

NEURAL NETWORKS AND DEEP LEARNING 2

-STATISTICAL MACHINE LEARNING-

Lecturer: Darren Homrighausen, PhD

NEURAL NETWORKS: GENERAL FORM

Generalizing to multi-layer neural networks:

(I'm eliminating the bias term for simplicity)

$$0 \text{ Layer} := \sigma(\alpha_{\text{lowest}}^\top X)$$

$$1 \text{ Layer} := \sigma(\alpha_{\text{lowest}+1}^\top (0 \text{ Layer}))$$

$$\vdots$$

$$\text{Top Layer} := \sigma(\alpha_{\text{Top}}^\top (\text{Top} - 1 \text{ Layer}))$$

$$L(\mu_g(X)) = \beta_{g0} + \beta_g^\top (\text{Top Layer}) \quad (g=1, \dots, G)$$

This looks like iterated matrix multiplications

- $Z_1 = \sigma(X\alpha_1)$
- \vdots
- $Z_{L-1} = \sigma(Z_{L-2}\alpha_{L-1})$
- $Z_L = \sigma(Z_{L-1}\alpha_L)$
- $L^{-1}(\beta^\top Z_L)$

$$(\alpha_l \in \mathbb{R}^{K_{l-1} \times K_l}, \text{ and } K_0 = p)$$

NEURAL NETWORKS: GENERAL FORM

Some comments on adding layers:

- It has been shown that one hidden layer is sufficient to approximate any piecewise continuous function
(“Approximation by superpositions of sigmoidal function” (1989). However, this may take a huge number of hidden units (i.e. $K \gg 1$))
- By including multiple layers, we can have fewer hidden units per layer. Also, we can encode (in)dependencies that can speed computations
- Also, another “universal approximator” is

$$f(X) = \sum_{q=1}^Q c_q \mathbf{1}(a_q \leq X \leq b_q)$$

But functions of this form wouldn't make for good neural networks

Returning to Doppler function

NEURAL NETWORKS: EXAMPLE

We can try to fit it with a single layer NN with different levels of hidden units K

A notable difference with B-splines is that 'wiggleness' doesn't necessarily increase with K due to regularization

Some specifics:

- I used the R package **neuralnet**
(This uses the **resilient backpropagation** version of the gradient descent)
- I regularized via a stopping criterion ($\|\partial \ell\|_{\infty} < 0.01$)
- I did 3 replications
(This means I did three starting values and then averaged the results)
- The layers and hidden units are specified like

(# Hidden Units on Layer 1) (# Hidden Units on Layer 2)...

NEURAL NETWORKS: EXAMPLE

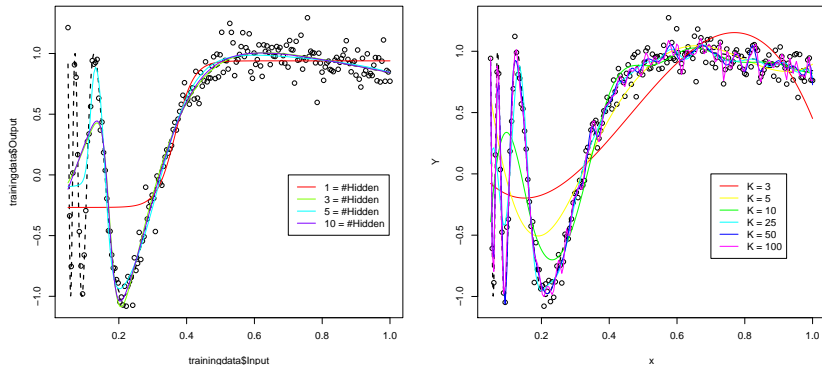


FIGURE: Single layer NN vs. B-splines

NEURAL NETWORKS: RISK

What's the estimation equality? $\text{MSE} = \mathbb{E}(\hat{f}(X) - f_*(X))^2$

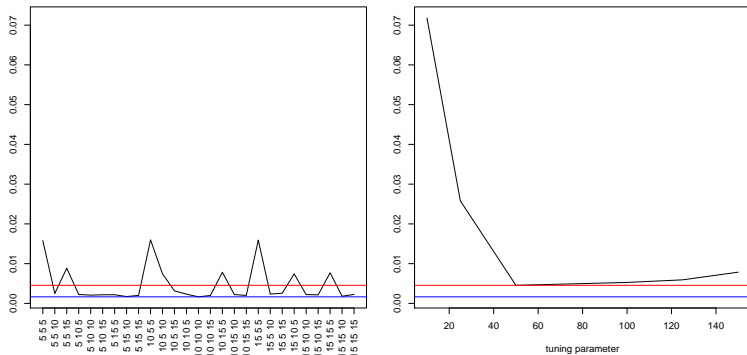


FIGURE: 3 layer NN¹ vs. B-splines

¹The numbers mean (#(layer 1) #(layer 2) #(layer 3))

