Machine Learning for Economists **Summer 2024**

University of Cologne **Due: May 1, 23:59**

Lab 2: Optimization

Please send your solutions (RMarkdown/Quarto or Jupyter Notebook with code and answers plus a version compiled into pdf format) to tom.zimmermann@uni-koeln.de

In this lab, we will consider gradient descent for function optimization. I will provide some code snippets that you can use to get started.

1. Gradient descent

In this exercise, we use gradient descent to find the minimum of

$$f(\theta) = (\theta - 2)^2 + 5$$

You can define and plot the function f with

```
# Define objective function
f =  function(theta) {
 (theta-2)^2 + 5
}

# Plot
plot(f, from = 0, to = 4)  # From and To define the range of X

# Alternatively
plot(f(seq(from = 0, to = 4, length.out = 200)),  # Plug in sequence of Xs in function f
     type = 'l',  # "lineplot")
```

1. What's the gradient of our function f? Define a gradient function g and plot it.
2. Assume a constant learning rate of $\lambda = .8$. Write down the general update step for gradient descent.
3. Implement gradient descent for minimizing f making use of your defined gradient function g. Compute 20 iterations to find the θ that minimizes $f(\theta)$. Plot the sequence of θ_ts against the iteration t. Start with $\theta_0 = 5$.
4. Replace the analytical gradient by a two-sided numerical approximation. This is often necessary in practice when the analytical gradient is hard to compute. Use a two-sided approximation such that

$$\hat{g}(\theta) = \frac{f(\theta + h) - f(\theta - h)}{2h}$$

   Repeat part 3 using the numerical gradient.

2. Ordinary least squares

In this part, we will use gradient descent to find coefficient estimates in a simple linear regression. Our model is

$$y_i = \beta_0 + \beta_1 x_i + u_i$$

and we are interested in minimizing the average loss over a sample of observations

$$\min_{\beta_0, \beta_1} \frac{1}{N} \sum_{i=1}^{N} L(\beta_0, \beta_1; y_i, x_i)$$

with the loss function being the squared error.

$$L(\beta_0, \beta_1; y_i, x_i) = (y_i - (\beta_0 + \beta_1 x_i))^2$$

We work with data from Kiva, a large provider of microcredit in developing countries. On their website, a potential borrower can upload a loan proposal, and creditors can chip in until the requested loan amount is reached, i.e. an initial loan request of, say 200 USD, will be split among, say 10 creditors, not necessarily with equal contributions. Load the dataset `Lab2_Optimization.csv` into R.

We are interested in the time it takes to fund a loan proposal as a function of the requested loan amount.

$$TimeToFund_i = \beta_0 + \beta_1 LoanAmount_i + \epsilon_i$$

1. Plot `LoanAmount` against `TimeToFund` to get a sense of the relationship between these two variables.
2. Even though we know how to solve OLS in closed form, we want to use gradient descent to find the coefficients. The objective function can be defined as

```
# Define objective function
MSE = function(beta0, beta1, y, X) {

  return(
    sum( (y - beta0 - beta1*X)^2)/length(y)
  )
}
```

Plot the objective function (average loss) as a function of $\beta_1 \in [0, .01]$, keeping β_0 fixed at 7.

3. The following function returns the analytical gradient of the objective function

```
gradientOLS = function(beta0, beta1, y, X){

  error = beta0 + beta1*X - y
  g0 = 2*sum(error)/length(y)    # Gradient wrt to beta0
  g1 = 2*sum(error*X)/length(y) # Gradient wrt to beta1

  return(c(g0, g1))    # Return gradient
}
```

Make sure you understand why this is the analytical gradient. Use this gradient function to optimize the MSE via gradient descent, starting at $\beta_0 = 5$ and $\beta_1 = .005$. Use a learning rate of $\lambda = .0001$ and 1000 iterations. Why does the algorithm yield NaNs for β_0 and β_1?

4. Does it help to change the learning rate?
5. What happens when we express `LoanAmount` in *1000USD terms* rather than in raw dollar terms? Try a learning rate of $\lambda \in \{.1, .01\}$.