1 What is model training?

In machine learning, many problems boil down (sprowadza się do) to optimization. This means finding the best settings (or parameters) for a model to make its predictions as accurate (dokładne) as possible. We measure accuracy using a loss function, which tells us how far off (daleko są) our predictions are from the actual data. The process of "training" or "fitting" a model is really about tweaking (dostrajaniu) its parameters to make this loss as small as possible.

2 Discuss linear regression

$$\hat{y}^{(i)} = \sum_{j=1}^{d} w_j x_j^{(i)} + b = w^T x^{(i)} + b$$

- w is a vector of **weights**, one for each feature, controlling how much each feature affects the prediction.
- b is the **bias**, a constant that shifts the line up or down.
- $\hat{y}^{(i)}$ is the predicted value for the i-th sample.

3 How could you Simplifying a code?

To make coding easier, we can roll the bias b into the weight vector. We do this by adding a "1" to every feature vector:

$$\tilde{x}^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix}$$

$$\tilde{w} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

Now, the prediction is just:

$$\hat{y}^{(i)} = b \cdot 1 + w_1 \cdot x_1^{(i)} + w_2 \cdot x_2^{(i)} + \cdots + w_d \cdot x_d^{(i)} = \tilde{w}^T \tilde{x}^{(i)}$$

This trick lets us handle (obsługiwać) all parameters (including bias) in one vector, which simplifies our math and code.

4 Propose a loss function for linear regression

Loss Function: Mean Squared Error (MSE)

To figure out how good our predictions are, we need a loss function. For linear regression, we use the **Mean Squared Error (MSE)**, which measures the average squared difference between actual targets $y^{(i)}$ and predictions $\hat{y}^{(i)}$:

$$L = \frac{1}{N} \sum_{i=1}^{N} \left(\hat{y}^{(i)} - y^{(i)} \right)^2$$

- N is the number of samples.
- The smaller the MSE, the better our model fits the data.

Our goal is to adjust (dostosować) $\tilde{w}$ to make this loss as small as possible.

5 Why can't we use mathematical methods to find the bottom of the bowl in machine learning? What method should we use?

The MSE is a function of our parameters, and we want to find the lowest point of this function — think of it as finding the bottom of a bowl. For simple problems like this, we could solve it directly with math, but in machine learning, the functions are often too complex (złożony) for that. Instead, we use **gradient descent**, an iterative method that "walks" toward the minimum.

6 What is the fundamental idea of gradient descent?

Gradient descent is a simple but powerful method for finding the minimum of a function. Here's how it works:

1. Start at some point on the function.
2. Find which direction leads (prowadzi) downhill (w dół) most steeply (najbardziej stromo) (the negative gradient).
3. Take a small step in that direction.
4. Repeat until you reach the bottom.

Mathematically, we update our position using:

$$w_{t+1} = w_t - \alpha \nabla L(w_t)$$

Where:

- w_t is our current position.
- $\nabla L(w_t)$ is the gradient (slope) at that position.
- α is the learning rate (how big a step to take).
- w_{t+1} is our new position.

7 Why is choosing the learning rate so important?

The learning rate α is important – too large and we might overshoot, too small and progress will be slow.

8 What information does a gradient give us?

Calculate the Gradient: The gradient tells us the direction of the steepest (najbardziej stromy) increase in the loss (straty). We want to go the opposite way to decrease it.

9 Derive gradient formula step by step

First, recall that our loss function is:

$$L = \frac{1}{N} \sum_{i=1}^{N} \left(\hat{y}^{(i)} - y^{(i)} \right)^2$$

And our predictions are:

$$\hat{y}^{(i)} = \tilde{w}^T \tilde{x}^{(i)}$$

Substituting (podstawiając) the prediction into the loss:

$$L = \frac{1}{N} \sum_{i=1}^{N} \left(\tilde{w}^T \tilde{x}^{(i)} - y^{(i)} \right)^2$$

In matrix form, this is:

$$L = \frac{1}{N} (\tilde{X}\tilde{w} - y)^T (\tilde{X}\tilde{w} - y)$$

because:

$$(a-b)^T(a-b) = a^Ta - 2a^Tb + b^Tb$$

$$a^Tb = b^Ta$$

To find the gradient, we expand (rozkładamy) this:

$$L = \frac{1}{N}(\tilde{w}^T\tilde{X}^T\tilde{X}\tilde{w} - 2y^T\tilde{X}\tilde{w} + y^Ty)$$

Taking the gradient with respect (względem) to $\tilde{w}$:

$$\nabla_{\tilde{w}}L = \frac{1}{N}(2\tilde{X}^T\tilde{X}\tilde{w} - 2\tilde{X}^Ty)$$

Since $\tilde{X}\tilde{w} = \hat{y}$, we get:

$$\nabla_{\tilde{w}}L = \frac{2}{N}\tilde{X}^T(\hat{y} - y)$$

Where:

- $\tilde{X}$ is the matrix of all $\tilde{x}^{(i)}$ samples stacked together.

$$\tilde{X} = \begin{bmatrix} x_{11} & 1 \\ x_{21} & 1 \\ x_{31} & 1 \end{bmatrix}$$

- y is the vector of all true targets.
- $\hat{y}$ is the vector of all predictions.

