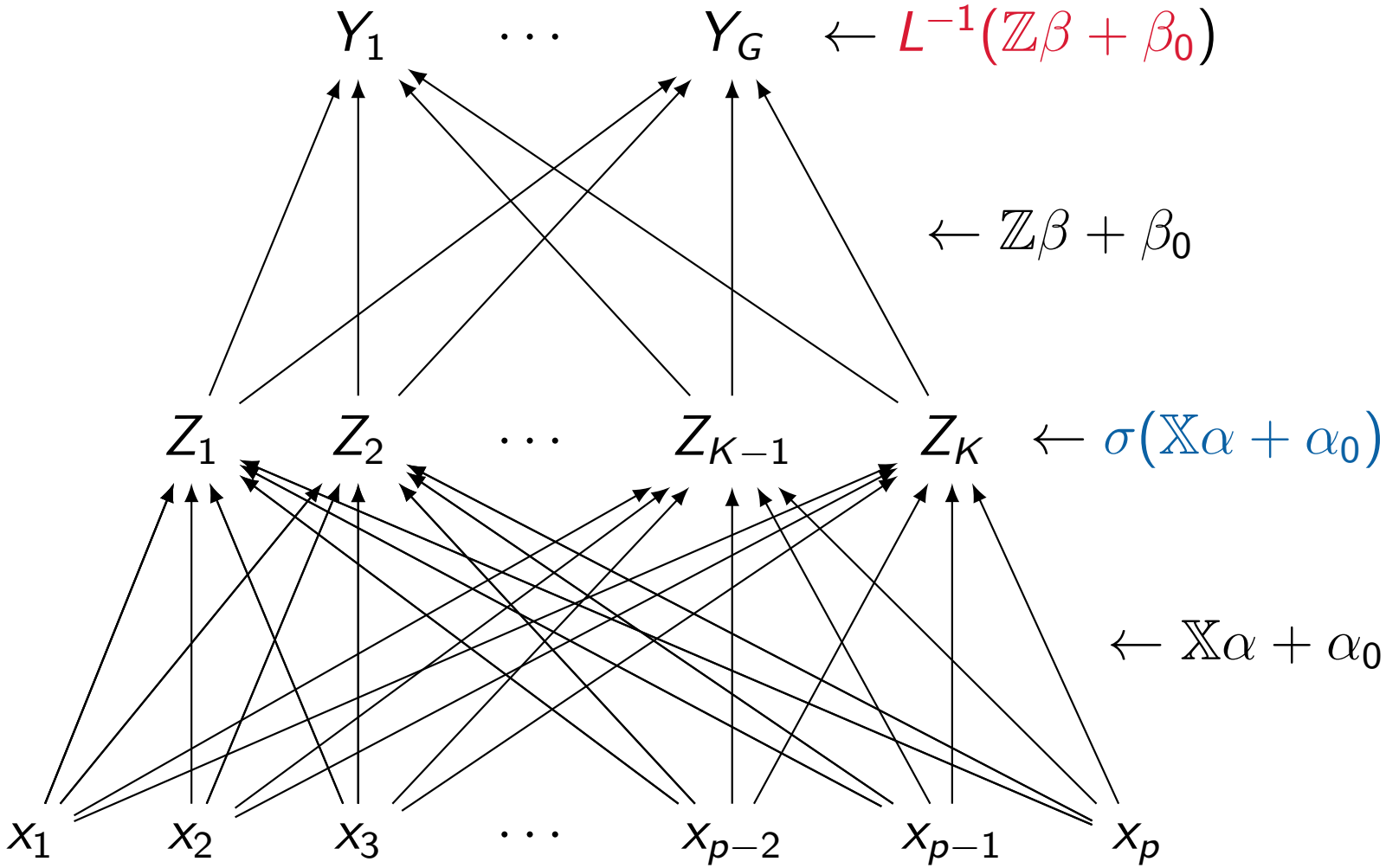
# Neural Networks and Deep Learning 2

## -Introduction to Data Science-

Lecturer: Darren Homrighausen, PhD

# Hierarchical view




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

# 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

  (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

# Back to nonparametric regression

# Nonparametric regression

Suppose $Y \in \mathbb{R}$ and we are trying to nonparametrically fit the regression function

$$\mathbb{E}Y|X = f_*(X)$$

A common approach (particularly when $p$ is small) is to specify a fixed basis of functions, $(\phi_k)_{k=1}^K$

(here $K$ is a tuning parameter)

An example of this approach is splines

# Nonparametric regression

Reminder: Nonparametric regression looks like:

1. Write

$$f_*(X) = \sum_{k=1}^{K} \beta_k \phi_k(X) = \beta^\top \Phi(X)$$

(Again, this is a feature map)

