Gov 2018: Lab 6 Neural networks

Your name:

Tuesday March 1, 2022

Neural networks

For this lab exercise, we will be writing a neural network from scratch. You may find it useful to take a look at the neural networks slides to write out your forward and back propagation functions.

Plot the data `p` to take a look at what the data actually looks like. `p` is composed of two parts, the `x` data, which consists of two features, and the `classes` output vector, a factor that takes 0 and 1 as values.

```r
library(tidyverse)
```

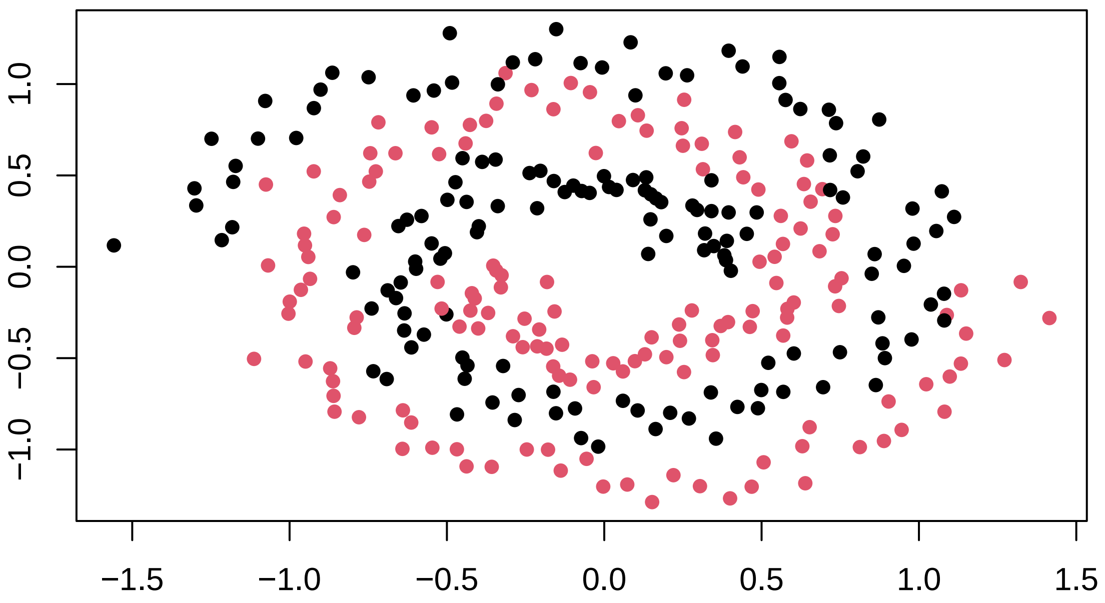
```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(mlbench)
library(ggplot2)
p<-readRDS("p.RDS")
## Recall that aim is to predict Y based on information from X
## Save x and y as separate items (two different versions for future use)
# features (x1, x2)
px <- data.frame(x1=p$x[,1],x2=p$x[,2]) # data.frame version
x <- data.matrix(p$x) # matrix version
# outcome label (y)
py <- p$classes
py <- as.factor(ifelse(p$classes==2,1,0)) # factor version
y <- p$classes == 2 # logical version


## TODO: Plot the data `p` as in slides (scatter plot of (x1, x2) where the color corresponds to class
# plot
# px %>%
#   ggplot(aes(x = x1, y = x2, color = y)) +
#   geom_point() +
#   theme_bw()
plot(p, pch=16)
```



