Course – Introduction to ML

Fadel Mamar Seydou

MSc. Computational Science and Engineering

Sampled from EPFL CS-433 Machine learning course of 2020

Table of contents

Table of contents

Today

History of machine learning

From Turing to Transformers

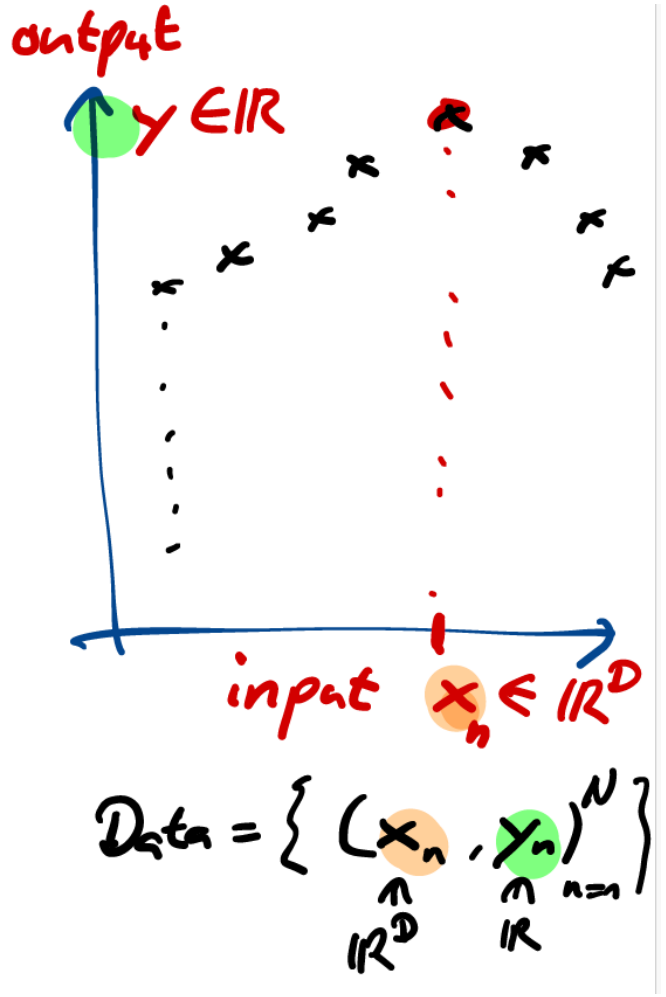
Regression

What is regression?

Regression is to relate input variables to the output variable, to either predict outputs for new inputs and/or to understand the effect of the input on the output.

Dataset for regression

In regression, data consists of pairs $(\mathbf{x}_n, y_n)$, where y_n is the n'th output and $\mathbf{x}_n$ is a vector of D inputs. The number of pairs N is the data-size and D is the dimensionality.



Regression

Two goals of regression

In prediction, we wish to predict the output for a new input vector, e.g. what is the weight of a person who is 170 cm tall?

In interpretation, we wish to understand the effect of inputs on output, e.g. are taller people heavier too?

The regression function

For both the goals, we need to find a function that approximates the output "well enough" given inputs.

$$y_n \approx f(\mathbf{x}_n), \text{ for all } n$$

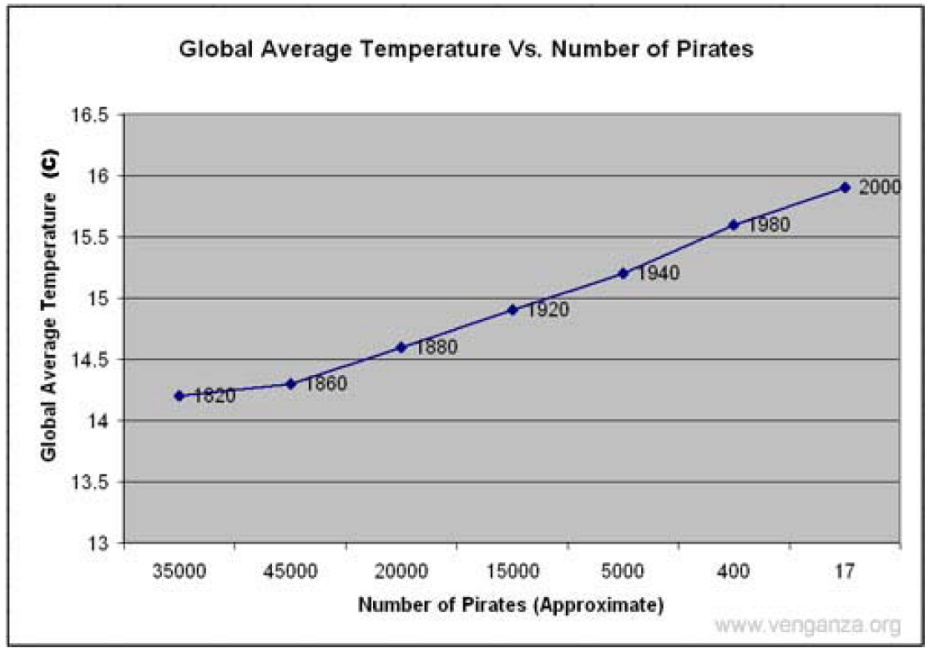
Machine Learning Jargon for Regression

Input variables are also known as features, covariates, independent variables, explanatory variables, exogenous variables, predictors, regressors. *Output variables* are also known as target, label, response, outcome, dependent variable, endogenous variables, measured variable, regressands.