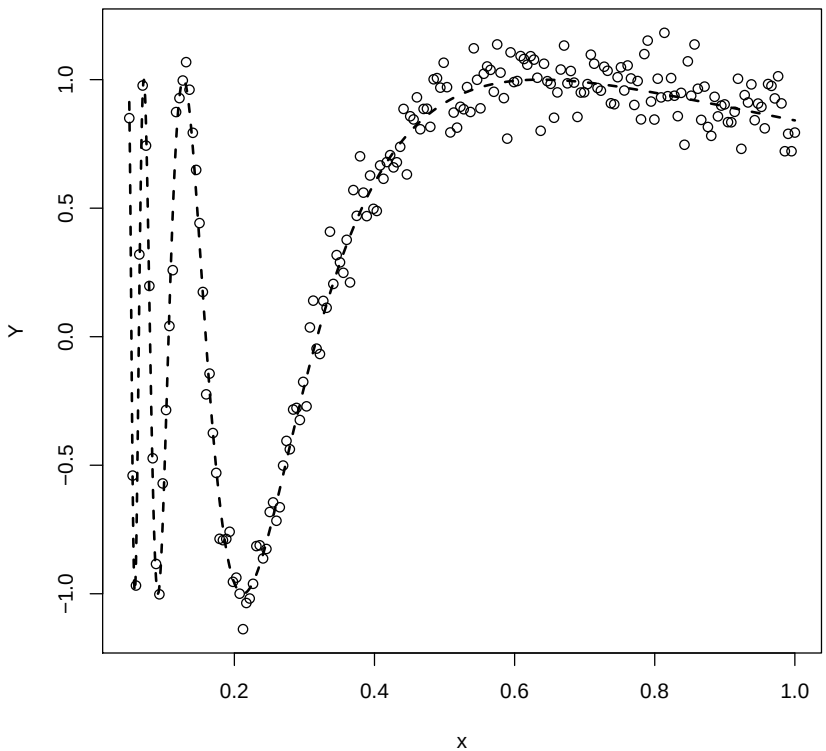
2. Estimate $\beta$ with least squares

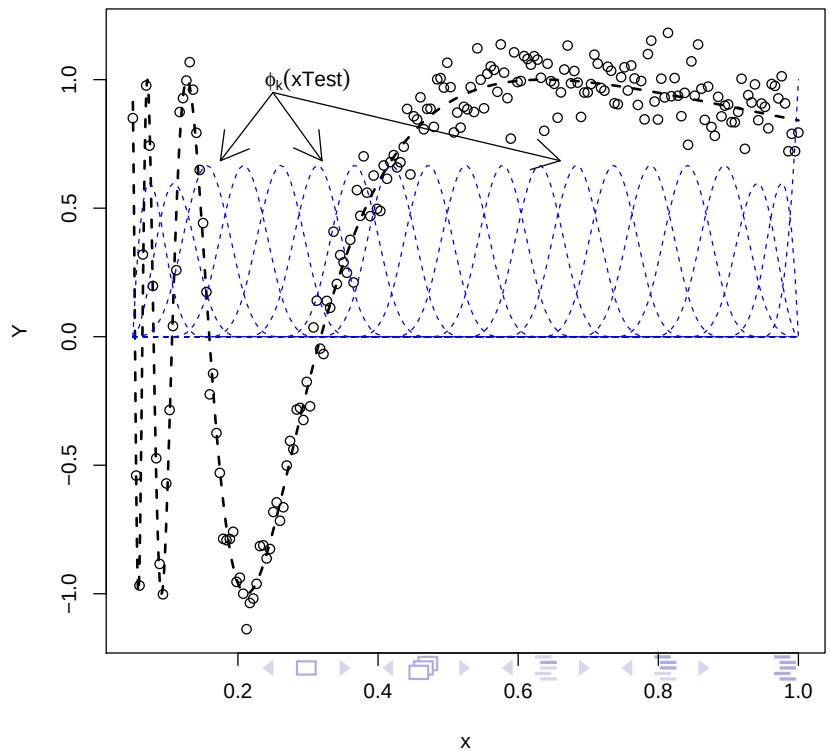
(Often a basis function $\phi_k$ for larger $k$ are rougher $\Rightarrow$ choosing $K$ controls smoothness)

Example: $\phi_k$ could be the $k^{th}$ b-spline basis function

# Back to the doppler function

```
x = seq(.05,1,length=200)
Y = sin(1/x) + rnorm(100,0,.1)
plot(x,Y)
xTest = seq(.05,1,length=1000)
lines(xTest,sin(1/xTest),col='black',lwd=2,lty=2)
```



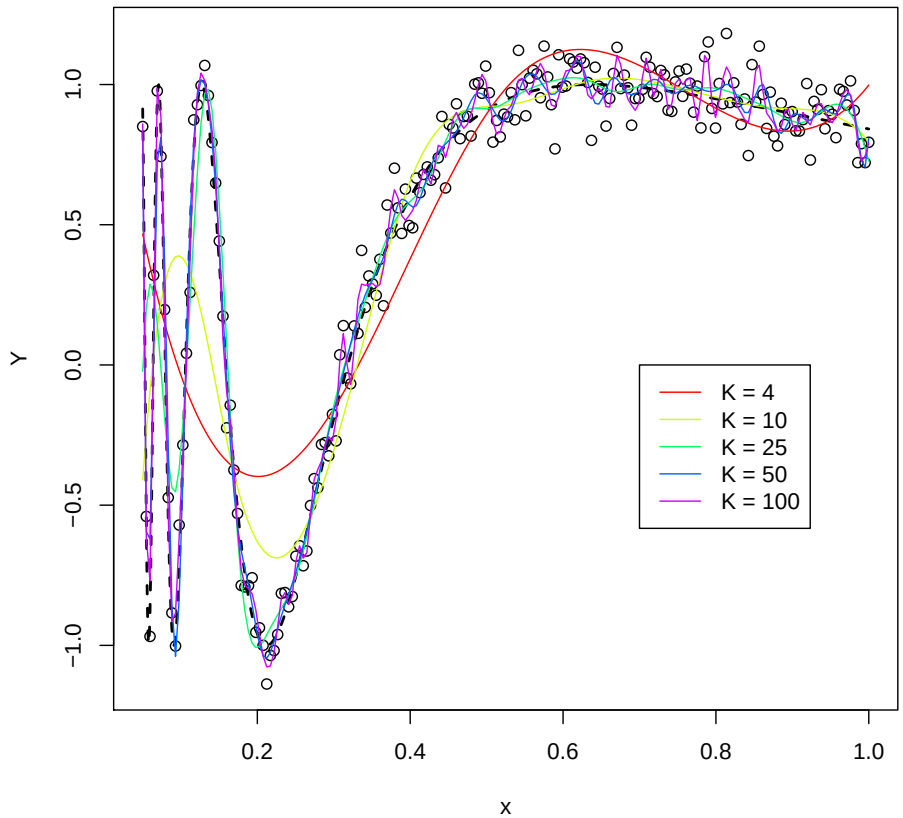
# Nonparametric regression: Example

(Code for second plot on previous slide)

```
require(splines)
X = bs(x,df=20)
plot(x,Y)
lines(xTest,sin(1/xTest),col='black',lwd=2,lty=2)
matlines(x=x,X,lty=2,type='l',col='blue')
label = bquote(phi[k](xTest))
text(x=.3,y=1,label)
arrows(x0=.25,y0=.95,x1=.175,y1=max(X[,df/2])+0.02)
arrows(x0=.25,y0=.95,x1=.32,y1=max(X[,df/2])+0.02)
arrows(x0=.25,y0=.95,x1=.66,y1=max(X[,df/2])+0.02)
```

# Nonparametric regression: Example

```
require(splines)
X    = bs(x,df=K)
Yhat = predict(lm(Y~.,data=X))
```



# Nonparametric regression

The weaknesses of this approach are:

- The basis, $\phi_k$, is fixed and independent of the supervisor
  (Just the coefficients $\beta$ are estimated)
- If $p$ is large, then nonparametrics doesn't work well at all
  (See previous discussion on curse of dimensionality)
- If the basis doesn't 'agree' with $f_*$, then $K$ will have to be large to capture the structure
- What if parts of $f_*$ have substantially different structure?
  (Such is the case in this example.. )

An alternative would be to have the data tell us what kind of basis to use

(We have done a version of this twice in the class: kernel methods and boosting)

# Neural networks: Example

Let's return to the example

We can fit the data with a single layer NN with varying #'s of hidden units $K$

A notable difference with B-splines is that 'wiggliness' doesn't necessarily increase with $K$

