University of BRISTOL

Supervised learning in non-linear settings

Henry W J Reeve

henry.reeve@bristol.ac.uk

Statistical Computing & Empirical Methods (EMATM0061)

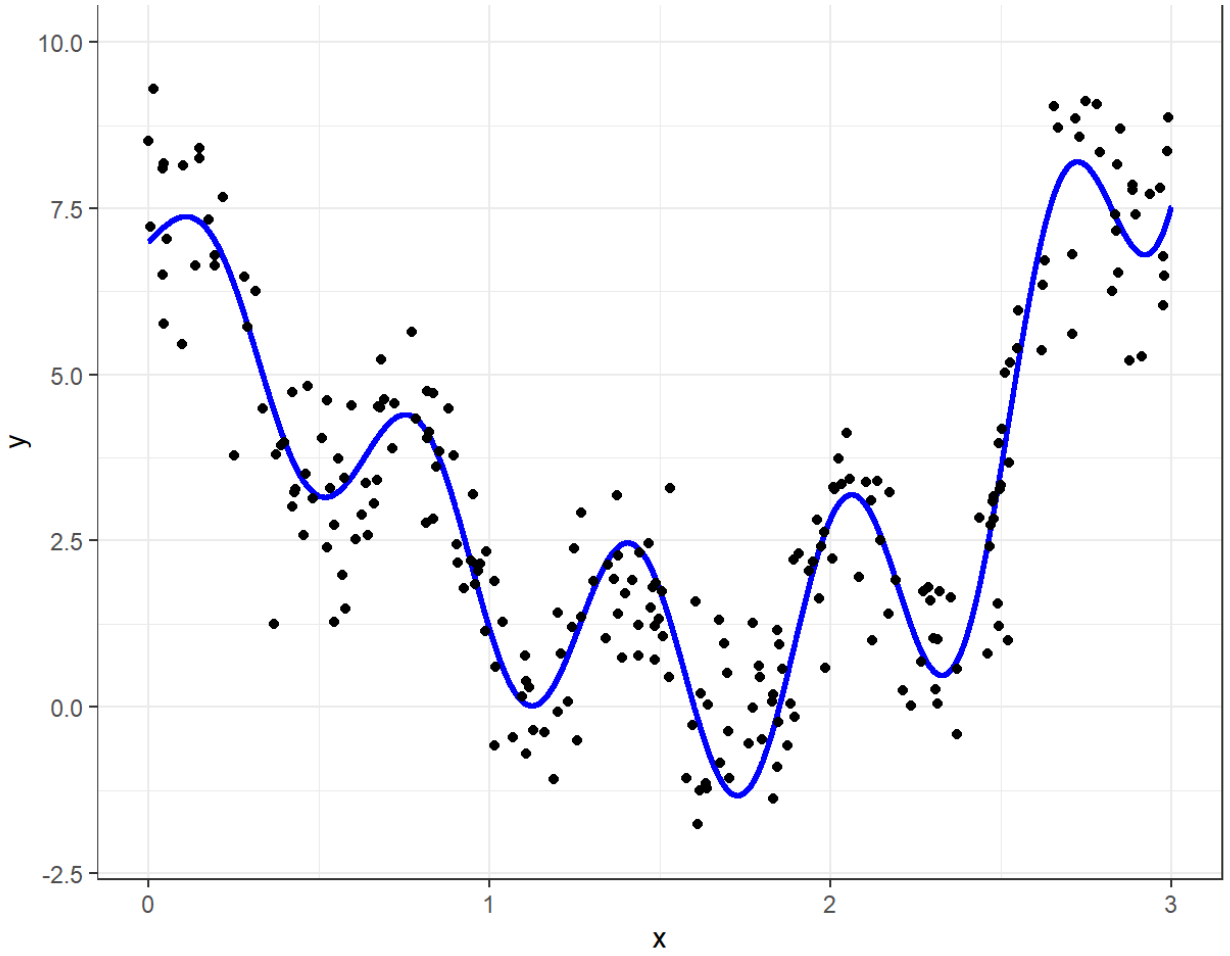
MSc in Data Science, Teaching block 1, 2021.

What will we cover today?

- We will investigate a small selection of different approaches for non-linear regression.
- We will begin with an approach for applying linear methods to non-linear problems with random non-linear feature mappings.
- We will then consider the neural network approach where the feature mapping is learnt from the data.
- We will also talk about the k-nearest neighbour method – a flexible distance-based approach.

A simple regression problem

Let's consider the following regression problem.



(X_i, Y_i)

$(x, \eta(x))$

A simple regression problem

There is an unknown distribution P over pairs (X, Y) taking values in $\mathcal{X} \times \mathbb{R}$.

In our simple example we have $\mathcal{X} = \mathbb{R}$.

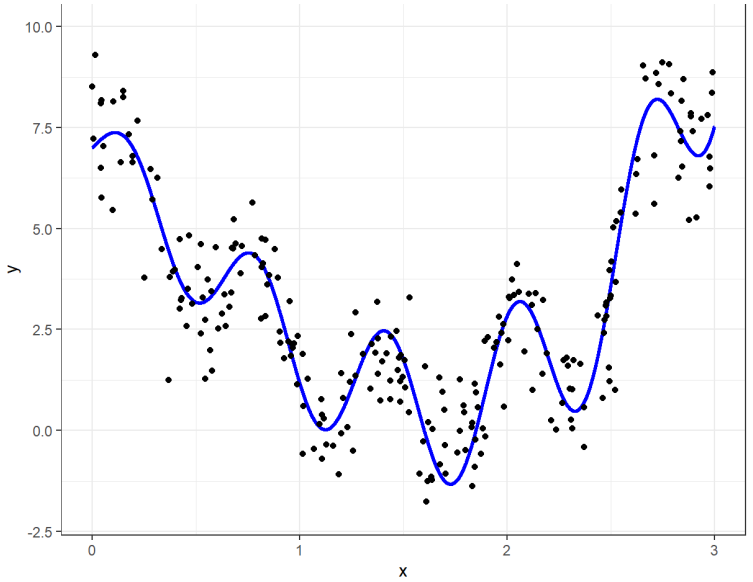
We want to learn a regression model $\phi : \mathcal{X} \to \mathbb{R}$ to minimize the mean squared error,

$$\mathcal{R}_{\mathrm{MSE}}(\phi) := \mathbb{E}[(\phi(X) - Y)^2].$$

The optimal function is $\eta(x) = \mathbb{E}(Y|X = x)$.

We learn a regression model $\hat{\phi} : \mathcal{X} \to \mathbb{R}$ based on data

$\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ with $(X_i, Y_i) \sim \mathrm{P}$ i.i.d.



A simple regression problem

Note that we want to obtain a regression model $\hat{\phi} : \mathcal{X} \to \mathbb{R}$ which does well on unseen data.

This corresponds to achieving a low mean squared error:

$$\mathcal{R}_{\mathrm{MSE}}(\phi) := \mathbb{E}[(\phi(X) - Y)^2]$$

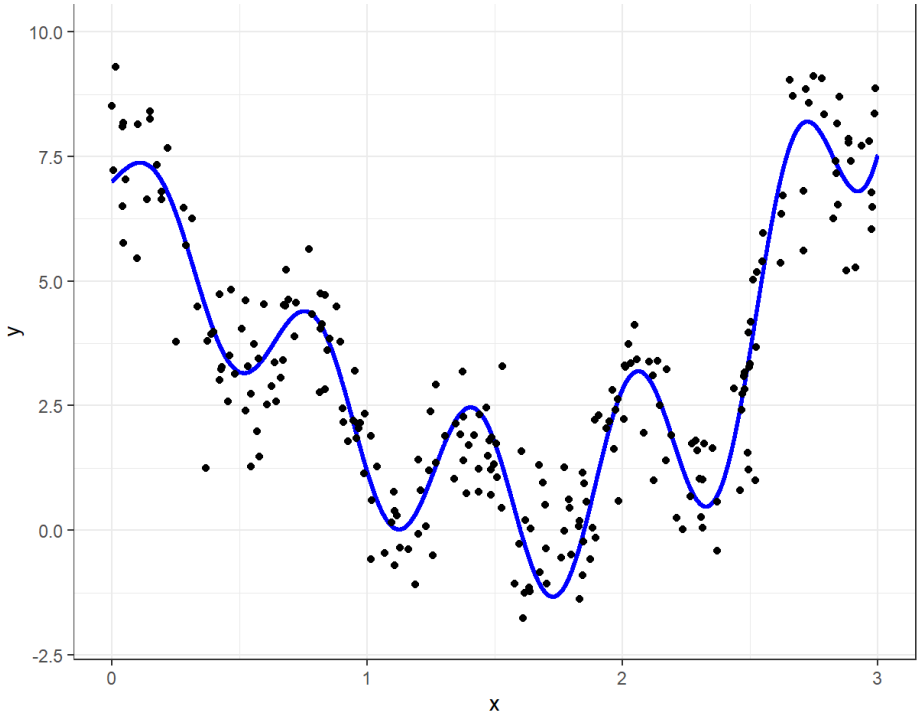
We don't just want to do well on training data & achieve

a low training error

$$\hat{\mathcal{R}}_{\mathrm{MSE}}(\phi) := \frac{1}{n} \sum_{i=1}^{n} (\phi(X_i) - Y_i)^2$$

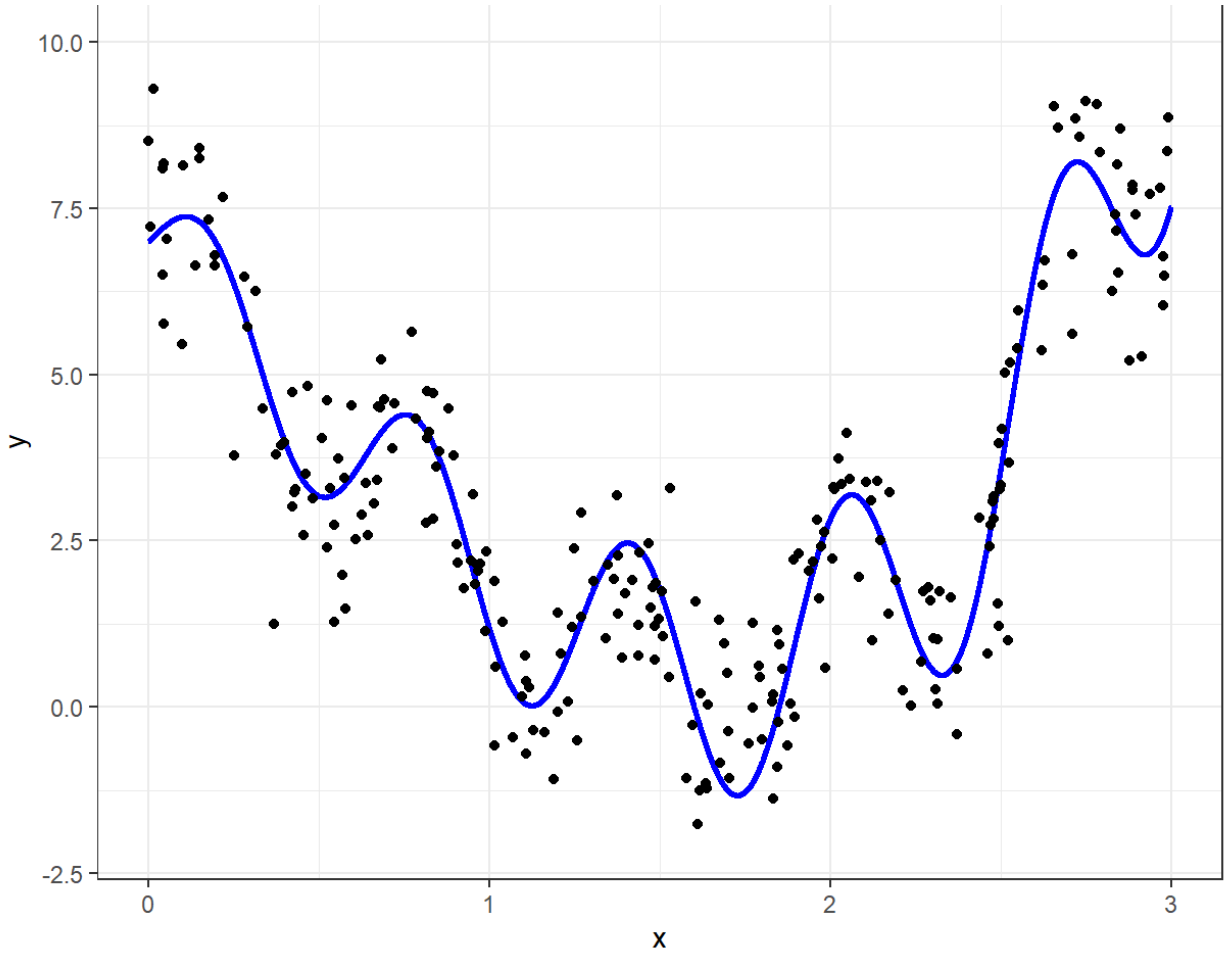
Doing well on unseen data corresponds to having

$\hat{\phi}(X) \approx \eta(X)$ for typical $(X, Y) \sim \mathrm{P}$.



A simple regression problem

Let's consider the following regression problem.



- (X_i, Y_i)
- $(x, \eta(x))$

Here $\eta(x) = \mathbb{E}(Y|X = x)$ is the optimal predictive function.

Linear methods

Previously we considered linear regression models of the form $\phi_{w,w_0} : \mathbb{R}^d \to \mathbb{R}$

defined by $\phi_{w,w_0}(x) = x\ w^\top + w_0$

where $w = (w_1, \cdots, w_d) \in \mathbb{R}^d$ and $w_0 \in \mathbb{R}$.

Example learning algorithms include:

1) Ordinary Least Squares.

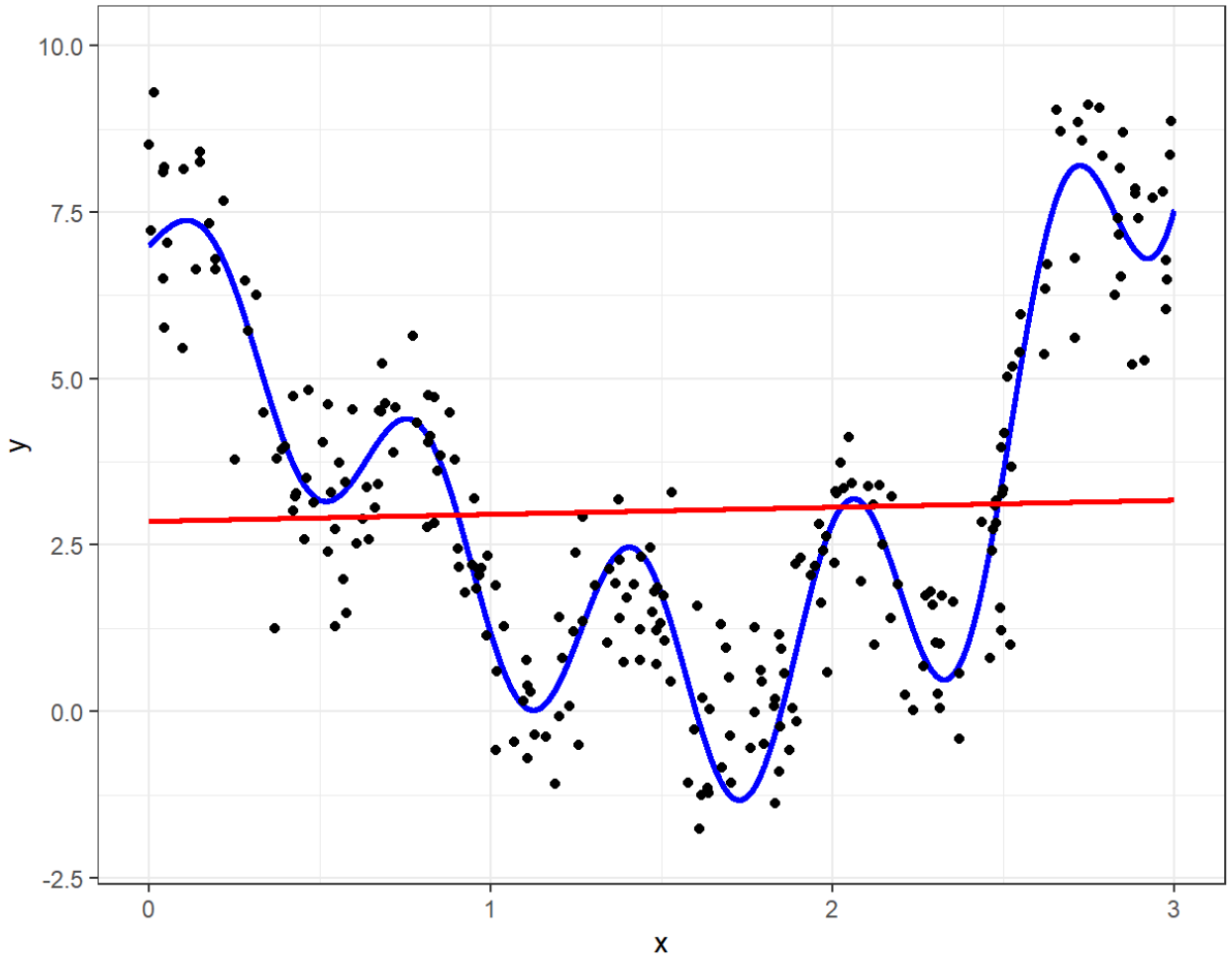
2) Ridge regression.