Regression

Correlation $\neq$ Causation

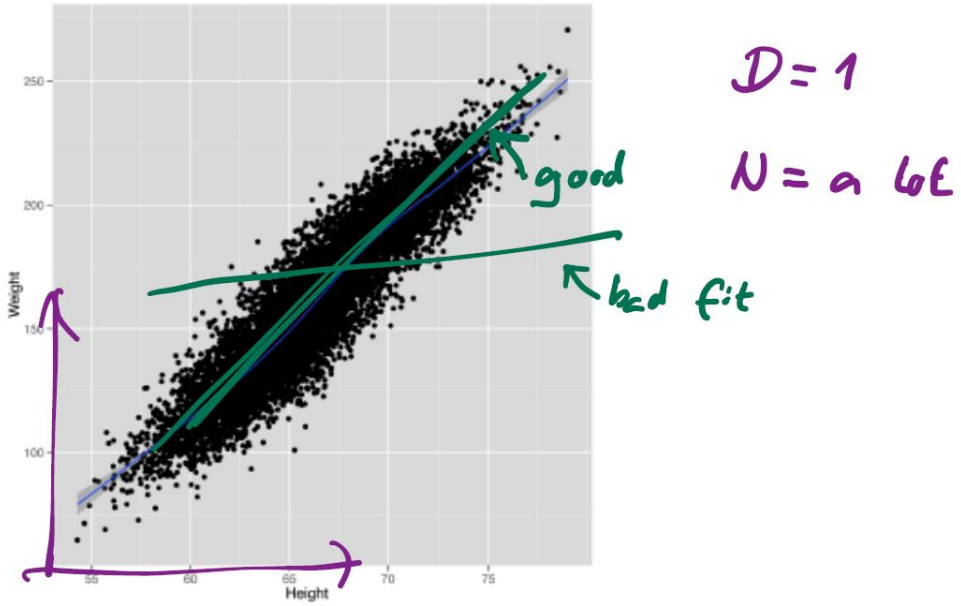
Regression finds correlation not a causal relationship, so interpret your results with caution.



Regression: linear regression

What is it?

Linear regression is a model that assumes a linear relationship between inputs and the output.



Why learn about linear regression?

Plenty of reasons: simple, easy to understand, most widely used, easily generalized to non-linear models. Most importantly, you can learn almost all fundamental concepts of ML with regression alone.

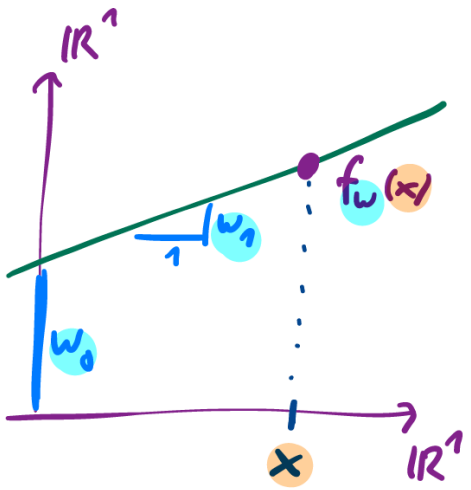
Regression: linear regression

Simple linear regression

With only one input dimension, we get simple linear regression.

$$y_n \approx f(\mathbf{x}_n) := w_0 + w_1 x_{n1}$$

Here, $\mathbf{w} = (w_0, w_1)$ are the two parameters of the model. They describe f.



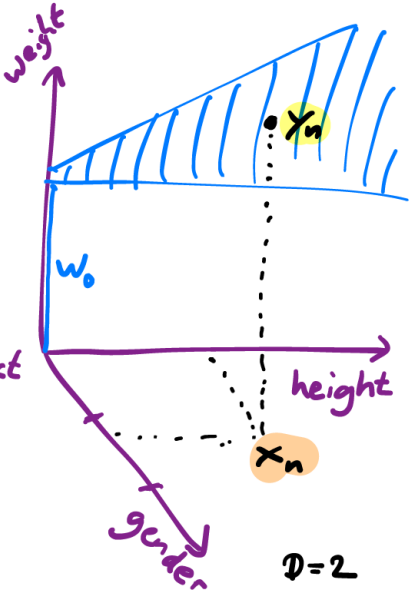
Multiple linear regression

If our data has multiple input dimensions, we obtain multivariate linear regression.

$$\begin{aligned} y_n \approx f(\mathbf{x}_n) &:= w_0 + w_1 x_{n1} + \ldots + w_D x_{nD} \\ &= w_0 + \mathbf{x}_n^\top \begin{pmatrix} w_1 \\ \vdots \\ w_D \end{pmatrix} \\ &=: \tilde{\mathbf{x}}_n^\top \tilde{\mathbf{w}} \end{aligned}$$

inner product

Note that we add a tilde over the input vector, and also the weights, to indicate they now contain the additional offset term (a.k.a. bias term).



$$\tilde{\mathbf{x}}_n := \begin{pmatrix} 1 \\ \mathbf{x}_n \end{pmatrix} = \begin{pmatrix} 1 \\ x_{n1} \\ \vdots \\ x_{nD} \end{pmatrix} \in \mathbb{R}^{D+1}$$

$\tilde{\mathbf{w}}$

Cost functions

Consider the following models.

1-parameter model: $y_n \approx w_0$

2-parameter model: $y_n \approx w_0 + w_1 x_{n1}$

How can we estimate (or guess) values of $\mathbf{w}$ given the data $\mathcal{D}$?

Cost functions: definition

Consider the following models.

1-parameter model: $y_n \approx w_0$

2-parameter model: $y_n \approx w_0 + w_1 x_{n1}$

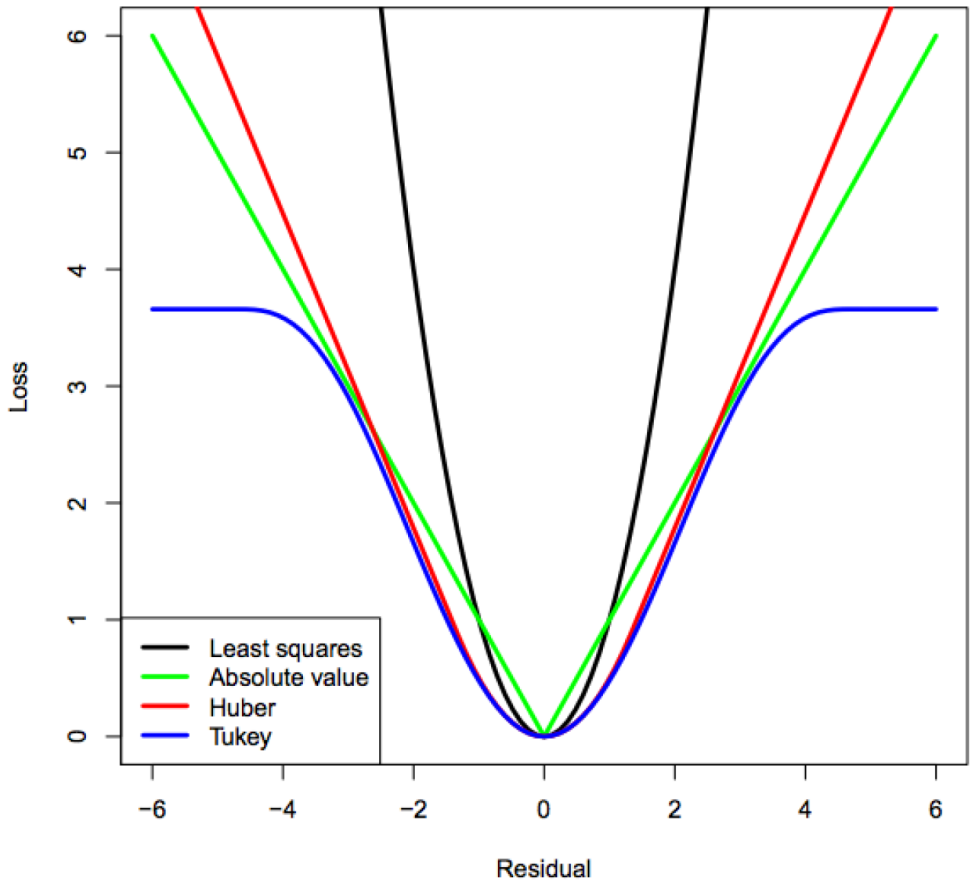
How can we estimate (or guess) values of $\mathbf{w}$ given the data $\mathcal{D}$?