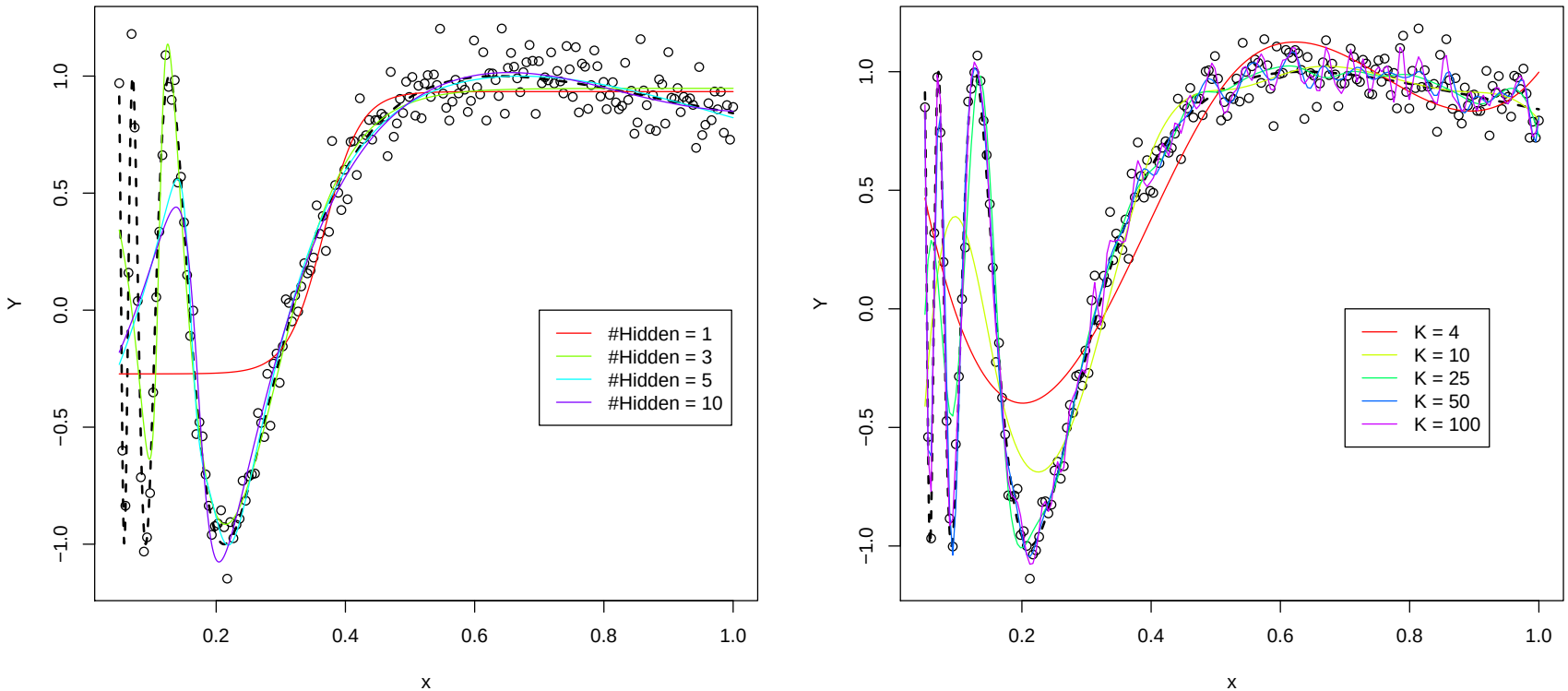
Some specifics:

- 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)

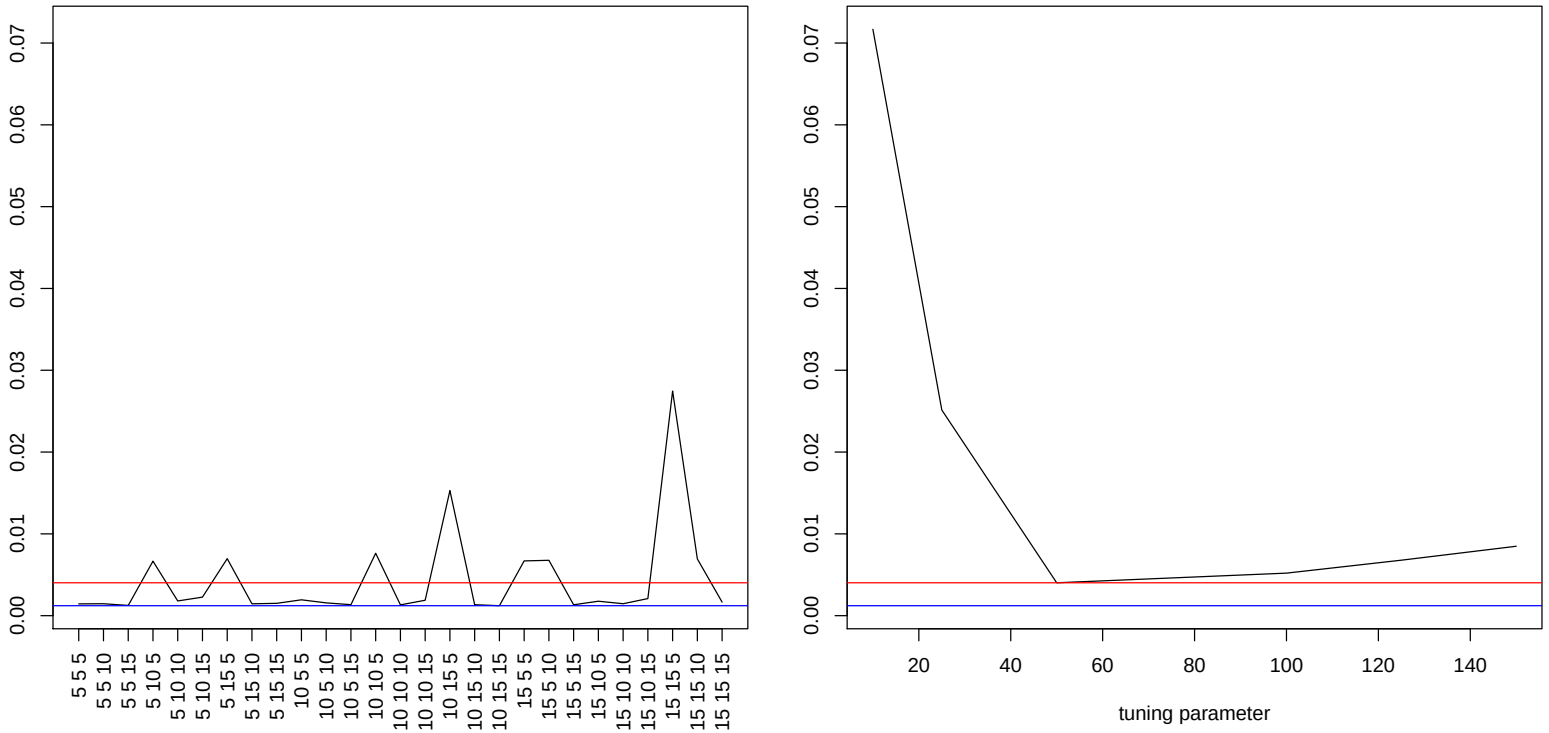
# Neural networks: Example




Single layer NN (left) vs. B-splines (right)

