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)

$$
\begin{aligned}
0 \text{ Layer} :&= \sigma(\alpha_{\text{lowest}}^{\top} X) \\
1 \text{ Layer}: &= \sigma(\alpha_{\text{lowest}+1}^{\top}(0 \text{ Layer})) \\
&\vdots \\
\text{Top Layer} :&= \sigma(\alpha_{\text{Top}}^{\top}(\text{Top - 1 Layer})) \\
L(\mu_g(X)) &= \beta_{g0} + \beta_g^{\top}(\text{Top Layer}) \quad (g=1,\ldots G)
\end{aligned}
$$

This looks like iterated matrix multiplications

- $\mathbb{Z}_1 = \sigma(\mathbb{X}\alpha_1)$

  $\vdots$
- $\mathbb{Z}_{L-1} = \sigma(\mathbb{Z}_{L-2}\alpha_{L-1})$
- $\mathbb{Z}_L = \sigma(\mathbb{Z}_{L-1}\alpha_L)$
- $L^{-1}(\beta^{\top}\mathbb{Z}_L)$

($\alpha_l \in \mathbb{R}^{K_{l-1} \times K_l}$, 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. $K >> 1$))

- 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 'wiggliness' 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

  $$(\text{\# Hidden Units on Layer 1})\,(\text{\# Hidden Units on Layer 2})\ldots$$

Neural networks: Example

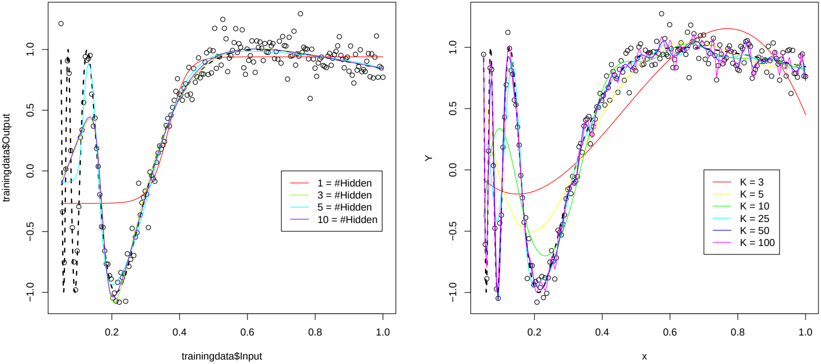


Figure: Single layer NN vs. B-splines

Neural networks: Risk

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

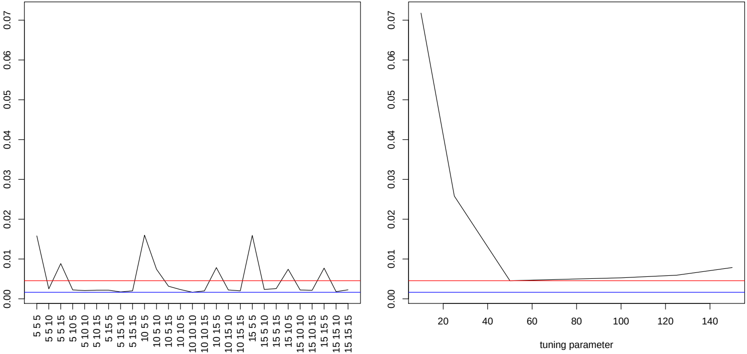


Figure: 3 layer NN[1] vs. B-splines

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

Neural networks: Example

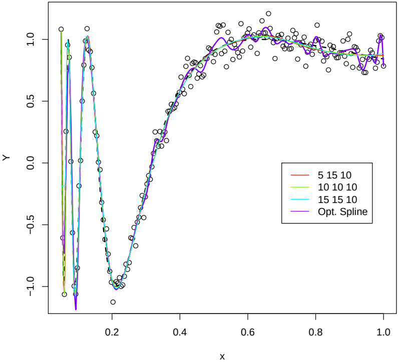


Figure: Optimal NNs vs. Optimal B-spline fit

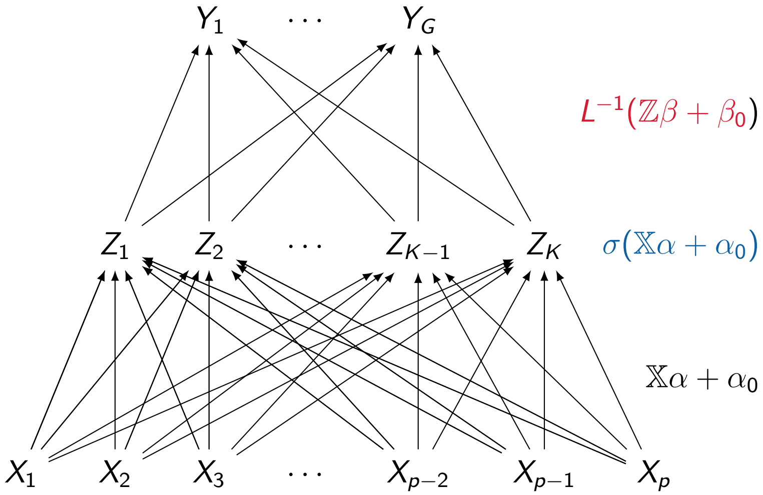
Neural networks: Code for Example

```
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)
```

