Weekly Summary Template

Brady Miller

Table of contents

Thursday, March 30

! TIL

Include a *very brief* summary of what you learnt in this class here.
Today, I learnt the following concepts in class:

1. More on plotting boundaries using decision trees
2. Creating neural networks
3. Regression with Neural Networks (compared to other types)

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.1     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.1     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
library(torch)
library(glmnet)
```

```
Loading required package: Matrix

Attaching package: 'Matrix'

The following objects are masked from 'package:tidyr':

    expand, pack, unpack

Loaded glmnet 4.1-7
```

```r
library(caret)
```

```
Loading required package: lattice

Attaching package: 'caret'

The following object is masked from 'package:purrr':

    lift
```

```r
library(dplyr)
library(tidyr)
library(dplyr)
library(nnet)
library(mlbench)
library(class)
library(rpart.plot)
```

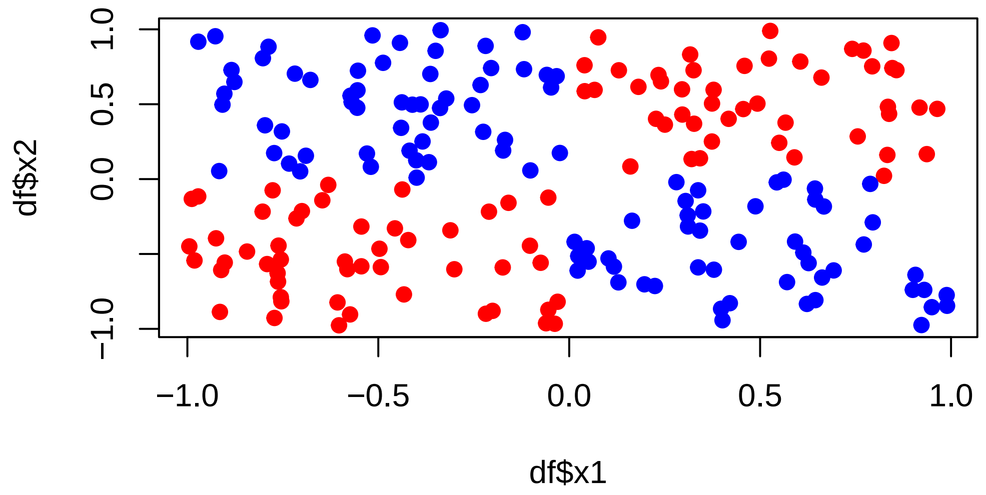
```
Loading required package: rpart
```

```r
library(e1071)
```

Plotting boundaries using decision trees

Creating plot of points

```
ex1 <- \(x) ifelse(
  sign(x[1] * x[2]) +0.01 * rnorm(1) <= 0, 0, 1)
n <- 200
X <- t(replicate(n, 2 * runif(2) -1))
y <- apply(X, 1, ex1) %>% as.factor()
col <- ifelse(y == 0, 'blue', 'red')
df <- data.frame(y = y, x1 = X[,1], x2 = X[,2])
plot(df$x1, df$x2, col = col, pch = 19)
```



Creating the logistic function and the predictions on the model that we will graph

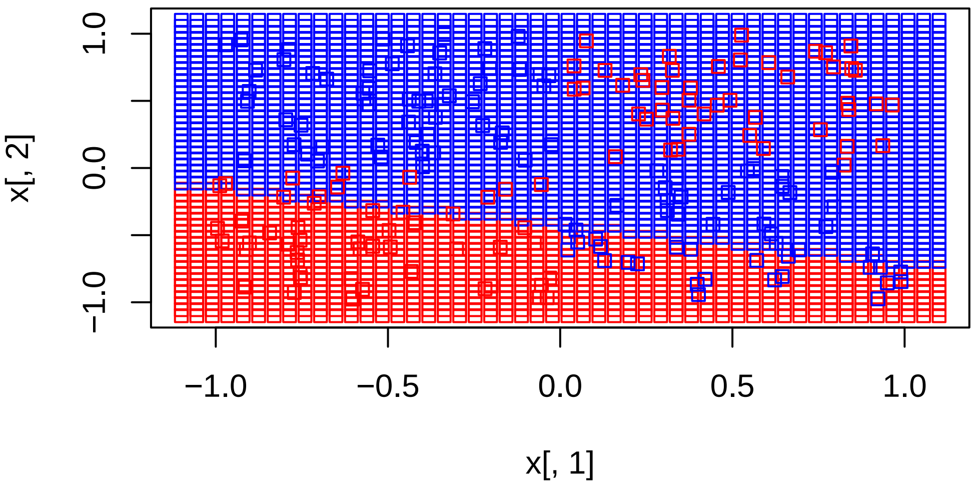
```
model <- glm(y~., df, family = binomial())