# Neural networks: Risk

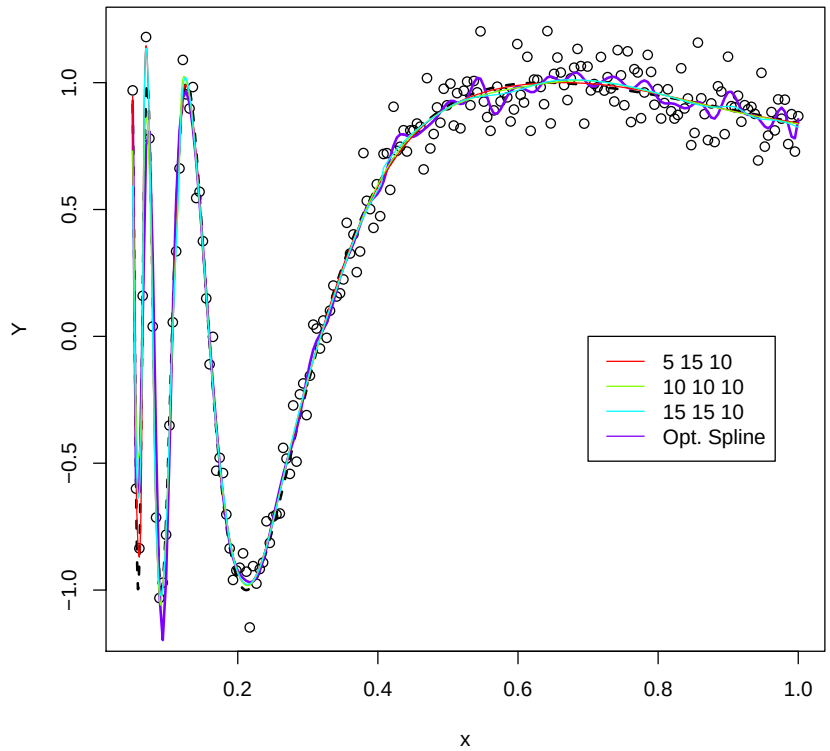
What's the estimation equality? $\text{MSE} = \mathbb{E}(\hat{f}(X) - f_*(X))^2$




3 layer[1] NN vs. B-splines

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

# Neural networks: Example




Optimal NNs vs. Optimal B-spline fit

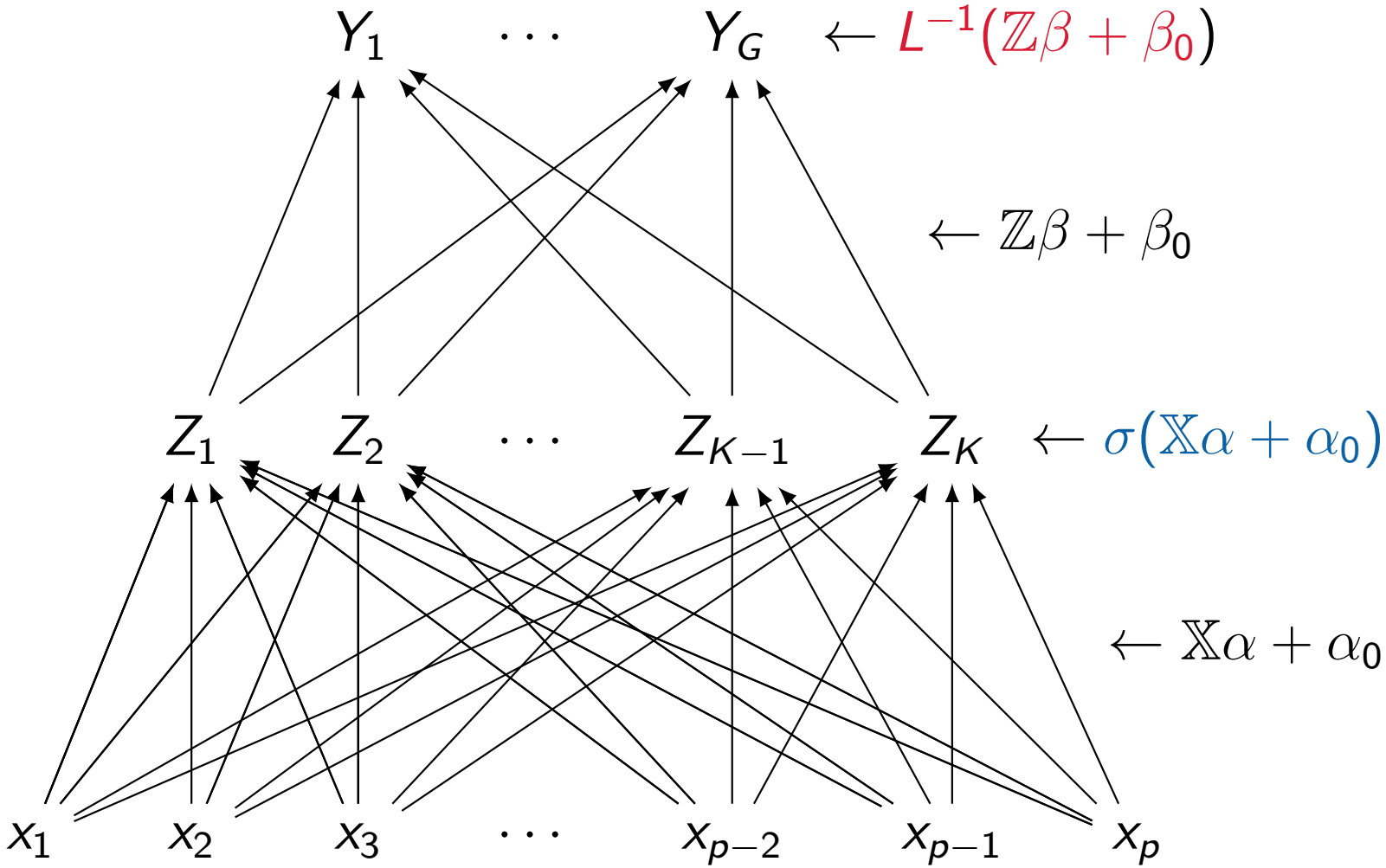
# Neural networks: Code for Example

```
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)
```