3) Lasso regression.

4) Elastic net regression.

Linear methods

Linear methods yield poor results for many problems.



(X_i, Y_i)

$(x, \eta(x))$

$(x, \hat{\phi}_{\hat{w}, \hat{w}_0}(x))$

Three approaches to non-linear problems

Approach 1 First apply a stochastic transformation of the feature space.

Then apply a linear technique in the transformed feature space.

Approach 2 Jointly learn a transformation of the feature space and a regression model.

Approach 3 Apply a non-parametric technique based on the proximity of test feature vectors to training examples.

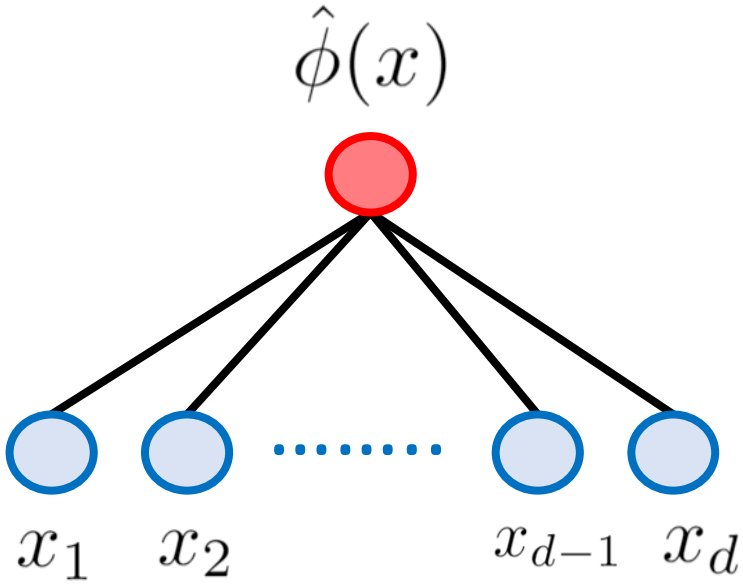
Now take a break!

Statistical Computing & Empirical Methods

Linear methods

We considered linear regression models of the form $\phi_{w,w_0} : \mathbb{R}^d \longrightarrow \mathbb{R}$ defined by

$$\phi_{w,w_0}(x) = w_1 \cdot x_1 + \cdots w_d \cdot x_d + w_0$$



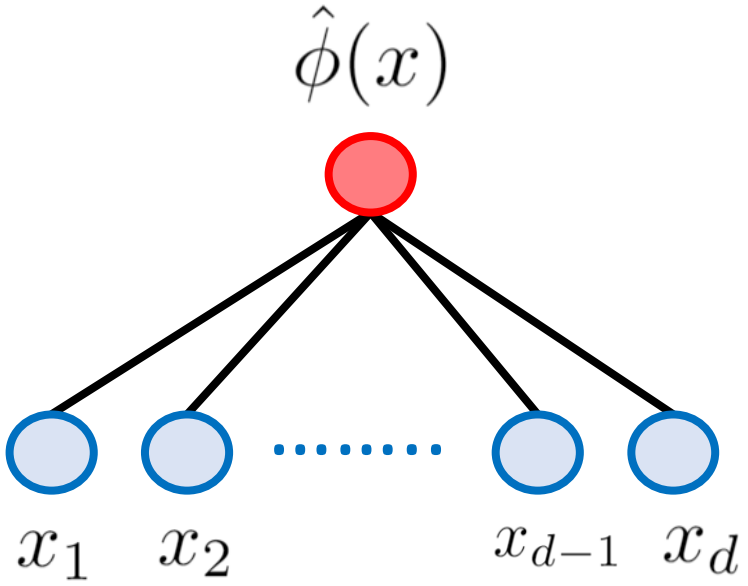
Linear mapping

Linear methods

We considered linear regression models of the form $\phi_{w,w_0} : \mathbb{R}^d \longrightarrow \mathbb{R}$ defined by

$$\phi_{w,w_0}(x) = w_1 \cdot x_1 + \cdots w_d \cdot x_d + w_0$$

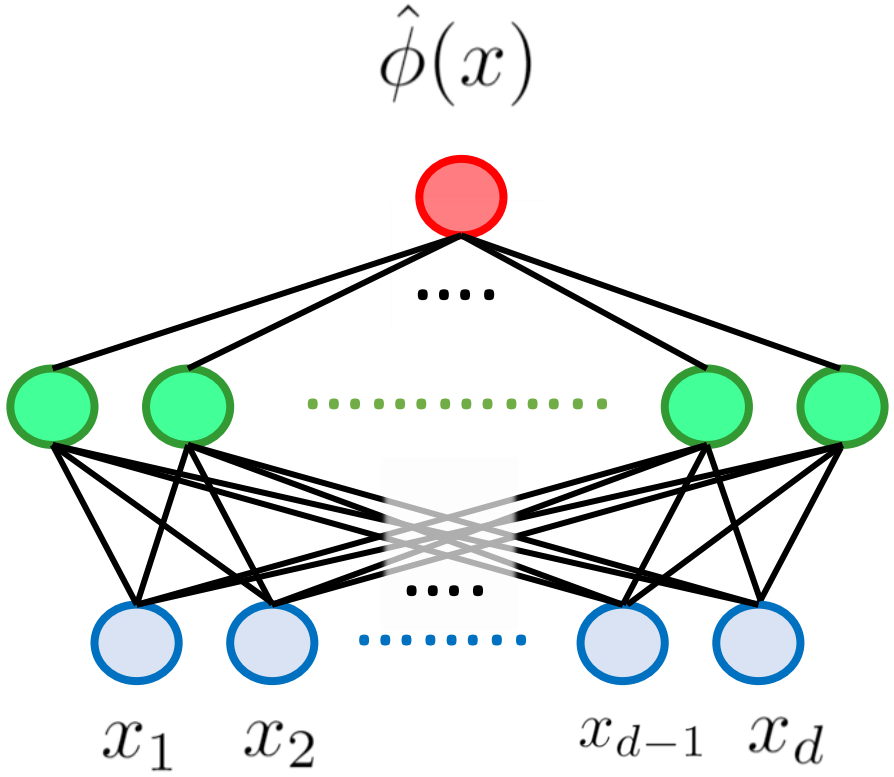
$$= x\ w^\top + w_0.$$



Linear mapping

Neural networks and non-linear feature mappings

We can create non-linear regression models $\hat{\phi} : \mathcal{X} \rightarrow \mathbb{R}$ can be constructed by combining linear mappings with simple non-linear functions.



Linear mapping

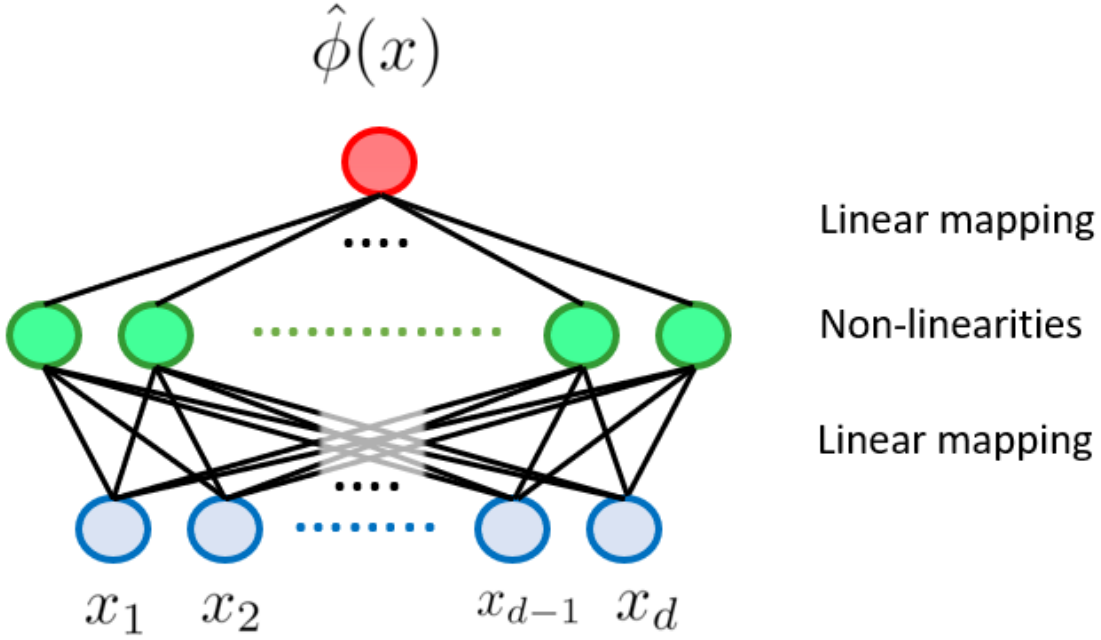
Non-linearities

Linear mapping

Neural networks and non-linear feature mappings

Let's suppose we want to learn a regression model $\hat{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}$.

First choose a non-linear activation function $\psi : \mathbb{R} \rightarrow \mathbb{R}$.



Neural networks and non-linear feature mappings

Let's suppose we want to learn a regression model $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$.

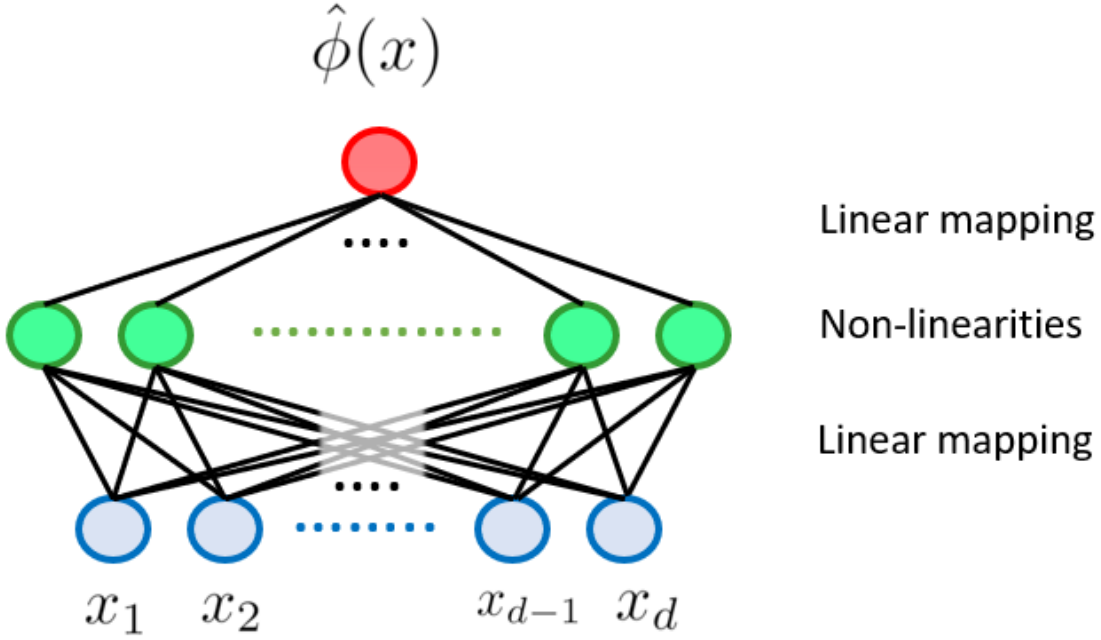
First choose a non-linear activation function $\psi : \mathbb{R} \to \mathbb{R}$.

<u>Examples</u>

a) $\psi(z) = \cos(z)$

b) $\psi(z) = \tanh(z)$

c) $\psi(z) = \max\{0, z\}$



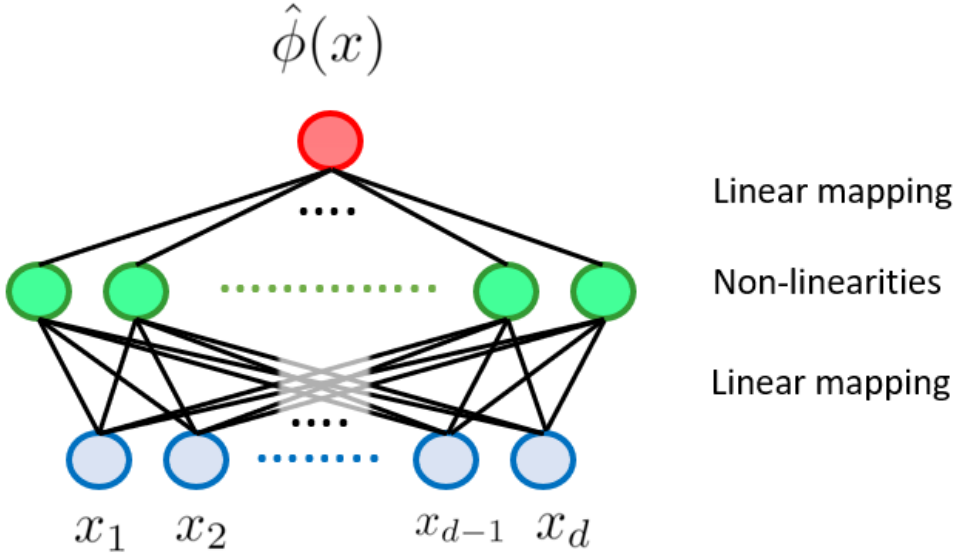
Neural networks and non-linear feature mappings

Let's suppose we want to learn a regression model $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$.

First choose a non-linear activation function $\psi : \mathbb{R} \to \mathbb{R}$.

Take a weight matrix $A \in \mathbb{R}^{q \times d}$

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qd} \end{pmatrix} \in \mathbb{R}^{q \times d}$$



and a bias vector $b = (b_1, \cdots, b_q) \in \mathbb{R}^q$.

Neural networks and non-linear feature mappings

Let's suppose we want to learn a regression model $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$.

First choose a non-linear activation function $\psi : \mathbb{R} \to \mathbb{R}$.

Take a weight matrix $A \in \mathbb{R}^{q \times d}$ and a bias vector $b = (b_1, \cdots, b_q) \in \mathbb{R}^q$.