Forward propagation

Write a forward propagation function (we'll use the sigmoid function for the activation). The function should take `x`, `w1` and `w2` as arguments and output the sigmoid of z2 and `h` (sigmoid of z1). Check the lecture slides ("Forward propagation").

```
## Forward propagation function
# Prepare activation function: we'll use the sigmoid function
# (Recall that activation function is a function that turns each hidden node 'on' or 'off')
sigmoid <- function(x) 1 / (1 + exp(-x))

# TODO: Write a forward propagation function
feedforward <- function(x, # your feature matrix. n by d matrix (n: # of data points, d: # of features)
                        w1, # weight matrix for generating hidden nodes (W_in: (d+1) by h matrix; h: #
                            # row: for each features. e.g., length of 3 = each for (1, x1, x2)
                            # col: for each hidden nodes. e.g., length of 2 = each for the first and
                        w2  # weight matrix for generating output (W_out: (h+1) by (k-1) matrix; k: # o
                            # row: for each hidden nodes. e.g., length of 2 = each for the first and
                            # col: for each classes. e.g., length of 1 = for the class (label) of 1
                        ) {
  # 1. Compute linear combination of features using weight matrix (W_in)
  z1 <- cbind(1, x) %*% w1
   # 2. Apply activation fn to get nodes in hidden layer
  h <- sigmoid(z1)
  # 3. For output layer, compute linear combination of hidden variables, this time using another weight
  z2 <- cbind(1, h) %*% w2
  # 4. Apply one more function to get the output
  output <- sigmoid(z2)
  return(list(output = output, # your predicted probs. n by (k-1) matrix
              h = h # hidden layer. n by h matrix
              ))
}
```

Back propagation

Write a back propagation function. The function should take as arguments: `x`, `y`, `yhat`, `w1`, `w2`, `h`, and the learning rate. It should return a list with `w1` and `w2`. Check the lecture slides (Back propagation).

```
## How do we get the weight matrices? Optimize the log-likelihood via gradient descent by iterating the
# TODO: Write a back propagation function
backpropagate <- function(x, # your features. n by d matrix
                          y, # your true labels (length of n)
                          y_hat, # your prediction (length of n)
                          w1, # current W_in. (d+1) by h matrix
                          w2, # current W_out. (h+1) by (k-1) matrix
                          h, # current hidden layer. n by h matrix
                          learn_rate # learning rate. e.g., 1e-2
                          ) {
  # 1. Gradient of the loglikelihood w.r.t the output weights (W_out)
  dw2 <- t(cbind(1, h)) %*% (y_hat - y)
  # 2. Gradient of the log-likelihood w.r.t the input weights (W_in)
  dh  <- (y_hat - y) %*% t(w2[-1, , drop = FALSE]) # n by h matrix
  # When drop is FALSE, the dimensions of the object are kept
  dw1 <- t(cbind(1, x)) %*% (h * (1 - h) * dh) # (d+1) by h matrix

  w1 <- w1 - learn_rate * dw1
  w2 <- w2 - learn_rate * dw2

  list(w1 = w1, w2 = w2)
}
```

Training the network

Write a training function for your neural network. It should take as arguments, `x`, `y`, the number of hidden nodes, the learning rate (set as default `1e-2`), and the number of iterations (set default at `1e4`). Return the *final* forward feeding output, `w1` and `w2`.

```
## Training the model
# TODO: Write a training function
# Training function
train.nn <- function(x, y, hidden = 5, learn_rate = 1e-2, iterations = 1e4) {
  # 1. Initialize weights
  dp1 <- ncol(x) + 1 # d+1
  w1 <- matrix(rnorm(dp1 * hidden), dp1, hidden)
  w2 <- as.matrix(rnorm(hidden + 1))
  for (i in 1:iterations) { # 2. Iterate forward and back propagation:
    # 2-1. Propagate forward to get output estimate
    ff <- feedforward(x, w1, w2)
    # 2-2. Propagate error backwards to update the weights towards a better solution
    bp <- backpropagate(x, y,
                        y_hat = ff$output,
                        w1, w2,
                        h = ff$h,
                        learn_rate = learn_rate)
    w1 <- bp$w1; w2 <- bp$w2
  }
  ff <- feedforward(x, w1, w2) # final forward feeding output
  list(output = ff$output, w1 = w1, w2 = w2)
}
```

`train.nn` your neural network with 5 hidden nodes on the data. Calculate the number of objects it classifies correctly. Let the cutoff be 0.5.

```
nnet5 <- train.nn(x, y, hidden = 5, iterations = 1e5)
mean((nnet5$output > .5) == y)
```

```
## [1] 0.7
```