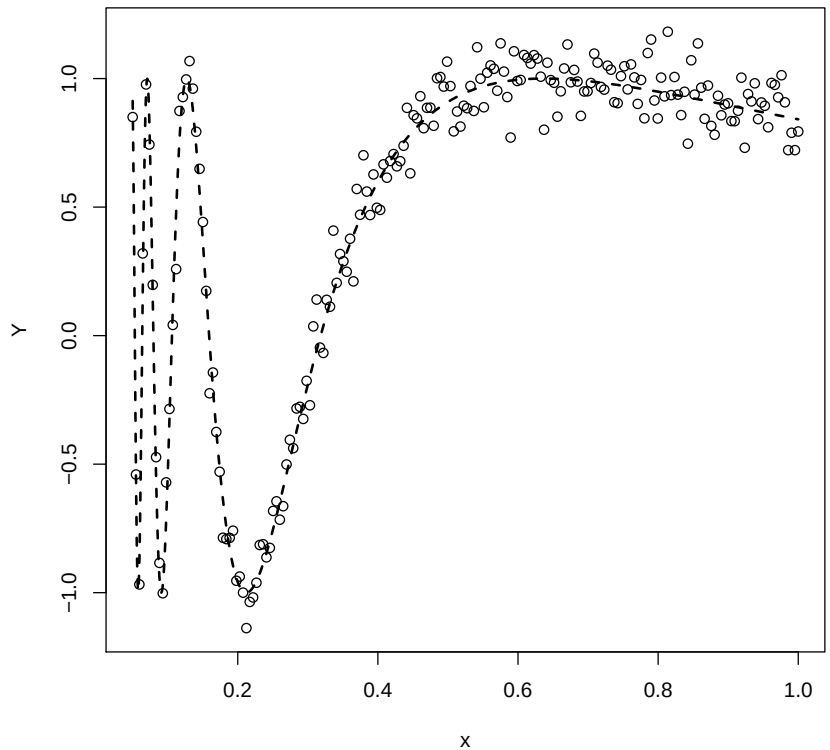
# Hierarchical view

# Hierarchical view

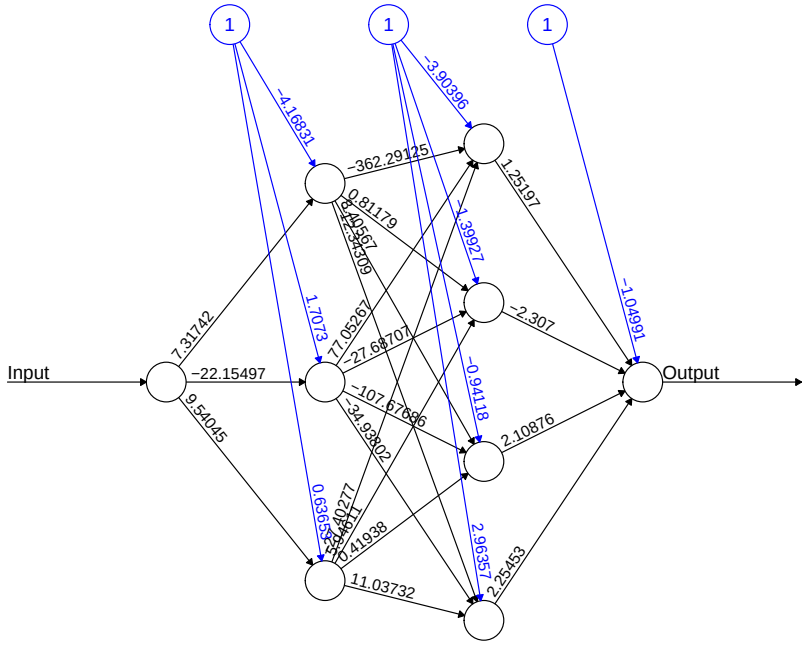



Recall: Single hidden layer neural network. Note the similarity to latent factor models

# Still using the Doppler function..



# Hierarchical from example




Error: 2.472645 Steps: 5792

This is a directed acyclic graph (DAG)

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

# Neural networks: Localization

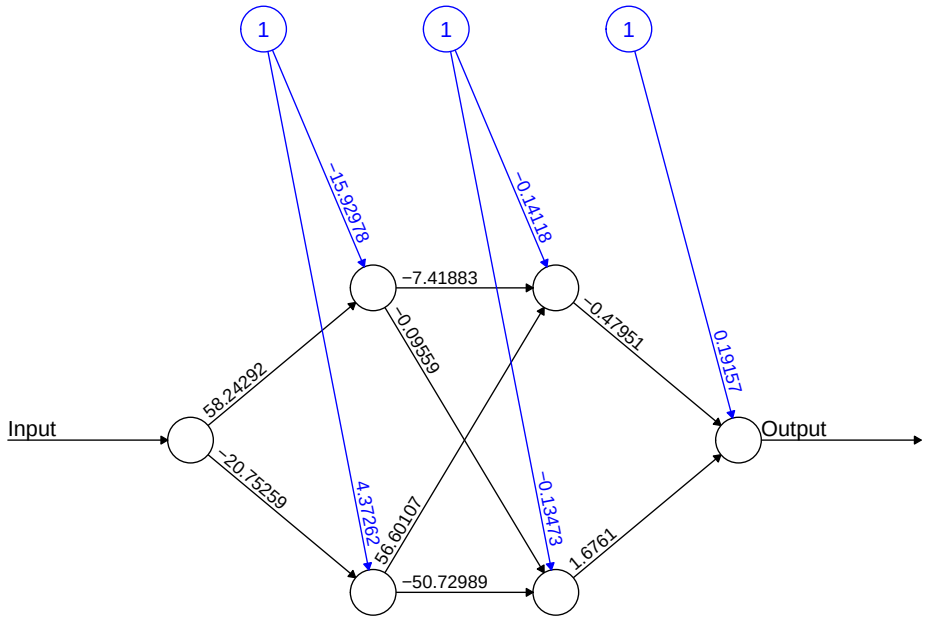
One of the main curses/benefits of neural networks is the ability to localize

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

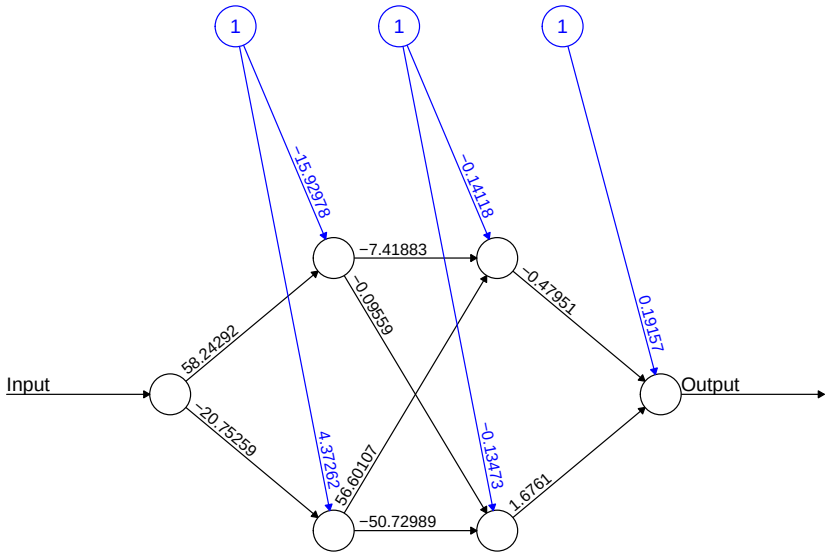
That is, we might want to constrain some of the weights to 0