A cost function (or energy, loss, training objective) is used to learn parameters that explain the data well. The cost function quantifies how well our model does - or in other words how costly our mistakes are.

Cost functions: desirable properties

When the target y is real-valued, it is often desirable that the cost is symmetric around 0, since both positive and negative errors should be penalized equally.

Also, our cost function should penalize "large" mistakes and "very-large" mistakes similarly.



Cost functions: desirable properties

- Robust to outliers
- Convexity

No free lunch: it's difficult to have both

Outliers are data examples that are far away from most of the other examples. Unfortunately, they occur more often in reality than you would want them to!

A strictly convex function has a unique global minimum $\mathbf{w}^\star$. For convex functions, every local minimum is a global minimum.

Cost functions: examples

$$\text{MSE}(\mathbf{w}) := \frac{1}{N}\sum_{n=1}^{N}\left[y_n - f_{\mathbf{w}}(\mathbf{x}_n)\right]^2$$

$$\text{MAE}(\mathbf{w}) := \frac{1}{N}\sum_{n=1}^{N}\left|y_n - f_{\mathbf{w}}(\mathbf{x}_n)\right|$$

Huber loss

$$Huber(e) := \begin{cases} \frac{1}{2}e^2 & \text{, if } |e| \leq \delta \\ \delta|e| - \frac{1}{2}\delta^2 & \text{, if } |e| > \delta \end{cases} \quad (1)$$

Huber loss is convex, differentiable, and also robust to outliers. However, setting δ is not an easy task.

Cost function: exercise

Select the best **weights** according to the loss

An exercise for MSE

Compute MSE for 1-param model:

$$\mathcal{L}(w_0) := \frac{1}{N}\sum_{n=1}^{N}\left[y_n - w_0\right]^2$$

	1	2	3	4	5	6	7
$y_1 = 1$ $y_2 = 2$ $y_3 = 3$ $y_4 = 4$							
$\mathrm{MSE}(\mathbf{w}) \cdot N$							
$y_5 = 20$							
$\mathrm{MSE}(\mathbf{w}) \cdot N$							

Some help: $19^2 = 361, 18^2 = 324, 17^2 = 289, 16^2 = 256, 15^2 = 225, 14^2 = 196, 13^2 = 169$.

Cost function: exercise

An exercise for MSE

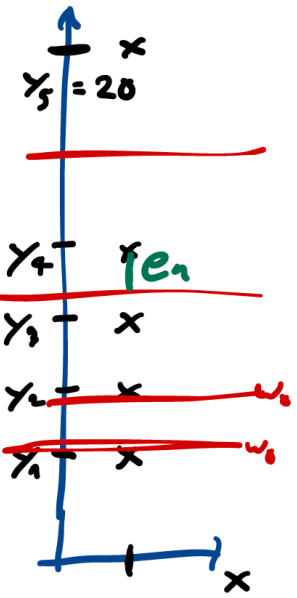
Compute MSE for 1-param model: $f_w(x) = w_0$

$$\mathcal{L}(w_0) := \frac{1}{N}\sum_{n=1}^{N}\left[y_n - w_0\right]^2$$

"$f_w(x)$

$w_0 =$	1	2	3	4	5	6	7
$y_1 = 1$	0^2	1^2	2^2	3^2	.	.	
$y_2 = 2$	1^2	0^2	1^2	2^2	.	.	
$y_3 = 3$	2^2	1^2	0^2	1^2	.	.	
$y_4 = 4$	3^2	2^2	1^2	0^2	.	.	
$\mathrm{MSE}(\mathbf{w}) \cdot N$	14	6	6	14	30	54	
$y_5 = 20$	19^2	18^2	17^2	16^2	15^2	14^2	13^2
$\mathrm{MSE}(\mathbf{w}) \cdot N$	.	.	.	.	.	250	

Some help: $19^2 = 361, 18^2 = 324, 17^2 = 289, 16^2 = 256, 15^2 = 225, 14^2 = 196, 13^2 = 169$.



best model with $w_0 = 4$

best model with 5

Optimization

Learning / Estimation / Fitting

Given a cost function $\mathcal{L}(\mathbf{w})$, we wish to find $\mathbf{w}^\star$ which minimizes the cost:

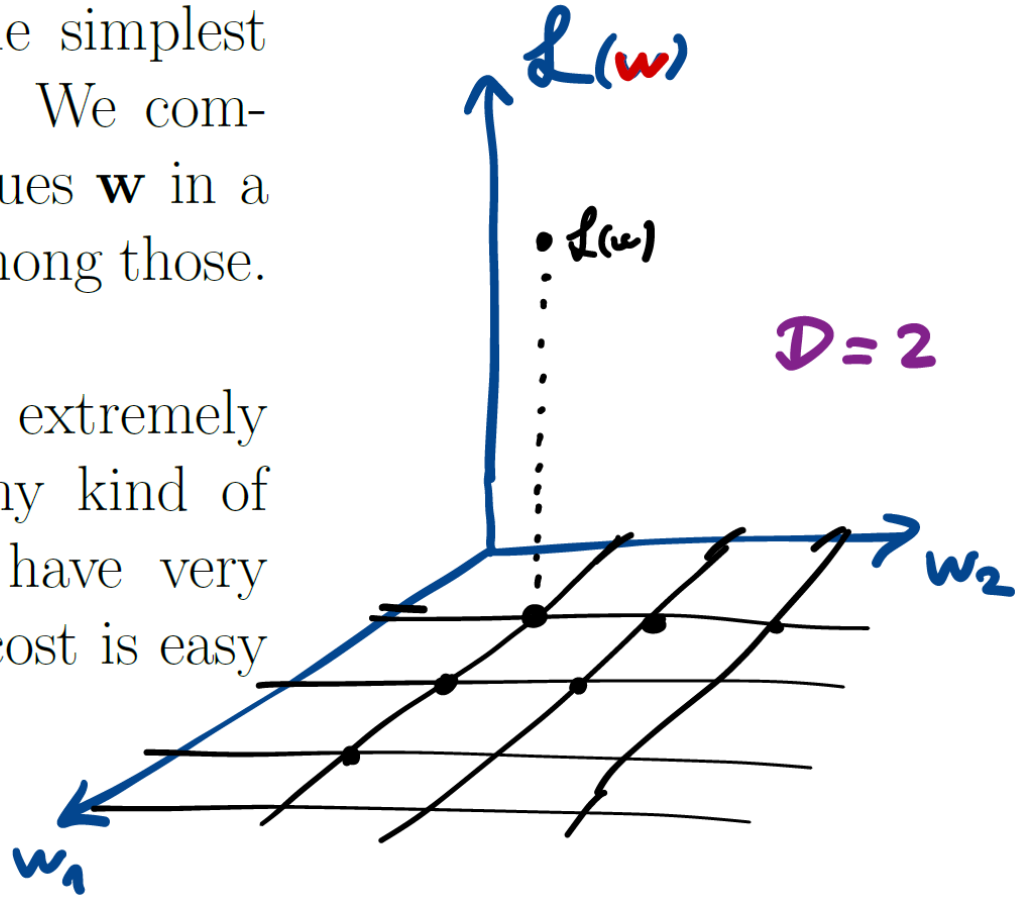
$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \quad \text{subject to } \mathbf{w} \in \mathbb{R}^D$$

This means the *learning* problem is formulated as an optimization problem.
We will use an optimization algorithm to solve the problem (to find a good $\mathbf{w}$).

Optimization: Grid search

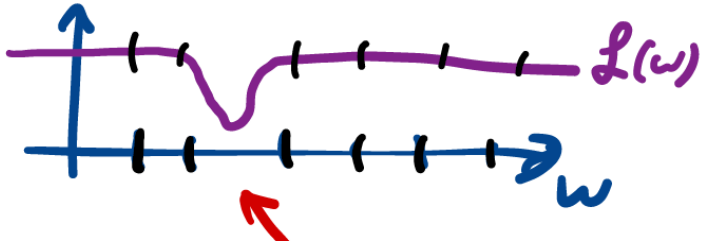
Grid search is one of the simplest optimization algorithms. We compute the cost over all values $\mathbf{w}$ in a grid, and pick the best among those.

This is brute-force, but extremely simple and works for any kind of cost function when we have very few parameters and the cost is easy to compute.



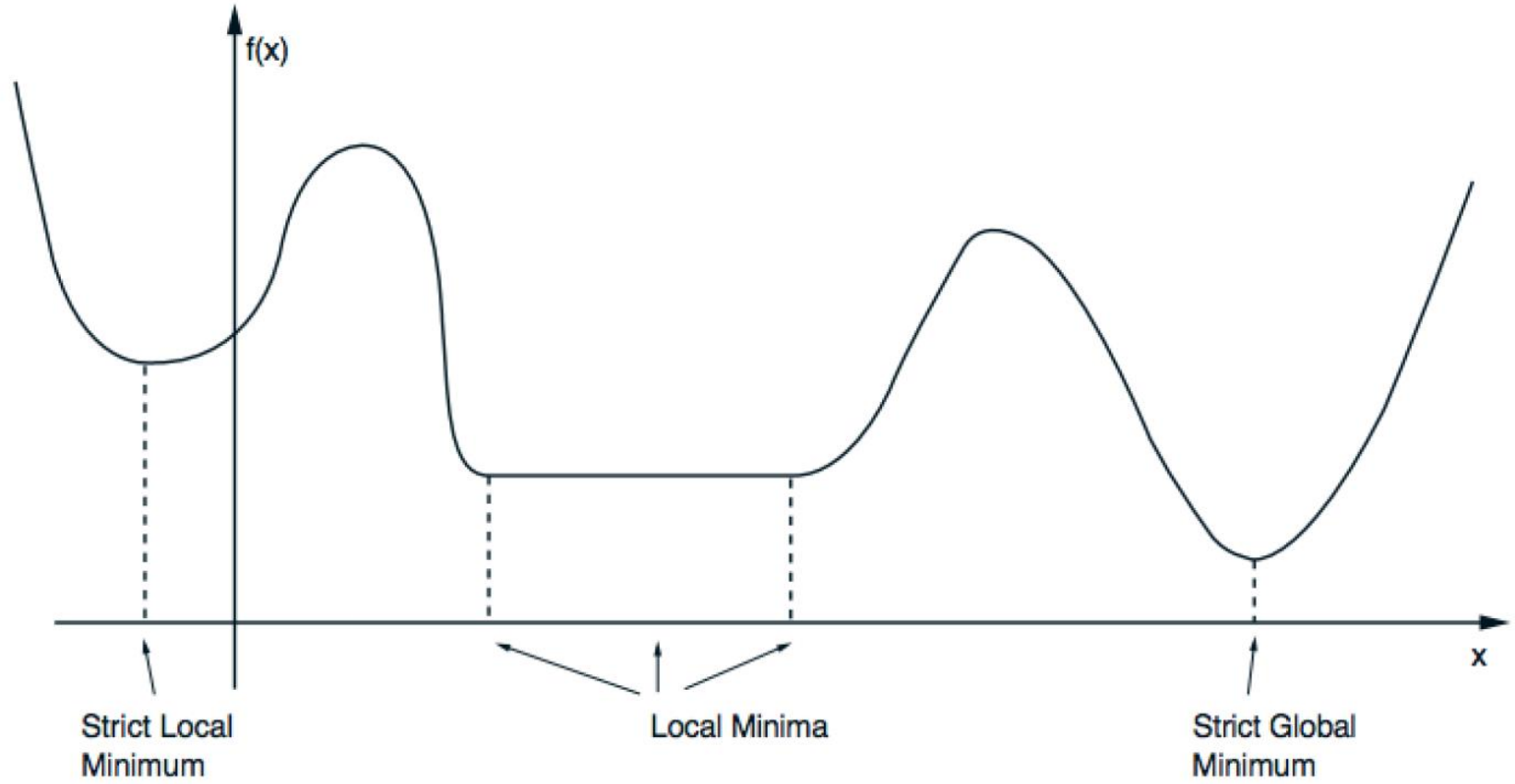
Optimization: Disadvantages of Grid search

