Lecturer: Prof. Homrighausen | Scribe: Veronica Burt

1 Previous Lectures

We've been discussing

- non-convexity of estimating a neural network, making them inefficient to solve
- Methods for training neural networks

2 Hierarchical View

- See Figure 1 for a hierarchical view of an example neural network.
- The hidden layer is represented by the Z's, which are linear combinations of the X's. Linear combinations of the Z's are used to create the Y's.
- See Figure 2 for a Directed Acyclic Graph of a neural network.
- Figure 2 has 2 hidden layers. We refer to creating hidden layers as dimension expansion.
- It is a directed graph since the edges are directed; the edges consist of ordered pairs. The $\hat{\beta}_j$'s the are weights on the hidden layer.
- We need to estimate the weights of the edges and the parameters at each level, making this a very difficult optimization problem.

3 Localization

- The ability to localize is both an advantage and disadvantage of neural networks. It makes a neural network very difficult computationally, but also makes the network customizable to the data.
- The idea behind localizing is that distance gives us a metric for how similar our data points are. We can group together close objects to make our predictions more accurate.
- For example, a Fourier series is global over time, but local over frequency.
- We can constrain a neural network by setting some of the weights equal to zero, allowing data points that are "far apart" to have no influence on each other.
- See Figure 3 for a picture of a constrained network compared to an unconstrained network.

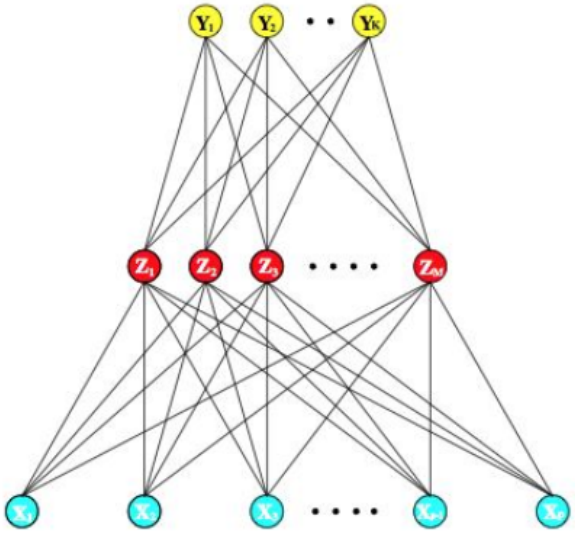

Figure 1: A single hidden layer (represented by the Z's) neural network.

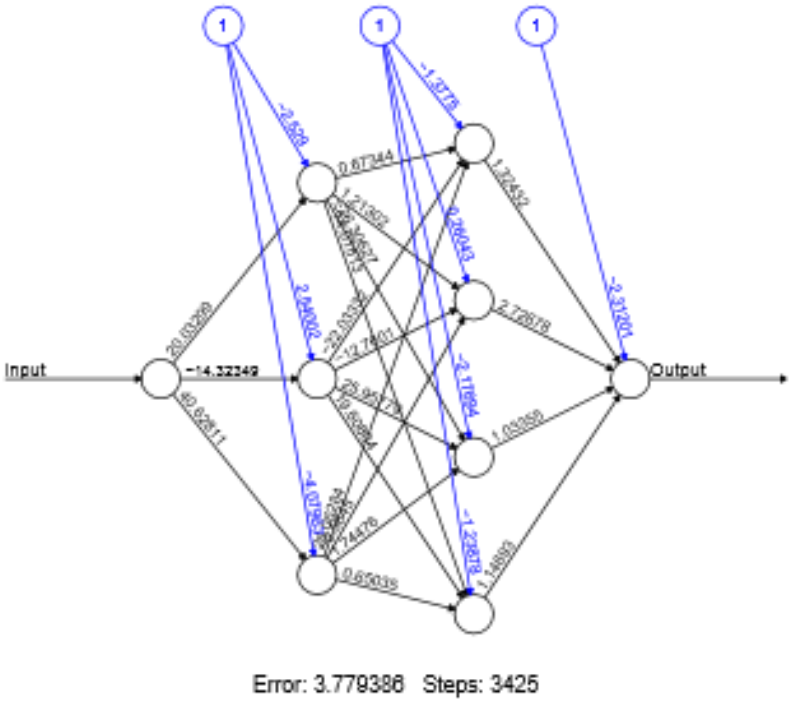

Figure 2: Directed Acyclic Graph of a neural network with 2 hidden layers.