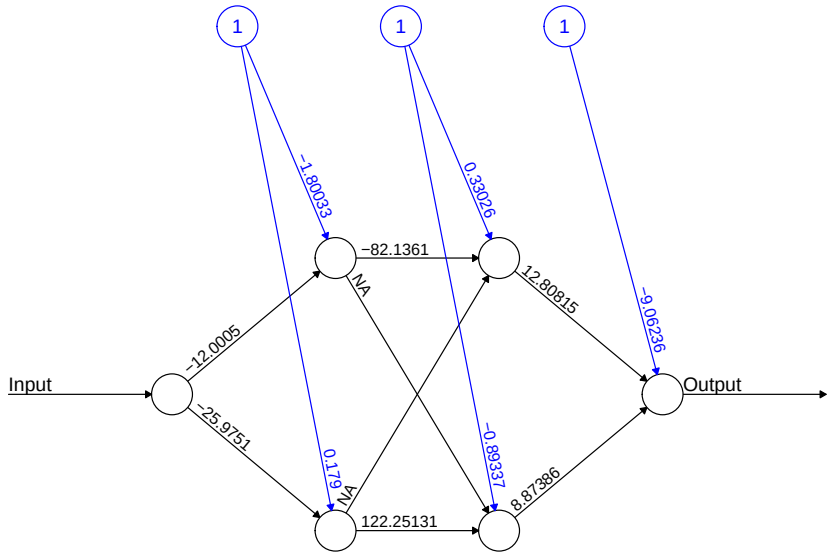

Unconstrained neural network

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

# Neural networks: Localization




Error: 13.010836 Steps: 5272




Error: 4.890935 Steps: 78362

Not-constrained vs. constrained

# 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

(I'm using the notation $Z = UD$ to connect it to the neural network notation)

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

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

(Note that PCA isn't applicable in the Doppler problem.. why?)

# Scores

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

$$\hat{A}_1 = \begin{bmatrix} \hat{\alpha}_1 0 & \hat{\alpha}_2 0 \\ \hat{\alpha}_1 & \hat{\alpha}_2 \end{bmatrix} = \begin{bmatrix} -15.93 & 4.37 \\ 58.24 & -20.75 \end{bmatrix}$$

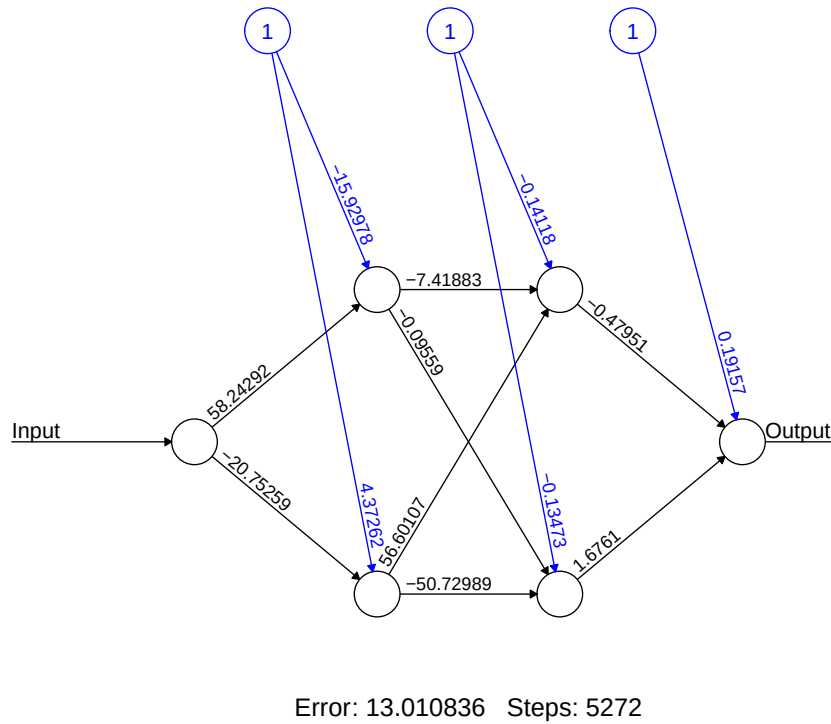
Augment $X \in \mathbb{R}^{n \times 1}$ with intercept column:

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

$1^{st}$ 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)



# Scores: First Layer

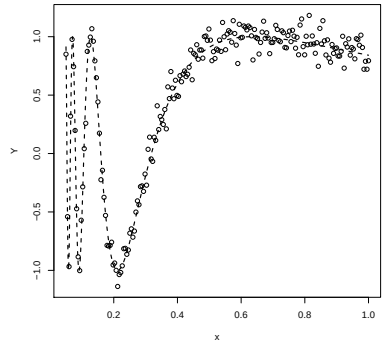
We can get scores from neural networks as well

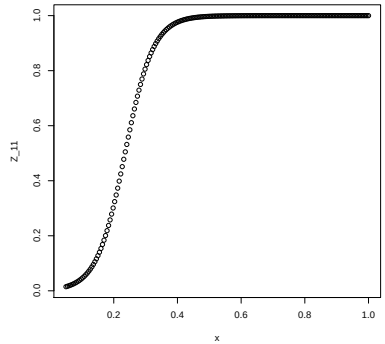
$$\hat{A}_1 = \begin{bmatrix} \hat{\alpha}_1 0 & \hat{\alpha}_2 0 \\ \hat{\alpha}_1 & \hat{\alpha}_2 \end{bmatrix} = \begin{bmatrix} -15.93 & 4.37 \\ 58.24 & -20.75 \end{bmatrix}$$

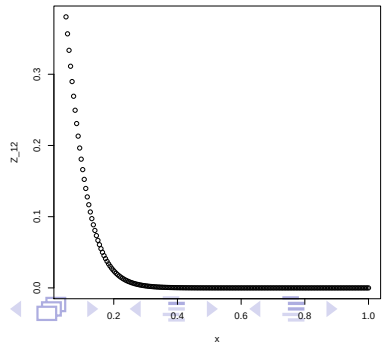
Augment $X \in \mathbb{R}^{n \times 1}$ with intercept column:

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

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

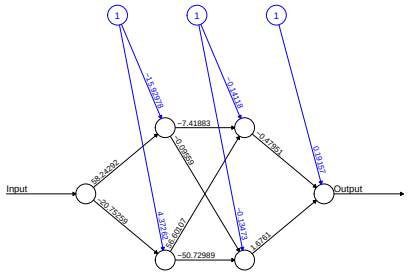


# Scores: Second Layer

We can get scores from neural networks as well




$$\hat{A}_2 = \begin{bmatrix} \hat{\alpha}_1 0 & \hat{\alpha}_2 0 \\ \hat{\alpha}_1 & \hat{\alpha}_2 \end{bmatrix} = \begin{bmatrix} -0.14181 & -0.13473 \\ -7.41883 & 56.60107 \\ -0.09559 & -50.72989 \end{bmatrix}$$

