Optimization for machine learning

Anastasia Chanbour

22/10/2021

Load the required packages

```
library(tidyverse) # required
```

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

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.5     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()
```

```
library(broom)     # required
library(modelr)    # for data_grid
```

```
##
## Attaching package: 'modelr'

## The following object is masked from 'package:broom':
##
##     bootstrap
```

```
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
theme_set(theme_minimal(base_size = 22))
```

1-d smooth regression example

Example data

Generate a one-dimensional example with a non-linear relationship

```
f <- function(x) sin(4*pi*x)
n <- 200
train1d <-
  data.frame(
    x = rbeta(n, 1, 3)
    ) %>%
```

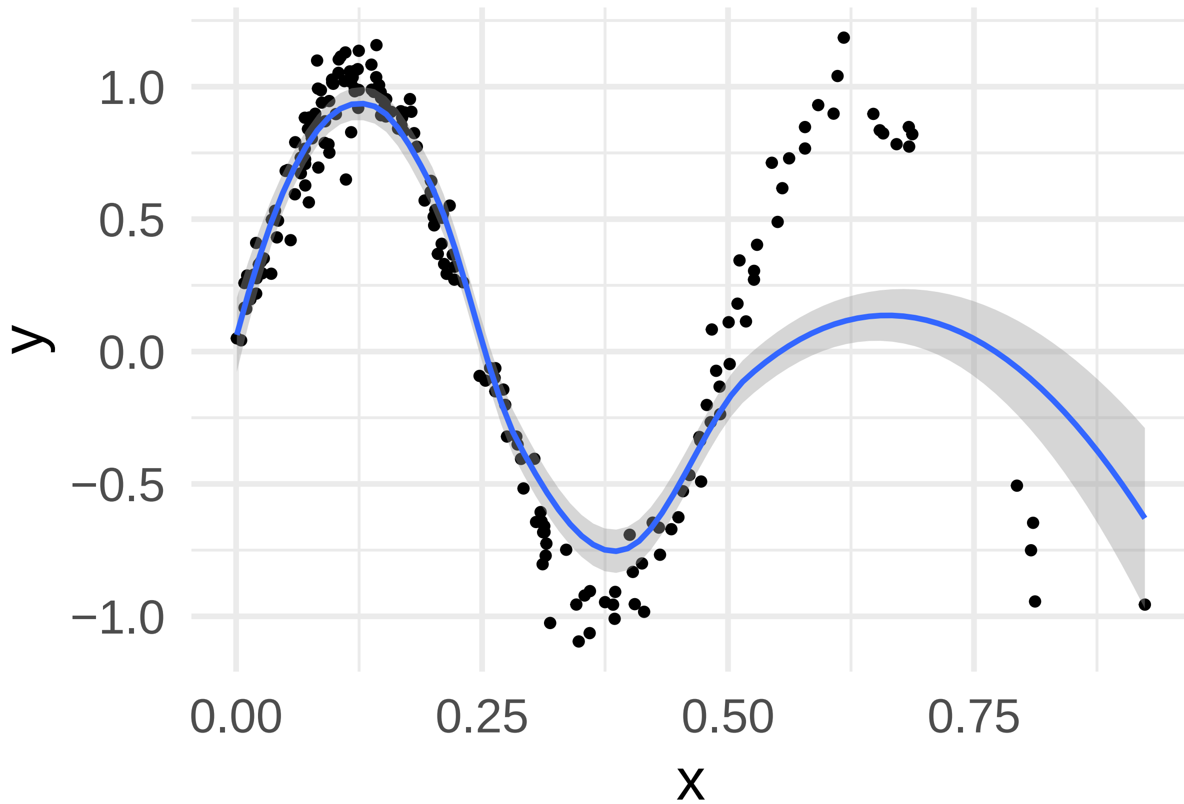
```
  # change the noise level sd

  mutate(y = f(x) + rnorm(n, sd = .1))

