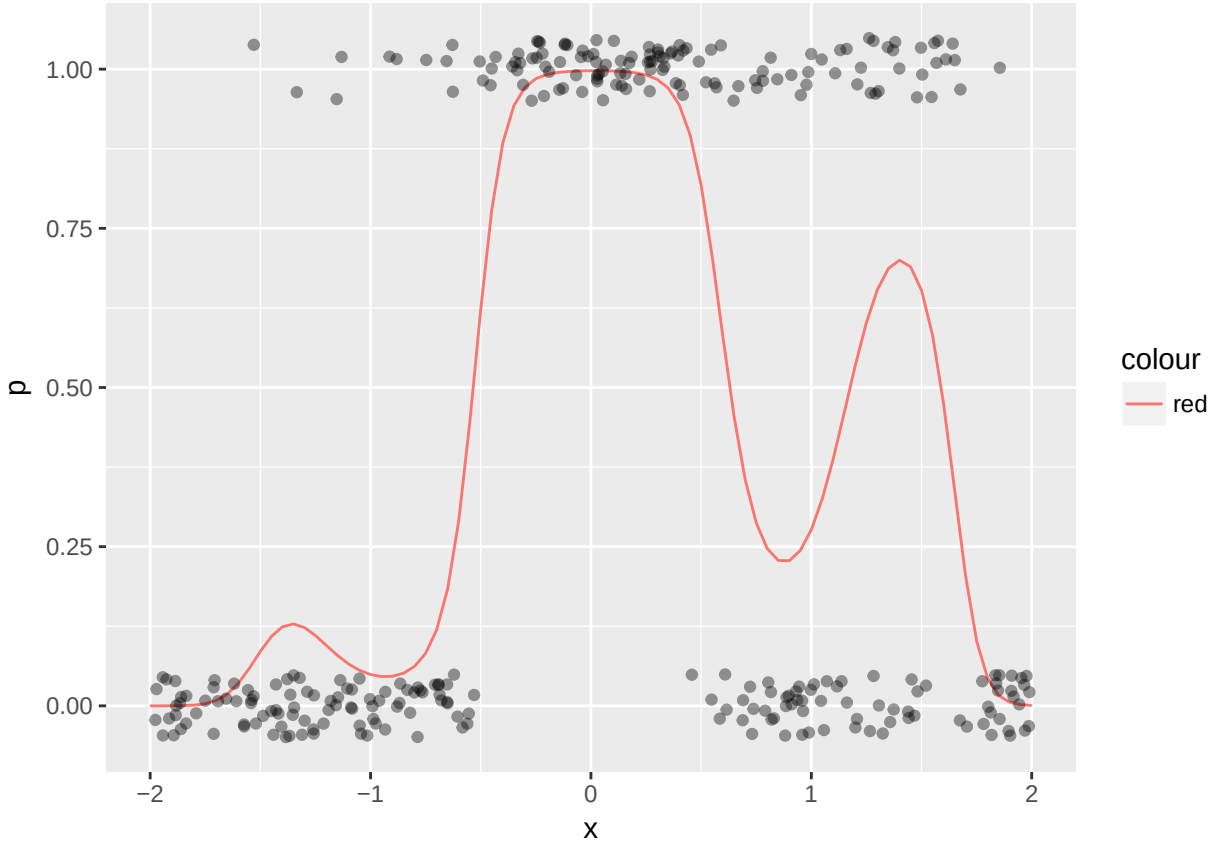
Redes neuronales

```
library(ggplot2)
h <- function(x){
    exp(x)/(1+exp(x))
}
x <- seq(-2,2,0.05)
p <- h(3 + x- 3*x^2 + 3*cos(4*x))
set.seed(280572)
x.2 <- runif(300, -2, 2)
g.2 <- rbinom(300, 1, h(3 + x.2- 3*x.2^2 + 3*cos(4*x.2)))
datos <- data.frame(x.2,g.2)
dat.p <- data.frame(x,p)
g <- qplot(x,p, geom='line', col='red')
g + geom_jitter(data = datos, aes(x=x.2,y=g.2), col ='black',
  position =position_jitter(height=0.05), alpha=0.4)
```




```
library(nnet)
set.seed(12)
nn <- nnet(g.2 ~ x.2, data=datos, size = 4, decay=0.0, entropy = T)
```

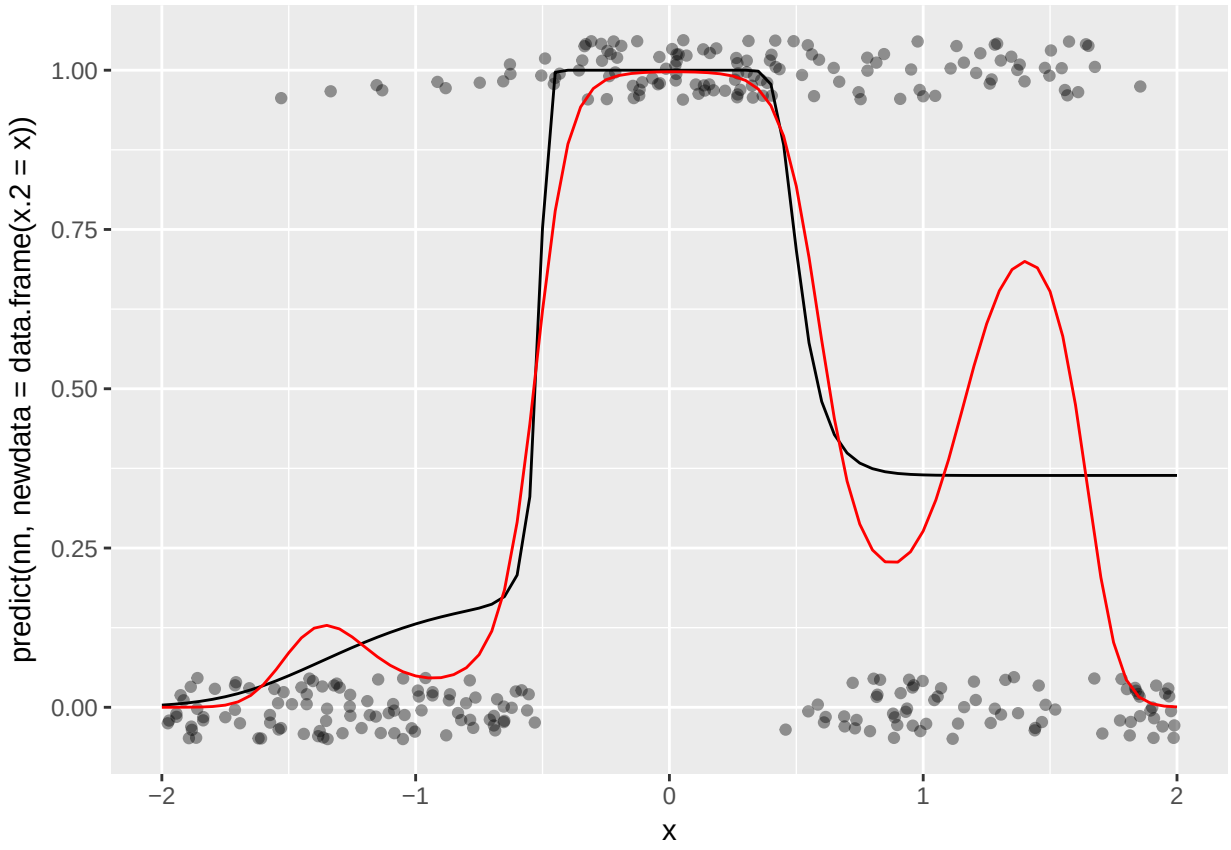
```
## # weights:  13
## initial  value 225.478422
## iter  10 value 166.155642
## iter  20 value 127.399514
## iter  30 value 124.561292
```

```
## iter  40 value 123.891360
## iter  50 value 122.886843
## iter  60 value 120.523224
## iter  70 value 116.443713
## iter  80 value 113.083930
## iter  90 value 112.182966
## iter 100 value 110.788798
## final  value 110.788798
## stopped after 100 iterations
```

```
nn$wts
```

```
##  [1]  -9.011126 -22.921790 -13.264391   1.720134  -2.759160  11.632168
##  [7]   6.093405   2.884352  22.844622 -34.249506   1.636633 -33.298600
## [13]   9.895719
```

```
w <- qplot(x, predict(nn, newdata=data.frame(x.2 = x)), geom='line')