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

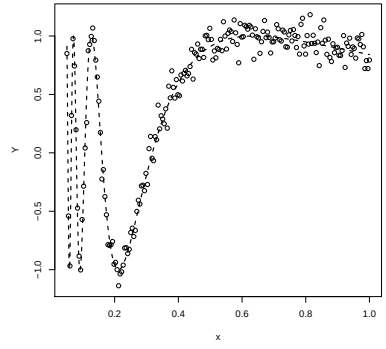
$$[1, \mathbb{Z}_1] \in \mathbb{R}^{n \times 3}$$

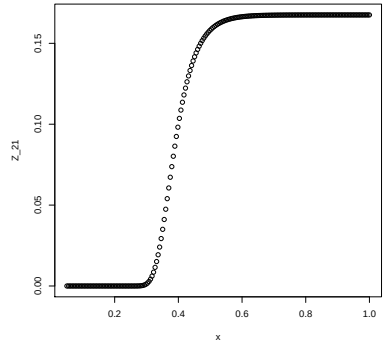
$1^{st}$ layer's scores: $\mathbb{Z}_1 = \sigma(\mathbb{X}\hat{A}_1) \in \mathbb{R}^{n \times 2}$

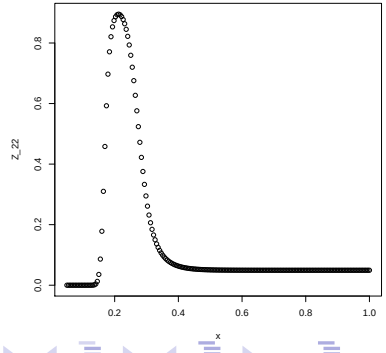
$2^{nd}$ layer's scores:

$\mathbb{Z}_2 = \sigma([1, \mathbb{Z}_1]\hat{A}_2) \in \mathbb{R}^{n \times 2}$

($[1, \mathbb{Z}_1]$ indicates augmenting $\mathbb{Z}_1$ with an intercept column)



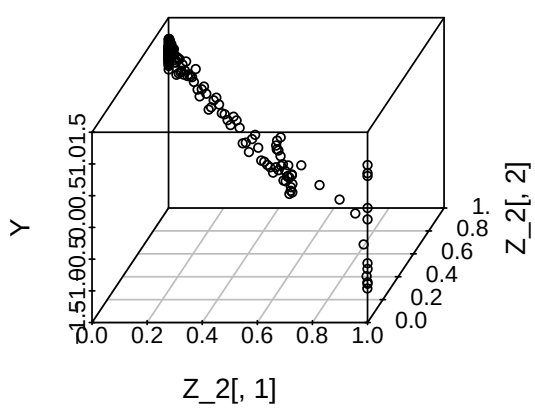



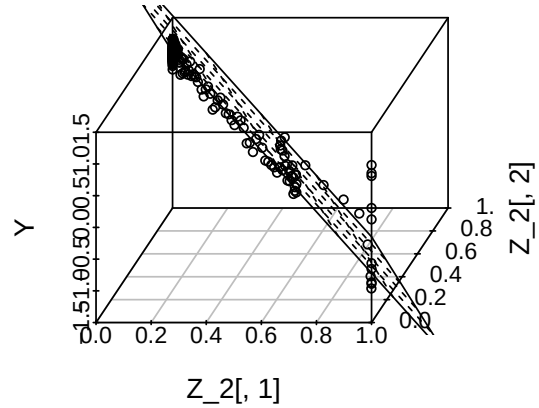
# Back to localization

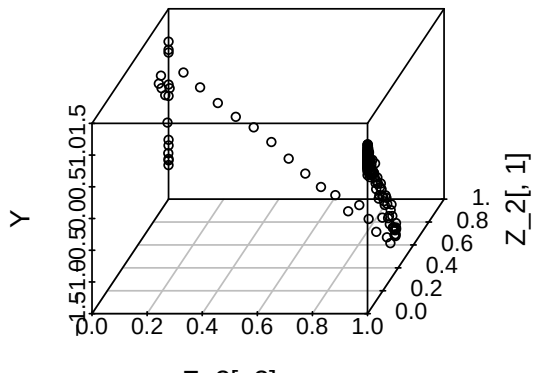
# Neural networks: Localization

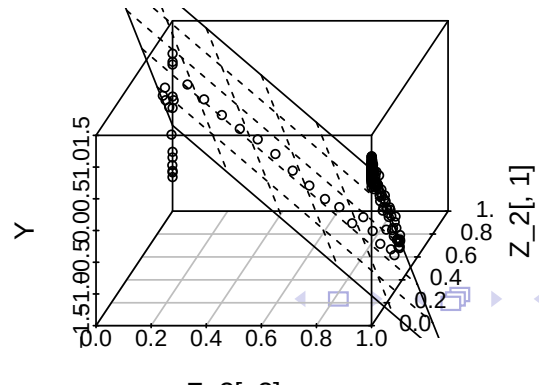
## Scores of the second hidden layer vs. the supervisor

($2^{nd}$ layer's scores: $\mathbb{Z}_2 = \sigma([1, \mathbb{Z}_1]\hat{A}_2) \in \mathbb{R}^{n \times 2}$, the supervisor is $Y$ (the noisy version of the doppler function)



# Neural networks: Localization

We can localize in neuralnet via the exclude parameter

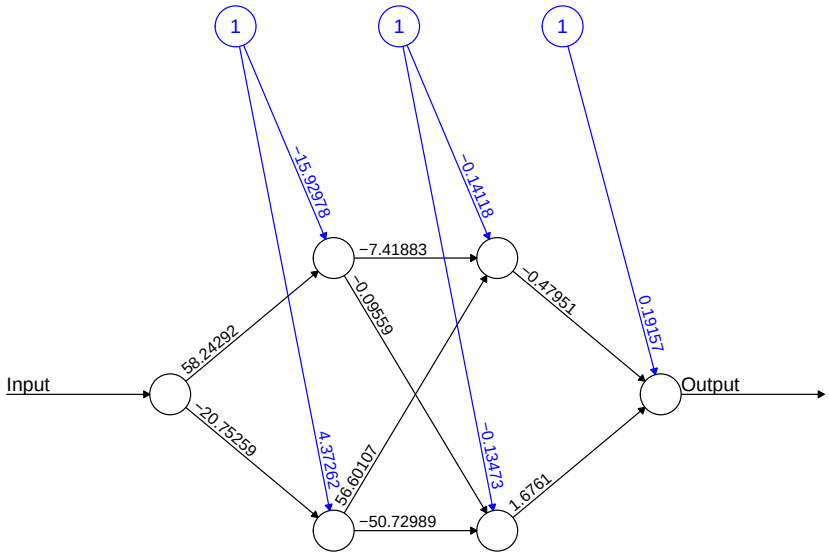
To use it, do the following:

```
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)
```