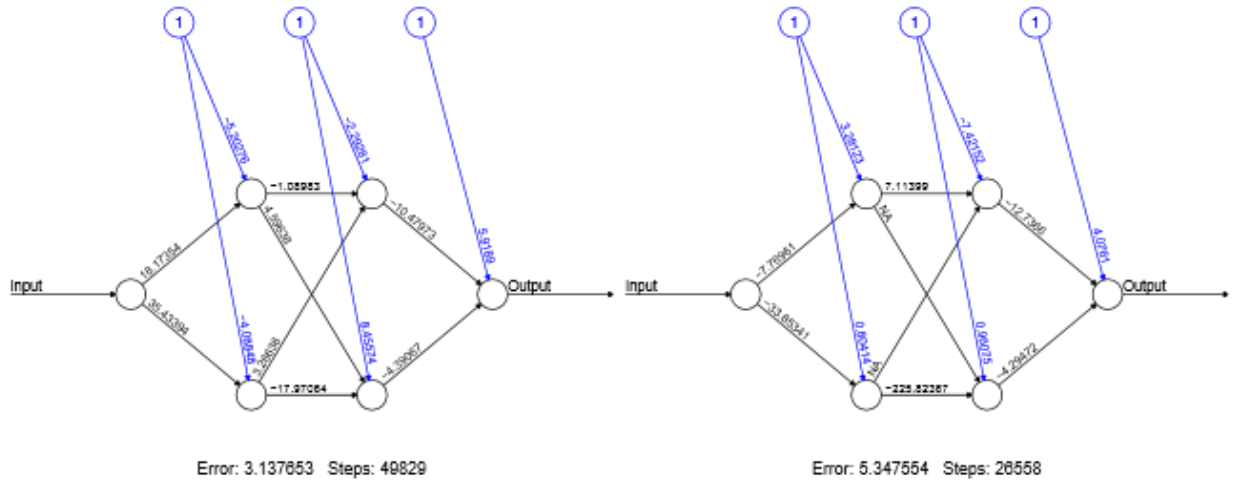

Figure 3: An unconstrained neural network (left) compared to a constrained network (right). Note that in the constrained network, the edges connecting the first element of the first hidden layer to the second element of the second hidden layer and the second element of the first hidden layer to the first element of the second hidden layer read "NA".

- We can constrain a neural network in R by utilizing the neuralnet package and the exclude parameter. Exclude is an $E \times 3$ matrix, where E is the number of exclusions:

  The first column stands for the layer

  The second column stands for the input neuron

  The third column stands for the output neuron

- The exclude parameters makes the desired entries of $\hat{\alpha} = 0$ and skips them in the updates. This may make sense given the context of our data, and also eases computation.

- For example, in our crime dataset, we may want to constrain the neural network to have neurons specifically for demographic variables, police expenditures, and economics. This is a great feature of neural networks.

4 Tuning Parameters

- There are 3 tuning parameters that we need to choose for neural networks: the number of hidden units, the number of hidden layers, and the regularization parameter (or a stopping criterion λ for an iterative solver). Note that the sum over the layers of the hidden units needs to equal the total hidden units.

- There are two main strategies for choosing the tuning parameters:

  We can set $\lambda = 0$ and use risk estimation to choose the number of hidden units and hidden layers. This requires a 2-dimensional risk estimation grid for every combination of units and layers, which is computationally expensive.

  We can set the number of hidden layers and the total number of units to be very large, and then choose λ via risk estimation. This method results in a 1-dimensional risk estimation grid, which is generally preferred.

- For risk estimation, we can use a Generalized Information Criterion method or cross validation.

- AIC = training error $+2\hat{df} * \hat{\sigma}^2$
- The 2 present in the penalty term makes the estimate unbiased. However, it can be difficult to estimate degrees of freedom for neural networks.
- The neuralnet package provides rather coarse estimates of AIC and BIC:

  It uses the number of parameters rather than the rank of the design matrix, which overestimates the degrees of freedom when the columns of the design matrix are linearly dependent.

