Modeling the Exposure and the Outcome: Neural Networks

Ashley I Naimi

June 2022

Contents

1 Neural Networks

Artificial neural networks (NNs) are a class of supervised learning algorithms inspired by the structure of the human brain (Brand et al., 2020). The basic idea behind NNs is to create a system of regression equations, where each equation is related to the other in terms of the relations we might find between neurons in the brain (Figure 1).

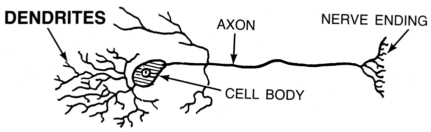

Figure 1: Depiction of a neuronal cell, with dendrites, axon, cell body, and nerve endings. (Image taken from Wikimedia Commons)

In simple terms, in an NN, each "neuron" in the network can be thought of as a regression model where the regression coefficients and covariates are inputs (which are also the outputs of other "neurons"), which are then transformed through a function (e.g. sigmoid[1], rectified linear unit) that outputs a value between 0 and 1. This function is often referred to as the "activation function."

The elements of a neural network are usually organized into three layers: the input layer, which consists of the covariates going into the model, the hidden layers, which consist of the regression functions that mimic neurons, and the output layer, which consists of the predictions from the neural network.

A regression network with no hidden layers and a sigmoid activation function corresponds exactly to a logistic regression model. When "hidden layers" are added between the input and output layers, the neural network is called a "feed-forward network" (aka multilayer perceptron). A neural network with multiple hidden layers is called a "deep" neural network, which does "deep learning" (LeCun et al., 2015).

In terms of what is actually occurring in a neural network, particularly with the hidden layers, consider three covariates in our NHEFS dataset: `qsmk`, `age`, and `dbp`, which we'd like to use to predict the probability of having greater than median weight gain (`wt_delta`). Let's suppose that we create a neural network with a single hidden layer, and 4 nodes in this layer. Each of these four nodes can be thought of as a regression model that take as inputs the three variables in the (`qsmk`, `age`, and `dbp`), with a certain set of coefficients that differ from one another across nodes. For example, the coefficients for one node may be positive for `age`, negative for `dbp`, and zero for `qsmk`, whereas the coeffecients for another node may be negative for all three.

This formulation with four nodes in a single hidden layer gives us four chances to guess the probability of `wt_delta`. In this sense, neural networks are more flexible than standard regression models, since a number of different

[1] Note that the sigmoid function is also known as the inverse logit, the expit, and is related to the softmax function. Different names for (roughly) the same thing.

regression formulations can be combined, depending on the number of nodes we include in the single layer.[2]

We can further increase the flexibility of a neural network by increasing the number of nodes in the hidden layer, or by increasing the number of hidden layers. Indeed, some of the most flexible modeling approaches are huge neural networks with many nodes and many hidden layers, i.e., deep learning algorithms. There are some important practical challenges and theoretical questions that remain to be addressed when deep learning methods are used to estimate causal effects (see Technical Note). For this reason, we focus here on single-layer neural networks estimated via the `nnet` package in R.

> **Technical Note**:
>
> There are very important practical distinctions to be made between deep and “shallow” (e.g., single-layer) learning models. In particular, the software and hardware requirements for fitting deep learning models are very different from those for fitting single-layer models. Deep learning models will often require the use of GPU processing, which can add several layers of complexity and time to the analysis. Additionally, datasets that are typically used to estimate causal effects are often not large enough to warrant the use of deep learning models. Specifically, it is difficult to justify fitting a deep learning model with thousands, or hundreds of thousands of parameters, when the dataset only has several thousand or several ten-thousand of observations.
>
> Deep learning for estimating causal effects is currently an active area of research, and there are a handful of important and difficult questions that need to be evaluated. Among these include hardware/software related questions of how to implement deep learning methods with double robust estimators, and the extent to which deep learning causal effect estimators are affected by problems due to the curse of dimensionality (bias, mean squared error, confidence interval coverage, variance estimation).

2 Neural Networks in R

There are a few packages in the R programming language that can be used to fit neural networks. Here, we will explore the use of the `nnet` package, and in particular explore the arguments of the `nnet` function that can be used to optimize the fit of the algorithm.

```
library(tidyverse)
library(nnet)
```