2. **Update the Parameters:** Move the weights a small step in the opposite direction of the gradient:

$$\tilde{w} \leftarrow \tilde{w} - \alpha\nabla_{\tilde{w}}L$$

3. **Repeat:** Keep updating until the loss stops getting smaller (or we've done enough steps).

10 Analytic Solution

Gradient:

$$\nabla_{\tilde{w}} L = \frac{1}{N} \left(2\tilde{X}^T \tilde{X} \tilde{w} - 2\tilde{X}^T y \right)$$

where:

- $\tilde{X}$ is the **design matrix** (with a bias term).
- y is the **vector of target values**.
- $\tilde{w}$ is the **weight vector**.
- N is the **number of samples**.

Setting the Gradient to Zero.
To find the **minimum** of the loss function L, we solve:

$$\frac{1}{N} \left(2\tilde{X}^T \tilde{X} \tilde{w} - 2\tilde{X}^T y \right) = 0$$

Steps:

1. Multiply by $N/2$ on both sides to simplify:

$$\tilde{X}^T \tilde{X} \tilde{w} - \tilde{X}^T y = 0$$

2. Rearrange to solve for $\tilde{w}$:

$$\tilde{X}^T \tilde{X} \tilde{w} = \tilde{X}^T y$$

3. Multiply by the inverse of $\tilde{X}^T \tilde{X}$ (assuming it is invertible):

$$\tilde{w} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T y$$

This formula provides the **closed-form solution** for linear regression using the Normal Equation.

11 exercise

In this laboratory session, you will implement linear regression with gradient descent from scratch (od podstaw) . The focus is on understanding the optimization process through hands-on (praktyczną) implementation. You will analyze the behavior of the algorithm under different conditions and visualize the results to gain insights (uzyskać wgląd) into the training dynamics.

11.1 Generate Synthetic Data

Generate a synthetic (syntetyczny) dataset with a known linear relationship. Create 100 data points with a single feature, using a true weight (rzeczywista waga) of 2 and a bias of 5, plus a small amount of Gaussian noise. This will create data that follows the relationship:

$$y = 2x + 5 + \epsilon$$

where ϵ is random noise.
Set a random seed for reproducibility. Your feature matrix X should have shape $(100, 1)$ and your target vector y should have shape $(100,)$.

11.2 Prepare the Data

To include the bias term (wyraz przesunięcia) in our weight vector, add a column of 1s (kolumna jedynek) to your feature matrix X. This allows us to represent both the bias and feature weight in a single vector, simplifying our implementation.

11.3 Initialize Parameters

Initialize your weight vector with zeros. Since we have one feature plus the bias term, your weight vector should have shape $(2,)$, where the first element represents the bias and the second represents the feature weight.

11.4 Implement Gradient Descent

Compute the gradient (formula is provided in the introduction) and update weights.

11.5 Run the Training Loop

Put it all together to train the model. Track values of the loss function during this process.

11.6 Check Your Results

After training, compare (porównaj) your weights to the true ones (prawdziwymi). Plot the data and compare the fitted model with the ground truth line (linią rzeczywistych wartości).

11.7 Experiment with Learning Rate

The learning rate α controls how big of a step we take during each iteration of gradient descent. Try training your model with different learning rates:

$$\alpha \in [10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$$

For each value:

- Plot the loss curve (loss vs. epoch)
- Record the final weights and final loss
- Observe how quickly (or slowly) the model converges

What happens when the learning rate is too small? Too large? Is there an optimal value in this range?

11.8 Explore Training Duration

Set `num_epochs` to larger values like 100 or 1000 and observe the training process:

- Does the loss continue to decrease throughout training, or does it plateau?
- At what point does the model effectively converge?
- Plot the loss curve for the extended training and identify where diminishing returns begin
- Calculate how close your final weights are to the true weights after different numbers of epochs

This exercise helps you understand when to stop training to avoid wasting computational resources.

11.9 Visualize the Loss Landscape

Create a visualization of the loss function across different weight values:

- Generate a grid of weight values
- Calculate the loss at each point in this grid
- Create a contour plot showing the "landscape" of the loss function
- Mark your initial weights, final weights, and the true weights on this plot

This visualization will help you understand the shape of the objective function you're optimizing and how gradient descent navigates this landscape to find the minimum.

11.10 Analytical Solution

For linear regression, the closed-form solution is:

$$\hat{w} = (X^T X)^{-1} X^T y$$

Compare the parameters and final MSE you get via gradient descent with those from the analytical solution. Are they similar?

11.11 High Dimensional Example

Extend your implementation to work with higher-dimensional data. Generate a synthetic dataset with 10 features ($d = 10$) and:

- Create new true weight vector and bias
- Generate the higher-dimensional X and corresponding y
- Train your model using gradient descent on this data
- Compare the training process with the 1D case:
  - Does it take more epochs to converge?
  - How does the learning rate affect convergence in higher dimensions?
  - Compare the final loss and accuracy of your model
  - Visualize how well your model fits the data (you may need to use partial dependence plots or other visualization techniques for high-dimensional data)