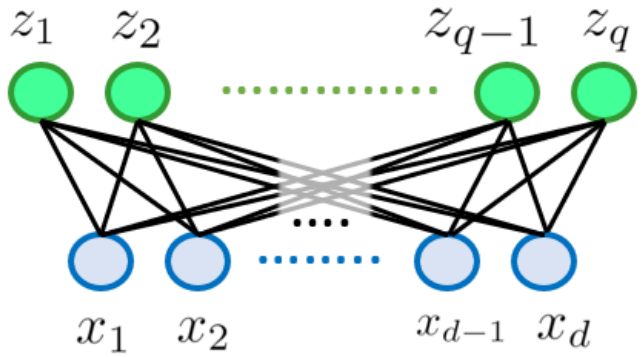
Non-linear feature mappings $z(x) = (z_1(x), z_2(x), \cdots, z_q(x))$

$$z_1(x) = \psi\left(a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots + a_{1d} \cdot x_d + b_1\right)$$

$$z_2(x) = \psi\left(a_{21} \cdot x_1 + a_{22} \cdot x_2 + \cdots + a_{2d} \cdot x_d + b_2\right)$$

$$\vdots$$

$$z_q(x) = \psi\left(a_{q1} \cdot x_1 + a_{q2} \cdot x_2 + \cdots + a_{qd} \cdot x_d + b_q\right).$$



Neural networks and non-linear feature mappings

Let's suppose we want to learn a regression model $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$.

First choose a non-linear activation function $\psi : \mathbb{R} \to \mathbb{R}$.

Take a weight matrix $A \in \mathbb{R}^{q \times d}$ and a bias vector $b = (b_1, \cdots, b_q) \in \mathbb{R}^q$.

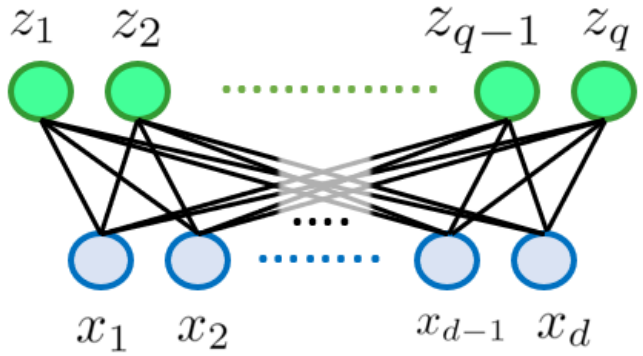
Non-linear feature mappings $z(x) = (z_1(x), z_2(x), \cdots, z_q(x))$

defined by $z(x) = \psi\left(xA^\top + b\right) \in \mathbb{R}^q$

$$z_1(x) = \psi\left(a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots + a_{1d} \cdot x_d + b_1\right)$$

$$\vdots$$

$$z_q(x) = \psi\left(a_{q1} \cdot x_1 + a_{q2} \cdot x_2 + \cdots + a_{qd} \cdot x_d + b_q\right).$$



Neural networks and non-linear feature mappings

Let's suppose we want to learn a regression model $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$.

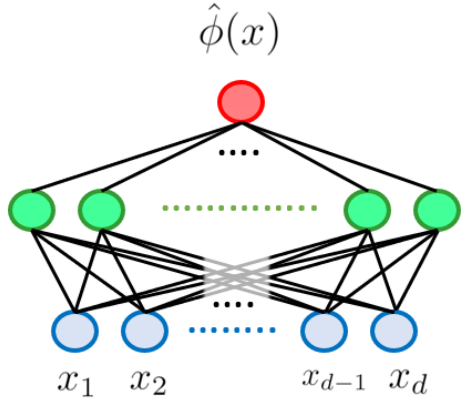
First choose a non-linear activation function $\psi : \mathbb{R} \to \mathbb{R}$.

Take a weight matrix $A \in \mathbb{R}^{q \times d}$ and a bias vector $b = (b_1, \cdots, b_q) \in \mathbb{R}^q$.

Feature mappings $z(x) = \psi\left(xA^\top + b\right) = (z_1(x), z_2(x), \cdots, z_q(x)) \in \mathbb{R}^q$.

Take weights vector $w = (w_1, \cdots, w_q) \in \mathbb{R}^q$ and a bias $w_0 \in \mathbb{R}$.

We take $\hat{\phi}(x) = w_1 \cdot z_1(x) + \cdots + w_q \cdot z_q(x) + w_0$



Neural networks and non-linear feature mappings

Let's suppose we want to learn a regression model $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$.

First choose a non-linear activation function $\psi : \mathbb{R} \to \mathbb{R}$.

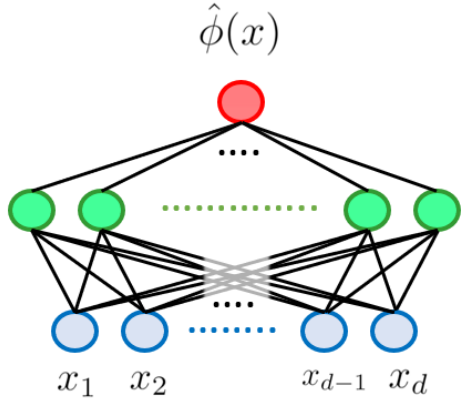
Take a weight matrix $A \in \mathbb{R}^{q \times d}$ and a bias vector $b = (b_1, \cdots, b_q) \in \mathbb{R}^q$.

Feature mappings $z(x) = \psi\left(xA^\top + b\right) = (z_1(x), z_2(x), \cdots, z_q(x)) \in \mathbb{R}^q$.

Take weights vector $w = (w_1, \cdots, w_q) \in \mathbb{R}^q$ and a bias $w_0 \in \mathbb{R}$.

We take $$\hat{\phi}(x) = w_1 \cdot z_1(x) + \cdots + w_q \cdot z_q(x) + w_0$$

$$= z(x)\, w^\top + w_0$$



Neural networks and non-linear feature mappings

Let's suppose we want to learn a regression model $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$.

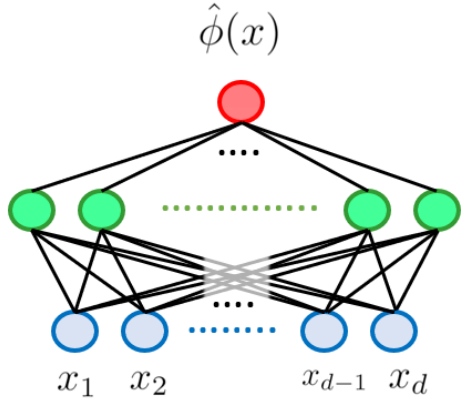
First choose a non-linear activation function $\psi : \mathbb{R} \to \mathbb{R}$.

Take a weight matrix $A \in \mathbb{R}^{q \times d}$ and a bias vector $b = (b_1, \cdots, b_q) \in \mathbb{R}^q$.

Feature mappings $z(x) = \psi\left(xA^\top + b\right) = (z_1(x), z_2(x), \cdots, z_q(x)) \in \mathbb{R}^q$.

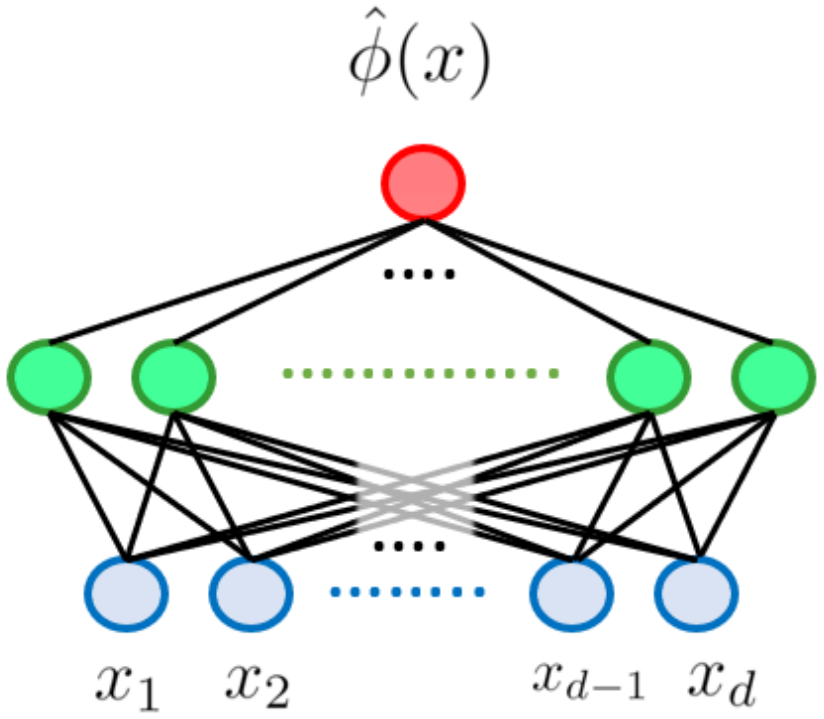
Take weights vector $w = (w_1, \cdots, w_q) \in \mathbb{R}^q$ and a bias $w_0 \in \mathbb{R}$.

We take $$\begin{aligned}\hat{\phi}(x) &= w_1 \cdot z_1(x) + \cdots + w_q \cdot z_q(x) + w_0 \\ &= z(x)\, w^\top + w_0 \\ &= \psi\left(x\, A^\top + b\right)\, w^\top + w_0.\end{aligned}$$



Neural networks and non-linear feature mappings

We can create non-linear regression models $\hat{\phi} : \mathcal{X} \rightarrow \mathbb{R}$ can be constructed by combining linear mappings with simple non-linear functions.



$$\hat{\phi}(x) = z(x)\, w^\top + w_0$$

$$z(x) = \psi\left(xA^\top + b\right)$$

x

How do we choose these non-linear feature transformations?

Now take a break!

Statistical Computing & Empirical Methods

Random feature mappings

Approach 1 First apply a stochastic transformation of the feature space.

Then apply a linear technique in the transformed feature space.

1. Randomly choose weights $A \in \mathbb{R}^{q \times d}$ and biases $b \in \mathbb{R}^q$.
2. Fix feature mappings $z(x) = \psi\left(xA^\top + b\right)$
3. Use the data $((X_1, Y_1), \cdots, (X_n, Y_n))$ to learn a linear mapping

$$\hat{\phi}(x) = z(x)\, w^\top + w_0$$

Ridge regression with random feature mappings

The regression model is $\hat{\phi}(x) = z(x)\, w^\top + w_0$ with $z(x) = \psi\left(xA^\top + b\right)$.

We choose $\sigma > 0$,

Randomly sample $A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qd} \end{pmatrix} \in \mathbb{R}^{q\times d}$ with $a_{rs} \sim \mathcal{N}\left(0, \sigma^2\right)$

Randomly sample $b = (b_1, \cdots, b_q) \in \mathbb{R}^q$ with $b_r \sim \mathrm{Unif}\left([0, 2\pi)\right)$ (i.i.d.)

Define $\psi : \mathbb{R} \to \mathbb{R}$ by $\psi(z) = \cos(z)$.

Ridge regression with random feature mappings

The regression model is $\hat{\phi}(x) = z(x)\, w^\top + w_0$ with $z(x) = \psi\left(xA^\top + b\right)$.

Randomly sample $A \in \mathbb{R}^{q \times d}$ and $b \in \mathbb{R}^q$ with $a_{rs} \sim \mathcal{N}\left(0, \sigma^2\right)$ & $b_r \sim \mathrm{Unif}\left([0, 2\pi)\right)$.

Fix an activation function $\psi : \mathbb{R} \to \mathbb{R}$ by $\psi(z) = \cos(z)$.

We then fix the non-linear feature map $z : \mathbb{R}^d \to \mathbb{R}^q$ by $z(x) = \psi\left(xA^\top + b\right)$.

Ridge regression with random feature mappings

The regression model is $\hat{\phi}(x) = z(x)\, w^\top + w_0$ with $z(x) = \psi\left(xA^\top + b\right)$.

Randomly sample $A \in \mathbb{R}^{q\times d}$ and $b \in \mathbb{R}^q$ with $a_{rs} \sim \mathcal{N}\left(0, \sigma^2\right)$ & $b_r \sim \mathrm{Unif}\left([0, 2\pi)\right)$.

Fix an activation function $\psi : \mathbb{R} \to \mathbb{R}$ by $\psi(z) = \cos(z)$.

We then fix the non-linear feature map $z : \mathbb{R}^d \to \mathbb{R}^q$ by $z(x) = \psi\left(xA^\top + b\right)$.

Now given a data set $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ we set $Z_i = z(X_i)$.

Ridge regression with random feature mappings

The regression model is $\hat{\phi}(x) = z(x)\, w^\top + w_0$ with $z(x) = \psi\left(xA^\top + b\right)$.

Randomly sample $A \in \mathbb{R}^{q\times d}$ and $b \in \mathbb{R}^q$ with $a_{rs} \sim \mathcal{N}\left(0, \sigma^2\right)$ & $b_r \sim \mathrm{Unif}\left([0, 2\pi)\right)$.

Fix an activation function $\psi : \mathbb{R} \to \mathbb{R}$ by $\psi(z) = \cos(z)$.

We then fix the non-linear feature map $z : \mathbb{R}^d \to \mathbb{R}^q$ by $z(x) = \psi\left(xA^\top + b\right)$.

Now given a data set $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ we set $Z_i = z(X_i)$.

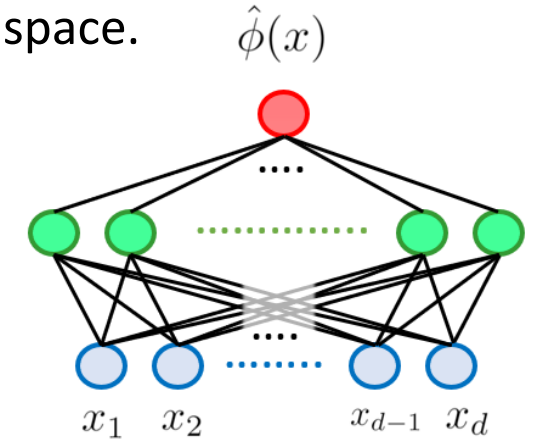
We choose the weights $w = (w_1, \cdots, w_q) \in \mathbb{R}^q$ and bias $w_0 \in \mathbb{R}$

by applying ridge regression to the transformed data $((Z_1, Y_1), \cdots, (Z_n, Y_n))$.

Ridge regression with random feature mappings

Approach 1 First apply a stochastic transformation of the feature space.

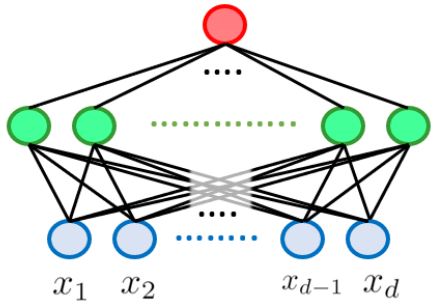
Then apply a linear technique in the transformed feature space.



1. Randomly choose weights $A \in \mathbb{R}^{q \times d}$ and biases $b \in \mathbb{R}^q$.
2. Fix feature mappings $z(x) = \psi\left(xA^\top + b\right)$
3. Transform $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ by $Z_i = z(X_i)$ and apply ridge regression to $((Z_1, Y_1), \cdots, (Z_n, Y_n))$ to learn $\hat{\phi}(x) = z(x)\, w^\top + w_0$.

Ridge regression with random feature mappings

1. Randomly choose weights $A \in \mathbb{R}^{q \times d}$ and biases $b \in \mathbb{R}^q$.



2. Fix feature mappings $z(x) = \psi\left(xA^\top + b\right)$

3. Transform $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ by $Z_i = z(X_i)$ and apply ridge regression

   to $((Z_1, Y_1), \cdots, (Z_n, Y_n))$ to learn $\hat{\phi}(x) = z(x)\, w^\top + w_0$.

This method has two hyper-parameters: (1) The noise parameter σ for the random weights.

(2) The norm penalty λ in the ridge regression.

These can be optimized by performance on the validation set.

Now take a break!

Statistical Computing & Empirical Methods

Ridge regression with random feature mappings in R

Suppose we have a train, validation and test split of our data set.