w + geom_jitter(data = datos, aes(x=x.2,y=g.2), col ='black',
  position =position_jitter(height=0.05), alpha=0.4)+geom_line(data=dat.p,aes(x=x,y=p),col="red")
```



Vamos a intentar con el método de optimización:

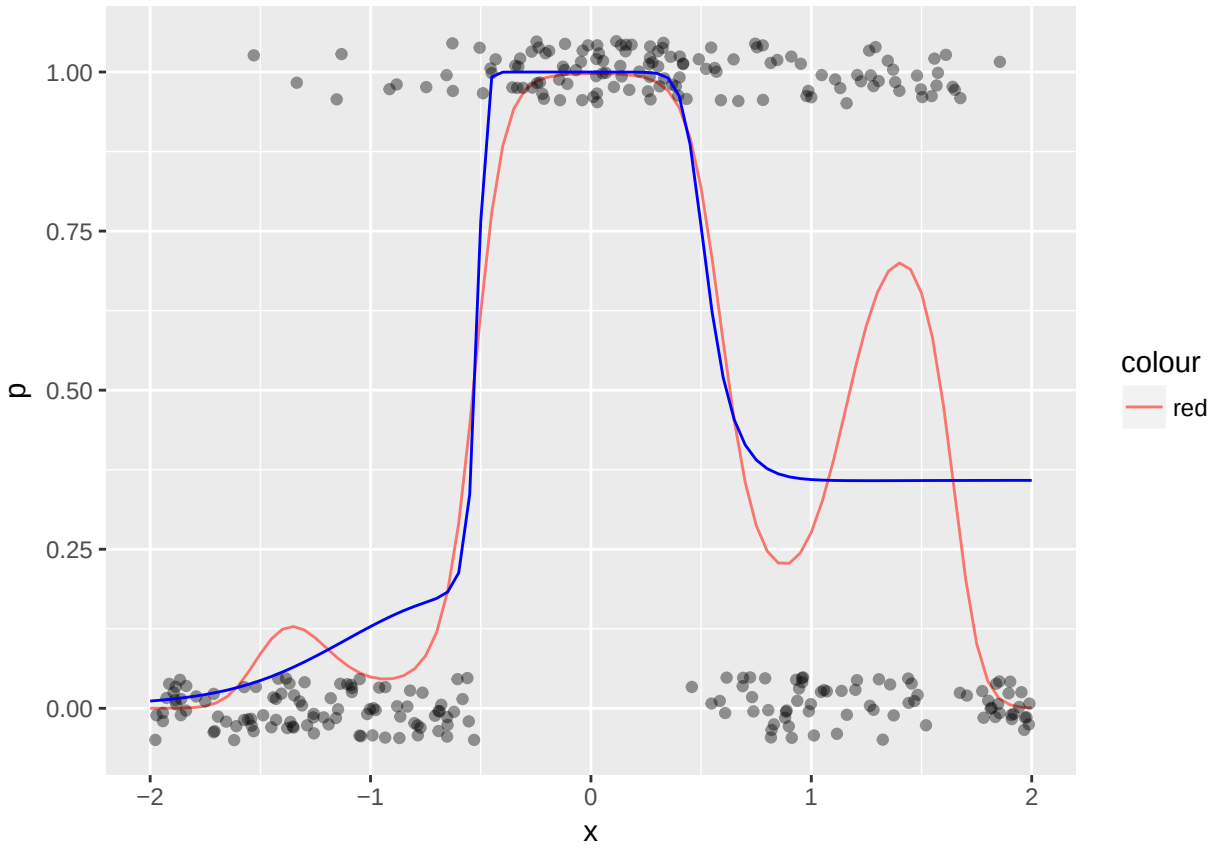
```
feed_fow <- function(beta, x){
  a_1 <- h(beta[1] + beta[2]*x) # calcula variable 1 de capa oculta
  a_2 <- h(beta[3] + beta[4]*x) # calcula variable 2 de capa oculta
  a_3 <- h(beta[5] + beta[6]*x)
  a_4 <- h(beta[7] + beta[8]*x)
  p <- h(beta[9]+beta[10]*a_1 + beta[11]*a_2 + beta[12]*a_3 + beta[13]*a_4) # calcula capa de salida
  p
```

```
}

devianza_fun <- function(x, y){
    # esta función es una fábrica de funciones
   devianza <- function(beta){
         p <- feed_fow(beta, x)
      - 2 * mean(y*log(p) + (1-y)*log(1-p))
   }
  devianza
}

dev <- devianza_fun(x.2, g.2)

set.seed(5)
salida <- optim(rnorm(13), dev, method='BFGS') # inicializar al azar punto inicial
salida
```

```
## $par
##  [1]  -3.556265  10.512723  -1.454699  -5.782094   4.031992   2.511931
##  [7] -11.810966 -26.227007   5.927853 -11.176326   1.284032   4.665916
## [13] -12.937752
##
## $value
## [1] 0.7403436
##
## $counts
## function gradient
##      100      100
##
## $convergence
## [1] 1
##
## $message
## NULL
```

```
beta<-salida$par
```

```
p.3<-feed_fow(beta,x)