  It also leaves out the variance in calculations. Even if the response variable is standardized, a variance of 1 is just an approximation.
- A better approach for estimating the degrees of freedom would be to use Taylor Series approximations to create a linear smoother and then take the trace. Or, we can use a cross-validation approach.

5 Representational Learning

- Neural networks can be sensitive to starting values since the problem is non-convex.
- Representation Learning is the idea that the performance of machine learning methods are often highly dependent on the choice of data representation.
- Our data is based on a coordinate system of measurements. Another scale or combination of measurements is probably a better representation of the data.
- Much of machine learning is based on the idea that another representation works better, and our $\hat{\beta}$'s are data dependent. Rather than using the original data, transform them into features, and then calculate $\hat{\beta}$'s.
- Learned representations often obtain "low level" information, such as overall shapes of the data. "Higher level" information, such as images, are not captured.

6 Principal Component Analysis

- Principal Components Analysis is an unsupervised dimension reduction technique. It helps us find a representation of the data that may not be obvious from the original coordinate system.
- It also solves various optimization problems that are equivalent to dimension reduction, such as maximizing variance, minimizing L_2 distortions, and finding the closest subspace of a given rank.
- We find linear combinations of the original (centered) data, $z_{ij} = \alpha_j^T X_i$.
- These z's are similar to the latent z's in neural networks. The key difference is the activation function.
- Since we are trying to minimize squared error distortions, we use the Singular Value Decomposition: $\mathbb{X} - \bar{\mathbb{X}} = UDV^T$, where $\bar{\mathbb{X}}$ is a matrix where each column is a vector of means for that covariate. Then, we have $Z = \mathbb{X}V = UD$.
- In this decomposition, V is the coordinate system, and UD are the principal components, or the coordinates inside the coordinate system provided by V.
- See Figure 4. Suppose we have predictors x_1 and x_2. We can preserve the structure of the data by setting x_1 equal to zero, which gives us the blue dots, and then setting x_2 equal to zero, which gives us the red dots. Note that the predictors are not correlated in this example.

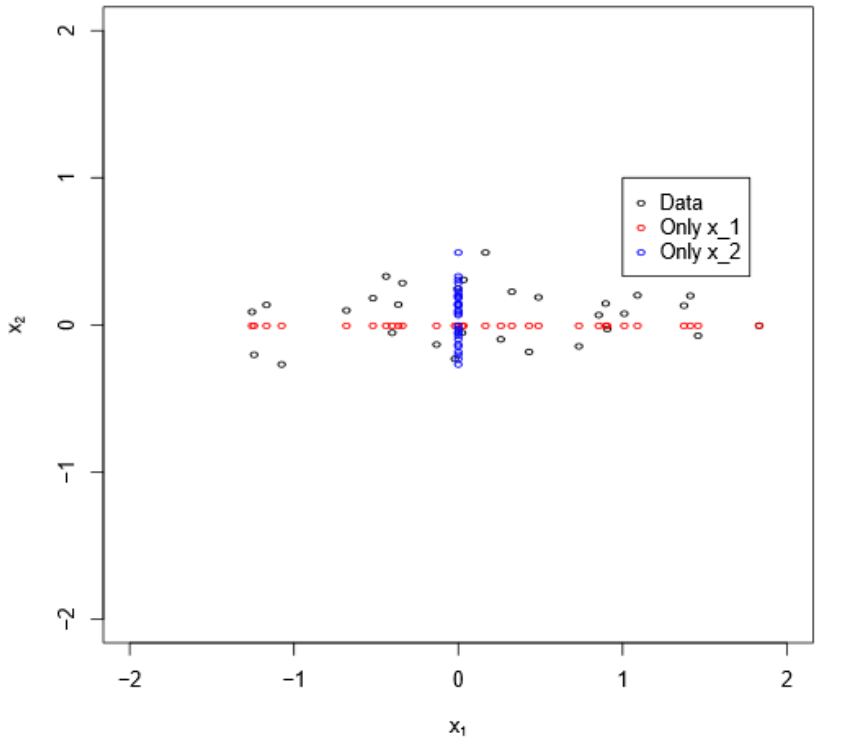

Figure 4: A plot of predictors x_1 and x_2. Note that our predictors are uncorrelated. $(0, x_2)$ is represented by the blue dots, and $(x_1, 0)$ is represented by the red dots.