Let’s extract feature vectors from labels as follows.

```
train_x<-train_data%>%select(-y)%>%as.matrix()
train_y<-train_data%>%pull(y)
```

```
val_x<-val_data%>%select(-y)%>%as.matrix()
val_y<-val_data%>%pull(y)
```

```
test_x<-test_data%>%select(-y)%>%as.matrix()
test_y<-test_data%>%pull(y)
```

Ridge regression with random feature mappings in R

Next we generate our random feature mapping $z(x) = \psi\left(xA^\top + b\right)$ where

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qd} \end{pmatrix} \in \mathbb{R}^{q \times d}$$

sampled with $a_{rs} \sim \mathcal{N}\left(0, \sigma^2\right)$

$b = (b_1, \cdots, b_q) \in \mathbb{R}^q$ sampled with $b_r \sim \text{Unif}\left([0, 2\pi)\right)$ (i.i.d.)

```
num_random_features<-500
sigma=0.01
```

```
d<-train_x%>%ncol()
rand_weights_matrix<-matrix(rnorm(num_random_features*d,sd=sigma),ncol=d)
rand_bias<-runif(num_random_features,max=(2*pi))
```

Ridge regression with random feature mappings in R

Next we generate our random feature mapping $z(x) = \psi\left(xA^\top + b\right)$.

```
d<-train_x%>%ncol()
rand_weights_matrix<-matrix(rnorm(num_random_features*d,sd=sigma),ncol=d)
rand_bias<-runif(num_random_features,max=(2*pi))
```

We specify the random feature mapping in terms of the random weights and bias as follows.

```
random_feature_transform<-function(x){
  n_x<-x%>%nrow()
  z<-cos(x%*%t(rand_weights_matrix)+
           matrix(rep(rand_bias,n_x),nrow=n_x,byrow=TRUE))
  return(z)
}
```

Ridge regression with random feature mappings in R

We now fix our feature mapping $z(x) = \psi\left(xA^\top + b\right)$.

Suppose we have training data $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$.

We transform the data by $Z_i = z(X_i)$ to obtain $((Z_1, Y_1), \cdots, (Z_n, Y_n))$.

```
train_z<-random_feature_transform(train_x)
```

Ridge regression with random feature mappings in R

We now fix our feature mapping $z(x) = \psi\left(xA^\top + b\right)$.

Suppose we have training data $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$.

We transform the data by $Z_i = z(X_i)$ to obtain $((Z_1, Y_1), \cdots, (Z_n, Y_n))$.

```
train_z<-random_feature_transform(train_x)
```

We train a ridge regression model on the transformed data $((Z_1, Y_1), \cdots, (Z_n, Y_n))$.

```
library(glmnet)
lambda<-10
```

```
nonlinear_ridge_model<-glmnet(x=train_z,y=train_y,lambda=lambda,alpha=0)
```

Ridge regression with random feature mappings in R

We now fix our feature mapping $z(x) = \psi\left(xA^\top + b\right)$.

Suppose we have training data $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$.

We transform the data by $Z_i = z(X_i)$ to obtain $((Z_1, Y_1), \cdots, (Z_n, Y_n))$.

We train a ridge regression model on the transformed data $((Z_1, Y_1), \cdots, (Z_n, Y_n))$.

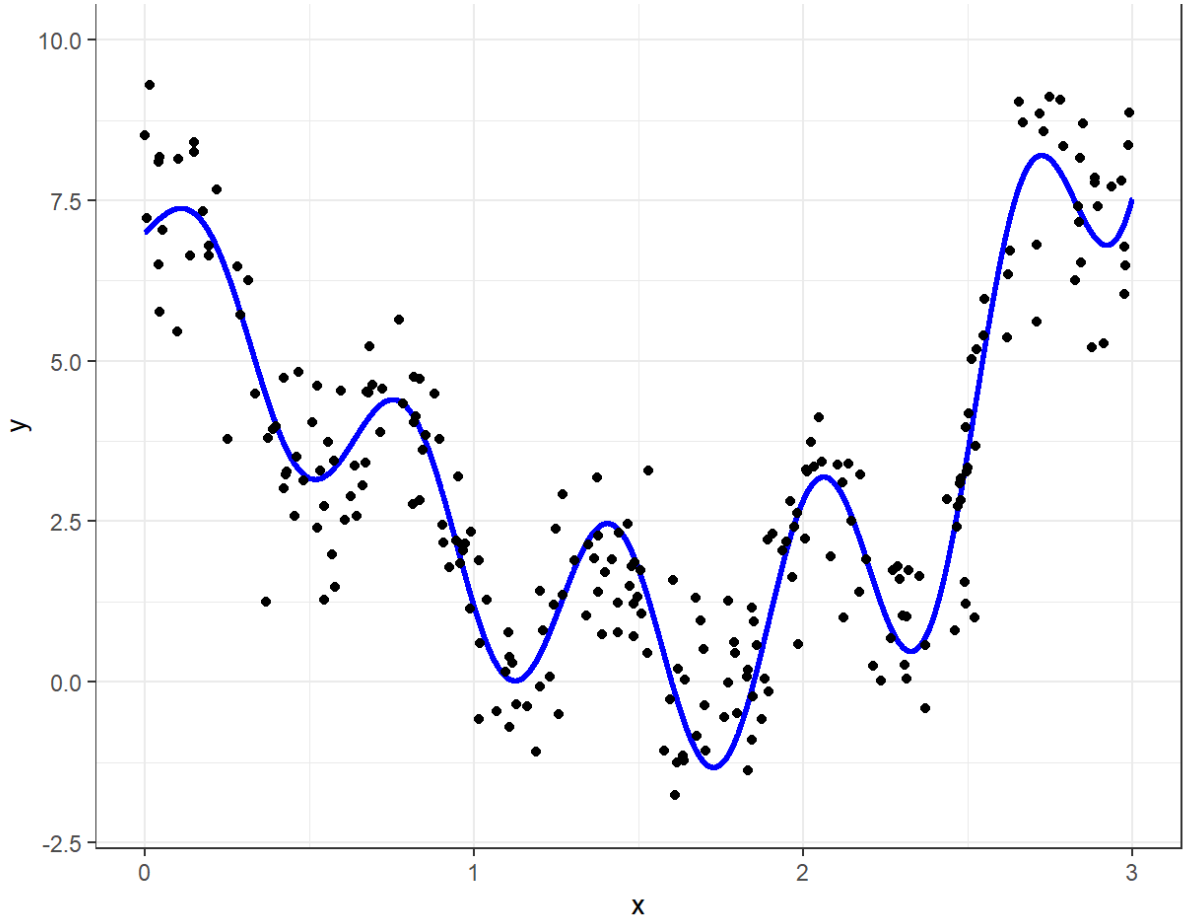
```
nonlinear_ridge_model<-glmnet(x=train_z,y=train_y,lambda=lambda,alpha=0)
```

We can then apply our model to validation (or test data) and compute the mean squared error.

```
val_z<-random_feature_transform(val_x)
val_predictions<-predict(nonlinear_ridge_model,newx=val_z)
val_msq_error<-mean((val_predictions-val_y)^2)
```

Ridge regression with random feature mappings in R

Returning to our example non-linear regression problem.



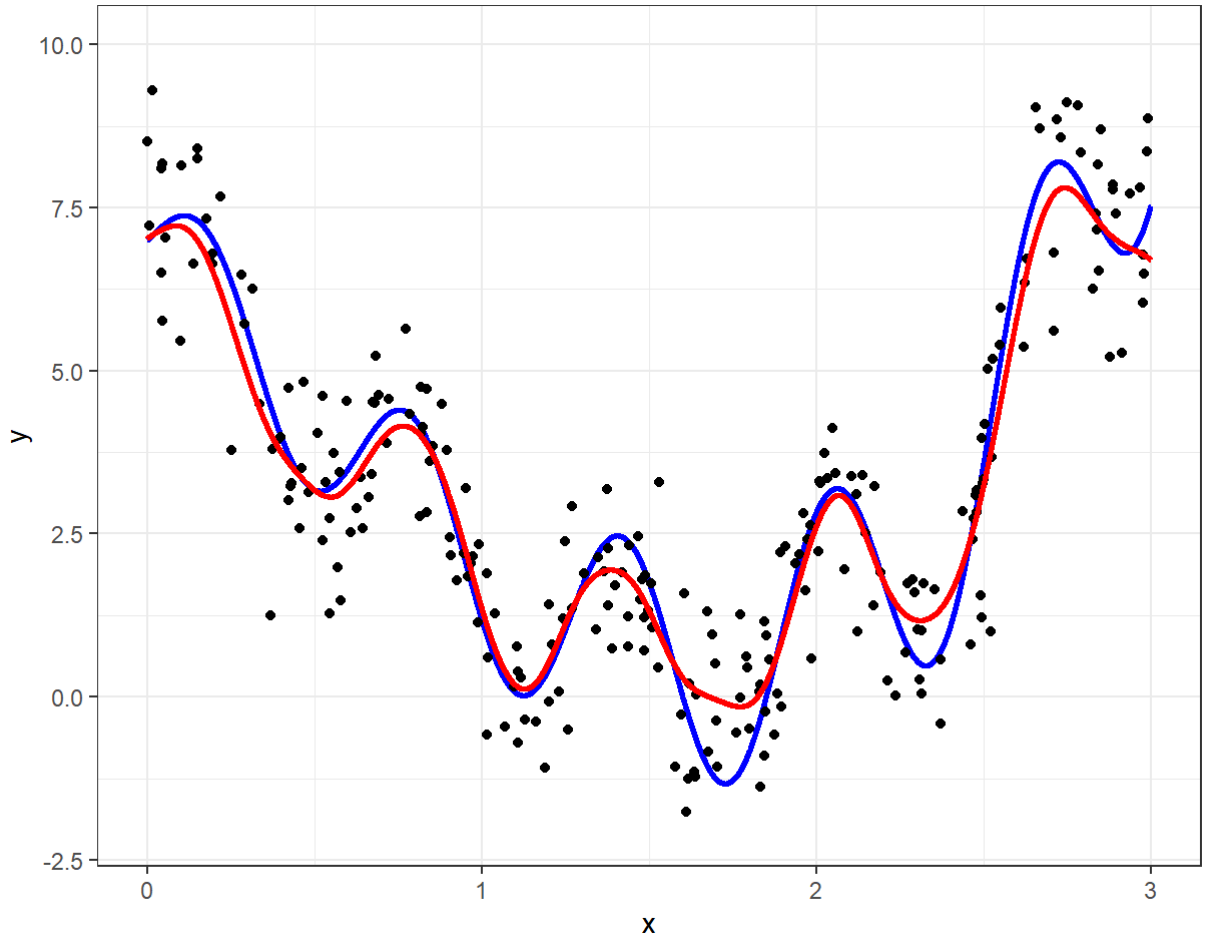
(X_i, Y_i)

$(x, \eta(x))$

Here $\eta(x) = \mathbb{E}(Y|X = x)$ is the optimal predictive function.

Ridge regression with random feature mappings in R

Returning to our example non-linear regression problem.



(X_i, Y_i)

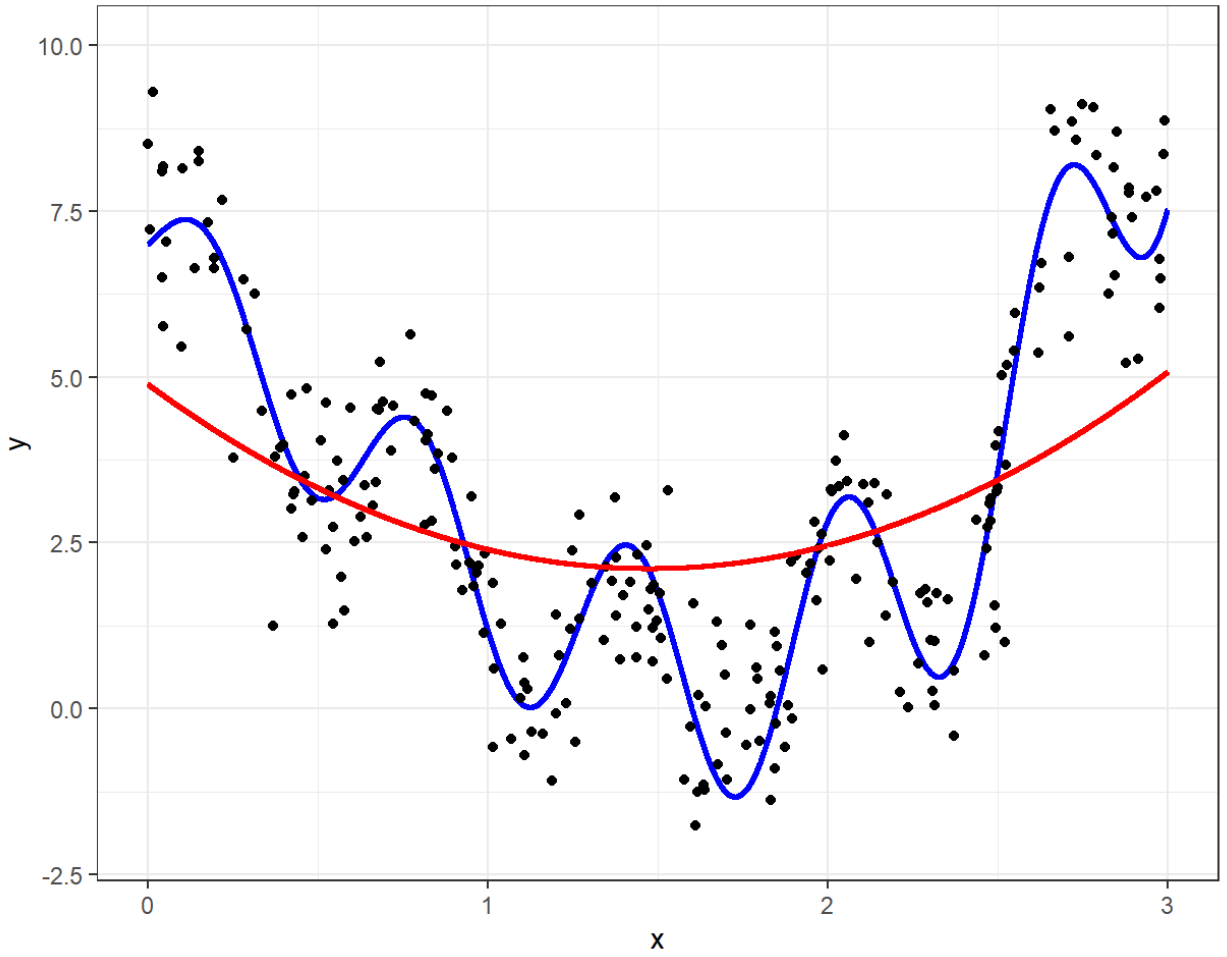
$(x, \eta(x))$

$(x, \hat{\phi}(x))$

With $\sigma = 10$ and $\lambda = 10$

Ridge regression with random feature mappings in R

Returning to our example non-linear regression problem.