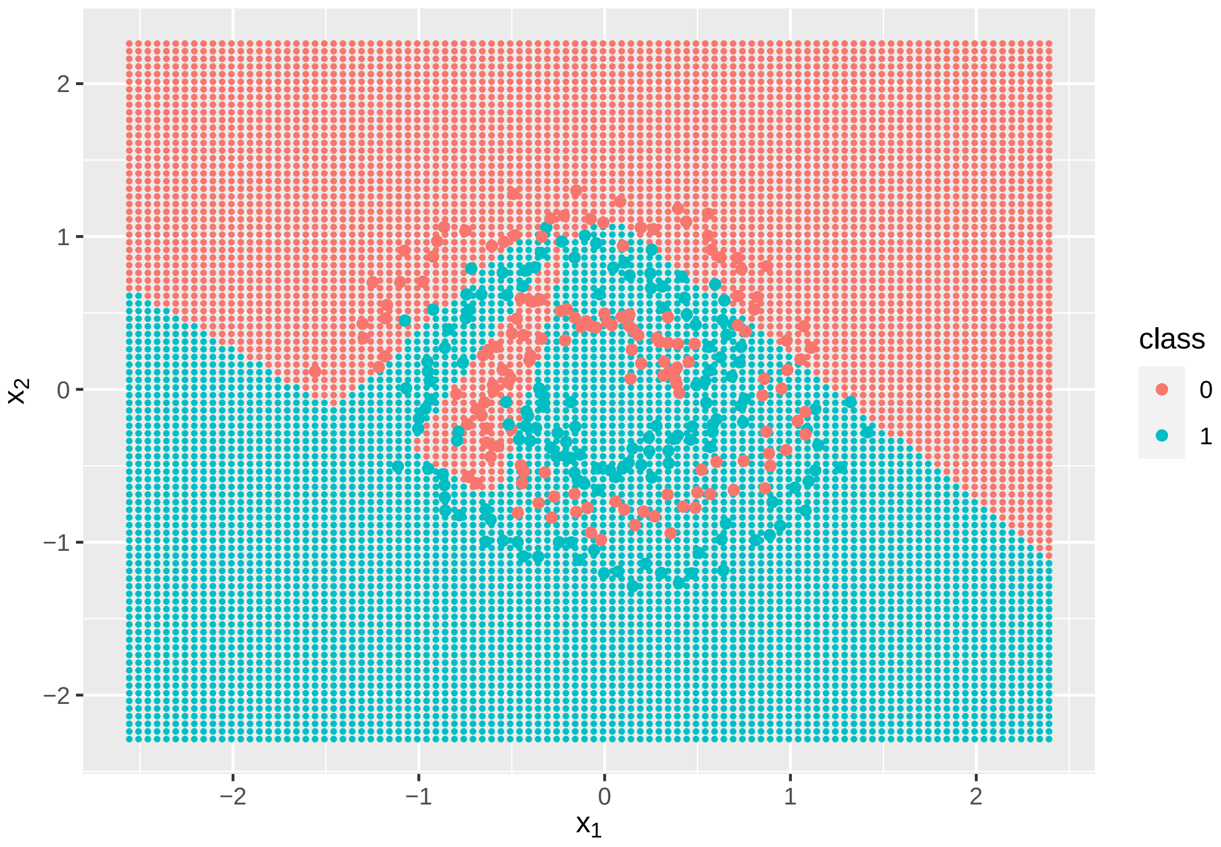
Visualization of the decision boundaries

Draw a picture to see what the decision boundaries look like.

1. Make a grid of points around the input space (`expand.grid`)
2. Feed the points through the trained neural network and get the classes (let the cut off be 0.5)
3. Plot the predicted classes on a grid behind the observed points.

```
# 1. Make a grid of points around the input space (`expand.grid`)
#assuming x is a two column matrix, with first column as variable 1, and second col as var 2
grid <- expand.grid(x1 = seq(min(x[,1]) - 1,
                             max(x[,1]) + 1,
                             by = .05),
                    x2 = seq(min(x[,2]) - 1,
                             max(x[,2]) + 1,
                             by = .05))
# TODO 2. Feed the points through the trained neural network and get the classes (let the cut off be 0.
ff_grid <- feedforward(x = data.matrix(grid[, c('x1', 'x2')]),
                       w1 = nnet5$w1,
                       w2 = nnet5$w2)
grid$class <- factor((ff_grid$output > .5) * 1,
                     labels = levels(py))

# 3. Plot the predicted classes on a grid behind the observed points.
p<-data.frame(x1=px$x1,x2=px$x2,class=py)
ggplot(p) + aes(x1, x2, colour = class) +
  geom_point(data = grid, size = .5) +
  geom_point() +
  labs(x = expression(x[1]), y = expression(x[2]))
```