datos.3<-data.frame(x,p.3)

g + geom_jitter(data = datos, aes(x=x.2,y=g.2), col ='black',
  position =position_jitter(height=0.05), alpha=0.4)+geom_line(data=datos.3,aes(x=x,y=p.3),col="blue")
```



Ahora lo vemos con la devianza regularizada:

```
devianza_reg <- function(x, y, lambda){
    # esta función es una fábrica de funciones
   devianza <- function(beta){
         p <- feed_fow(beta, x)
         # en esta regularizacion quitamos sesgos, pero puede hacerse también con sesgos.
        - 2 * mean(y*log(p) + (1-y)*log(1-p)) + lambda*sum(beta[-c(1,3,5)]^2)
   }
  devianza
}

dev_r <- devianza_reg(x.2, g.2, 0.001) # crea función dev
set.seed(5)
salida <- optim(rnorm(13), dev_r, method='BFGS') # inicializar al azar punto inicial
salida
```

```
## $par
##  [1] -2.2996479  5.1338008  0.8503622 -0.3881671  2.0878686  3.7709767
##  [7] -1.7815387 -3.6552983  0.2353861 -4.7448789  0.3692797  3.6266207
## [13] -3.7210264
##
## $value
## [1] 0.9238334
##
## $counts
## function gradient
##      100      100
```

```
##
## $convergence
## [1] 1
##
## $message
## NULL
```

```
beta<-salida$par
p.3<-feed_fow(beta,x)
datos.3<-data.frame(x,p.3)

g + geom_jitter(data = datos, aes(x=x.2,y=g.2), col ='black',
  position =position_jitter(height=0.05), alpha=0.4)+geom_line(data=datos.3,aes(x=x,y=p.3),col="blue")
```