An Overview of Neural Networks with Principal Component Analysis

Harley Clifton, Becky Catlett, Natasha Gesker, and Eliot Liucci

2023-10-18

Contents

Summary

Neural Network Models and Principal Component Analysis (PCA) can be used together to better understand data. While neural networks have a lot of potential, they can get bogged down by their complexity and heavy computational load. Using PCA can help with these problems when it is properly executed. To fully understand the usefulness of Neural Network Models, we will explore the various layers, deep learning, and the difference between Single-Layer and Multi-Layer Neural Networks. We will use both theory and examples with data sets familiar to this class.

Introduction to Deep Learning

The Neural Network model is a deep learning technique based off the human brain and the neural networks that exist there. Deep learning is a concept similar to machine learning and artificial intelligence techniques. Neural Network modeling techniques were widely used in the 1980s, according to ISL. Currently, Neural Networks are not a common learning technique due to newer, more simple modeling techniques such as random forests. With the booming growth in the artificial intelligence, statisticians are often exposed to machine learning and data science techniques. We will be exploring Neural Networks in this chapter, along with its relationship to Principal Component Analysis. The examples and code will be written in R, with Google's *tensorflow* software playing a comprehensive roll in coding methods for Neural Network models.

Principal Component Analysis is a technique used to reduce the numbers of columns in a multivariate dataset in order to better visualize the data and create input for a regression model. Often using correlation matrices, principal component analysis converts the original variables into smaller dimensions. In the following text, we will go further into depth on neural networks and principal component analysis, and the connection between the two.

The Neural Network Model

A neural network is a type of deep learning algorithm that makes use of a web of nodes to predict or classify data. The complexity of a neural network can vary greatly based on the task at hand. As the name suggests, these models are similar to the function of neurons in a brain. The model takes in information through the *input layer*, which then activates various nodes in the *hidden layers*, and then produces a result.

The Input Layer

The input layer is where data can be inserted into the model. If we have p input variables, denoted as $X = X_1, X_2, ..., X_p$, then our network will have p input nodes. Each node in future layers will depend on the value that X_i holds.

Nonlinear Activation Functions

Before we get into the hidden layer, it is important to understand what is happening at each hidden layer node. Each hidden layer node is computed by taking a weighted linear combination of the input layer and then applying a *nonlinear activation function* so that the *activation*, which is the value the node will take based on input vector X, will be between 0 and 1.

We will discuss two of the most common activation functions. For simpler networks, the *sigmoid* function is effective. The sigmoid function is defined as

$$S(x) = \frac{1}{1 + e^{-x}}$$

As discussed previously, the purpose of the activation function is to bring the range of values for the input layer down to a value between 0 and 1.

An activation function that is more common in networks that require more "training" is the rectified linear activation unit function, or ReLU for short. The ReLU function is defined as

$$R(x) = \begin{cases} 0 & \text{if x < 0} \\ x & \text{otherwise} \end{cases}$$

The benefits of using ReLU over Sigmoid is that ReLU can be better used for *backpropagation*, which is the main technique used to train networks.

Hidden Layers

Hidden layers are the bread and butter of neural network models. Take, for example, the network pictured below with 4 input nodes and 2 hidden nodes.

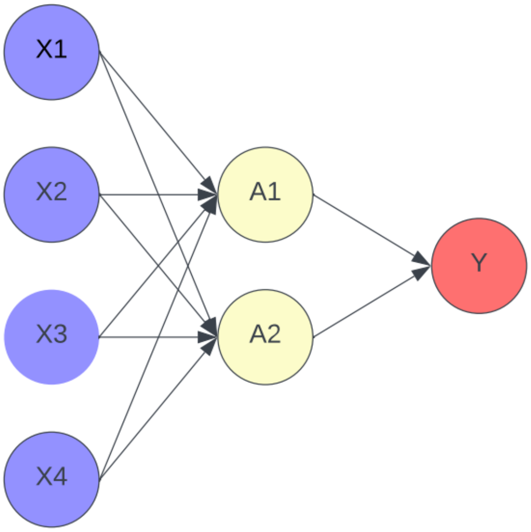

Figure 1: Example of a Simple Neural Network with 4 Input Nodes and 2 Hidden Nodes

Where $g(.)$ is the nonlinear activation function of choice and $w_{j,1}$ is the weight associated with activation 1 and the input node j. The value of $w_{0,1}$ is called the "bias", and can be added to offset the activation so the minimum value matches what it is expected to be. For each activation of the hidden layer, we are taking a weighted sum of all nodes in the input layer. The activation function restricts the range of the values the activation can hold. For Sigmoid, it would be between 0 and 1 and ReLU would just be greater than or equal to 0.