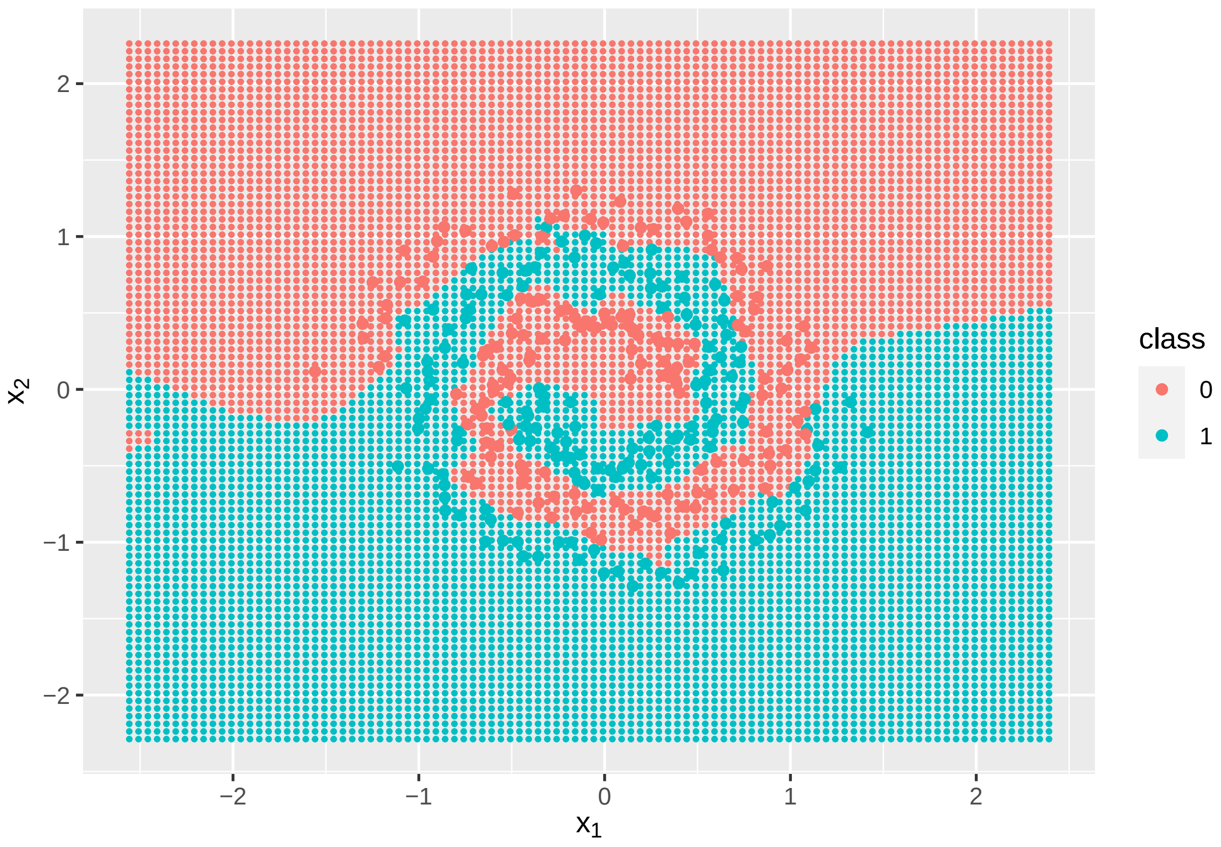
Try different specification

Try training your neural network again, but this time with 30 nodes. Plot the decision boundaries as before. Do you see any differences? What happens with just one hidden node?