# plot data and loess curve

ggplot(train1d, aes(x, y)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

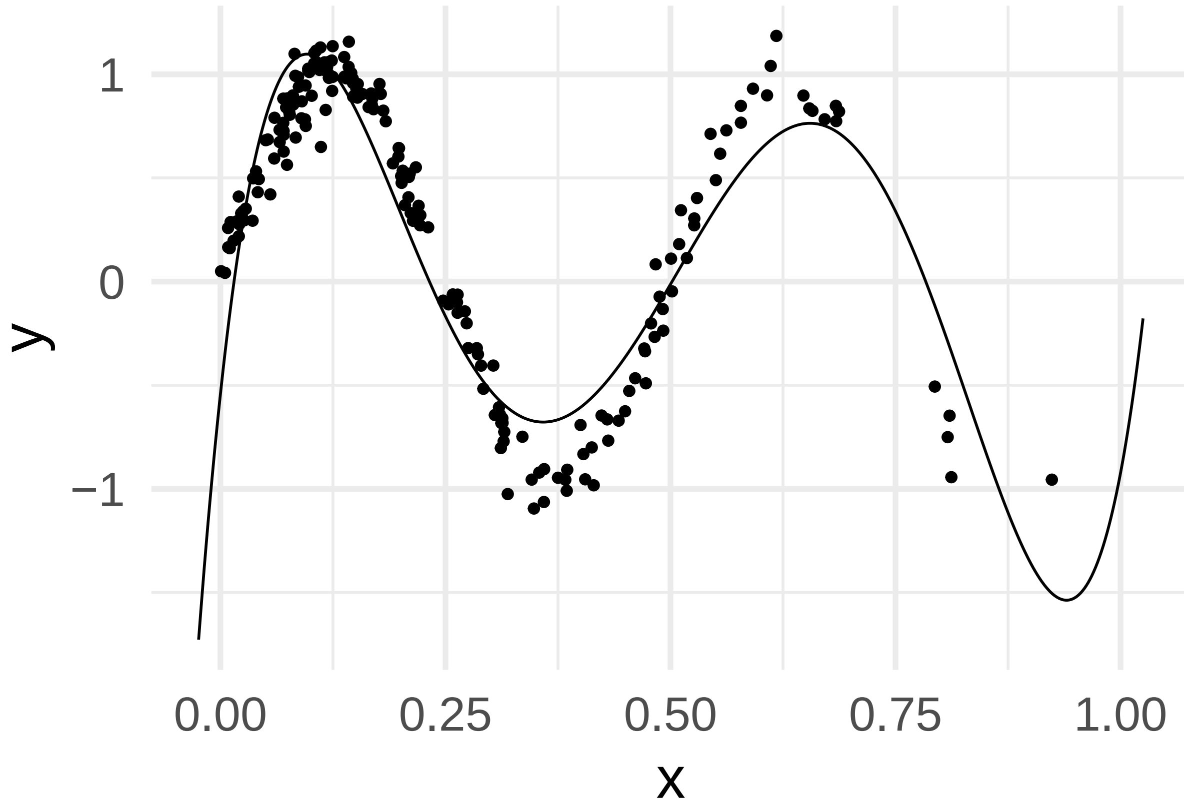


Linear regression with a polynomial transform of x

```
model_lm <- lm(y ~ poly(x, 5), data = train1d)

train1d_grid <-
  data_grid(train1d,
          x = seq_range(c(x, 1), 500, expand = .05))

augment(model_lm,
        newdata = train1d_grid) %>%
  ggplot(aes(x, y)) +
  geom_point(data = train1d) +
  geom_line(aes(y = .fitted))
```



Gradient descent

Goal: implement gradient descent and use it to solve for the coefficients of the above linear model

Update is

$$\beta_{k+1} = \beta_k - \gamma \nabla L(\beta_k)$$

where $\gamma > 0$ is the step size.