1. There is no guarantee that we end up close to an optimum



2. For a large number of parameters D, grid search results in an exponential computational complexity
   - Example: for 10 possible values, we need to check *10^D*

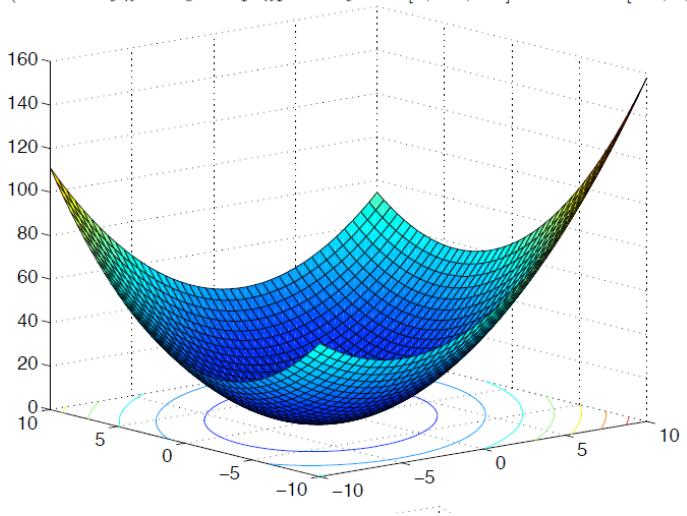
Optimization landscapes



Optimization: gradient descent

A gradient (at a point) is the slope of the tangent to the function (at that point). It points to the direction of largest increase of the function.

For a 2-parameter model, MSE($\mathbf{w}$) and MAE($\mathbf{w}$) are shown below.
(We used $\mathbf{y}_n \approx w_0 + w_1 x_{n1}$ with $\mathbf{y}^\top = [2, -1, 1.5]$ and $\mathbf{x}^\top = [-1, 1, -1]$).



Definition of the gradient:

$$\nabla \mathcal{L}(\mathbf{w}) := \left[\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_D} \right]^\top$$

This is a vector, $\nabla \mathcal{L}(\mathbf{w}) \in \mathbb{R}^D$.

Gradient Descent

To minimize the function, we iteratively take a step in the (opposite) direction of the gradient

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma \nabla \mathcal{L}(\mathbf{w}^{(t)})$$

where $\gamma > 0$ is the step-size (or learning rate). Then repeat with the next t.

Optimization: gradient descent

For linear regression

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1D} \\ x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{ND} \end{bmatrix}$$

← data point 1

We define the error vector $\mathbf{e}$:

$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{w}$$

and MSE as follows:

$$\mathcal{L}(\mathbf{w}) := \frac{1}{2N} \sum_{n=1}^{N} (y_n - \mathbf{x}_n^\top \mathbf{w})^2$$

$$= \frac{1}{2N} \mathbf{e}^\top \mathbf{e}$$

$$= \frac{1}{2N} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

then the gradient is given by

$$\nabla \mathcal{L}(\mathbf{w}) = -\frac{1}{N} \mathbf{X}^\top \mathbf{e}$$

matrixcalculus.org

$$\frac{\partial}{\partial w_1} \mathcal{L} = \frac{1}{2N} \sum_{n=1}^{N} -2(y_n - \mathbf{x}_n^\top \mathbf{w}) x_{n1}$$

$$= -\frac{1}{N} (\mathbf{X}_{:1})^\top \mathbf{e}$$

$$\vdots$$

$$\frac{\partial}{\partial w_D} \mathcal{L} = -\frac{1}{N} (\mathbf{X}_{:D})^\top \mathbf{e}$$

Optimization: stochastic gradient descent

Sum Objectives. In machine learning, most cost functions are formulated as a sum over the training examples, that is

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n(\mathbf{w}) \ ,$$

↖ cost of one datapoint n

where $\mathcal{L}_n$ is the cost contributed by the n-th training example.

Q: What are the $\mathcal{L}_n$ for linear MSE?

The SGD Algorithm. The stochastic gradient descent (SGD) algorithm is given by the following update rule, at step t:

① Sample one datapoint $n \in \{1, \dots, N\}$ uniformly at random

② $$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma \nabla \mathcal{L}_n(\mathbf{w}^{(t)}) \ .$$

g „stochastic gradient"

Optimization: stochastic gradient descent

Theoretical Motivation. *Idea:* Cheap but unbiased estimate of the gradient!
In expectation over the random choice of n, we have

$$\mathbb{E}\left[\nabla \mathcal{L}_n(\mathbf{w})\right] = \nabla \mathcal{L}(\mathbf{w})$$

which is the true gradient direction. (check!)

$$g = \mathbb{E}\left[\nabla \ell_n(w)\right] = \frac{1}{N}\sum_{n=1}^{N} \nabla \ell_n(w) = \nabla \frac{1}{N}\sum_n \ell_n = \nabla \ell(w)$$

$\Rightarrow$ unbiased

Optimization: Mini-batch SGD

Mini-batch SGD. There is an intermediate version, using the update direction being

$$\mathbf{g} := \frac{1}{|B|} \sum_{n \in B} \nabla \mathcal{L}_n(\mathbf{w}^{(t)})$$

again with

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma \mathbf{g}.$$

In the above gradient computation, we have randomly chosen a subset $B \subseteq [N]$ of the training examples. For each of these selected examples n, we compute the respective gradient $\nabla \mathcal{L}_n$, at the same current point $\mathbf{w}^{(t)}$.

