STATISTICS 452/652: Statistical Learning and Prediction

October 9, 2020

Lecture 12b: Neural Nets

(Reading: not in book)

1 Goals of lecture

- We have several SL prediction methods that operate on linear combinations of explanatory variables,
$$Z = \phi_1 X_1 + \ldots + \phi_p X_p$$
  - Principal Component Regression (PCR)
  - Partial Least Squares' (PLS)
  - Projection Pursuit (PPR)
- We now learn one more, (artificial) Neural Nets ([A]NN)
  - NNs are considered by many to be one of the premiere tools in statistical learning
  - SHOULD DEFINITELY be a part of the SL toolkit.
    - Why is it not in the book???

2 (Artificial) Neural Nets: A "Universal Predictor"

(Both Hastie, Tibshirani, and Friedman (HTF), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Ed* (Chapter 11) and Izenman (IZ), *Modern Multivariate Statistical Techniques* (Chapter 10) cover the essential material for this lecture. I found IZ had a longer buildup, with history and a description of the biology that the prediction model is trying to mimic. If you like computing, HTF has a more detailed description of how the model parameters are estimated.)

Neural net (NN) model development started in the 1960s. They gained popularity in the 1980s and 1990s as statisticians started to analyze them and understand their statistical properties. Today they are considered one of the premiere tools for "automated" prediction. It turns out that it's not so "automatic," though.

2.1 Basic NN architecture

1. A neural net is constructed as follows
   (a) A set of inputs (X) "feeds forward" into a HIDDEN LAYER of HIDDEN NODES
      i. A "node" in a NN is a function of a linear combination of inputs
         A. similar to projection pursuit, and PLS
         B. Key difference: *several of them may be constructed simultaneously, rather than in a sequence*.
      ii. Nodes are called "hidden" because they are neither the measurable inputs nor the measurable output. They are an interim, temporary calculation, like the Z components.
   (b) (Optionally): Iterate the feed-forward into one or more additional hidden layers
   (c) Combine the results of the final hidden layer into a prediction
2. Details are given later; see Figure 1.
3. Theory says that any continuous non-random function can be approximated arbitrarily closely by a certain version of the NN architecture
   (a) Closer approximation requires more hidden nodes.
4. Thus, in theory, NNs have the capacity to be *excellent* predictors...
   (a) ...or excellent over-fitters.

2.2 Details on model

Structure: $X \to Z \to T \to Y$

For now, assume that there is only one response variable Y being predicted.

1. Create hidden nodes $Z_1, \ldots Z_M$ from inputs $X_1, \ldots, X_p$ such that

$$Z_m = f_o(\alpha_{m0} + \alpha_{m1}X_1 + \ldots + \alpha_{mp}X_p)$$

   (a) $\alpha_{0m}, \alpha_{1m}, \ldots, \alpha_{pm}$, $m = 1, \ldots, M$ are "weights"
      i. $p + 1$ parameters per hidden node
      ii. Unlike other linear combination components, add the constant α_{m0}
   (b) f_0 is any function
      i. Called the ACTIVATION FUNCTION
      ii. *The same function for all* m
      iii. Often taken to be a "sigmoidal", $f_0(a) = 1/(1 + e^{-a})$
         A. Scales each hidden node's output to be potentially between 0 and 1
      iv. PPR used a different spline for each m.

Figure 1: Basic architecture of a single-layer neural net with K different response variables that are being predicted simultaneously. In most applications $K = 1$. Inputs start at the bottom and get combined into hidden nodes, which are subsequently combined into a prediction. From HTF

Elements of Statistical Learning (2nd Ed.) ©Hastie, Tibshirani & Friedman 2009 Chap 11

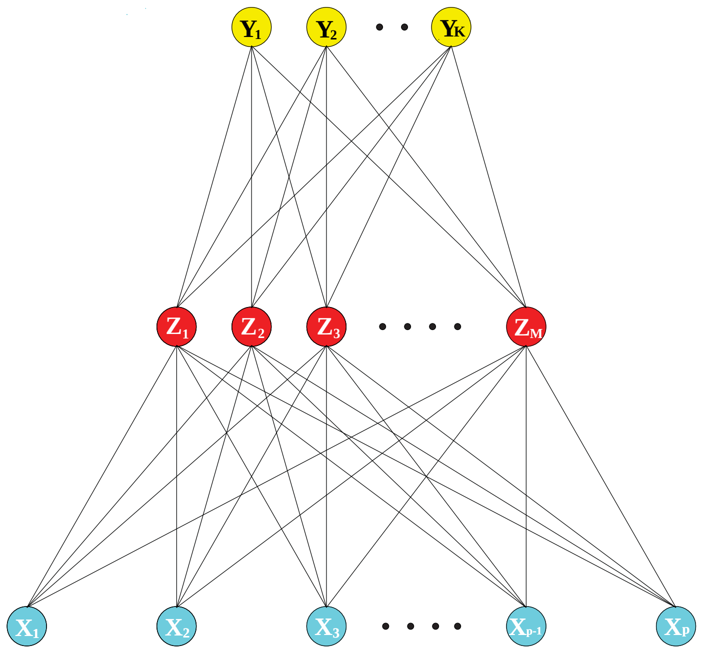

FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

   (c) M is a tuning parameter! (as always...)