(X_i, Y_i)

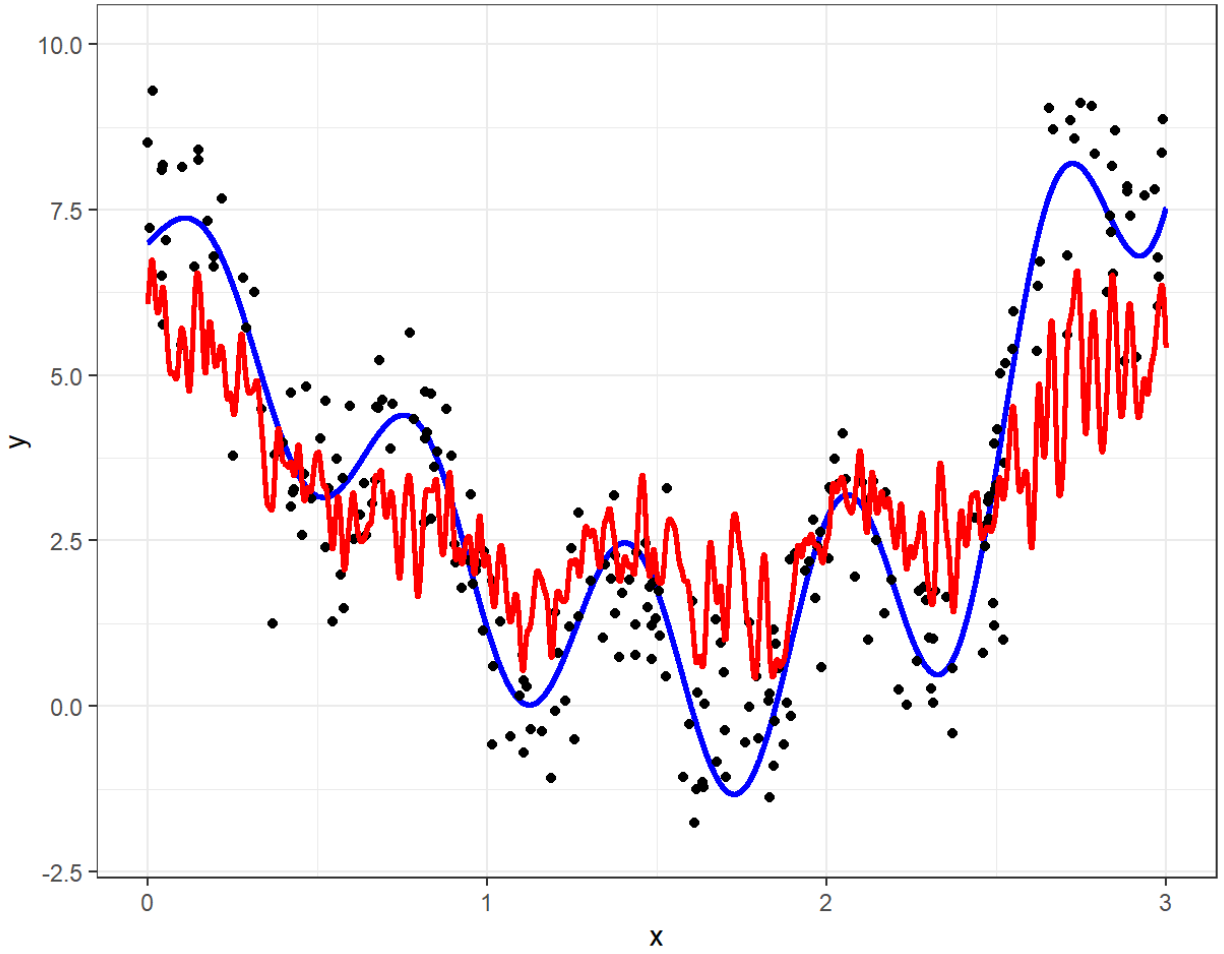
$(x, \eta(x))$

$(x, \hat{\phi}(x))$

With $\sigma = 0.01$ and $\lambda = 10$

Ridge regression with random feature mappings in R

Returning to our example non-linear regression problem.



(X_i, Y_i)

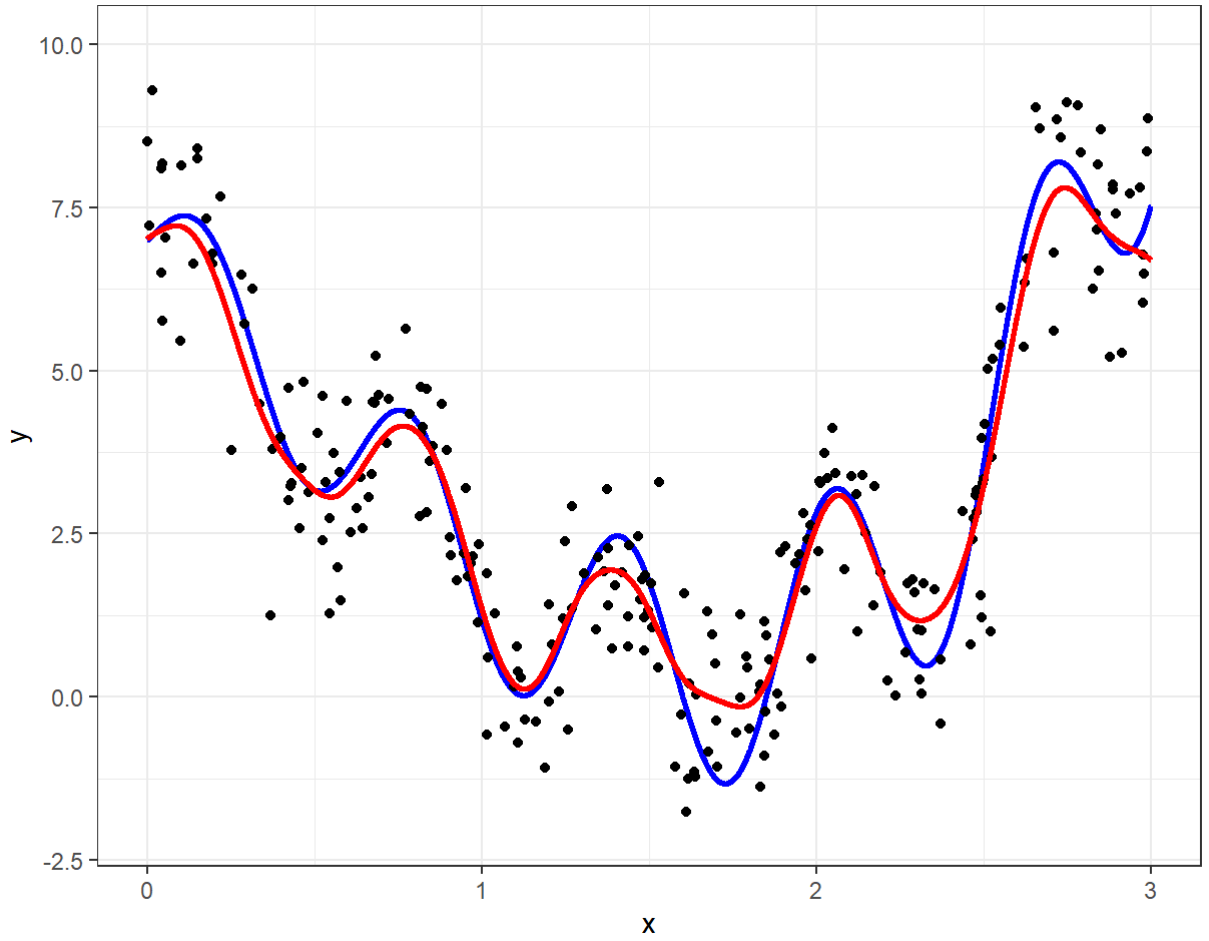
$(x, \eta(x))$

$(x, \hat{\phi}(x))$

With $\sigma = 100$ and $\lambda = 10$

Ridge regression with random feature mappings in R

Returning to our example non-linear regression problem.



(X_i, Y_i)

$(x, \eta(x))$

$(x, \hat{\phi}(x))$

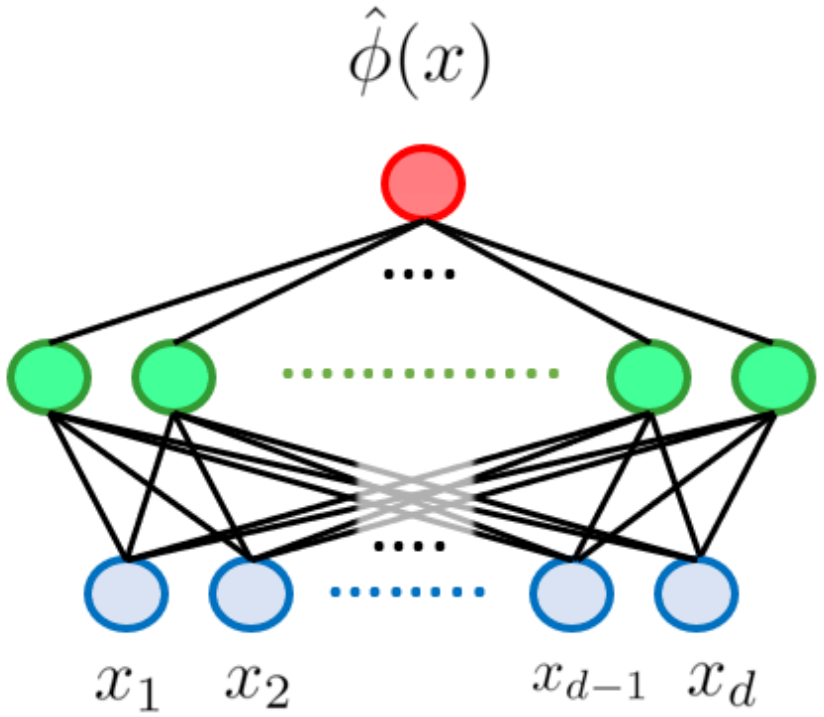
With $\sigma = 10$ and $\lambda = 10$

Now take a break!

Statistical Computing & Empirical Methods

Non-linear methods

We can create non-linear regression models $\hat{\phi} : \mathcal{X} \rightarrow \mathbb{R}$ can be constructed by combining linear mappings with simple non-linear functions.



$$\hat{\phi}(x) = z(x)\, w^\top + w_0$$

$$z(x) = \psi\left(xA^\top + b\right)$$

x

How do we choose these non-linear feature transformations?

Neural networks

Approach 2 Jointly learn a transformation of the feature space and a regression model.

Neural networks

Approach 2 Jointly learn a transformation of the feature space and a regression model.

Our goal is to find a mapping $\phi : \mathcal{X} \to \mathbb{R}$ with small test error,

$$\mathcal{R}_{\mathrm{MSE}}(\phi) := \mathbb{E}[(\phi(X) - Y)^2]$$

We consider neural networks of the form $\phi(x) = \psi\left(x\ A^\top + b\right)\ w^\top + w_0$.

Given a data sample $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ with $(X_i, Y_i) \sim \mathrm{P}$ we minimise

$$\hat{\mathcal{R}}_{\mathrm{MSE}}(\phi) = \frac{1}{n} \sum_{i=1}^{n} (\phi(X_i) - Y_i)^2$$

Neural networks

Our goal is to find a mapping $\phi : \mathcal{X} \to \mathbb{R}$ with small test error,

$$\mathcal{R}_{\mathrm{MSE}}(\phi) := \mathbb{E}[(\phi(X) - Y)^2]$$

We consider neural networks of the form $\phi(x) = \psi\left(x\ A^\top + b\right)\ w^\top + w_0$.

Given a data sample $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ with $(X_i, Y_i) \sim \mathrm{P}$ we minimise

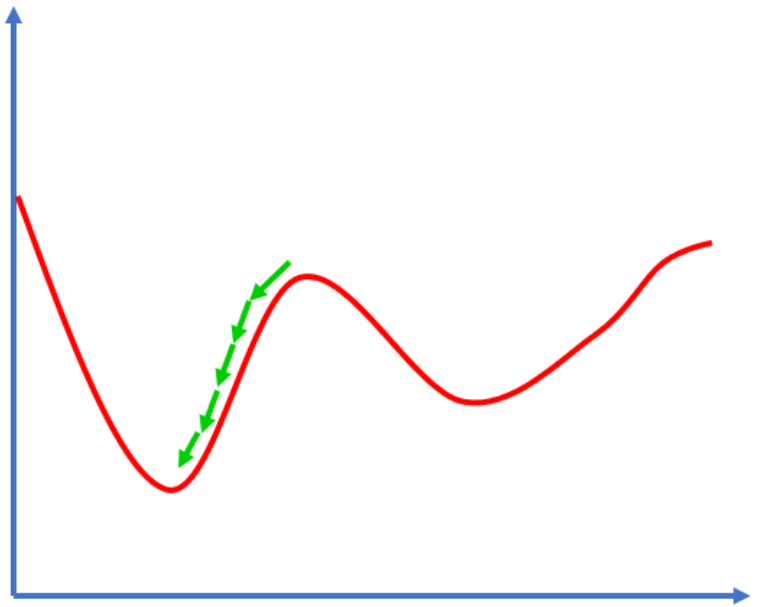
$$\hat{\mathcal{R}}_{\mathrm{MSE}}(\phi) = \frac{1}{n}\sum_{i=1}^{n}\left(\phi(X_i) - Y_i\right)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left\{\left(\psi\left(X_i\ A^\top + b\right)\ w^\top + w_0\right) - Y_i\right\}^2 .$$

Neural networks

Given a data sample $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ we attempt to minimise

$$\hat{\mathcal{R}}_{\text{MSE}}(\phi) = \frac{1}{n} \sum_{i=1}^{n} \left\{ \left(\psi \left(X_i \; A^\top + b \right) \; w^\top + w_0 \right) - Y_i \right\}^2 .$$



This is done iteratively through gradient descent.

Randomly initialize the parameters θ .

For $t = 1, \cdots, T$

$$\theta_{t+1} \mapsto \theta_t - \alpha \cdot \frac{\partial}{\partial \theta} \left\{ \hat{\mathcal{R}}_{\text{MSE}}(\phi) \right\} \Big|_{\theta_t}$$

Neural networks in R

We can create a simple neural network in R with the “nnet” library.

```
library(nnet) # load library
```

Set a random seed for reproducibility.

```
set.seed(123) # set random seed
```

Specify a maximum number of iterations.

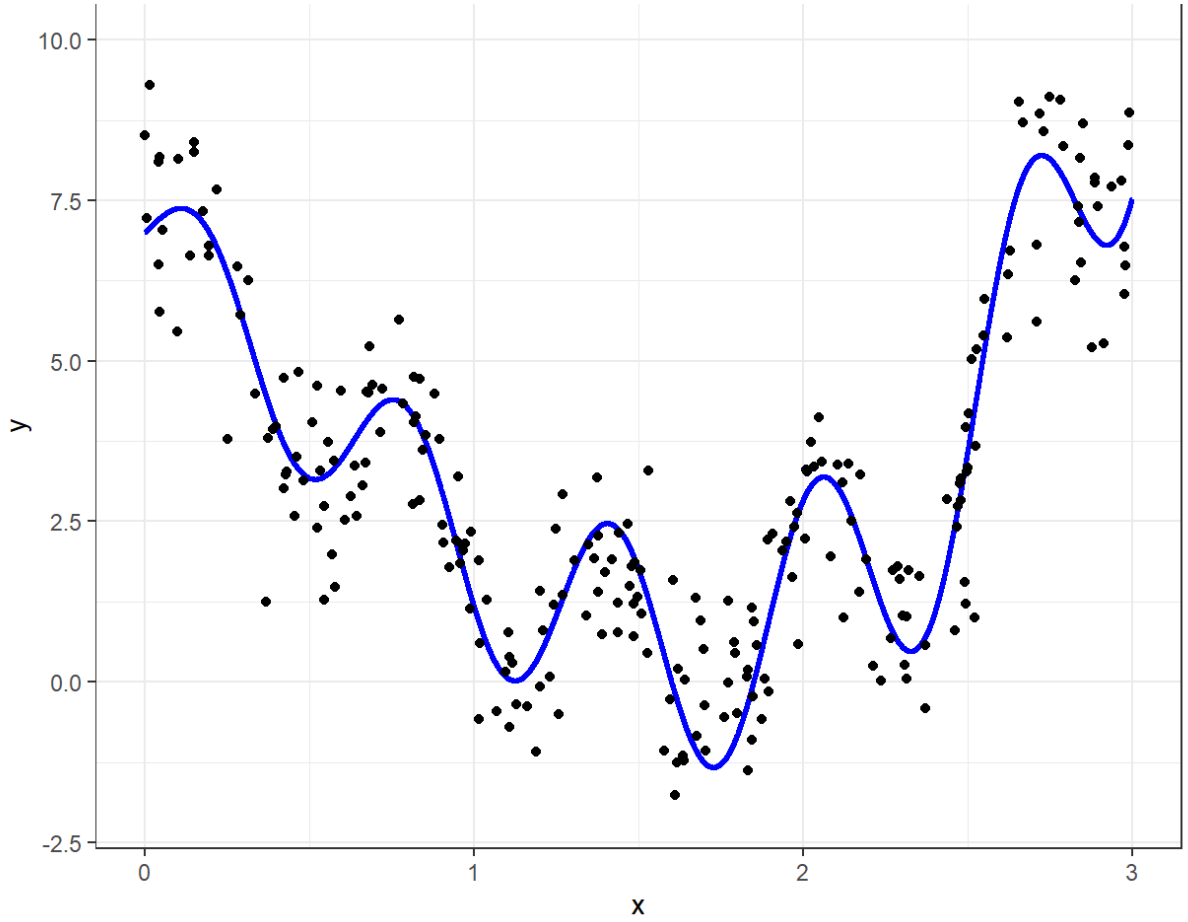
```
max_iterations<-10000 # maximum number of iterations
```

Train the neural network.

```
nnet_model<-nnet(train_x, train_y,size=500,
                 MaxNWts=2000,linout=TRUE,maxit=max_iterations) # train nnet
```

Neural networks

Returning to our example non-linear regression problem.