Output Layer

The output layer is what we are predicting. For a quantitative response, we would have a single node that would hold the value we predict based on the input vector X. For a categorical response with q levels, we would have q output nodes. The output can be thought of as a linear regression model fit using the hidden layer nodes as inputs. This is formally written as

$$f(x) = \beta_0 + \sum_{k=1}^{k} A_k \cdot \beta_k$$

Applications of Principal Component Analysis with Neural Networks

Neural networks can quickly grow in complexity. Given a dataset with 20 input variables, we could end up requiring many nodes in the Hidden Layer. The training process can be timely and computationally expensive.

Principal Component Analysis would allow for those 20 input variables to be trimmed down to 2 or 3 principal components. This would also theoretically cut down on the number of nodes in the Hidden Layer, thus reducing the computational cost of fitting the model while maintaining the accuracy of the model.

Single-Layer Neural Networks

A Single-Layer Neural Network is a type of *feed forward neural network*. This means information flows in one direction, from input to output, without any feedback loops. In other words, in a feed forward neural network, there are no loops or cyclical connections between nodes. The neural network builds a nonlinear function $f(X)$ to predict the response Y. The nonlinear function $f(X)$ is the output layer.

To go from the input layer to the output layer, the neural network creates and builds a hidden layer that computes $k = 1, ..., K$ activations $A_k = h_k(X)$. The activations are nonlinear transformations of linear combinations of the p input variables $X_1, X_2, ..., X_p$. Mathematically, we can write the *activation* of the 1st node of the hidden layer A_1 as

$$A_1 = h_k(X) = g(w_{0,1} + \sum_{j=1}^{p} w_{j,1} \cdot X_j)$$

These $h_k(X)$ are not fixed in advance, but rather learned during the training of the network. The K activations from the hidden layer feed into the output layer to create the nonlinear function $f(X)$.

Note on K: K is an arbitrary number chosen by the statistician. Generally, it is advised to chose a value of K that is less than $2p$.

A neural network with a single layer can be visualized as such:

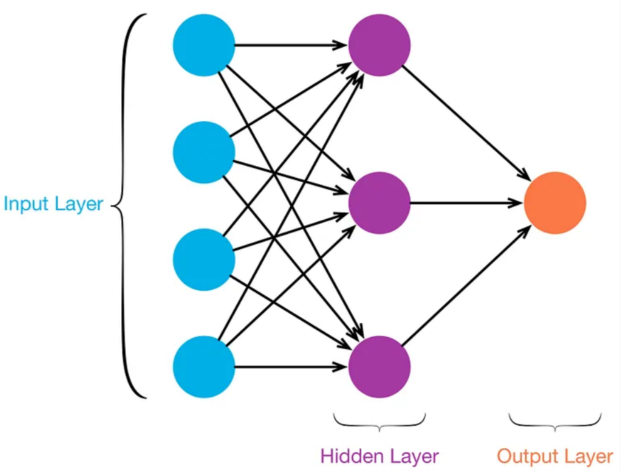

Figure 2: Single Layer Neural Network. This example has an input layer (blue) that consists of 4 variables. The hidden layer computes 3 activations (purple) that are nonlinear transformations of linear combinations of the input variables. The output layer (orange) is a linear model that uses these activations as inputs, resulting in a function f(X). In this example, there is only one output, but it is possible to have many.

However, as a model becomes more complex it potentially requires numerous nodes in the hidden layer, and it may be computationally expensive. In situations like this, it would be useful to reduce the number of nodes in the input layer.

Principal component analysis can also be used in conjunction with Single Layer Neural Networks. In this scenario, the investigator begins by conducting PCA on the data. Then, the input layer in the neural network will consist of q principal components, instead of the p variables in the dataset. Here, $q < p$, hence reducing the number of nodes in the input layer.

Thus, if we return the above generalized visual example in **Figure 2**, the four blue neurons of the input layer would represent the first four principal components. It is worth noting that the number of principal components that make up the input layer is not standardized, but rather it should consist of as many nodes (principal components) as are deemed necessary via principal component analysis.

The contents of the input layer is the main difference between typical Single-Layer Neural Networks and Single-Layer Neural Networks found using PCA. However, the steps of conducting a Single-Layer Neural Network remain the same.