f_logistic = \(x) predict(model, data.frame(x1=x[,1], x2 = x[,2]),
                          type = 'response')

xnew <- cbind(
  x1 = rep(seq(-1.1,1.1,length.out=50),50),
  x2 = rep(seq(-1.1,1.1,length.out=50),each = 50)
)
```

```
plt <- function(f, x) {
  plot(x[,1], x[,2], col=ifelse(f(x) < 0.5, 'blue', 'red'), pch =22)
  points(df$x1, df$x2, col=ifelse(y=='0', 'blue', 'red'), pch =22)
}
```

Creating an overview function that creates a table of the predictions vs actual values

```
overview <- function(f){
  11
  predicted <- ifelse(f(df[,-1])<0.5, 0, 1)
  actual <- df[,1]
  table(predicted, actual)
}
```

```
plt(f_logistic, xnew)
```



```
overview(f_logistic)
```

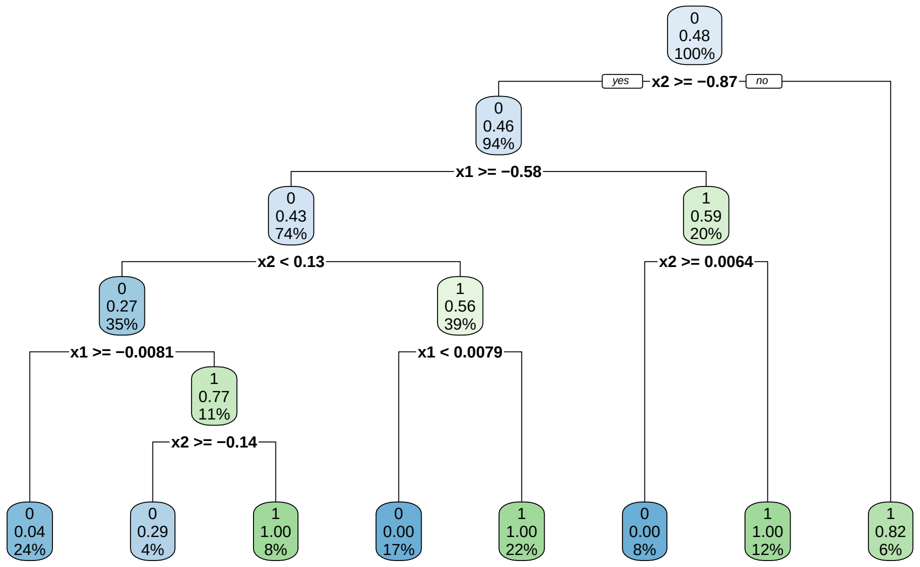
```
         actual
predicted  0  1
```

```
0 80 56
1 24 40
```

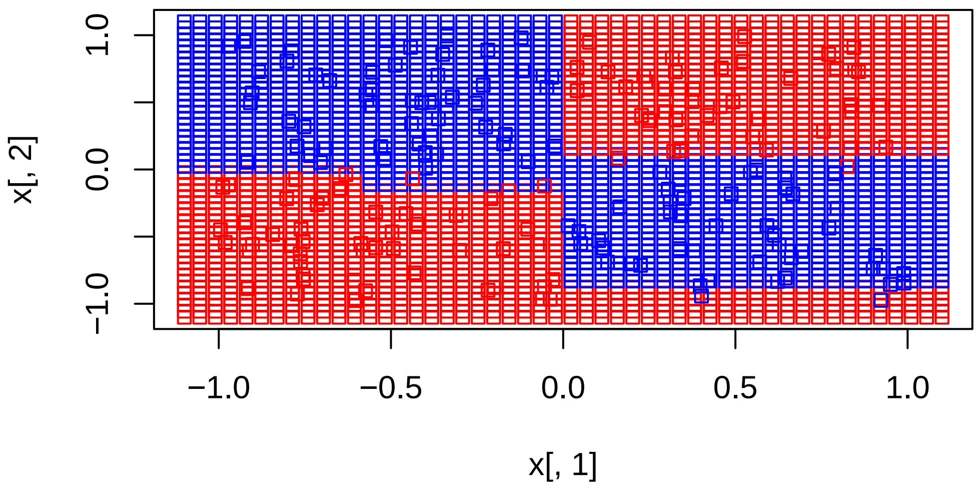
This plot demonstrates the linear regression would not be a good choice for this data set as many of the points are misclassified. This is also shown in the confusion matrix, as 41 of the 94 points that are actually blue are classified/ predicted as red, while 18 of the 106 points that are actually red are predicted as blue.

By using a decision tree model created below, we can better predict the class of the points.