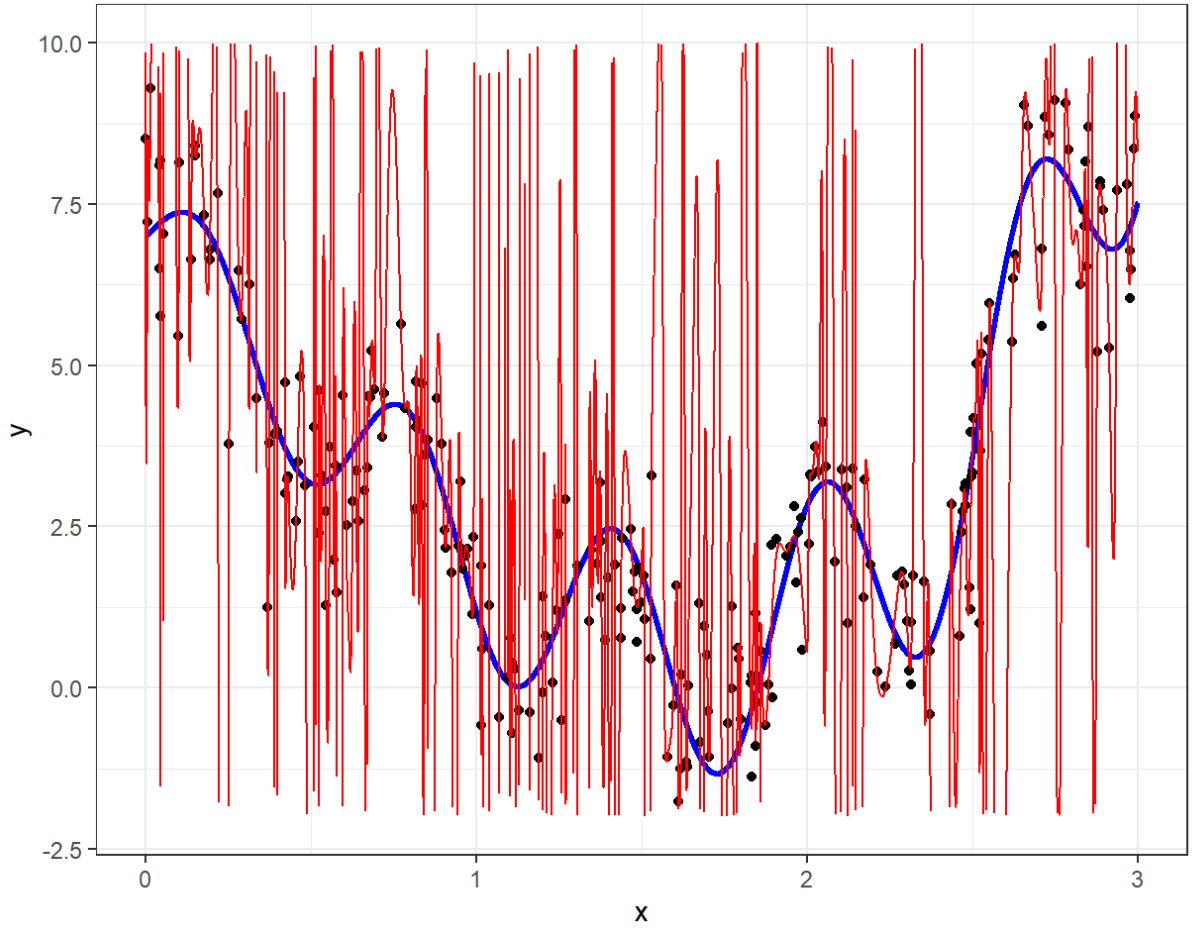


- (X_i, Y_i)
- $(x, \eta(x))$

Here $\eta(x) = \mathbb{E}(Y|X = x)$ is the optimal predictive function.

Neural networks

Minimising the mean squared error on training data leads to a high variance solution.



(X_i, Y_i)

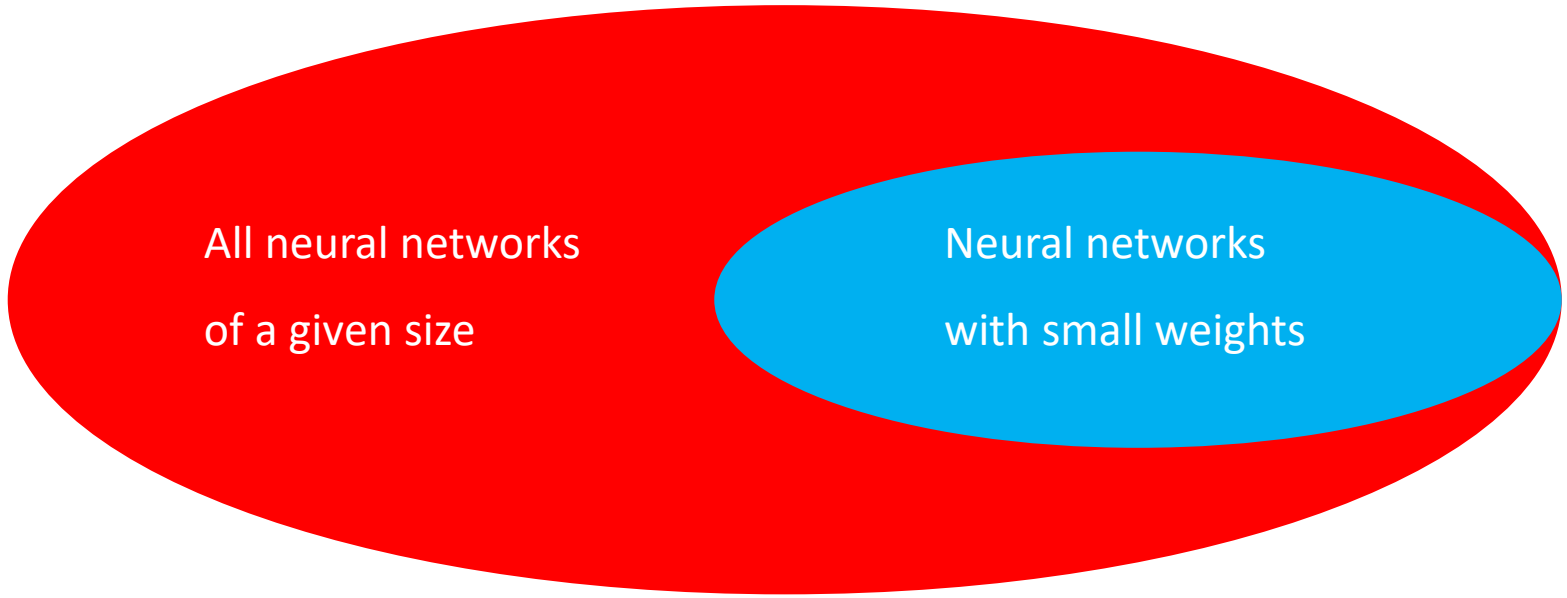
$(x, \eta(x))$

$(x, \hat{\phi}(x))$

Neural networks with weight decay

The source of the instability is a search over a large space of all neural networks of a given size.

Idea: To reduce the instability of our estimate we limit our search space to neural networks with “small” weights.



We focus on the space of neural networks with weights of low norm.

Neural networks with weight decay

Given a data sample $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ we minimize the regularized loss

$$\hat{\mathcal{R}}^{\lambda}_{\mathrm{MSE}}(\phi) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\left\{\left(\psi\left(X_i\, A^{\top} + b\right)\, w^{\top} + w_0\right) - Y_i\right\}^2}_{\text{Data fitting term}} + \lambda \cdot \underbrace{\left\{\|A\|_{\mathrm{F}}^2 + \|w\|_2^2\right\}}_{\text{Regularisation term}}.$$

where $\|A\|_{\mathrm{F}}^2 = \sum_{r=1}^{q}\sum_{s=1}^{d}(a_{rs})^2$ and $\|w\|_2^2 = \sum_{r=1}^{q}(w_r)^2$.

Minimising the regularised loss is known as ℓ_2 - **regularization** or **weight decay**

Neural networks with weight decay

Given a data sample $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ we minimize the regularized loss

$$\hat{\mathcal{R}}^{\lambda}_{\mathrm{MSE}}(\phi) = \frac{1}{n}\sum_{i=1}^{n}\left\{\left(\psi\left(X_i\, A^{\top} + b\right)\, w^{\top} + w_0\right) - Y_i\right\}^2 + \lambda \cdot \left\{\|A\|_{\mathrm{F}}^2 + \|w\|_2^2\right\}.$$

Let's set the weight decay and maximum iterations.

```
max_iterations<-10000 # maximum number of iterations
weight_decay<-0.01 # set weight decay
```

Train the neural network to mininimise the regularized loss function.

```
set.seed(123) # set random seed
nnet_model<-nnet(train_x, train_y,size=500,decay=weight_decay,
                 MaxNWts=2000,linout=TRUE,maxit=max_iterations) # train nnet
```

Neural networks with weight decay

Let’s set the weight decay and maximum iterations.

```
max_iterations<-10000 # maximum number of iterations
weight_decay<-0.01 # set weight decay
```

Train the neural network to mininimise the regularized loss function.

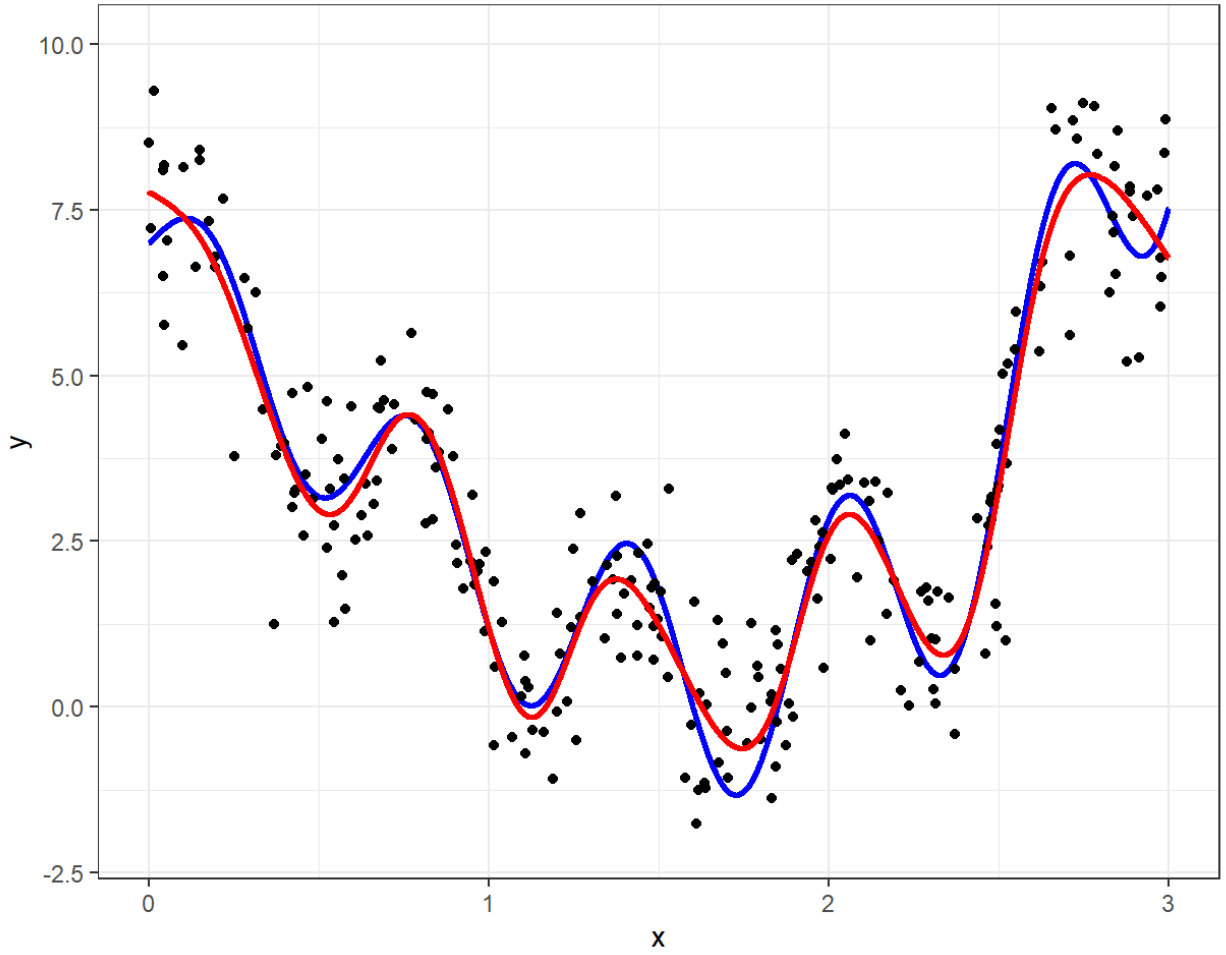
```
set.seed(123) # set random seed
nnet_model<-nnet(train_x, train_y,size=500,decay=weight_decay,
                 MaxNWts=2000,linout=TRUE,maxit=max_iterations) # train nnet
```

We can use our trained model to make predictions and estimate the validation error.

```
val_predictions<-predict(nnet_model,newdata=val_x) # validation predictions
val_msq_error<-mean((val_predictions-val_y)^2)  # validation error
```

Neural networks with weight decay

By minimizing the regularized loss function we achieve a better fit to unseen data.



(X_i, Y_i)

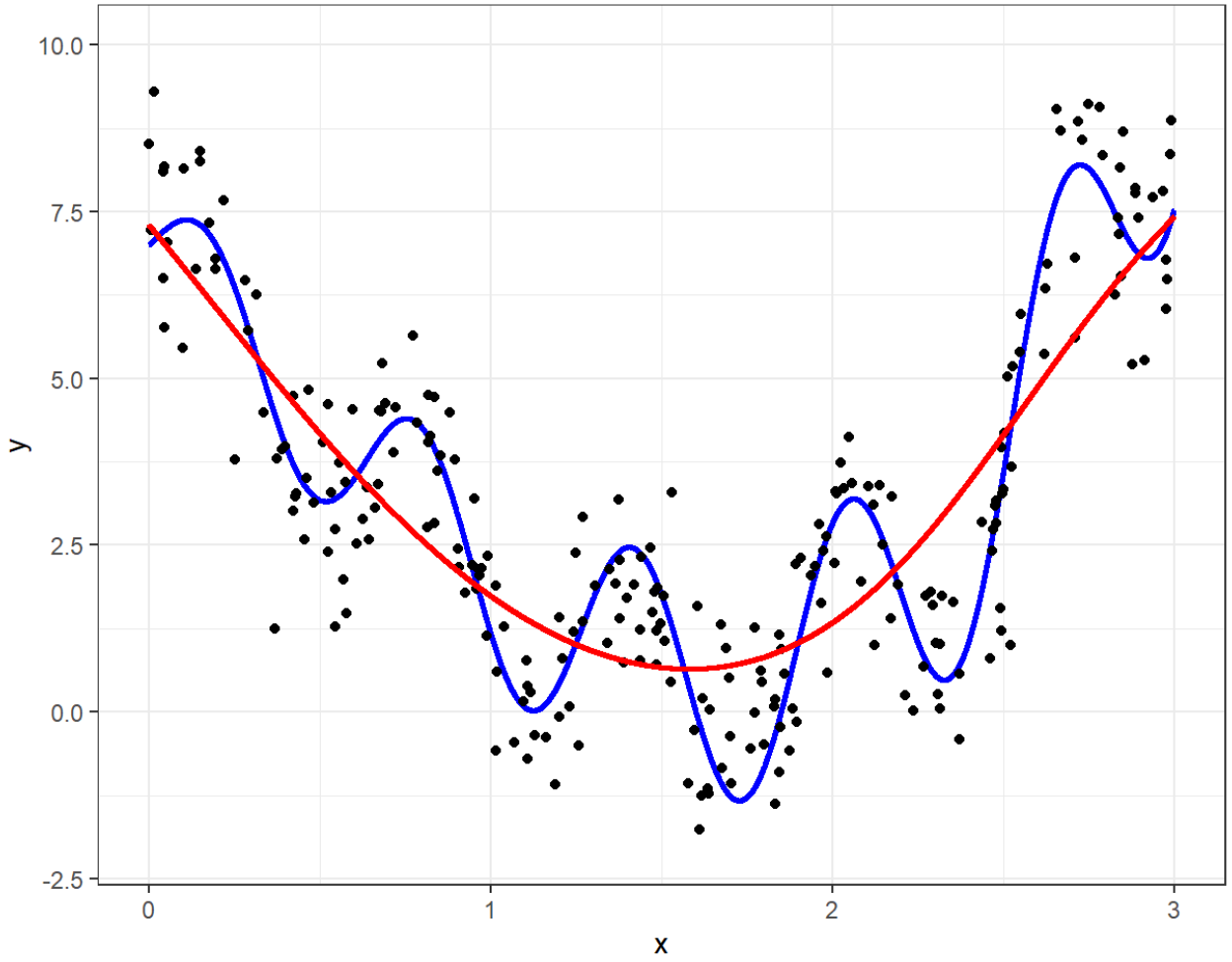
$(x, \eta(x))$

$(x, \hat{\phi}(x))$

Weight decay of $\lambda = 0.01$

Neural networks with weight decay

Too much weight decay leads to underfitting and gives a poor fit to both train and test data.