Example

For an example of a single-layer network, we used the `airquality` dataset from the `datasets` package in R. We trained a model to predict Ozone levels using the other predictors available. We then trained another model to predict Ozone levels using the first 3 principal components and compared the results.

Ozone	Solar.R	Wind	Temp	Month	Day
0.5675676	0.0192192	0.1981982	0.012012	0.000000	0.5675676
0.3513514	0.0210210	0.2132132	0.012012	0.003003	0.3513514
0.4444444	0.0348348	0.2192192	0.012012	0.006006	0.4444444
0.9369369	0.0315315	0.1831832	0.012012	0.009009	0.9369369
0.8948949	0.0228228	0.1921922	0.012012	0.018018	0.8948949

The network using the observed variables in the dataset can be seen below. The network was trained quickly and predicted Ozone levels with 5% error 90% of the time and predicted with 1% error 65% of the time. Impressive!

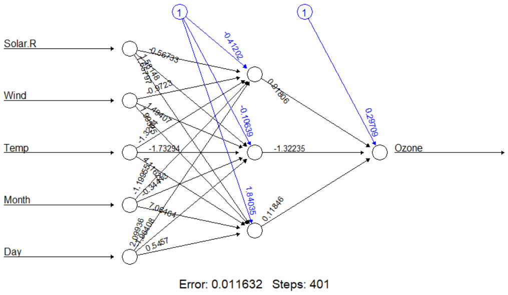

Figure 3: Neural network using observed variables to predict Ozone levels

Comparing those results to the network trained using the first 3 principal components, we got predictions within 5% of the observed values 73% of the time and predictions within 1% of the observed values 62% of the time. The performance of the network using principal components is similar with a 1% error rate, but falls behind within a 5% error rate.

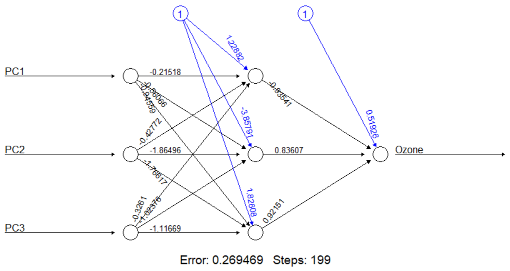

Figure 4: Neural network using first 3 principal components to predict Ozone levels

Multi-Layer Neural Networks

A Multi-Layer Neural Network relays on the same structure as a Single-Layer Neural Network, but typically has more than one hidden layer and multiple units per layer. While a single hidden layer with a large number of units could approximate most functions, using multiple hidden layers with a more modest number of units is simpler and more practical.

Now, there will be multiple output variables instead of one. The variables represent a single qualitative variable, and are dependent on each other.

A Multi-Layer Neural Network can be visualized as such:

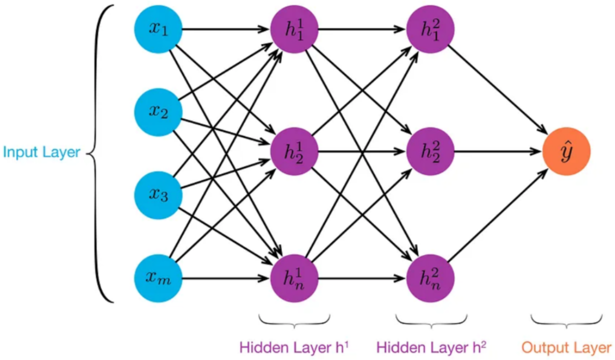

Figure 5: Multi-Layer Neural Network. This example has an input layer (blue) that consists of 4 variables. The first hidden layer computes 3 activations (purple) and the second hidden layer also computes 3 activations (purple). The output layer (orange) is a linear model that uses these activations as inputs, resulting in a function f(X). This example only has one output, but it is possible to have many.

When building the hidden layers, each new layer is build in context of the previous activation. For example, the first hidden layer will look the same as the activations for a Single-Layer Neural Network.

$$A_k^{(1)} = h_k^{(1)}(X) = g(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} X_j$$

The next hidden layer treats the activation $A_k^{(1)}$ of the first hidden layer as the inputs for the new activations. Here the activations $A_k^{(1)}$ from the first layer are functions of X.

$$A_l^{(2)} = h_l^{(2)}(X) = g(w_{l0}^{(2)} + \sum_{k=1}^{K_1} w_{lk}^{(2)} A_k^{(1)}$$

Each layer continues to use the activations of the prior layer to build a a more complex output layer.

Another difference between these activation formulas and the one used for Single-Layer is the new super scrips. These indicate which layer the activations and weights belong.