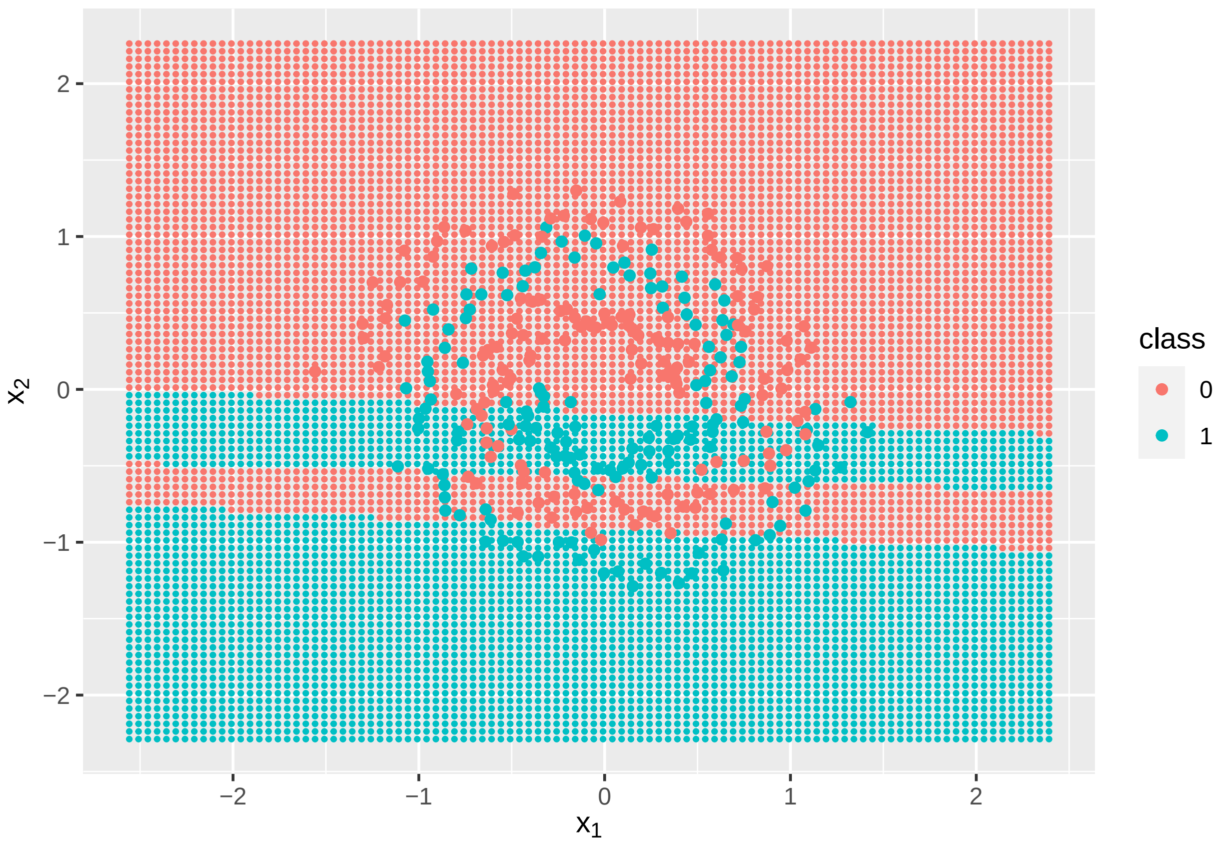
```
nnet25 <- train.nn(x, y, hidden = 25, iterations = 1e6)
ff_grid25 <- feedforward(x = data.matrix(grid[, c('x1', 'x2')]),
                         w1 = nnet25$w1,
                         w2 = nnet25$w2)

grid$class <- factor((ff_grid25$output > .5) * 1,
                     labels = levels(py))

ggplot(p) + aes(x1, x2, colour = class) +
  geom_point(data = grid, size = .5, aes(colour=class)) +
  geom_point() +
  labs(x = expression(x[1]), y = expression(x[2]))
```



```
nnet2 <- train.nn(x, y, hidden = 2, iterations = 1e5)
ff_grid2 <- feedforward(x = data.matrix(grid[, c('x1', 'x2')]),
                        w1 = nnet2$w1,
                        w2 = nnet2$w2)
grid$class <- factor((ff_grid2$output > .5) * 1,
                     labels = levels(py))
ggplot(p) + aes(x1, x2, colour = class) +
  geom_point(data = grid, size = .5) +
  geom_point() +
  labs(x = expression(x[1]), y = expression(x[2]))
```


2
1
0
−1
−2
x2
−2
−1
0
1
2
x1
class
0
1