NEURAL NETWORKS: EXAMPLE

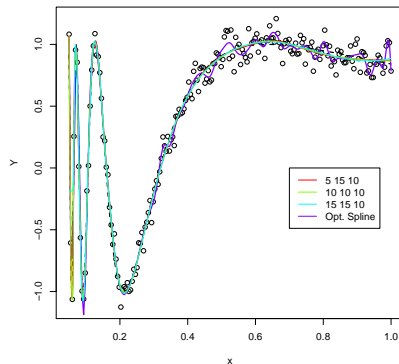


FIGURE: Optimal NNs vs. Optimal B-spline fit

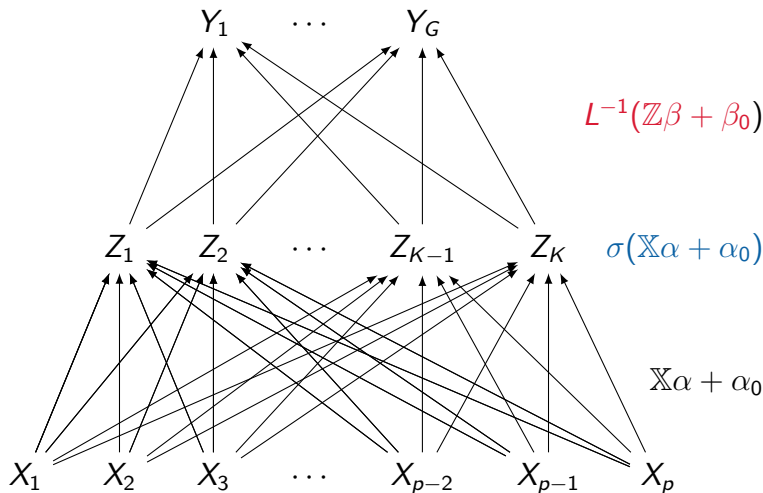
NEURAL NETWORKS: CODE FOR EXAMPLE

```
trainingdata = cbind(x,Y)
colnames(trainingdata) = c("Input","Output")
testdata      = xTest

require("neuralnet")
K              = c(10,5,15)
nRep           = 3
nn.out         = neuralnet(Output~Input,trainingdata,
                           hidden=K, threshold=0.01,
                           rep=nRep)
nn.results = matrix(0,nrow=length(testdata),ncol=nRep)
for(reps in 1:nRep){
  pred.obj = compute(nn.out, testdata,rep=reps)
  nn.results[,reps] = pred.obj$net.result
}
Yhat = apply(nn.results,1,mean)
```

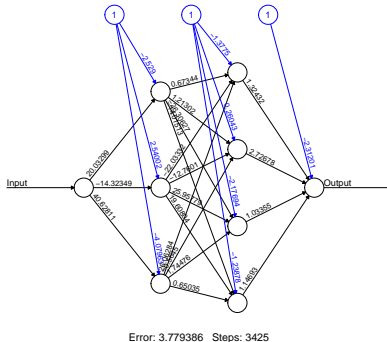
Hierarchical view

HIERARCHICAL VIEW



RECALL: Single hidden layer neural network. Note the similarity to latent factor models

HIERARCHICAL FROM EXAMPLE



This is a **directed acyclic graph** (DAG)

```
nn.out = neuralnet(Output~Input,trainingdata,  
                    hidden=c(3,4))  
plot(nn.out)
```

NEURAL NETWORKS: LOCALIZATION

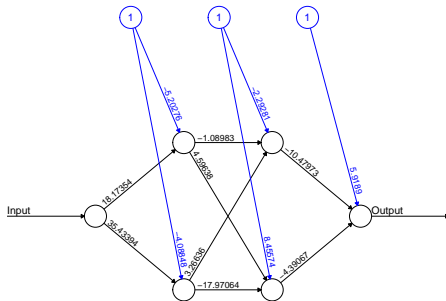
One of the main curses/benefits of neural networks is the ability to **localize**