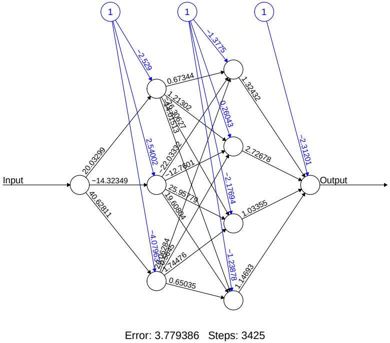
Hierarchical view

Hierarchical view


Y_1 $\cdots$ Y_G
$L^{-1}(\mathbb{Z}\beta + \beta_0)$
Z_1 Z_2 $\cdots$ Z_{K-1} Z_K
$\sigma(\mathbb{X}\alpha + \alpha_0)$
$\mathbb{X}\alpha + \alpha_0$
X_1 X_2 X_3 $\cdots$ X_{p-2} X_{p-1} X_p

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

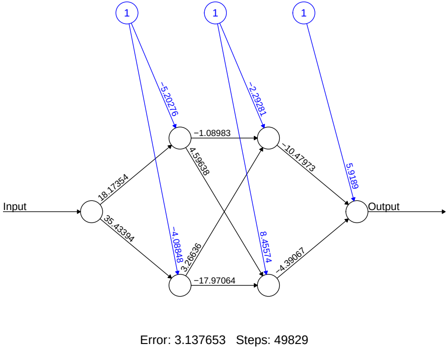


Figure: Unconstrained neural network

```
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 $\mathbb{X} - \overline{\mathbb{X}} = UDV^\top$ 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

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

Scores

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

$$\hat{A}_1 = \begin{bmatrix} \hat{\alpha}_{10} & \hat{\alpha}_{20} \\ \hat{\alpha}_1 & \hat{\alpha}_2 \end{bmatrix} = \begin{bmatrix} -5.20 & -4.08 \\ 18.17 & 35.43 \end{bmatrix}$$

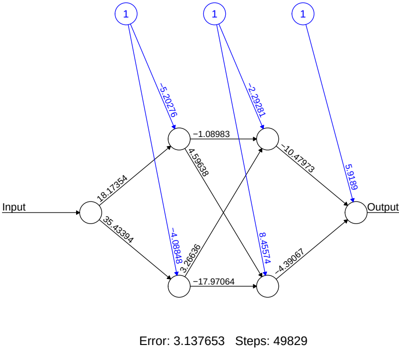
Augment $X \in \mathbb{R}^{n \times 1}$ with intercept column:

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

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

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

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

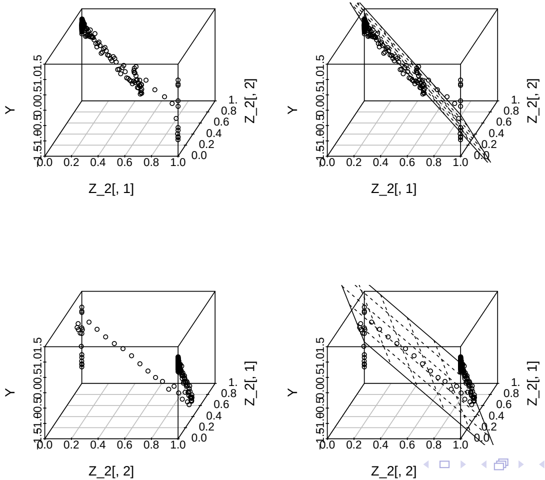


Back to localization

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

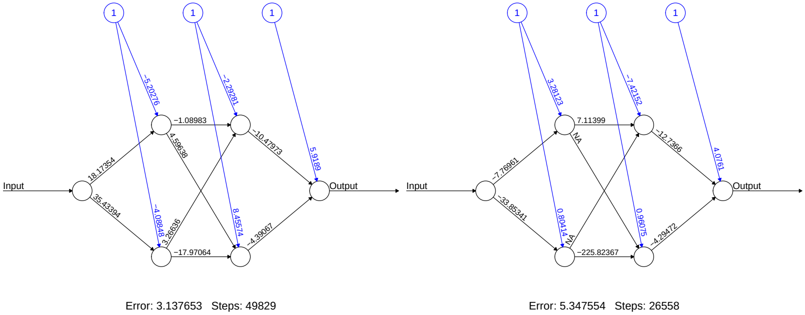


Figure: Not-constrained vs. constrained

Neural networks: Localization

Plots of scores for 2-neuron last hidden layer

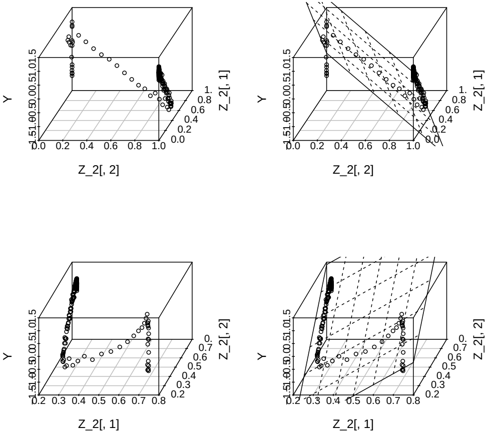
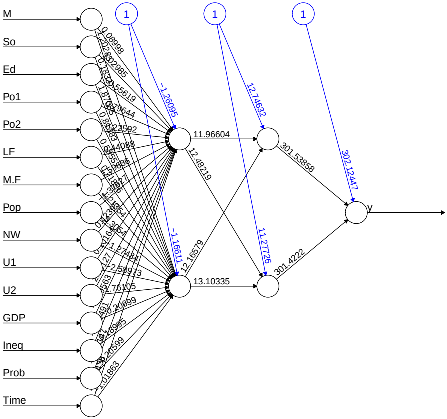


Figure: Top: Not-constrained. Bottom: constrained

Neural networks: Crime data

```
M
percentage of males aged 1424.
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:

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

(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