(X_i, Y_i)

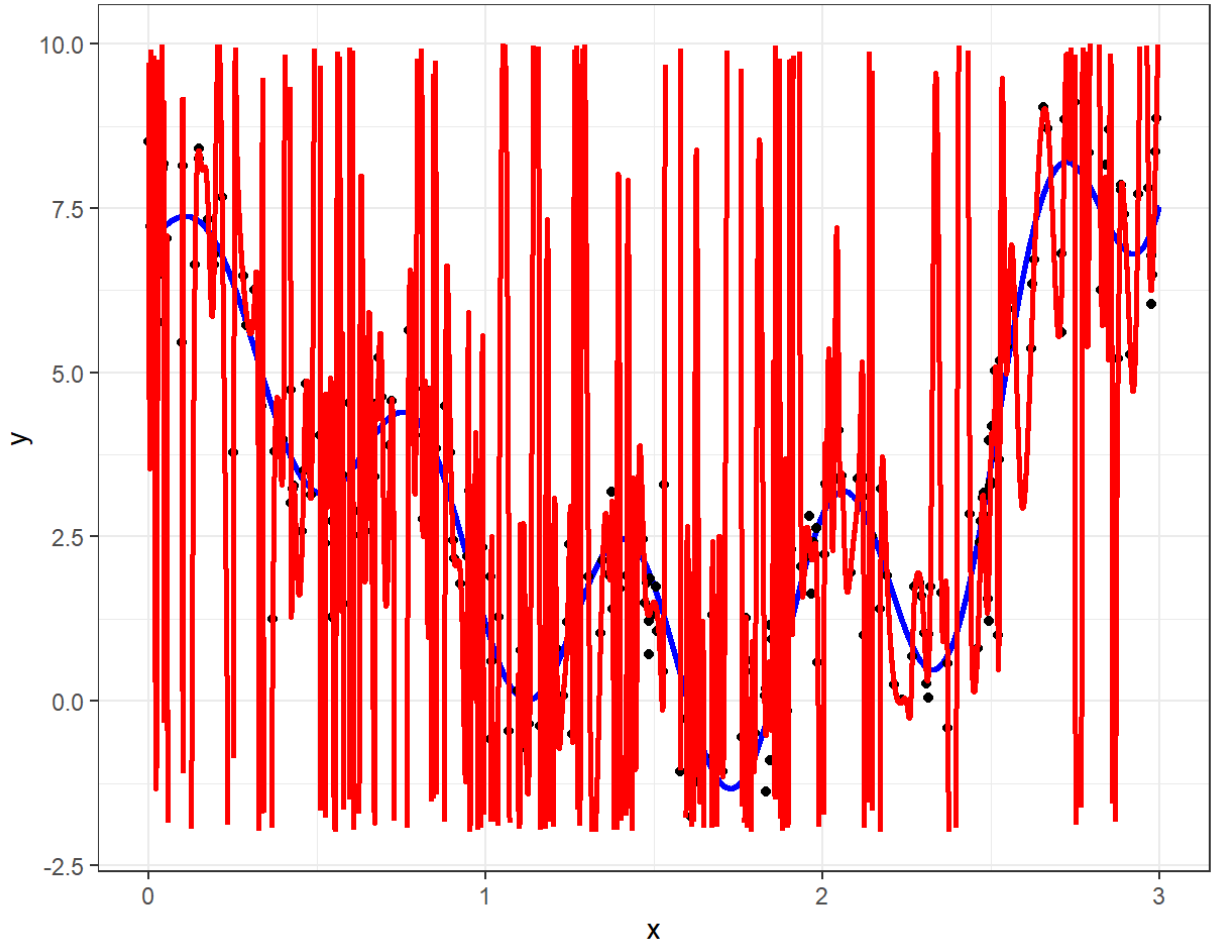
$(x, \eta(x))$

$(x, \hat{\phi}(x))$

Weight decay of $\lambda = 1$

Neural networks with weight decay

Too little weight decay leads to very low train error but high test error.



(X_i, Y_i)

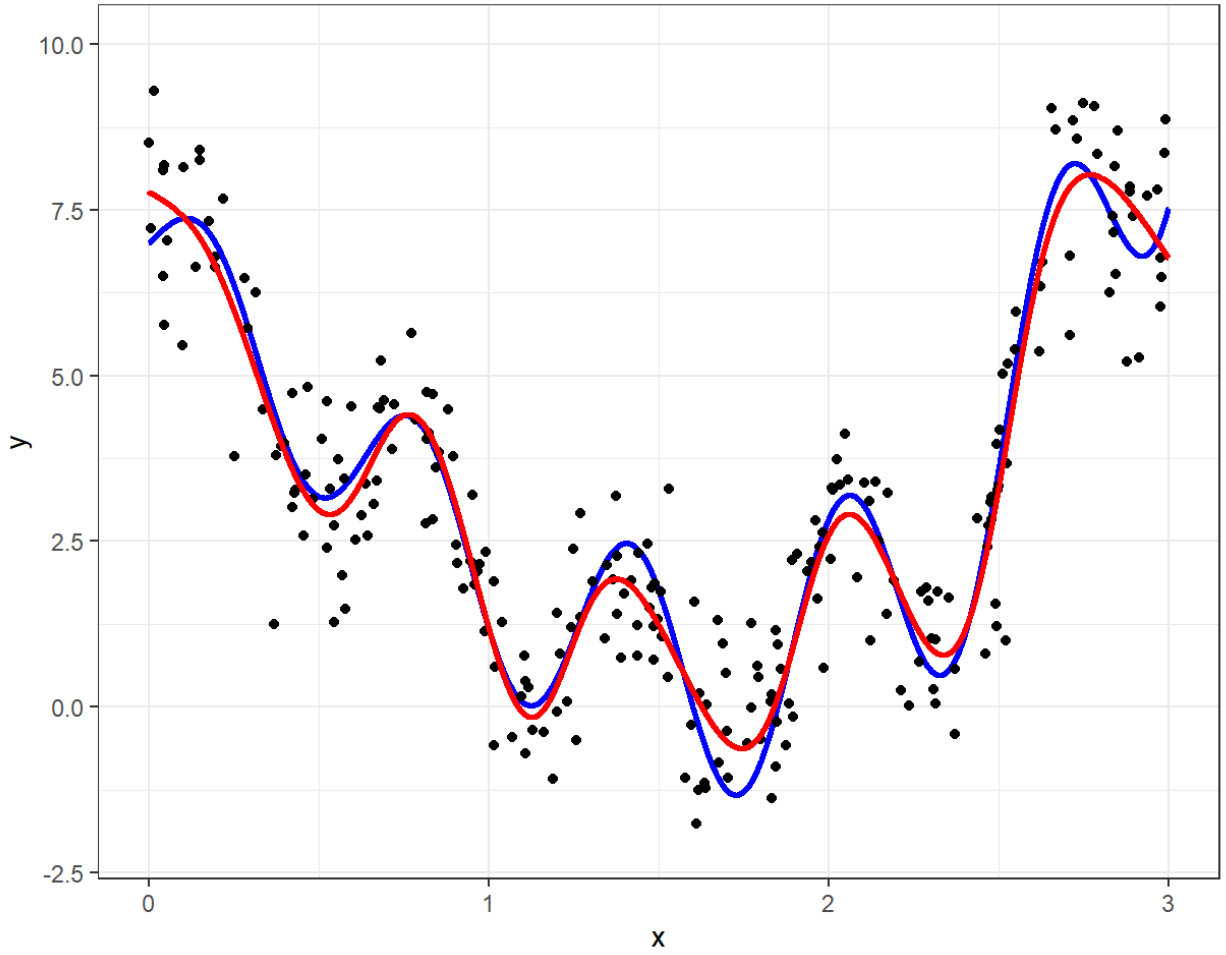
$(x, \eta(x))$

$(x, \hat{\phi}(x))$

Weight decay of $\lambda = 10^{-8}$

Neural networks with weight decay

By minimizing the regularized loss function we achieve a better fit to unseen data.



(X_i, Y_i)

$(x, \eta(x))$

$(x, \hat{\phi}(x))$

Weight decay of $\lambda = 0.01$

Now take a break!

Statistical Computing & Empirical Methods

A non-parametric approach

Approach 3 Apply a non-parametric technique based on the proximity of input feature vectors to feature vectors in the training data.

The k-nearest neighbour method

Approach 3 Apply a non-parametric technique based on the proximity of input feature vectors to feature vectors in the training data.

Our goal is to find a mapping $\phi : \mathcal{X} \to \mathbb{R}$ with small test error,

$$\mathcal{R}_{\mathrm{MSE}}(\phi) := \mathbb{E}[(\phi(X) - Y)^2]$$

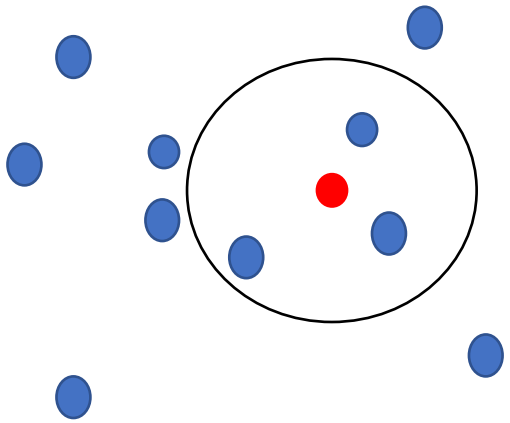
The optimal solution is $\eta : \mathcal{X} \to \mathbb{R}$ defined by $\eta(x) = \mathbb{E}(Y|X = x)$.

IDEA: If $x_0 \in \mathcal{X}$ is close to $x_1 \in \mathcal{X}$ then we expect $\eta(x_0)$ to be close to $\eta(x_1)$.

The k-nearest neighbour method

The optimal solution is $\eta : \mathcal{X} \to \mathbb{R}$ defined by $\eta(x) = \mathbb{E}(Y|X = x)$.

IDEA: If $x_0,\ x_1 \in \mathcal{X}$ are close then we expect $\eta(x_0)$ to be close to $\eta(x_1)$.



$$\hat{\phi}_k(x) = \frac{1}{k}\sum_{j=1}^{k} Y_{\tau_j(x)}$$

where $X_{\tau_1(x)}, \cdots, X_{\tau_k(x)}$

are the k –nearest neighbours of x.

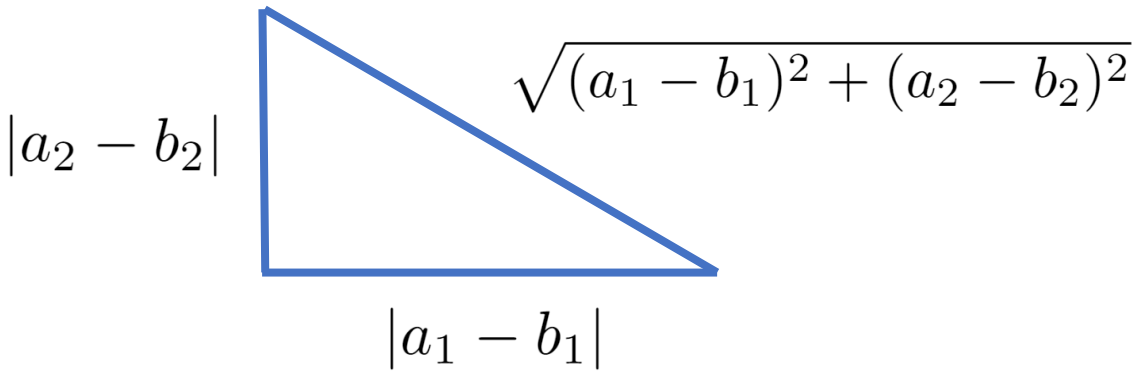
The k-nearest neighbour method

The optimal solution is $\eta : \mathcal{X} \to \mathbb{R}$ defined by $\eta(x) = \mathbb{E}(Y|X = x)$.

IDEA: If $x_0,\ x_1 \in \mathcal{X}$ are close then we expect $\eta(x_0)$ to be close to $\eta(x_1)$.

Given vectors $a = (a_1, \cdots, a_d) \in \mathbb{R}^d$ and $b = (b_1, \cdots, b_d) \in \mathbb{R}^d$ the

Euclidean distance is $\|a - b\|_2 = \sqrt{(a_1 - b_1)^2 + \cdots + (a_d - b_d)^2}$.



The k-nearest neighbour method

IDEA: If $x_0,\ x_1 \in \mathcal{X}$ are close then we expect $\eta(x_0)$ to be close to $\eta(x_1)$.

Suppose we have a data set $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ with (X_i, Y_i) in $\mathbb{R}^d \times \mathbb{R}$.

The k-nearest neighbour method

IDEA: If $x_0,\ x_1 \in \mathcal{X}$ are close then we expect $\eta(x_0)$ to be close to $\eta(x_1)$.

Suppose we have a data set $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ with (X_i, Y_i) in $\mathbb{R}^d \times \mathbb{R}$.

Fix $k \in \mathbb{N}$.

The k-nearest neighbour method

IDEA: If $x_0,\ x_1 \in \mathcal{X}$ are close then we expect $\eta(x_0)$ to be close to $\eta(x_1)$.

Suppose we have a data set $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ with (X_i, Y_i) in $\mathbb{R}^d \times \mathbb{R}$.

Fix $k \in \mathbb{N}$. Given a test point $x \in \mathbb{R}^d$,

We take a permutation (a reordering) $\tau_1(x), \cdots, \tau_n(x)$ of the set $\{1, \cdots, n\}$ with

$$\|x - X_{\tau_1(x)}\|_2 \leq \|x - X_{\tau_1(x)}\|_2 \leq \cdots \leq \|x - X_{\tau_n(x)}\|_2.$$

The k-nearest neighbour method

IDEA: If $x_0,\ x_1 \in \mathcal{X}$ are close then we expect $\eta(x_0)$ to be close to $\eta(x_1)$.

Suppose we have a data set $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ with (X_i, Y_i) in $\mathbb{R}^d \times \mathbb{R}$.

Fix $k \in \mathbb{N}$. Given a test point $x \in \mathbb{R}^d$,

We take a permutation (a reordering) $\tau_1(x), \cdots, \tau_n(x)$ of the set $\{1, \cdots, n\}$ with

$$\|x - X_{\tau_1(x)}\|_2 \leq \|x - X_{\tau_1(x)}\|_2 \leq \cdots \leq \|x - X_{\tau_n(x)}\|_2.$$

Set $\hat{\phi}_k(x) = \frac{1}{k}\sum_{j=1}^{k} Y_{\tau_j(x)}$ where $X_{\tau_1(x)}, \cdots, X_{\tau_k(x)}$ are the k-nearest neighbours.