This makes neural networks very customizable, but commits the data analyst to intensively examining the data

Suppose we are using 1 input and we want to restrict the implicit DAG

NEURAL NETWORKS: LOCALIZATION

That is, we might want to constrain some of the weights to 0



Error: 3.137653 Steps: 49829

FIGURE: Unconstrained neural network

```
nn.out = neuralnet(Output~Input,trainingdata,  
                    hidden=c(2,2))
```

Detour: Scores

SCORES

A frequently used term in statistics is **scores**

EXAMPLE A: In PCA we form $\mathbb{X} - \bar{\mathbb{X}} = UDV^\top$ and $Z = UD$ are called the (PCA) scores

Although it might not look like it, the scores are **fitted values**

EXAMPLE B: In OLS, we estimate $\hat{\beta}$ via least squares and form the fitted values

$$\hat{Y} = \mathbb{X}\hat{\beta}$$

EXAMPLE A: We can recover the PCA scores via

$$(\mathbb{X} - \bar{\mathbb{X}})V = UD \underbrace{V^\top V}_{\text{identity}} = UD$$

SCORES

We can get the same sort of information as the PCA scores from **neural networks**

$$\hat{A}_1 = \begin{bmatrix} \hat{\alpha}_{10} & \hat{\alpha}_{20} \\ \hat{\alpha}_1 & \hat{\alpha}_2 \end{bmatrix} = \begin{bmatrix} -15.93 & 4.37 \\ 58.24 & -20.75 \end{bmatrix}$$

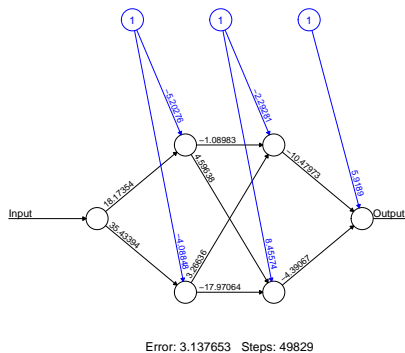
Augment $X \in \mathbb{R}^{n \times 1}$ with intercept column:

$$\mathbb{X} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ & \vdots \\ 1 & X_n \end{bmatrix}$$

1st layer's scores: $Z_1 = \sigma(\mathbb{X}\hat{A}_1) \in \mathbb{R}^{n \times 2}$

2nd layer's scores: $Z_2 = \sigma(\mathbf{Z}_1\hat{A}_2) \in \mathbb{R}^{n \times 2}$

(Technically, we need to augment \mathbf{Z}_1 with intercept column)

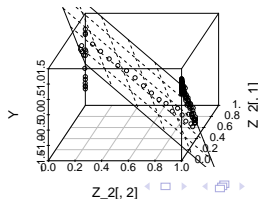
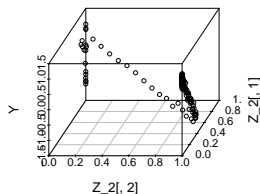
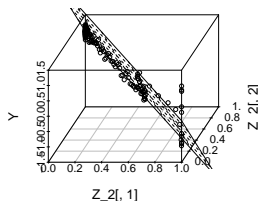
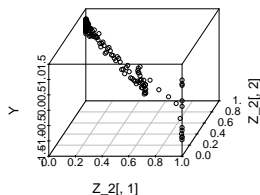


Back to localization

NEURAL NETWORKS: LOCALIZATION

Plots of scores for 2-neuron last hidden layer

(2nd layer's scores: $\mathbb{Z}_2 = \sigma(\mathbb{Z}_1 \hat{A}_2) \in \mathbb{R}^{n \times 2}$)



NEURAL NETWORKS: LOCALIZATION

We can do this in `neuralnet` via the `exclude` parameter

To use it, do the following:

```
exclude = matrix(1,nrow=2,ncol=3)
exclude[1,] = c(2,2,2)
exclude[2,] = c(2,3,1)
nn.out = neuralnet(Output~Input,trainingdata,
                    hidden=c(2,2), threshold=0.01,
                    exclude=exclude)
```

`exclude` is a $E \times 3$ matrix, with E the number of **exclusions**

- first column stands for the layer
- the second column for the input neuron
- the third column for the output neuron