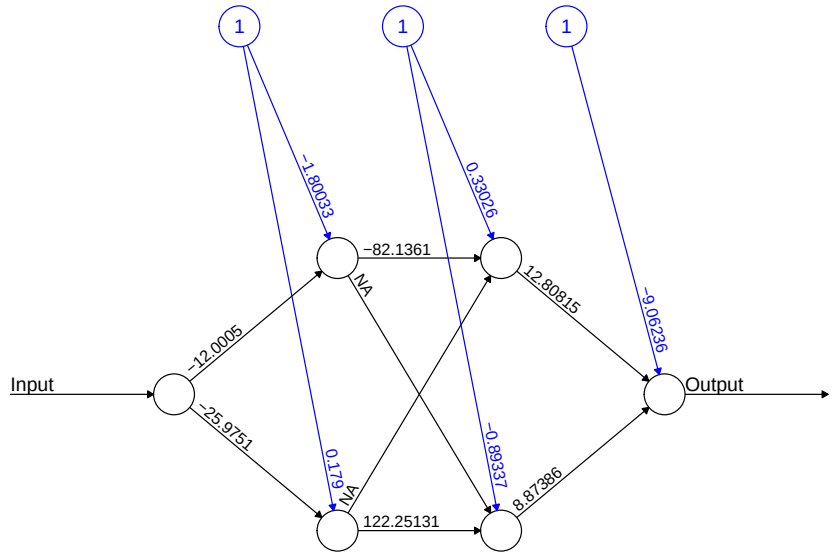
exclude is a $E \times 3$ matrix, with $E$ the number of exclusions

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

# Neural networks: Localization




Error: 13.010836 Steps: 5272



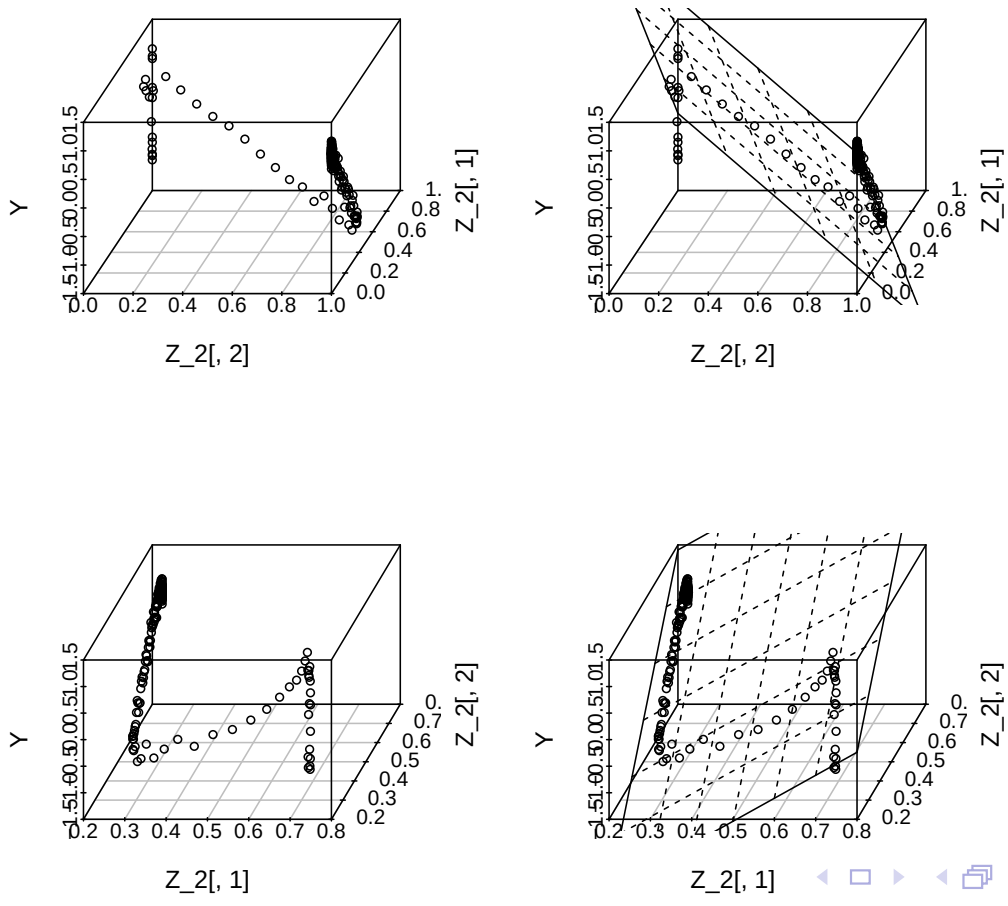

Error: 4.890935 Steps: 78362

Not-constrained vs. constrained

# Neural networks: Localization

Scores of the second hidden layer vs. the supervisor for constrained NN

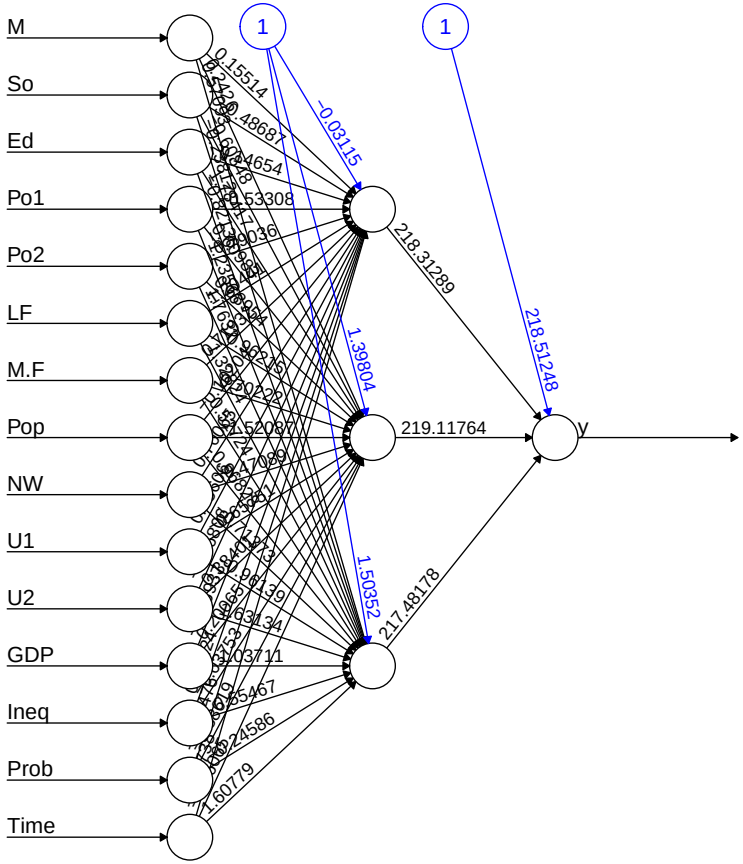
($2^{nd}$ layer's scores: $\mathbb{Z}_2 = \sigma([1, \mathbb{Z}_1]\hat{A}_2) \in \mathbb{R}^{n \times 2}$, the supervisor is $Y$ (the noisy version of the doppler function)



# 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, in my opinion, when neural networks work well)

# Tuning parameters

# Neural networks: Tuning parameters

We can use a GIC method:

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

(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!