EE 046211 - Technion - Deep Learning

HW1 - Optimization and Automatic Differentiation

Keyboard Shortcuts

- Run current cell: **Ctrl + Enter**
- Run current cell and move to the next: **Shift + Enter**
- Show lines in a code cell: **Esc + L**
- View function documentation: **Shift + Tab** inside the parenthesis or `help(name_of_module)`
- New cell below: **Esc + B**
- Delete cell: **Esc + D, D** (two D's)

Students Information

- Fill in

Name	Campus Email	ID
Nir Elfasi	nirelfasi@campus.technion.ac.il	206094567
Itamar Nierenberg	itamarnie@campus.technion.ac.il	205939010

Submission Guidelines

- Maximal garde: 100.
- Submission only in **pairs**.
  - Please make sure you have registered your group in Moodle (there is a group creation component on the Moodle where you need to create your group and assign members).
- **No handwritten submissions.** You can choose whether to answer in a Markdown cell in this notebook or attach a PDF with your answers.
- SAVE THE NOTEBOOKS WITH THE OUTPUT, CODE CELLS THAT WERE NOT RUN WILL NOT GET ANY POINTS!
- What you have to submit:
  - If you have answered the questions in the notebook, you should submit this file only, with the name: `ee046211_hw1_id1_id2.ipynb` .
  - If you answered the questionss in a different file you should submit a `.zip` file with the name `ee046211_hw1_id1_id2.zip` with content:
    - `ee046211_hw1_id1_id2.ipynb` - the code tasks
    - `ee046211_hw1_id1_id2.pdf` - answers to questions.
  - No other file-types (`.py` , `.docx` ...) will be accepted.
- Submission on the course website (Moodle).
- **Latex in Colab** - in some cases, Latex equations may no be rendered. To avoid this, make sure to not use *bullets* in your answers ("* some text here with Latex equations" -> "some text here with Latex equations").

Working Online and Locally

- You can choose your working environment:
  1. `Jupyter Notebook`, **locally** with Anaconda or **online** on Google Colab
     - Colab also supports running code on GPU, so if you don't have one, Colab is the way to go. To enable GPU on Colab, in the menu: `Runtime` $\rightarrow$ `Change Runtime Type` $\rightarrow$ `GPU`.
  2. Python IDE such as PyCharm or Visual Studio Code.
     - Both allow editing and running Jupyter Notebooks.
- Please refer to `Setting Up the Working Environment.pdf` on the Moodle or our GitHub (https://github.com/taldatech/ee046211-deep-learning) to help you get everything installed.
- If you need any technical assistance, please go to our Piazza forum (`hw1` folder) and describe your problem (preferably with images).

Agenda

Part 1 - Theory

- You can choose whether to answser these straight in the notebook (Markdown + Latex) or use another editor (Word, LyX, Latex, Overleaf...) and submit an additional PDF file, **but no handwritten submissions**.
- You can attach additional figures (drawings, graphs,...) in a separate PDF file, just make sure to refer to them in your answers.

- $\LaTeX$ Cheat-Sheet (to write equations)
  - Another Cheat-Sheet

Question 1 - Convergence of Gradient Descent

Recall from the lecture notes:

- **Definition**: A function f is β-smooth if:

$$\forall w_1, w_2 \in \mathbb{R}^d : ||\nabla f(w_1) - \nabla f(w_2)|| \leq \beta ||w_1 - w_2||$$

- **Lemma**: If f is β-smooth then

$$f(w_1) - f(w_2) - \nabla f(w_2)^T(w_1 - w_2) \leq \frac{\beta}{2} ||w_1 - w_2||^2$$

Prove the lemma.

Hints:

- Represent f as an integral: $f(x) - f(y) = \int_0^1 \nabla f(y + t(x-y))^T(x-y)dt$
- Make use of Cauchy-Schwarz.

Answer

- We will show an additional argument:
- 1. $\forall x \in \mathbb{R}$ and $\forall g : \mathbb{R} \to \mathbb{R} \Rightarrow g(x) \leq |g(x)| \Rightarrow \int g(x)dx \leq \int |gx()|dx$
- 1. $\int_0^1 \nabla f(w_2)^T(w_1 - w_2)dt = \nabla f(w_2)^T(w_1 - w_2)$
- First we will use the first hint and represent $f(w_1) - f(w_2)$ as an integral:

$$f(w_1) - f(w_2) - \nabla f(w_2)^T(w_1 - w_2) = \int_0^1 \nabla f(w_2 + t(w_1 - w_2))^T(w_1 - w_2)dt -$$