[2] The combination of parameters or coefficients for each node is usually determined using gradient descent with backpropagation. We will not be covering these technical details here. For the interested reader, I’d recommend chapter 11 (page 362-8) of Raschka et al. (2022)

```
nhefs <- read_csv(here("data", "nhefs.csv")) %>%
    mutate(wt_delta = as.numeric(wt82_71 >
        median(wt82_71)))

head(nhefs)
```

```
## # A tibble: 6 x 12
##    seqn  qsmk   sex   age income   sbp   dbp price71 tax71  race wt82_71
##   <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl>   <dbl>
## 1   233     0     0    42     19   175    96    2.18 1.10      1  -10.1
## 2   235     0     0    36     18   123    80    2.35 1.36      0    2.60
## 3   244     0     1    56     15   115    75    1.57 0.551     1    9.41
## 4   245     0     0    68     15   148    78    1.51 0.525     1    4.99
## 5   252     0     0    40     18   118    77    2.35 1.36      0    4.99
## 6   257     0     1    43     11   141    83    2.21 1.15      1    4.42
## # ... with 1 more variable: wt_delta <dbl>
```

In this example, we are going to use a `nnet` algorithm to regress greater than median weight change (`wt_delta`) against `qsmk` and the other variables in the dataset.

```
outcome <- nhefs$wt_delta
covariates <- nhefs[, c("qsmk", "sex", "age",
    "income", "sbp", "dbp", "price71", "tax71",
    "race")]

nnet1 <- nnet(x = covariates, y = outcome,
    size = 5)
```

```
## # weights:  56
## initial  value 405.358925
## final  value 348.500008
## converged
```

We can explore the structure of this model graphically to get a sense of what is going on in this model. Figure 2 is showing us the five nodes in the single hidden layer, as well as the direction and magnitudes of the associations.

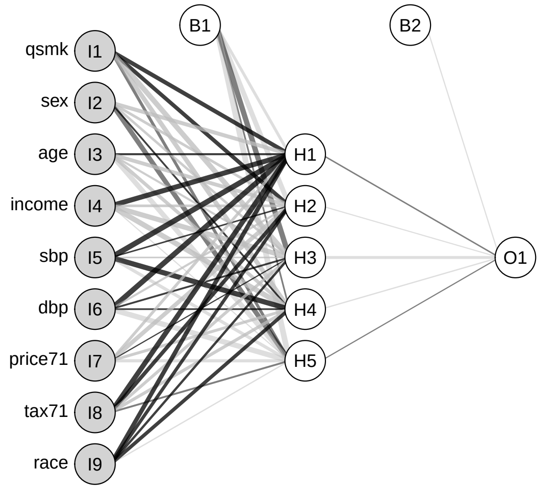

Figure 2: Structure of a simple neural net with 5 nodes fit to the NHEFS data. Black lines represent positive associations, gray lines represent negative associations. The size of each line corresponds roughly to the association magnitude. Code for this figure was obtained from: https://stackoverflow.com/questions/12447852/

3 Tuning the Neural Net

In terms of implementing a single-layer neural network, there are a wide variety of options or arguments that we could change to optimize the fit of the algorithm.[3] There are three key arguments that should be considered when fitting a single-layer NN to data using the `nnet` function:

3.1 size

The size argument determines the number of units created in hidden layer of the neural network. This argument is required, and there is no default. Thus, leaving it blank should return an error to the console. Generally, the larger the size of the neural network, the more flexibly it fits the data. However, if we choose too high a number, this may result in overfitting.[4] In the datasets that I tend to work with (N between ~1000 and ~15,000), I usually employ size values

[3] Generally, it is always important to become familiar with the arguments of whatever function you may be using to clean, code, or analyze data. In R, the help function is one of the best ways to do this. For the `nnet` package, just type "?nnet" in the console after loading the `nnet` library.

[4] In the section on Super Learning, we will learn an important strategy we can use to determine the optimal value for a tuning parameter.

ranging between 1 and 20.

3.2 linout

By default, the `nnet` function employs a logistic link function to connect the input layer to the hidden layer, and the hidden layer to the output layer. Occasionally, the data fit better with a linear link function. This can be accomplished by setting `linout = T`.

3.3 entropy

By default, the `nnet` function employs a least squares loss function to solve for the parameters (weights) in the model. This can be switched to an entropy loss function by setting `entropy = T`.

3.4 skip

By default, all of the variables in the input layer must be passed through the hidden layer before output predictions can be generated. Sometimes, particularly when overfitting is a problem, it may be advisable to enable some variables to skip the hidden layer and link them directly to the output layer.

3.5 decay

The `decay` argument is a regularization parameter that smooths the neural net function by "pulling" some of the parameters closer to their null values. By default the `decay` value is set to zero. It is often useful to explore values ranging between 0.001 and 0.15.

References

Jennie E. Brand, Bernard Koch, and Jiahui Xu. Machine learning. In Paul Atkinson, Sara Delamont, Alexandru Cernat, Joseph W. Sakshaug, and Richard A. Williams, editors, *SAGE Research Methods*. Sage Publications, 2020.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 2015.

S Raschka, YH Liu, and V Mirjalili. *Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python*. Packt, Birmingham-Mumbai, 2022.