The notation $\mathbf{W_1}$ represents the entire matrix of weights that feed from the input layer in to the first hidden layer, L_1. It has (p + 1) * (# of L_1) units. The '+ 1' is to account for the intercept. Each element from the first layer feeds into the second hidden layer, L_2, through $\mathbf{W_1}$. This matrix has (# of L_1 + 1) * (# of L_2) units. This pattern continuous on for teach hidden layer. All of these unit weights are stored in matrix **B**.

Multi-Layer also puts out a non-linear function to predict the response Y. In this case, for m = 0, 1, . . . , 9:

$$f_m(X) = Z_m = \beta_{m0} + \sum_{l=1}^{K_2} \beta_{ml} h_l^{(2)} = \beta_{m0} + \sum_{l=1}^{K_2} \beta_{ml} A_l^{(2)}$$

A special softmaz activation function can also be used to get the formula:

$$f_m(X) = Pr(Y = m|X) = e^{Zm} / \sum_{l=0}^{9} e^{X_l}$$

This ensures that all the outputs behave like probabilities.

Example

For an example of a multi-layer network, we will use the `Iris` dataset from the `datasets` package in R. We first trained a model to predict Species using Sepal Length, Sepal Width, Petal Length, and Petal Width, and then compared its performance to a model using the first 2 principal components to predict Species.

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0.6410256	0.4358974	0.1666667	0.0128205	setosa
0.6153846	0.3717949	0.1666667	0.0128205	setosa
0.5897436	0.3974359	0.1538462	0.0128205	setosa
0.5769231	0.3846154	0.1794872	0.0128205	setosa
0.6282051	0.4487179	0.1666667	0.0128205	setosa

The network that uses the 4 observed features of a flower performed well, with an accuracy of 98%. Below we can see the structure of the network.

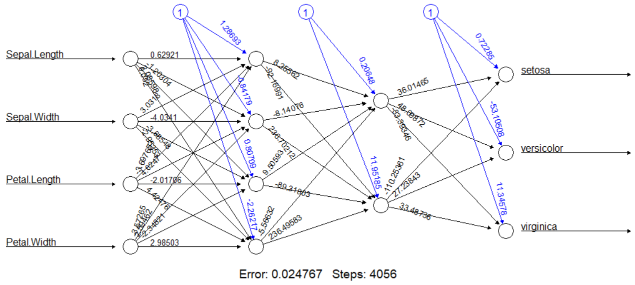

Figure 6: Neural network using observed variables to predict species

The network using the first 3 principal components also obtaining an accuracy of 98% with species classification, while only taking only $\frac{2}{3}$s of the time to train. We can see that network's structure below.

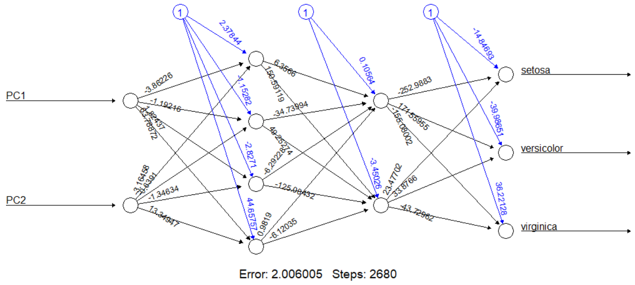

Figure 7: Neural network using first 3 principal components to predict species

The structure of these networks is the same, so it is unsurprising that the performance was similar. A takeaway from this example is that PCA can be used to allow for simpler input layers while achieving similar results. When a data set includes thousands of observations for training, reducing the time to train by $\frac{1}{3}$ can make a huge difference in the time it takes to analyze the data and create the model.

References

3Blue1Brown. 2017. “Neural Networks.” YouTube. 2017. https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi.

Abdi, Hervé, Dominique Valentin, and Betty Edelman. 1999. *Neural Networks*. 124. Sage.

Anderson, James A. 1995. *An Introduction to Neural Networks*. MIT press.

Awan, Abid Ali. 2023. “Building Neural Network (NN) Models in r.” Data Camp. 2023. https://www.datacamp.com/tutorial/neural-network-models-r.

James, Gareth, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. 2013. *An Introduction to Statistical Learning*. Vol. 112. Springer.

Kang, Nahua. 2017. “Multi-Layer Neural Networks with Sigmoid Function — Deep Learning for Rookies (2).” Medium.com. 2017. https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f.

Pramoditha, Rukshan. 2022. “Using PCA to Reduce Number of Parameters in a Neural Network by 30x Times.”