The output $\hat{\phi}_k(x)$ is the average over the k-nearest neighbours

The k-nearest neighbour method in R

We can carry out the k-nearest neighbor method in R with “knn” library.

```
library(kknn) # load knn library
```

The k-nearest neighbour method in R

We can carry out the k-nearest neighbor method in R with “knn” library.

```
library(kknn) # load knn library
```

Let’s choose the number of nearest neighbours “k”.

```
k<-15 # set number of neighbours
```

The k-nearest neighbour method in R

We can carry out the k-nearest neighbor method in R with “knn” library.

```
library(kknn) # load knn library
```

Let’s choose the number of nearest neighbours “k”.

```
k<-15 # set number of neighbours
```

We can then train our k-nearest neighbour model.

```
knn_model<-train.kknn(y~.,data=train_data, ks = k, kernel = "rectangular")
```

The k-nearest neighbour method in R

We can carry out the k-nearest neighbor method in R with “knn” library.

```
library(kknn) # load knn library
```

Let’s choose the number of nearest neighbours “k”.

```
k<-15 # set number of neighbours
```

We can then train our k-nearest neighbour model.

```
knn_model<-train.kknn(y~.,data=train_data, ks = k, kernel = "rectangular")
```

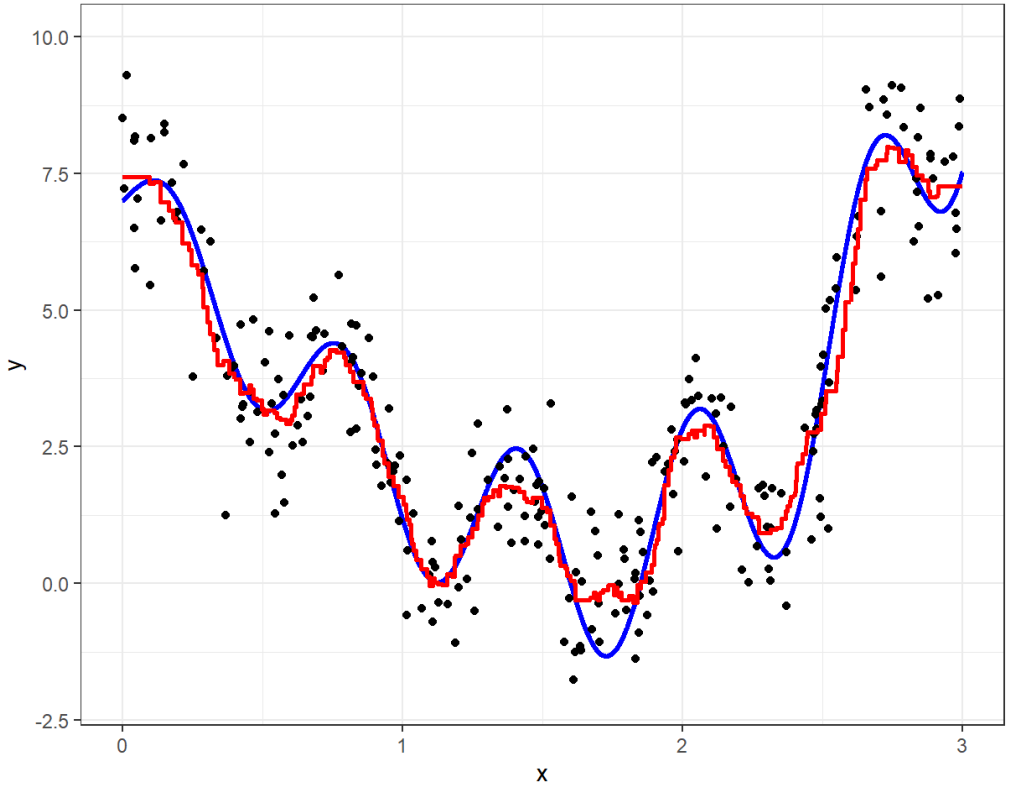
We can now make predictions and estimate our performance.

```
val_predictions<-predict(knn_model,val_x%>%data.frame()) # validation predictions
val_msq_error<-mean((val_predictions-val_y)^2)  # validation error
```

The k nearest neighbor method

The k-nearest neighbour method gives a flexible approach to non-linear regression.

The approach is non-parametric meaning the complexity of the model grows with the sample size.



(X_i, Y_i)

$(x, \eta(x))$

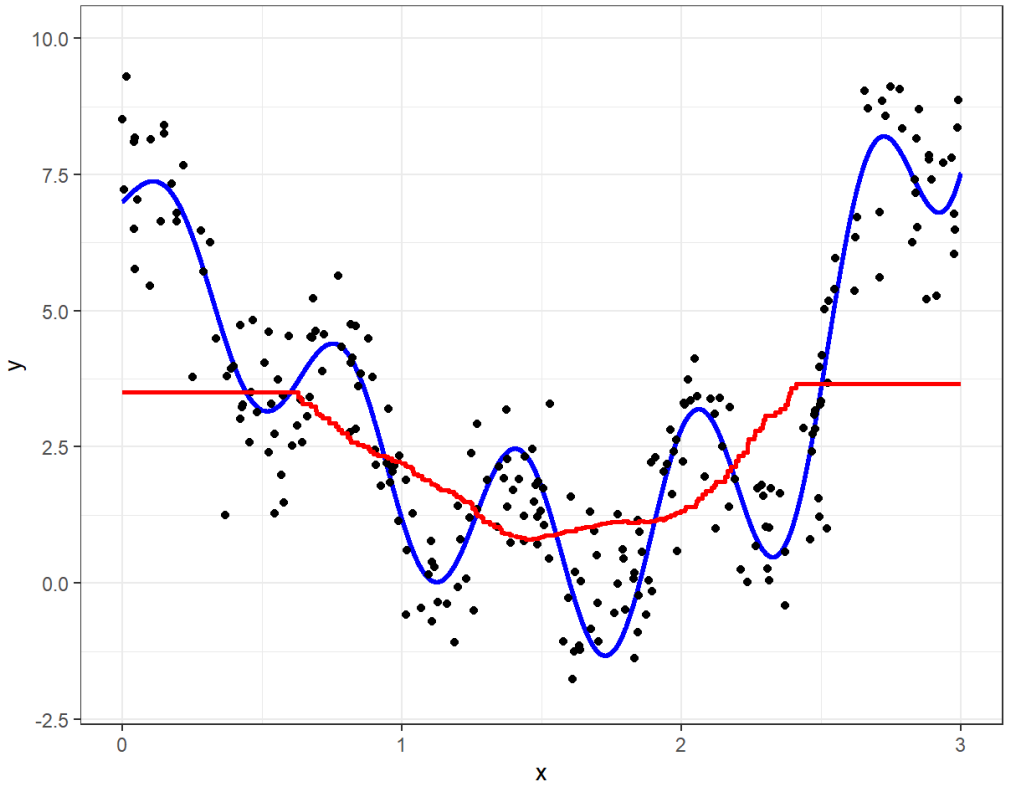
$(x, \hat{\phi}(x))$

Number of neighbours $k = 15$

The k nearest neighbor method

If the number of neighbours k is too large this method will under fit.

The method will have a very high bias and perform poorly on both train and test data.



(X_i, Y_i)

$(x, \eta(x))$

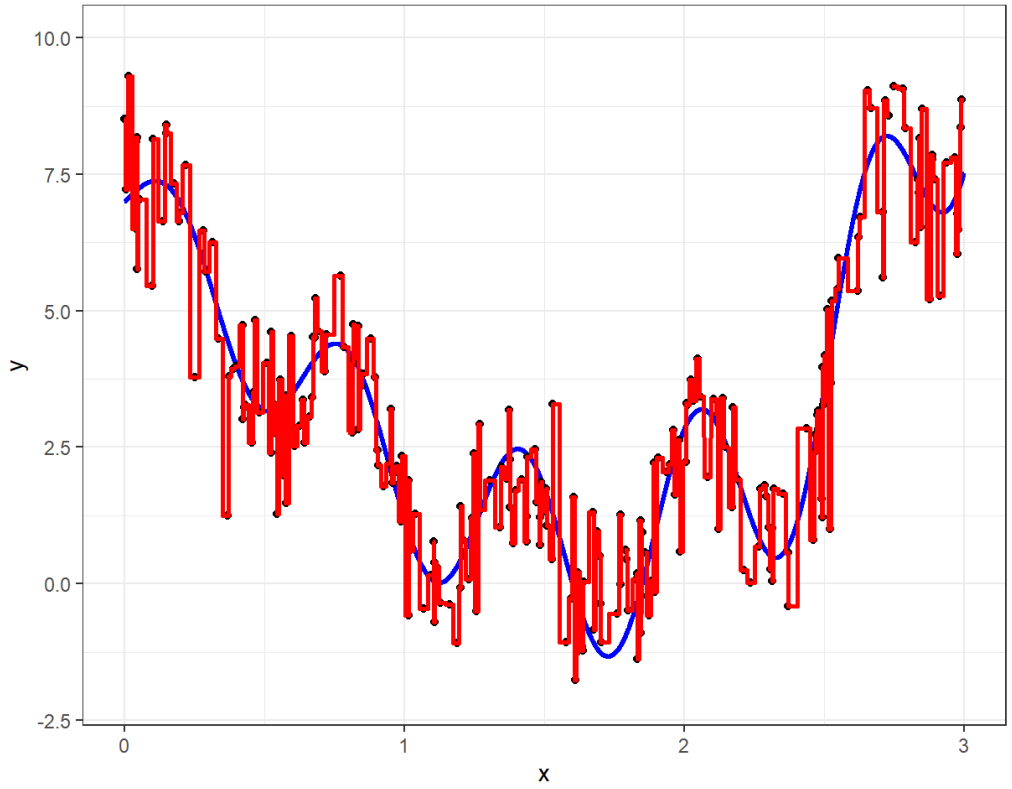
$(x, \hat{\phi}(x))$

Number of neighbours $k = 100.$

The k nearest neighbor method

If the number of neighbours k is too small this method will over fit.

The method will give a low train error but a very high variance with high test error.



(X_i, Y_i)

$(x, \eta(x))$

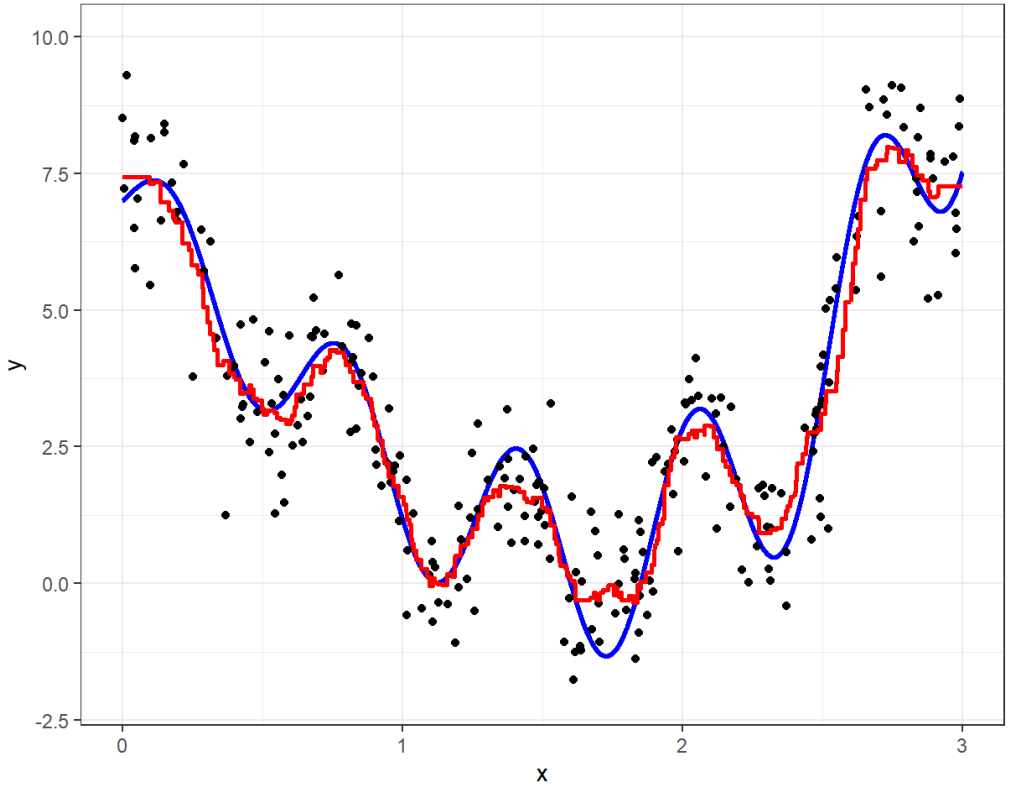
$(x, \hat{\phi}(x))$

Number of neighbours $k = 1$

The k nearest neighbor method

The k-nearest neighbour method gives a flexible approach to non-linear regression.

An optimal number of neighbours k will carefully balance bias and variance.



(X_i, Y_i)

$(x, \eta(x))$

$(x, \hat{\phi}(x))$

Number of neighbours $k = 15$

What have we covered today?

- We will investigate a small selection of different approaches for non-linear regression.
- We will begin with an approach for applying linear methods to non-linear problems with random non-linear feature mappings.
  - Hyperparameters include the standard deviation of random features and norm penalty.
- We will then consider the neural network approach where the feature mapping is learnt from the data.
  - Hyperparameters include the weight decay as well as the structure of the network.
- We will also talk about the k-nearest neighbour method – a flexible distance-based approach.
  - Hyperparameters include the number of neighbours and the distance metric.
- Each of these approaches can be applied to classification as well as regression.

University of BRISTOL

Thanks for listening!

henry.reeve@bristol.ac.uk

Include EMATM0061 in the subject of your email.

Statistical Computing & Empirical Methods

Non-linear feature mappings

Let's suppose we want to learn a regression model $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$.

First choose a non-linear a function $\sigma : \mathbb{R} \to \mathbb{R}$.

Take a weight matrix $A \in \mathbb{R}^{q \times d}$ and a bias vector $b = (b_1, \cdots, b_q) \in \hat{\phi}(x)$

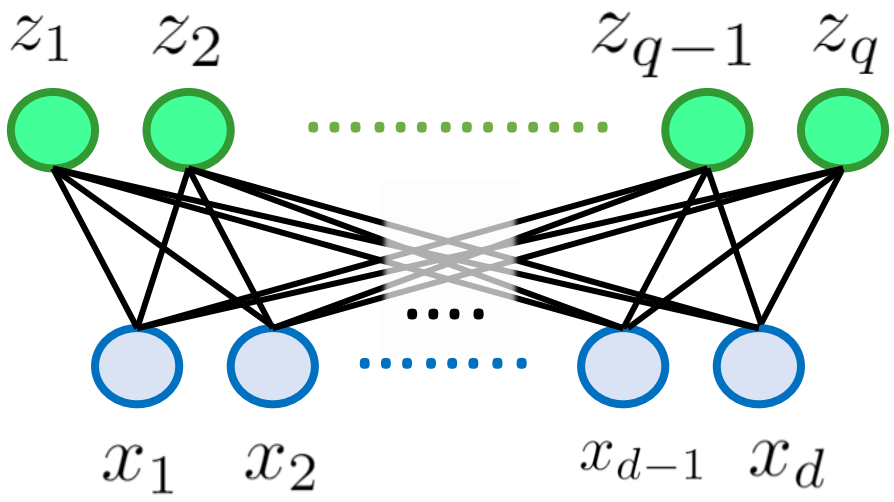
Non-linear feature mappings

$$z_1(x) = \sigma\left(a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots + a_{1d} \cdot x_d + b_1\right)$$

$$z_2(x) = \sigma\left(a_{21} \cdot x_1 + a_{22} \cdot x_2 + \cdots + a_{2d} \cdot x_d + b_1\right)$$

$$\vdots$$

$$z_q(x) = \sigma\left(a_{q1} \cdot x_1 + a_{q2} \cdot x_2 + \cdots + a_{qd} \cdot x_d + b_q\right).$$



Neural networks