```
dtree <- rpart(y ~ x1 + x2, df, method = 'class')
rpart.plot(dtree)
```



```
f_dtree <- \(x) as.numeric(predict(dtree, data.frame(x1 = x[,1], x2=x[,2]),
                                   type = 'class')) - 1
plt(f_dtree, xnew)
```



This plot does a much better job in correctly classifying the points

```r
overview(f_dtree)
```

```
          actual
predicted   0   1
        0 102   4
        1   2  92
```

Based on the confusion matrix, it actually perfectly predicts the values.

Creating Neural Networks

Neural Network with 2 hidden layer

```r
X_tensor <- torch_tensor(df[, -1] %>% as.matrix(), dtype = torch_float())
y_tensor <- torch_tensor(cbind(df[, 1] %>% as.numeric() - 1), dtype = torch_float())


Loss <- function(x, y, model){
  nn_bce_loss()(model(x), y)
}


Xnew <- cbind(
  x1 = rep(seq(-1, 1, length.out = 50), 50),
  x2 = rep(seq(-1, 1, length.out = 50), each = 50))


df_new <- data.frame(x1=Xnew[,1], x2=Xnew[,2])

p <- 2
q <- 20
q1 <- 100
q2 <- 20

hh2_module <- nn_module(
  initialize = function() {
    self$f <- nn_linear(p,q1)
    self$g <- nn_linear(q1,q2)
    self$h <- nn_linear(q2,1)
    self$a <- nn_relu()
    self$s <- nn_sigmoid()
  },
  forward = function(x) {
    x %>%
      self$f() %>%
      self$a() %>%
      self$g() %>%
      self$a() %>%
      self$h() %>%
      self$s()
  }
)




F <- hh2_module()
optimizer <- optim_adam(F$parameters, lr = 0.05)
epochs <- 200
```

```r
for(i in 1:epochs){
  loss <- Loss(X_tensor, y_tensor, F)

  optimizer$zero_grad()
  loss$backward()
  optimizer$step()

  if (i < 10 || i %% 20 == 0) {
    cat(sprintf("Epoch: %d, Loss: %.4f\n", i, loss$item()))
  }
}
```

```
Epoch: 1, Loss: 0.6932
Epoch: 2, Loss: 0.6875
Epoch: 3, Loss: 0.5852
Epoch: 4, Loss: 0.5208
Epoch: 5, Loss: 0.4156
Epoch: 6, Loss: 0.3046
Epoch: 7, Loss: 0.2367
Epoch: 8, Loss: 0.1857
Epoch: 9, Loss: 0.1423
Epoch: 20, Loss: 0.0190
Epoch: 40, Loss: 0.0024
Epoch: 60, Loss: 0.0007
Epoch: 80, Loss: 0.0004
Epoch: 100, Loss: 0.0003
Epoch: 120, Loss: 0.0002
Epoch: 140, Loss: 0.0002
Epoch: 160, Loss: 0.0002
Epoch: 180, Loss: 0.0001
Epoch: 200, Loss: 0.0001
```