2. Create $T = \beta_0 + \beta_1 Z_1 + \ldots + \beta_m Z_m$
   (a) $\beta_0, \beta_1, \ldots \beta_m$ are more weights.
3. Convert T into $\hat{Y}$ by some function according to the form of Y
   (a) $\hat{Y} = g(T)$
   (b) If doing regression, g is usually the identity function, $\hat{Y} = T$
   (c) For classification, need to predict a 0 or 1 for each observation, so need g to convert T into binary
4. Summarizing,

$$f(X) = g[\beta_0 + \sum_{m=1}^{M} \beta_m f_0(\alpha_{m0} + \sum_{j=1}^{p} \alpha_{mj} X_j)]$$

   (a) PPR and NN both make made a nonlinear function out of linear combinations of variables.
      i. Both take linear combinations of these functions
      ii. NN adds possible second function, g, onto linear combination
      iii. PPR fits them sequentially, NN fits components and final regression weights *simultaneously*.
         A. That makes fitting complicated but makes result potentially WAY better!
   (b) $(p + 2)M + 1$ parameters
5. LOTS of parameters if M and/or p are at all large
   (a) Bias-Variance tradeoff!!!

2.3 Issues with estimating the parameters

1. Parameter estimates aim to minimize residual sum of squares RSS, like other methods
2. It is a very difficult task to find a large number of linear combinations *simultaneously* that jointly optimize the fit of the NN to the data.
3. Parameters are "poorly identified": Different combinations of the parameters can lead to the same predictions
   (a) Permute the hidden nodes to get the same result.
   (b) Often multiple local minima, may or may not be as good as global minima
      i. Like trying to get to the lowest point in a mammatus cloud

4. Uses complicated iterative estimation method called BACK-PROPOGATION to "crawl" toward a solution
   (a) Start with a random guess at parameter values and take baby steps in a direction that improves fit a little
   (b) May take many iterations to reach solution (CONVERGE)
   (c) Can terminate prematurely at a sub-optimal solution
5. WARNING!!! Different starting values can lead to different solutions, so want to sample among them over several restarts of the iterative estimation process.

3 Notes

1. NNs are highly overparameterized predictors.
   (a) M is often set to be large, depending on n and p
      i. Larger M for larger n, p.
   (b) Often $5 \leq M \leq 100$ with large-ish data sets.
      i. n in thousands or more
   (c) The potentially large number of parameters in the NN can lead to considerable chance of overfitting.
      i. Need to use shrinkage (called WEIGHT DECAY) to prevent this.
      ii. Venables and Ripley (2002) recommend shrinkage between .0001 and .01.
      iii. I have found that more shrinkage is needed with larger M.
2. Can create NNs with *multiple hidden layers.*
   (a) Allows modeling more complex phenomena, especially "hierarchical" (multiple levels of inputs, like split plots or clustered data)
   (b) Adds MANY more parameters, and also more tuning parameters
   (c) DEEP LEARNING is essentially just a family of giant NNs with many hidden layers and possibly some additional architectural features (e.g., different hidden nodes use only subsets of previous layers, some nodes feed backward into past nodes)
3. Can use a single net to predict multiple responses simultaneously
   (a) One set of hidden nodes
   (b) At final step, multiple linear combinations $(T_1(\cdot), \ldots, T_K(\cdot))$ taken, leading to ultimate predictions of $Y_1, \ldots, Y_K$.
4. Looking at weights to understand effects of inputs is only useful in simple nets.
5. LOTS OF TUNING IS NEEDED!
   (a) Number of layers, Number of nodes, Decay parameter
   (b) Basic R function `nnet()` does only 1-layer NNs, so just two parameters to tune.

Example: Air Quality Data (`Neural Nets AirQ.R`) The function `nnet()` in the package `nnet` fits only single-layer NNs. It is fast but not stable. You need to run it several times to be more confident of achieving a proper fit. Alternatively, the function `neuralnet()` in the package `neuralnet` can also fit NNs, and can handle multiple layers. It uses a much more stable algorithm than `nnet()` and is consequently so slow that it is almost useless on many data sets.

