle with the congressional vote dataset even more than the two layer model struggled with the car evaluation dataset, with 64% correct predictions.
The last surprisng performance was the simple linear network on the forest fires dataset, with a large MSE. This is most likely due to the use of more continuous features than other datasets.
It is not terribly surprising to see the autoencoder performing slightly worse in some cases when compared to the other models. The auto encoder is attempting to find a compressed version of the inputs, so some extra error may be introduced.

5. Conclusion

After running 5x2 cross validation across all three models and all six datasets, we have found that the simple linear network seems to perform consistently better, or at least close to as good, as the other two models, with the exception of the forest fires dataset. It was surprising to see that a simple linear model can perform classification and regression so well considering the possibility of nonlinear functions being present within the data.
The benefit of the autoencoder, however, is dimension reduction. Once reduced dimensions have been realized then the memory and runtime benefits for processing new instances begins take hold and we may elect to select the autoencoder, even if it has slightly poorer performance than other models, such as in the Abalone dataset where the simple network as a slightly smaller MSE.

6. Appendix

6.1 Supplemental Tables

	Run 1	Run 2	Run 3	Run 4	Run 5
Simple Network	0.9784	0.9785	0.9785	0.9821	0.9714
Two Layer	0.9856	0.9821	0.9821	0.9857	0.9892
Autoencoder	0.9784	0.9857	0.9928	1.000	0.9750

Table 5: The accuracy in predicting the class of the breast cancer dataset for each type of ANN

	Run 1	Run 2	Run 3	Run 4	Run 5
Simple Network	0.9220	0.9043	0.9043	0.9043	0.9043
Two Layer	0.6994	0.6551	0.6551	0.6551	0.6551
Autoencoder	0.9220	0.9043	0.9043	0.9043	0.9043

Table 6: The accuracy in predicting the class of the car evaluation dataset for each type of ANN

	Run 1	Run 2	Run 3	Run 4	Run 5
Simple Network	0.9828	0.9540	0.9713	0.9885	0.9885
Two Layer	0.6149	0.9885	0.6207	0.9943	0.9943
Autoencoder	0.9770	0.6207	0.6207	0.3793	0.6207

Table 7: The accuracy in predicting the class of the congressional vote dataset for each type of ANN

	Run 1	Run 2	Run 3	Run 4	Run 5
Simple Network	4.776	3.468	3.759	3.640	3.560
Two Layer	5.495	4.498	4.866	4.767	4.994
Autoencoder	4.999	4.240	3.922	4.009	4.175

Table 8: The MSE of the regression on the Abalone dataset for each type of ANN

	Run 1	Run 2	Run 3	Run 4	Run 5
Simple Network	796.2	7331.4	2378.0	1450.6	4154.3
Two Layer	4326.6	3102.7	8549.0	24384.6	4200.8
Autoencoder	2055.2	5508.3	4454.8	5015.1	4420.3

Table 9: The MSE of the regression on the Computer Hardware dataset for each type of ANN

	Run 1	Run 2	Run 3	Run 4	Run 5
Simple Network	2.2E+28	3.1E+24	1.3E+27	3.4E+28	1.5E+27
Two Layer	2.4E+01	2.4E+01	2.0E+01	2.0E+01	1.7E+01
Autoencoder	7.8E+00	1.2E+01	4.7E+00	5.9E+00	7.7E+00

Table 10: The MSE of the regression on the Forest Fire dataset for each type of ANN

Acknowledgements

All data used for this assignment was obtained from http://archive.ics.uci.edu/ml, from the University of California, Irvine, School of Information and Computer Science. The breast cancer dataset was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg

References

1. D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations byerror propagation. In Parallel Distributed Processing. Vol 1: Foundations. MIT Press, Cambridge, MA, 1986

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. O. L. Mangasarian and W. H. Wolberg: "*Cancer diagnosis via linear programming*", SIAM News, Volume 23, Number 5, September 1990, pp 1 & 18.

4. P. Cortez and A. Morais. A Data Mining Approach to Predict Forest Fires using Meteorological Data. In J. Neves, M. F. Santos and J. Machado Eds., New Trends in Artificial Intelligence, Proceedings of the 13th EPIA 2007 - Portuguese Conference on Artificial Intelligence, December, Guimarães, Portugal, pp. 512-523, 2007. APPIA, ISBN-13 978-989-95618-0-9.

5. Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386–408.