NEURAL NETWORKS: LOCALIZATION

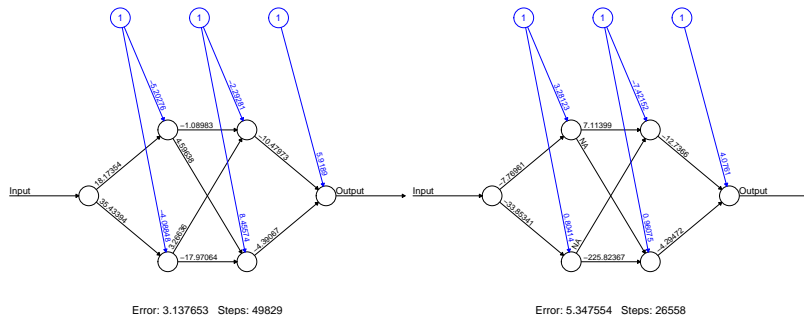


FIGURE: Not-constrained vs. constrained

NEURAL NETWORKS: LOCALIZATION

Plots of scores for 2-neuron last hidden layer

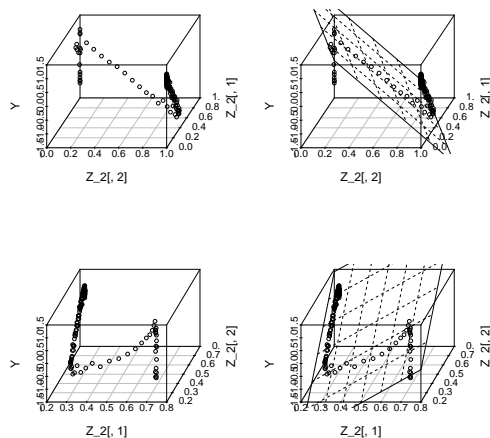


FIGURE: Top: Not-constrained. Bottom: constrained

NEURAL NETWORKS: CRIME DATA

M

percentage of males aged 14-24.

So

indicator variable for a Southern state.

Ed

mean years of schooling.

Po1

police expenditure in 1960.

LF

labour force participation rate.

M.F

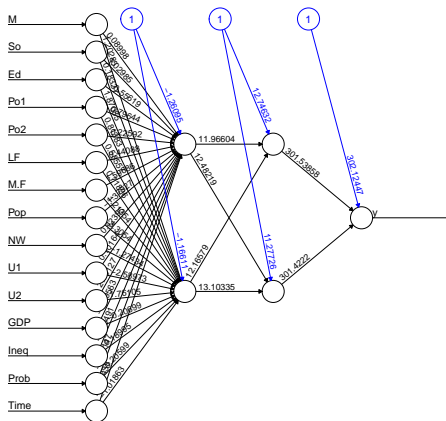
number of males per 1000 females.

...

y

rate of crimes in a particular category per capita

NEURAL NETWORKS: CRIME DATA



NEURAL NETWORKS: CRIME DATA

We may want to constrain the neural network to have neurons specifically about

- Demographic variables
- Police expenditure
- Economics

This type of prior information can be encoded via **exclude**

(This is really the only situation in which neural networks work well)

Tuning parameters

NEURAL NETWORKS: TUNING PARAMETERS

The most common recommendation I've seen is to take the 3 tuning parameters: The number of hidden units, the number of layers, and the regularization parameter λ

(or a stopping criterion λ for the iterative solver)

Either choose $\lambda = 0$ and use risk estimation to choose the number of hidden units

(This could be quite computationally intensive as we would need a reasonable 2-d grid over units \times layers)

Or, fix a large number of layers and hidden units and choose λ via risk estimation

(This is the preferred method)

NEURAL NETWORKS: TUNING PARAMETERS

We can use a GIC method:

$$\text{AIC} = \text{training error} + 2\hat{d}f\hat{\sigma}^2$$

(This is reported by `neuralnet`, by setting `likelihood = T`)

Or via cross-validation

NEURAL NETWORKS: TUNING PARAMETERS

Unfortunately, **neuralnet** provides a somewhat bogus measure of AIC/BIC

Here is the relevant part of the code

```
if (likelihood) {  
  synapse.count = length(weights) - length(exclude)  
  aic = 2 * error + (2 * synapse.count)  
  bic = 2 * error + log(nrow(response))*synapse.count  
}
```

They use the number of parameters for the degrees of freedom!

It is still an open question as to a good degrees of freedom estimator for neural networks