I will use `nnet()` To use `nnet()` requires some steps

1. Start by scaling the explanatory variables to lie between 0–1:

   $$x' = \frac{x - \min}{\max - \min}$$

   where min and max are the minimum and maximum observed values for a given X_j.

   (a) Repeat this separately for each X_j with each variable's own max and min values in the scaling

   (b) If there is a training set split from other data, use the min and max *only from the training data*

      i. *Then rescale the validation/test set using the max and min from the training data*

2. Choose the number of hidden nodes and amount of shrinkage/weight decay (`size=` and `decay=`)

   (a) These are complete guesses, but can have HUGE effect on prediction function

   (b) The really need to be tuned, like λ in LASSO, Ridge, and smoothing splines.

      i. Use any resampling/splitting procedure on training data.
      ii. Try a few different values of `size` and `decay` in a grid (all combinations of sizes and decays)
      iii. Use minimum mean or plots of relative MSPE to choose the best
      iv. **If best results are on boundary of grid, expand in that direction!**
      v. Can "fine-tune" by trying a finer grid around the best values
      vi. THIS TAKES TIME!!!
      vii. Some code in program

         A. The `caret` package has a complex function `train()` that can be used for exploring root-MSPE and other measures across different values of tuning parameters.
         B. I also offer a manually programmed version.

3. For a given `size` and `decay`, it is best to run `nnet`() multiple times

   (a) As mentioned above, different starting points lead to different conclusions, and some are better than others.

(b) Use training error to find best result, just like training error is optimized for finding LSEs for a given model size

See the program for a very detailed demonstration.

4 What to take away from this

- NNs are a family of incredibly powerful prediction functions
  - Flexible and adaptable to literally any shape
  - Related to PCR, PLS, PPR
- The basic implementation in R's `nnet()` function required lots of tuning
  - Can make finding a good NN challenging
- When properly tuned, NN can be one of the best predictors and is often at least as good as other top models
  - Badly tuned versions are worthless
- DEFINITELY worth having in your toolkit!

5 Exercises

Application

Refer to the Air Quality data described previously, and the analyses we have done with `Ozone` as the response variable, and the five explanatory variables (including the two engineered features).

Use Neural Nets (NN) to model the relationship between `Ozone` and all five explanatories as specified below.:

1. Create a matrix of the five explanatory variables. Rescale each of them to lie between 0 and 1 and same them as another matrix. **Print a `summary()` of each object to confirm that you have scaled properly.**

2. Fit 4 NNs nets using only `Temp` and `Wind`, using each combination of 2 and 6 hidden nodes with 0.001 and 1 shrinkage; i,e,; (2,0.001), (2,1), (6,0.001), (6,1)

   (a) Refit each one manually 20 times or more and compute the sMSE each time.

      i. **Report the sMSE for the optimal fit for each model.**

      ii. Comment on the stability of fits for different models. In other words, **which models were most/least consistent with the sMSE values produced by different fits?**

   (b) Make a 3-D plot of each model's fit.

      i. **Report a screenshot of each fit, rotated to roughly the same angle each time to show a good comparison of the fits.** (I find it best to look down through the corner with low temp and high wind, so that the high ozone values are in the back.)

      ii. **Comment separately on how increasing number of nodes or increasing shrinkage appears to affect the fits.**

3. Now try tuning the NN *on these two variables* using 5-fold CV. Use a grid of (1, 3, 5, 7, 9) nodes and (.001, .1, .5, 1, and 2) decay. Refit each combination of parameters 10 times to find the best sMSE for that combination.

   (a) Compute the overall MSPE for each combination, and add 95% confidence intervals. **Take square roots and report the results.**

   (b) **Show relative root-MSPE boxplots of the five splits.**

   (c) Use these results to **identify (i) the best combination, and (ii) other combinations that seem to perform just as well.**

   (d) Is further tuning necessary? **That is, is the best model (i) at a boundary or (ii) quite different from neighbouring models?**

4. **BONUS (This will take a long time and some extra programming, so it is optional. But it is actually closer to how I would conduct a comparison of methods for a real data set.)** Add NN *on all variables* to the 10-fold CV

comparison that has been used for LASSO, Ridge, and other methods. Use the same folds for NN that were used for the other methods. *Note that you will need to do some tuning withing each of the CV folds! That is, you will need to perform tuning* ***within*** *each 90% training set created by the 10-fold CV. Use one round of 5-fold CV for this, and refit each NN just 5 times to find the optimal fit.*

(a) **For the each of the 10 folds, report the chosen best values of the NN tuning parameters.** (I expect that they will vary from one fold to the next)

(b) **Compare the mean MSPE from the best-tuned NN to the other models tried so far.**

(c) **ADD tuned NN to the relative MSPE boxplots made previously. Comment on how well it performs compared to other methods**