Step 1: writing functions to output the least squares loss and its gradient

```r
# Loss function in linear regression RSS:
least_squares_loss <- function(x,y,beta) {
  sum((y-x%*%beta)^2)
}

# Loss function gradient:
least_squares_gradient <- function(x,y,beta) {
  -2*t(x)%*%(y-x%*%beta) #in matrix notation
}
```

Step 2: writing a loop to take multiple steps in the direction of the negative gradient, keeping step size fixed

```
# Model data and initialise coefficient vector
y <- train1d$y
x <- model.matrix(model_lm)
p <- ncol(x)

gamma <- 1 #step size within permitted values
beta0 <-rep(0,p) #initialise vector of parameters as 1xp vector of zeros

previous_loss <- least_squares_loss(x,y,beta0)
grad0 <- least_squares_gradient(x,y, beta0)
beta1 <- beta0 - gamma*grad0 #first update of beta
next_loss <- least_squares_loss(x,y,beta1)
previous_beta <- beta1

for(i in 1:5){
  gradn <- least_squares_gradient(x,y, previous_beta)
  next_beta <- previous_beta - gamma*gradn
  previous_beta <- next_beta
  print(previous_beta)
}
```

```
##                      [,1]
## (Intercept) -5.051689e+04
## poly(x, 5)1  1.776357e-13
## poly(x, 5)2  1.776357e-15
## poly(x, 5)3  4.041212e-14
## poly(x, 5)4 -2.664535e-14
## poly(x, 5)5 -4.440892e-14
##                      [,1]
## (Intercept)  2.015637e+07
## poly(x, 5)1 -8.127042e+00
## poly(x, 5)2  4.978637e+00
## poly(x, 5)3 -1.368841e+00
## poly(x, 5)4 -1.248647e+01
## poly(x, 5)5  7.562221e+00
##                      [,1]
## (Intercept) -8.042390e+09
## poly(x, 5)1  1.254445e-08
## poly(x, 5)2 -1.440070e-08
## poly(x, 5)3  1.728987e-08
## poly(x, 5)4 -7.831714e-09
## poly(x, 5)5  4.317514e-08
##                      [,1]
## (Intercept)  3.208914e+12
## poly(x, 5)1 -8.127050e+00
## poly(x, 5)2  4.978641e+00
## poly(x, 5)3 -1.368852e+00
## poly(x, 5)4 -1.248647e+01
## poly(x, 5)5  7.562204e+00
##                      [,1]
## (Intercept) -1.280357e+15
```

```
## poly(x, 5)1  3.722831e-03
## poly(x, 5)2 -1.974916e-04
## poly(x, 5)3 -8.992650e-04
## poly(x, 5)4 -1.057499e-03
## poly(x, 5)5  6.042957e-03
```

Step 3: writing a function to step in the direction of the negative gradient until the loss function no longer decreases by a certain amount, keeping step size fixed

```r
gamma <- 0.001
beta0 <- beta0 <-rep(0,p)
previous_loss<- previous_loss <- least_squares_loss(x,y,beta0)
grad0 <- least_squares_gradient(x,y, beta0)
beta1 <- beta0 - gamma*grad0
next_loss <- least_squares_loss(x,y,beta1)
steps <- 1 #for printing updates


#we iterate until the change in values of the loss function is small enough
while ((next_loss - previous_loss > 0.01)) {
   gradn <- least_squares_gradient(x,y, previous_beta)
  next_beta <- previous_beta - gamma*gradn
  grad_current <- gradn
  if (steps %% 50 == 0) print(previous_loss)
  steps <- steps + 1
  previous_beta <- next_beta
  previous_loss <- next_loss
  next_loss <- least_squares_loss(x,y,next_beta)

}
```

Step 4: experimenting with manually decreasing the stepsize and convergence threshold

```r
c(steps, previous_loss, next_loss)
```

```
## [1]   1.00000 107.01059  93.81674
```

```r
next_beta
```

```
##                      [,1]
## (Intercept) -1.280357e+15
## poly(x, 5)1  3.722831e-03
## poly(x, 5)2 -1.974916e-04
## poly(x, 5)3 -8.992650e-04
## poly(x, 5)4 -1.057499e-03
## poly(x, 5)5  6.042957e-03
```

Extra reference: use the Barzilai-Borwein method to choose step size

See https://en.wikipedia.org/wiki/Gradient_descent