- See Figure 5. What if our predictors were correlated? We can do that by simply rotating our data from Figure 4 in an upward or downward direction. Note that now, setting x_1 or x_2 to zero causes us to lose a lot of structure in our data.
- If we knew how to rotate our data, we could preserve more structure. PCA gives us the rotation to put the data back into an uncorrelated setting. See Figure 6.

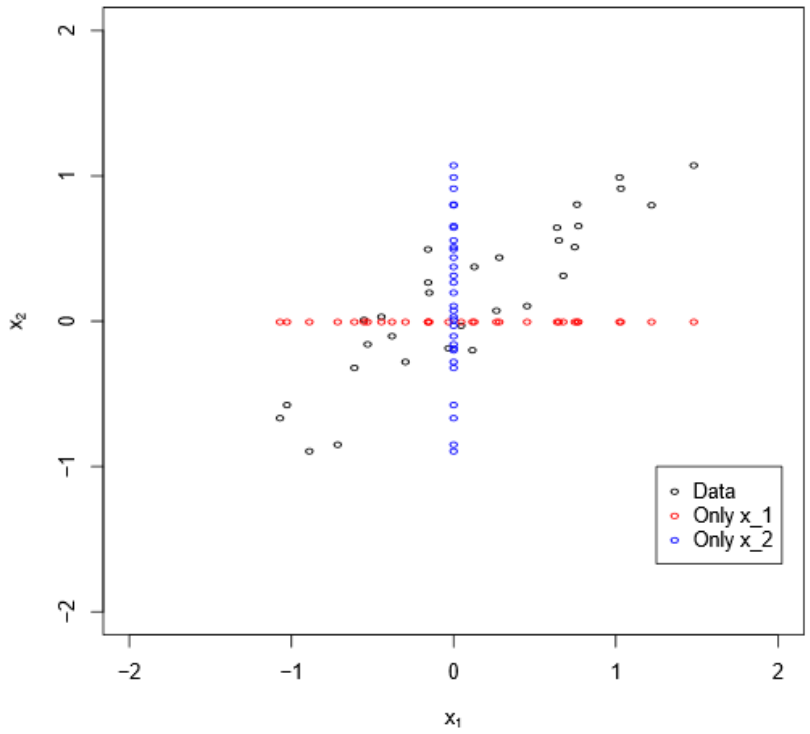

Figure 5: A plot of predictors x_1 and x_2. Note that our predictors are correlated, and this is simply a rotation of our previous data. $(0, x_2)$ is represented by the blue dots, and $(x_1, 0)$ is represented by the red dots.

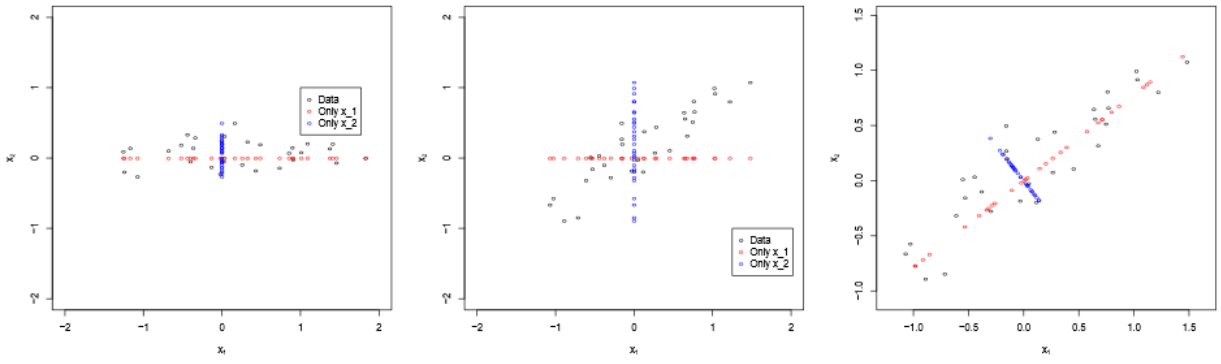

Figure 6: A plot of uncorrelated predictors (left), correlated predictors (center), and the PCA version of the correlated predictors (right).