Lecturer: Prof. Homrighausen Scribe: Henry Scharf

The portion of this lecture dedicated to the NN example, localization of NN, and projection pursuit are all fairly complete in the notes. Therefore, these scribe notes will focus around the challenge posed on slide 53: applying Stein's method to find the degrees of freedom in a neural network.

Because neural networks consist of many functions of many combinations of functions of many combinations of...some original covariates, there is likley a complex dependence structure between the parameters in the model, and the *effective degrees of freedom* is probably significantly less than the total number of parameters.

We need to have a reasonable estimate for degrees of freedom, as evidenced by the fact that PAC bounds depend on this quantity (worryingly, or perhaps just for lack of a good alternative, `neuralnet` just uses the number of parameters anyway).

This paper by Zou, Hastie, and Tibshirani is a good introduction (beginning at the very end of p. 2) to this idea of effective degrees of freedom. In it, they give a surprisingly simple unbiased estimator for the degrees of freedom in a LASSO model.

I have taken the liberty of barking up the wrong tree. For those interested in exploring this challenge, what follows is what I have decided is a dead-end approach. Hope it is revealing.

A bad idea

One potential avenue would be to pursue Stein's method, namely:

$$d.f. = \mathbb{E}\left[\nabla \cdot \hat{Y}\right]$$

where $\nabla \cdot \hat{Y} = \sum_{i=1}^{n} \frac{\partial}{\partial Y_i} \hat{Y}_i$. In the case of NN, our $\hat{Y}$ are quite complicated functions of the observed data Y (which is part of the point, their flexibility), so to begin thinking about this it might be useful to restrict our attention to a simple NN with one layer that is doing binary classification. We can define $\hat{Y} = g(Y)$ hierarchically as in the notes (where we've used the logistic link function):

$$\begin{aligned}
g(Y) &= \underset{\{-1,1\}}{\operatorname{argmax}}(\hat{\mu}_{\{-1,1\}}) \\
\hat{\mu}_1 &= \frac{e^{W_1}}{e^{W_{-1}} + e^{W_1}} \\
\hat{\mu}_{-1} &= \frac{e^{W_{-1}}}{e^{W_{-1}} + e^{W_1}} = \mathbf{1} - \hat{\mu}_1 \\
W_1 &= \beta_{0,1} + \beta_1^\top Z \\
W_{-1} &= \beta_{0,-1} + \beta_{-1}^\top Z \\
Z_j &= \sigma(\alpha_{j,0} + \alpha_j^\top X), \quad j = 1, \ldots, J
\end{aligned}$$

Now, we can begin to pick apart $\nabla \cdot g(Y)$.

$$\frac{\partial g_i(Y)}{\partial Y_i} = \begin{cases} \frac{\partial \hat{\mu}_{i,-1}}{\partial Y_i}; & \hat{\mu}_{i,-1} > \hat{\mu}_{i,1} \\ \frac{\partial \hat{\mu}_{i,1}}{\partial Y_i}; & \hat{\mu}_{i,-1} \leq \hat{\mu}_{i,1} \end{cases}$$

We know that

$$\frac{\partial \hat{\mu}_{i,-1}}{\partial Y_i} = -\frac{\partial \hat{\mu}_{i,1}}{\partial Y_i}$$

so we can focus on just

$$\begin{aligned} \frac{\partial \hat{\mu}_{i,1}}{\partial Y_i} &= \frac{\partial}{\partial Y_i} \frac{e^{W_{i,1}}}{e^{W_{i,-1}} + e^{W_{i,1}}} \\ &= \left(\frac{\partial W_{i,1}}{\partial Y_i}\right) \hat{\mu}_{i,-1} - \left[\left(\frac{\partial W_{i,-1}}{\partial Y_i}\right) e^{W_{i,-1}} + \left(\frac{\partial W_{i,1}}{\partial Y_i}\right) e^{W_{i,1}}\right] \frac{\hat{\mu}_{i,1}^2}{e^{W_{i,1}}} \end{aligned}$$

and now we see we need to focus on

$$\left(\frac{\partial W_{i,\pm 1}}{\partial Y_i}\right) = f\left(\frac{\partial \alpha_j}{\partial Y_i}, \frac{\partial \beta_{\pm 1}}{\partial Y_i}\right)$$

where this function of the derivatives in complicated due to the non-linearity of the sigmoid function σ.

To continue down this route, it seems at least two things are needed. First, some approximation or linearization of f and probably also $\frac{\partial \hat{\mu}_{i,1}}{\partial Y_i}$ will be required to allow us to take expectation. Second, we will need some closed form approximation for the way $\hat{\mu}_{i,1}$ depends on Y, since the typical neural networks fitting algorithm doesn't provide this.

Even more problematic, there are serious issues with what we mean by a partial derivative with respect to Y_i when Y_i can only take on two possible values. Perhaps our derivative should be more akin to a measure of influence, where we wish to know what the change in $g_i(Y)$ is when Y_i changes from -1 to 1. It would require some low-level mathematical investigation to come up with a valid Stein's method analog here.

In short, things seem a little hopeless.

Another possibility?

In lieu of this analytic approach, we could take a more numerical perspective. For instance, we could re-fit the neural network n times, changing one of the Y_i each time, and observing the effect on $\hat{Y}_i$. In the case of binary classification, this will be a vector of n 1s and 0s, whose sum may be a reasonable analog for $\mathbb{E}[\nabla \cdot g(Y)]$ in Stein's method.