```r
f_nn = \(x) as_array(F(torch_tensor(x %>% as.matrix(), dtype = torch_float())))
overview(f_nn)
```

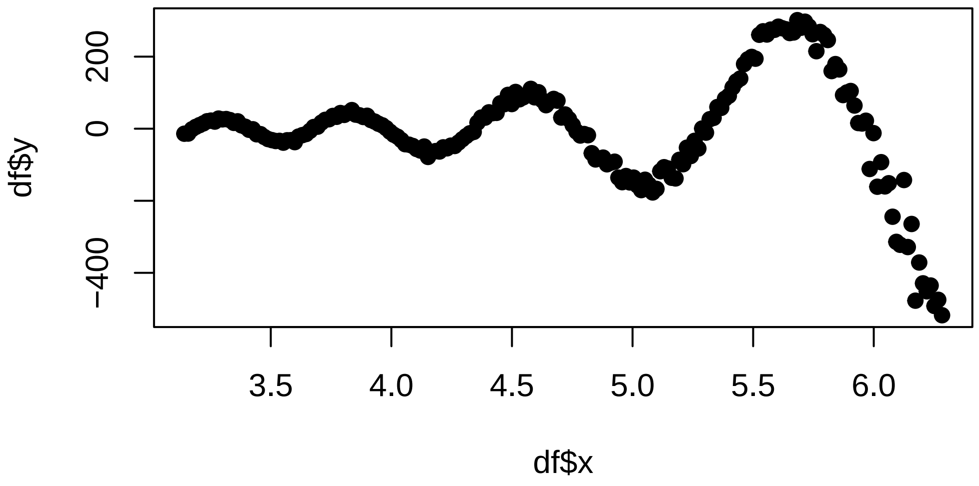
```
         actual
predicted   0   1
        0 104   0
        1   0  96
```

From this neural network, we can see that it perfectly predicts the values.

Regression with Neural Networks

This code chunk is creating a graph of points in a sinusodial like curve which we will use to demonstrate the various regression techniques on and see which ones work the best

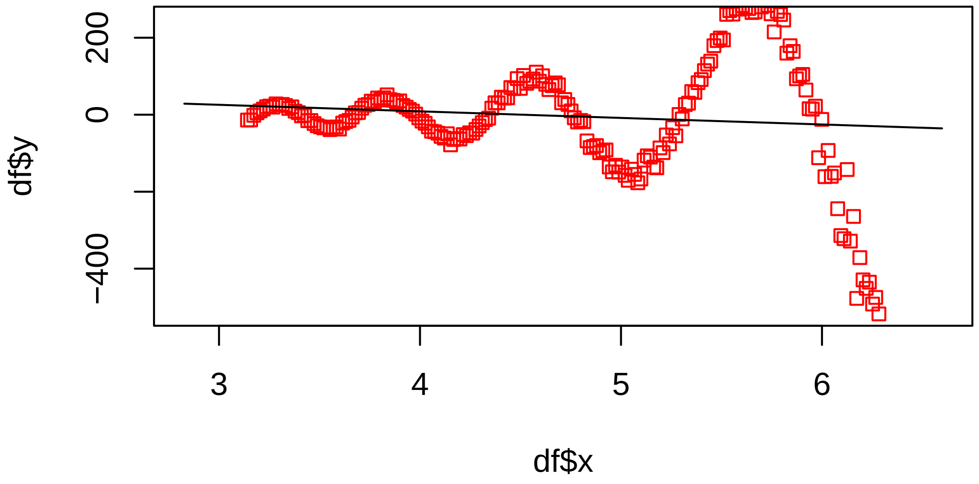
```r
generate_data <- function(n, noise = 0.1) {
  x <- seq(1*pi, 2*pi, length.out = n)
  y <- exp(x) * (sin(150/x) + rnorm(n, 0, noise))
  data.frame(x = x, y = y)
}

df <- generate_data(200, noise = 0.1)
plot(df$x, df$y, pch=19)
```



```r
x_new <- seq(0.9 * pi, 2.1*pi,  length.out = 1000)
df_new <- data.frame(x = x_new)