Randomly select $|B|$ datapoints

$B \subseteq \{1, \dots N\}$

Examples:

- $B = \{n\}$ $\quad |B| = 1$
  $\Rightarrow$ SGD
- $B = \{1, 2, \dots, N\}$
  $\Rightarrow$ (full) gradient descent
- $|B| = 5$
  $\Rightarrow$ mini-batch SGD

Optimization: Computational cost

Computational cost. For linear MSE, what is the complexity (# operations) of computing the stochastic gradient?
(using only $|B| = 1$ data examples)

a) starting from $\mathbf{w}$?

GD: $\mathcal{L}(w) = \frac{1}{N}\sum_{n=1}^{N} \frac{1}{2}(y_n - x_n^T w)^2$

$\nabla\mathcal{L}(w) = -\frac{1}{N} X^T (y - Xw)$, $e \in \mathbb{R}^N$

cost: $\mathcal{O}(N \cdot D)$

SGD: $\mathcal{L}_n(w) = \frac{1}{2}(y_n - x_n^T w)^2$

$\nabla\mathcal{L}_n(w) = -x_n^T (y_n - x_n^T w)$, $e_n \in \mathbb{R}$

cost: $\mathcal{O}(D)$

N times cheaper per iteration

Optimization: sub gradient SGD

Subgradients

A vector $\mathbf{g} \in \mathbb{R}^D$ such that

$$\mathcal{L}(\mathbf{u}) \geq \mathcal{L}(\mathbf{w}) + \mathbf{g}^\top(\mathbf{u} - \mathbf{w}) \quad \forall \mathbf{u}$$

is called a subgradient to the function $\mathcal{L}$ at $\mathbf{w}$.

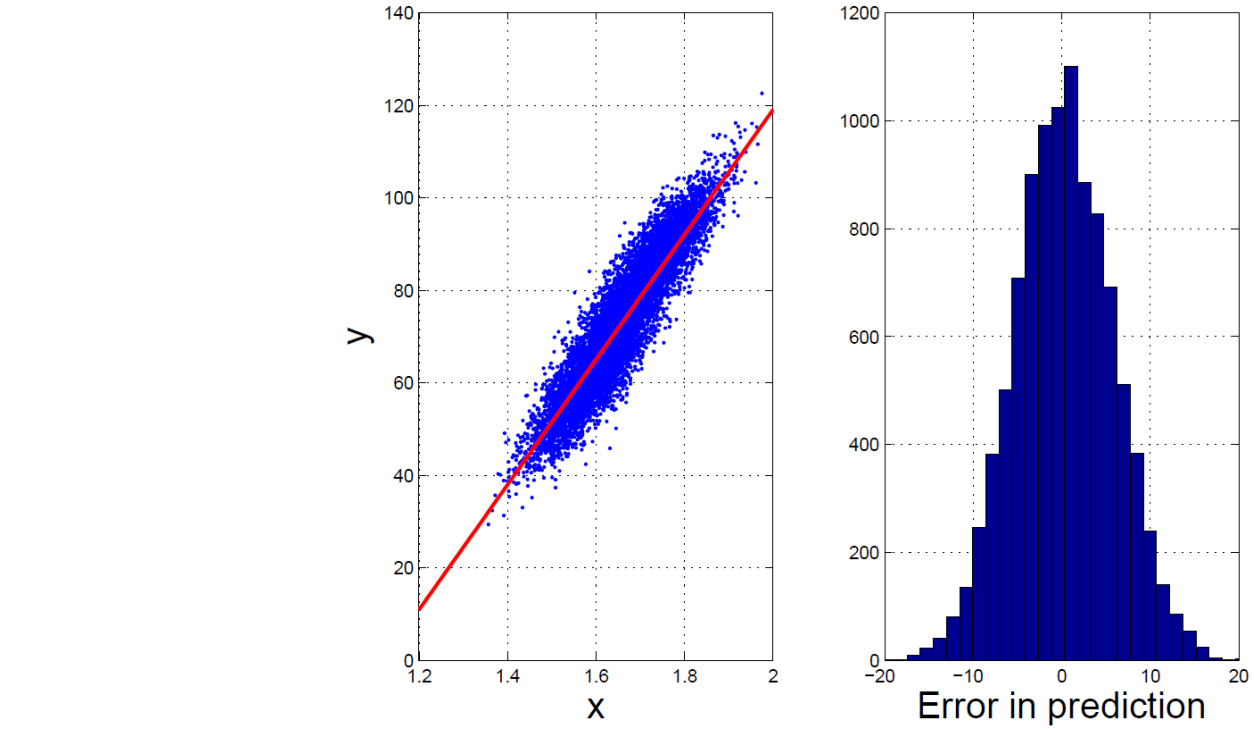
This definition makes sense for objectives $\mathcal{L}$ which are not necessarily differentiable (and not even necessarily convex).

Identical to the gradient descent algorithm, but using a subgradient instead of gradient. Update rule

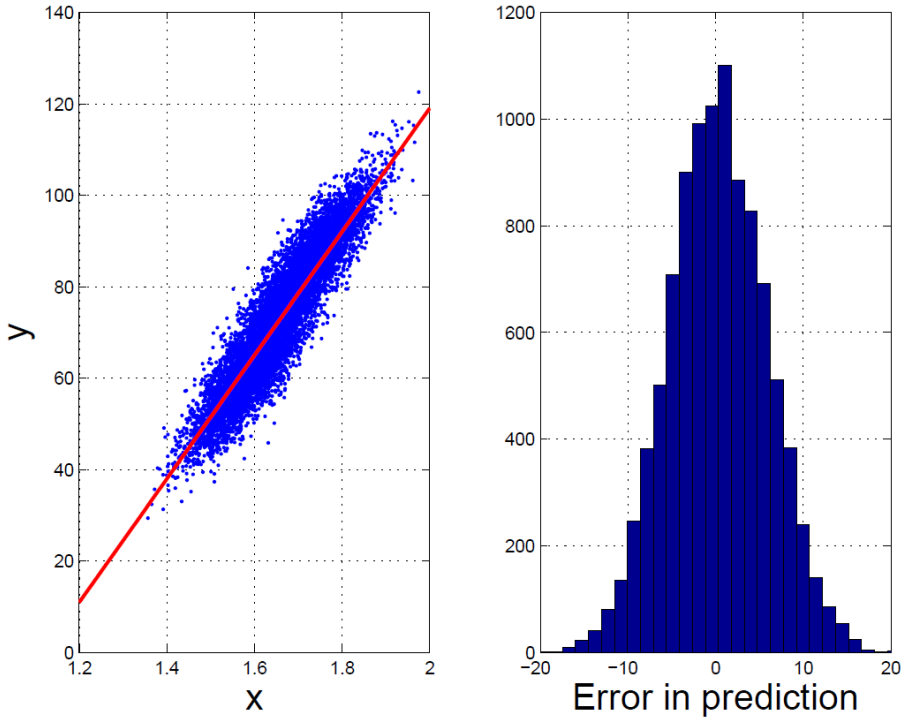
$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma\, \mathbf{g}$$

for $\mathbf{g}$ being a subgradient to $\mathcal{L}$ at the current iterate $\mathbf{w}^{(t)}$.

