

NEURAL NETWORKS AND DEEP LEARNING 2

-STATISTICAL LEARNING AND DATA MINING-

Lecturer: Darren Homrighausen, PhD

NEURAL NETWORKS: GENERAL FORM

Generalizing to multi-layer neural networks, we can specify any number of hidden units:

(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

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

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

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
(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

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

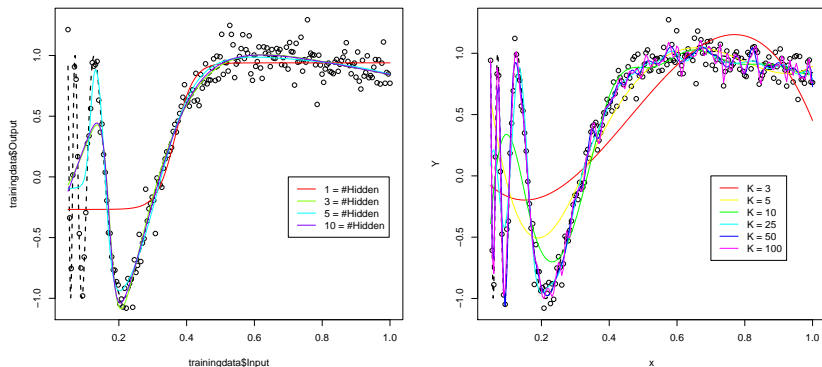


FIGURE: Single layer NN vs. B-splines

NEURAL NETWORKS: RISK

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

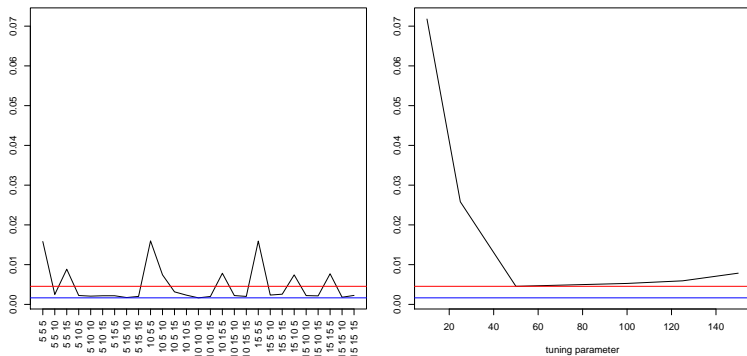


FIGURE: 3 layer NN¹ vs. B-splines

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

NEURAL NETWORKS: EXAMPLE

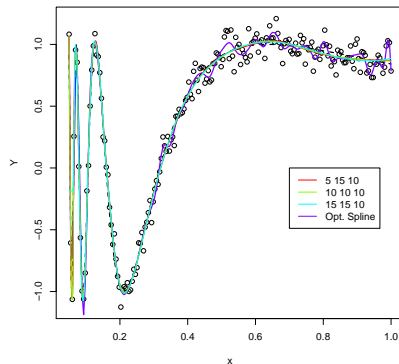


FIGURE: Optimal NNs vs. Optimal B-spline fit

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

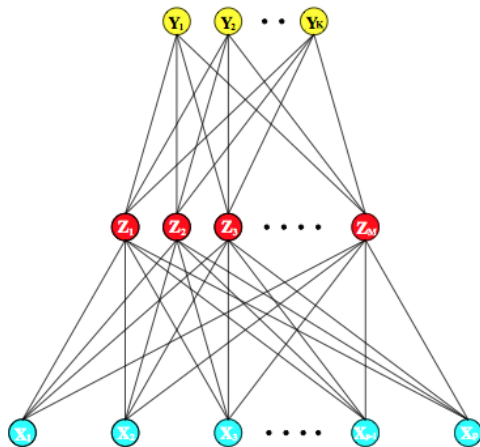
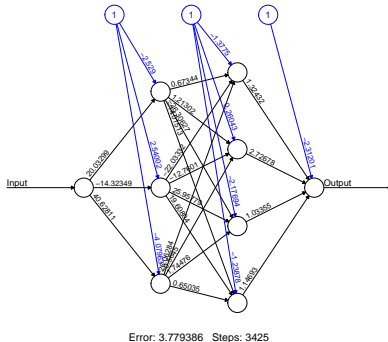


FIGURE: **RECALL:** Single hidden layer neural network. Note the similarity to latent factor models

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

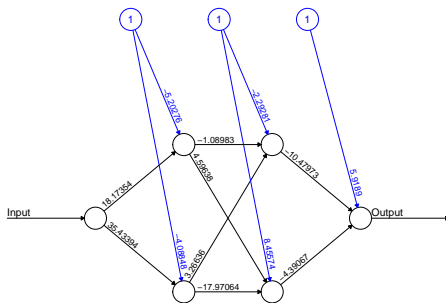
One of the main curses/benefits of neural networks is the ability to **localize**

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))
```

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

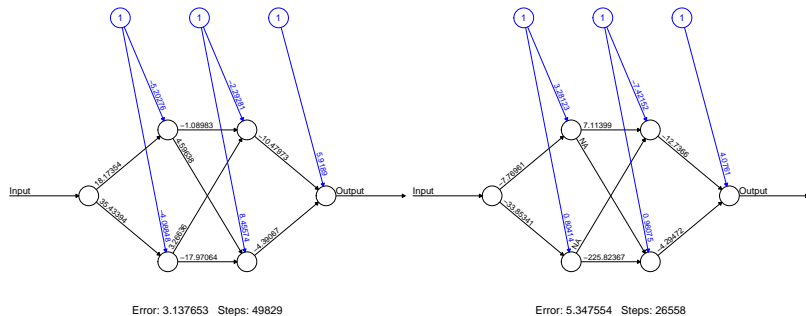


FIGURE: Not-constrained vs. constrained

NEURAL NETWORKS: 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

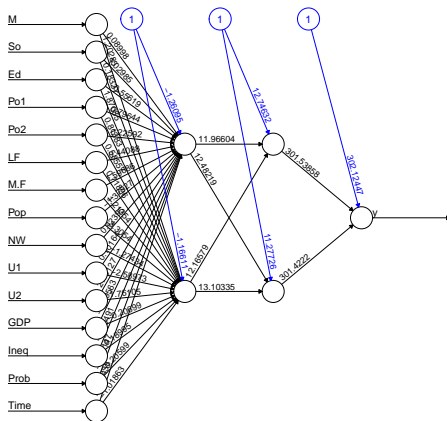
number of males per 1000 females.

...

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, in my opinion, when 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:

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

(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!