plt_reg <- function (f, x){
  ynew <- f(x)
  ylim <- range(c(ynew, df$y))
  ylim[1] <- max(c(-800, ylim[1]))
  ylim[2] <- min(c(250, ylim[2]))
  xlim <-range(x)
  plot(df$x, df$y, pch = 22, col = 'red', xlim=xlim, ylim = ylim)
  points(x[,1], ynew, pch=22, type='l')
}
```

Linear regression on that plot

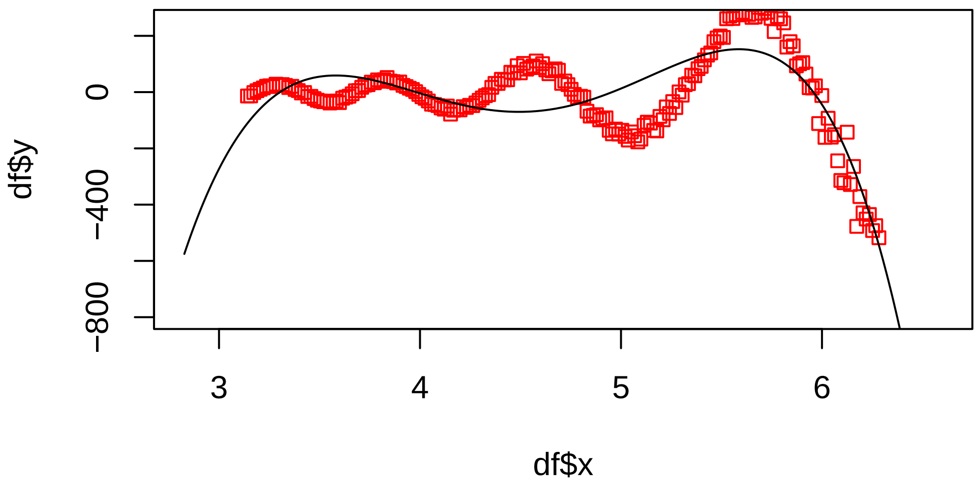
```
f_lm = \(x)
    lm(y ~ x, df) %>%
    predict(., x)
plt_reg(f_lm, df_new)
```



- This linear regression doesn't do a good job fitting the data (too simple)

Polynomial regression on that plot

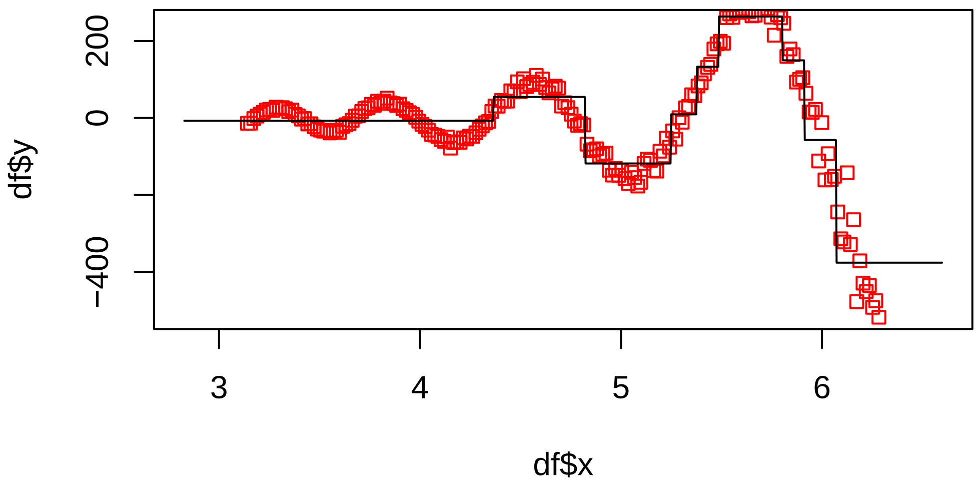
```
f_polynomial = \(x)
    lm(y ~ x + I(x^2) + I(x^3) + I(x^5), df) %>%
    predict(., x)
plt_reg(f_polynomial, df_new)
```



- The polynomial regression does a better job and creates less error as it ‘moves’ more with the curve than the linear regression.

Decision Tree regression

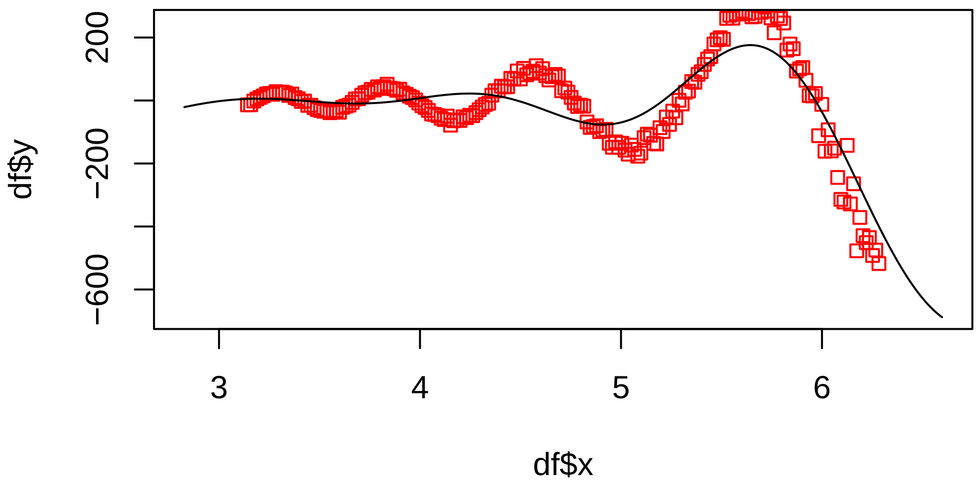
```
f_dtree = \(x)
    rpart(y ~ x, df) %>%
    predict(., x)
plt_reg(f_dtree, df_new)
```



- Decision tree does good job of modeling funtion especially in the range of (5,6), but it has some weakness in the beginning

Support Vector Machine