Maximum likelihood estimator (MLE)



Maximum likelihood estimator (MLE)



Another take at MSE from a probabilistic point of view.

Recall the definition of a Gaussian random variable in $\mathbb{R}$ with mean μ and variance σ^2. It has a density of

$$p(y \mid \mu, \sigma^2) = \mathcal{N}(y \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-\mu)^2}{2\sigma^2}\right].$$

MLE: probabilistic view point for MSE

We assume that our data is generated by the model,

$$y_n = \mathbf{x}_n^\top \mathbf{w} + \epsilon_n,$$

where the ϵ_n (the noise) is a zero-mean Gaussian random variable with variance σ^2 and the noise that is added to the various samples is independent of each other, and independent of the input. Note that the model $\mathbf{w}$ is unknown.

Therefore, given N samples, the likelihood of the data vector $\mathbf{y} = (y_1, \cdots, y_N)$ given the input $\mathbf{X}$ (each row is one input) and the model $\mathbf{w}$ is equal to

$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} p(y_n \mid \mathbf{x}_n, \mathbf{w}) = \prod_{n=1}^{N} \mathcal{N}(y_n \mid \mathbf{x}_n^\top \mathbf{w}, \sigma^2).$$

The probabilistic view point is that we should maximize this likelihood over the choice of model $\mathbf{w}$. I.e., the "best" model is the one that maximizes this likelihood.

MLE: log-likelihood

Defining cost with log-likelihood

Instead of maximizing the likelihood, we can take the logarithm of the likelihood and maximize it instead. Expression is called the log-likelihood (LL).

$$\mathcal{L}_{\mathrm{LL}}(\mathbf{w}) := \log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = -\frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - \mathbf{x}_n^\top \mathbf{w})^2 + \text{cnst}.$$

Compare the LL to the MSE (mean squared error)

$$\mathcal{L}_{\mathrm{LL}}(\mathbf{w}) = -\frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - \mathbf{x}_n^\top \mathbf{w})^2 + \text{cnst}$$

$$\mathcal{L}_{\mathrm{MSE}}(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (y_n - \mathbf{x}_n^\top \mathbf{w})^2$$

MLE can also be interpreted as finding the model under which the observed data is most likely to have been generated from (probabilistically). This interpretation has some advantages that we discuss now.

Exercises

1 Computing the Cost Function

In this exercise, we will focus on simple linear regression which takes the following form,

$$y_n \approx f(x_{n1}) = w_0 + w_1 x_{n1}. \tag{1}$$

We will use height as the input variable x_{n1} and weight as the output variable y_n. The coefficients w_0 and w_1 are also called *model parameters*. We will use a mean-square-error (MSE) function defined as follows,

$$\mathcal{L}(w_0, w_1) = \frac{1}{2N} \sum_{n=1}^{N} \left(y_n - f(x_{n1})\right)^2 = \frac{1}{2N} \sum_{n=1}^{N} \left(y_n - w_0 - w_1 x_{n1}\right)^2. \tag{2}$$

Our goal is to find $w_0^\star$ and $w_1^\star$ that minimize this *cost*.

Let us start by the array data type in *NumPy*. We store all the (y_n, x_{n1}) pairs in a vector and a matrix as shown below.

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \qquad \widetilde{\boldsymbol{X}} = \begin{bmatrix} 1 & x_{11} \\ 1 & x_{21} \\ \vdots & \vdots \\ 1 & x_{N1} \end{bmatrix} \tag{3}$$

Exercise 1:

To understand this data format, answer the following warmup questions:

- What does each *column* of $\tilde{\boldsymbol{X}}$ represent?
- What does each *row* of $\tilde{\boldsymbol{X}}$ represent?
- Why do we have 1's in $\tilde{\boldsymbol{X}}$?
- If we have heights and weights of 3 people, what would be the size of $\boldsymbol{y}$ and $\tilde{\boldsymbol{X}}$? What would $\tilde{\boldsymbol{X}}_{32}$ represent?
- In `helpers.py`, we have already provided code to form arrays for $\boldsymbol{y}$ and $\tilde{\boldsymbol{X}}$. Have a look at the code, and make sure you understand how they are constructed.
- Check if the sizes of the variables make sense (use functions `shape`).

a) Now we will compute the MSE. Let us introduce the vector notation $\boldsymbol{e} = \boldsymbol{y} - \tilde{\boldsymbol{X}}\boldsymbol{w}$, for given model parameters $\boldsymbol{w} = [w_0, w_1]^\top$. Prove that the MSE can also be rewritten in terms of the vector $\boldsymbol{e}$, as

$$\mathcal{L}(\boldsymbol{w}) = \ldots \tag{4}$$

b) Complete the implementation of the notebook function `compute_loss(y, tx, w)`. You can start by setting $\boldsymbol{w} = [1, 2]^\top$, and test your function.

2 Grid Search

Now we are ready to implement our first optimization algorithm: Grid Search. Revise the lecture notes.

Exercise 2:

a) Fill in the notebook function `grid_search(y, tx, w0, w1)` to implement grid search. You will have to write one for-loop per dimension, and compute the cost function for each setting of w_0 and w_1. Once you have all values of cost function stored in the variable `loss`, the code finds an approximate minimum (as discussed in the class).

The code should print the obtained minimum value of the cost function along with the found $w_0^\star$ and $w_1^\star$. It should also show a contour plot and the plot of the fit, as shown in Figure 1.

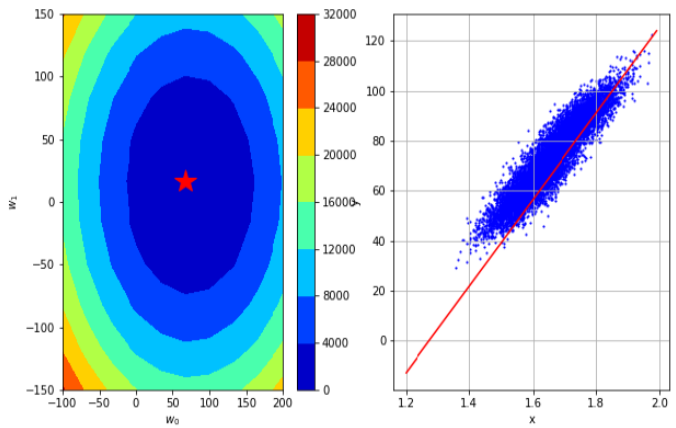


Figure 1: Grid Search Visualization

b) Does this look like a good estimate? Why not? What is the problem? Why is the MSE plot not smooth?

Repeat the above exercise by changing the grid spacing to 10 instead of 50. Compare the new fit to the old one.

c) Discuss with your peers:

- To obtain an accurate fit, do you need a coarse grid or a fine grid?
- Try different values of grid spacing. What do you observe?
- How does increasing the number of values affect the computational cost? How fast or slow does your code run?

3 Gradient Descent

In the lecture, we derived the following expressions for the gradient (the vector of partial derivatives) of the MSE for linear regression,

$$\frac{\partial \mathcal{L}(w_0, w_1)}{\partial w_0} = -\frac{1}{N}\sum_{n=1}^{N}\left(y_n - w_0 - w_1 x_{n1}\right) = -\frac{1}{N}\sum_{n=1}^{N} e_n \tag{5}$$

$$\frac{\partial \mathcal{L}(w_0, w_1)}{\partial w_1} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_n - w_0 - w_1 x_{n1}\right)x_{n1} = -\frac{1}{N}\sum_{n=1}^{N} e_n x_{n1} \tag{6}$$

Denoting the gradient by $\nabla\mathcal{L}(\boldsymbol{w})$, we can write these operations in vector form as follows,

$$\nabla\mathcal{L}(\boldsymbol{w}) := \begin{bmatrix} \frac{\partial \mathcal{L}(w_0,w_1)}{\partial w_0} & \frac{\partial \mathcal{L}(w_0,w_1)}{\partial w_1} \end{bmatrix} = -\frac{1}{N}\begin{bmatrix} \sum_{n=1}^{N} e_n \\ \sum_{n=1}^{N} e_n x_{n1} \end{bmatrix} = -\frac{1}{N}\tilde{\boldsymbol{X}}^\top \boldsymbol{e} \tag{7}$$

Exercise 3:

a) Now implement a function that computes the gradients. Implement the notebook function `compute_gradient(y, tx, w)` using Equation (7). Verify that the function returns the right values. First, manually compute the gradients for hand-picked values of $\boldsymbol{y}$, $\tilde{\boldsymbol{X}}$, and $\boldsymbol{w}$ and compare them to the output of `compute_gradient`.

b) Once you make sure that your gradient code is correct, get some intuition about the gradient values:

Compute the gradients for

- $w_0 = 100$ and $w_1 = 20$
- $w_0 = 50$ and $w_1 = 10$

What do the values of these gradients tell us? For example, think about the norm of this vector. In which case are they bigger? What does that mean?

Hint: Imagine a quadratic function and estimate its gradient near its minimum and far from it.

Hint 2: As we know from the lecture notes, the update rule for gradient descent at step t is

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \gamma\nabla\mathcal{L}(\boldsymbol{w}^{(t)}) \tag{8}$$

where $\gamma > 0$ is the step size, and $\nabla\mathcal{L} \in \mathbb{R}^2$ is the gradient vector.

c) Fill in the notebook function `gradient_descent(y, tx, initial_w, ...)`. Run the code and visualize the iterations. Also, look at the printed messages that show $\mathcal{L}$ and values of $w_0^{(t)}$ and $w_1^{(t)}$. Take a detailed look at these plots,

- Is the cost being minimized?
- Is the algorithm converging? What can be said about the convergence speed?
- How good are the final values of w_1 and w_0 found?

d) Now let's experiment with the value of the step size and initialization parameters and see how they influences the convergence. In theory, gradient descent converges to the optimum on convex functions, when the value of the step size is chosen appropriately.

- Try the following values of step size: 0.001, 0.01, 0.5, 1, 2, 2.5. What do you observe? Did the procedure converge?
- Try different initializations with fixed step size $\gamma = 0.1$, for instance:
  - $w_0 = 0$, $w_1 = 0$
  - $w_0 = 100$, $w_1 = 10$
  - $w_0 = -1000$, $w_1 = 1000$

  What do you observe? Did the procedure converge?

4 Stochastic Gradient Descent

Exercise 4:

Let us implement stochastic gradient descent. Recall from the lecture notes that the update rule for stochastic gradient descent on an objective function $\mathcal{L}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{w})$ at step t is

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \gamma \nabla \mathcal{L}_n(\boldsymbol{w}^{(t)}) \ . \tag{9}$$

HINT: You can use the function `batch_iter()` in the file of `helpers.py` to generate mini-batch data for stochastic gradient descent.

5 Effect of Outliers and MAE Cost Function

In the course we talked about *outliers*. Outliers might occur due to measurement errors. For example, in the weight/height data, a coding mistake could introduce points whose weight is measured in pounds rather than kilograms.

Such outlier points may have a strong influence on model parameters. For example, MSE (the one you implemented above) is known to be sensitive to outliers, as discussed in the class.

Exercise 5:

Let's simulate the presence of two outliers, and their effect on linear regression under MSE cost function,

- Reload the data through function `load_data()` by setting `sub_sample=True` to keep only a few data examples.
- Plot the data. You should get a cloud of points similar, but less dense, than what you saw before with the whole dataset.
- As before, find the values of w_0, w_1 to fit a linear model (using MSE cost function), and plot the resulting f together with the data points.
- Now we will add two outliers points simulating the mistake that we entered the weights in pounds instead of kilograms. For example, you can achieve this by setting `add_outlier=True` in `load_data()`. Feel free to add more outlier points.
- Fit the model again to the augmented dataset with the outliers. Does it look like a good fit?

One way to deal with outliers is to use a more *robust* cost function, such as the Mean Absolute Error (MAE), as discussed in the class.