```
f_svm = \(x)
    svm(y ~ x, df, kernel = 'radial') %>%
    predict(., x)
plt_reg(f_svm, df_new)
```

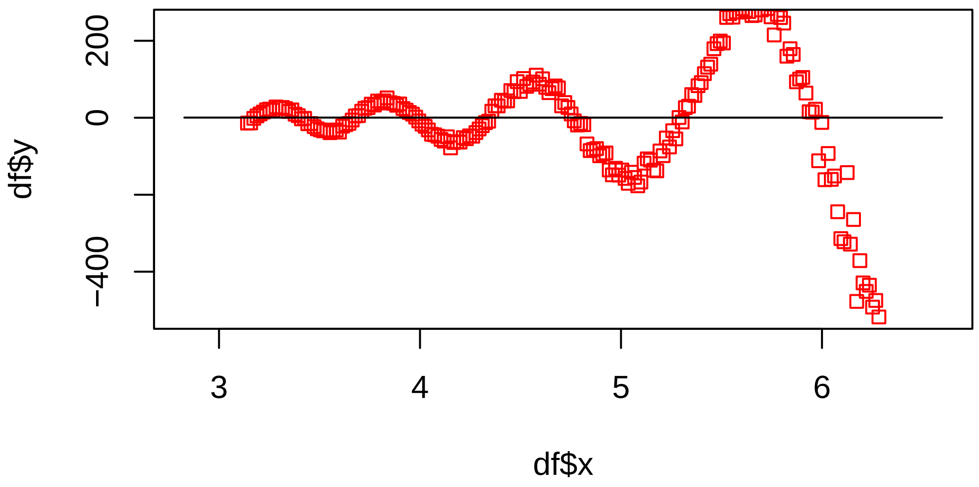


- SVM might do the best job as it consistently flows in directions similar to the data displayed which results in less error.

Neural Network

```
reg_module <- nn_module(
  initialize = function() {
    self$f <- nn_linear(1,50)
    self$g <- nn_linear(50,200)
    self$h <- nn_linear(200,1)
    self$a <- nn_relu()
    self$s <- nn_sigmoid()
  },
  forward = function(x) {
    x %>%
      self$f() %>%
      self$a() %>%
      self$g() %>%
      self$a() %>%
      self$h() %>%
      self$s()
  }
)
```

```r
f_nn <- function(x){
  F <- reg_module()
  X_tensor <- torch_tensor(df$x %>% as.matrix(), dtype=torch_float())
  y_tensor <- torch_tensor(cbind(df$y), dtype=torch_float())
  optimizer <- optim_adam(F$parameters, lr = 0.001)
  epochs <- 2000

  for(i in epochs){
    loss <- nn_mse_loss()(F(X_tensor), y_tensor)
    optimizer$zero_grad()
    loss$backward()
    optimizer$step()
  }
  return(as_array(F(torch_tensor(x %>% as.matrix(), dtype=torch_float()))))
}

plt_reg(f_nn, df_new)
```



- I’m not sure why the neural network does such a poor job with the data, and I don’t know how to fix it but remember that it performed much better in the demonstration in class.