- From argument 2:

$$\int_0^1 \nabla f(w_2 + t(w_1 - w_2))^T(w_1 - w_2)dt - \nabla f(w_2)^T(w_1 - w_2) = \int_0^1 \nabla f(w_2 + t(w_1 -$$

$$= \int_0^1 \nabla(f(w_2 + t(w_1 - w_2)) - f(w_2))^T(w_1 - w_2)dt$$

- When in the last transition we used the linearity of the gradient.
- Now we will apply argument 1:

$$\int_0^1 \nabla(f(w_2 + t(w_1 - w_2)) - f(w_2))^T(w_1 - w_2)dt \leq \int_0^1 ||\nabla(f(w_2 + t(w_1 - w_2)) -$$

$$= ||w_1 - w_2|| \cdot \int_0^1 ||\nabla(f(w_2 + t(w_1 - w_2)) - f(w_2))||dt$$

- Now we will use the fact that f is β - smooth:

$$||w_1 - w_2|| \cdot \int_0^1 ||\nabla(f(w_2 + t(w_1 - w_2)) - f(w_2))||dt \leq ||w_1 - w_2|| \cdot \beta \cdot \int_0^1 ||t \cdot (w_1$$

$$= ||w_1 - w_2||^2 \cdot \beta \cdot \int_0^1 tdt = \frac{\beta}{2} \cdot ||w_1 - w_2||^2$$

■

Question 2 - Optimization and Gradient Descent

The function $f:\mathbb{R}^d \to \mathbb{R}$ is infinitely continuously differentiable, and satisfies $\min_{w\in\mathbb{R}^d} f(w) = f_* > -\infty$.

We wish to minimize this function using a version of Gradient Descent (GD) with step-size η, where in each iteration the gradients are multiplied by matrix A

$$(*)\; w(t+1) = w(t) - \eta A \nabla f\left(w(t)\right).$$

Matrix A is strictly positive, i.e., $\lambda_{min} \triangleq \lambda_{min}(A) > 0$, and denote $\lambda_{max} \triangleq \lambda_{max}(A)$.

1. In section only assume that $f(w) = \frac{1}{2}w^T H w$, where H is strictly positive. Find/choose A and η such that the algorithm $(*)$ converges in minimal number of steps. Why is that choice is infeasible when d is large? What is a common applicable approximation?
2. Prove that Gradient Flow (i.e., GD in the limit $\eta \to 0$):

$$\dot{w}(t) = -A\nabla f\left(w(t)\right)$$

   converges to a critical point for all f and A that satisfy the conditions in the given question.
   - **Hint**: from the properties of eigenvalues it satifies that $\forall v \in \mathbb{R}^d: \lambda_{min}||v||^2 \leq v^T A v \leq \lambda_{max}||v||^2$.
3. Given that the function f is $\beta\text{-smooth}$, find a condition on the step-size η such that we get convergence to a critical point in algorithm $(*)$. Prove convergence under this condition.
   - **Hint**: for a $\beta\text{-smooth}$ function, one can write:

$$f\left(w(t+1)\right) - f\left(w(t)\right) \leq \left(w(t+1) - w(t)\right)^T \nabla f\left(w(t)\right) + \frac{\beta}{2}||w(t+1) - w(t$$

Answer

Q1.

$\nabla f(w) = Hw \Rightarrow w(t+1) = w(t) - \eta A H w \Rightarrow$if we will choose $A = H^{-1}$ and $\eta = 1$ We will converge to $w(t) = 0$ which is a stationary point for f in a single step. The problem with this solution when d is large is that the inverse calculation of H is infeasible a common applicable approximation we could use to solve that problem, is using the concept of a preconditioning matrix.

Q2.

$\dot{f}\left(w(t)\right) = \nabla f(w(t))^T \cdot \dot{w}(t) = -\nabla f(w(t))^T \cdot A \cdot \nabla f(w(t)) \leq -\lambda_{min}||\nabla f(w(t))||^2 \leq$
$f(w(t)$ is monotonic decreasing which means:

$$\lim_{t\to+\infty} f(w(t)) = min f(w(t)) \Rightarrow \lim_{t\to+\infty} \dot{f}\left(w(t)\right) = 0 \Rightarrow \lim_{t\to+\infty} -\lambda_{min}||\nabla f(w(t))||^2 = 0$$

Q3.

from $(*)$ and hint number 3, we get that:

$$f\left(w(t+1)\right) - f\left(w(t)\right) \leq \left(-\eta A \nabla f\left(w(t)\right)\right)^T \nabla f\left(w(t)\right) + \frac{\beta}{2}||-\eta A \nabla f\left(w(t)\right)||^2$$

$$= -\eta \nabla f(w(t))^T A \nabla f\left(w(t)\right) + \frac{\beta\eta^2}{2}||A \nabla f\left(w(t)\right)||^2$$

from hint number 2 we get that:

$$-\nabla f(w(t))^T A \nabla f\left(w(t)\right) \leq -\lambda_{min}(A)||\nabla f\left(w(t)\right)||^2$$

and

$$||A \nabla f\left(w(t)\right)||^2 \leq \lambda_{max}(A^T A)||\nabla f\left(w(t)\right)||^2$$

Therefore:

$$f\left(w(t+1)\right) - f\left(w(t)\right) \leq -\eta\lambda_{min}(A)||\nabla f\left(w(t)\right)||^2 + \frac{\beta\eta^2}{2}\lambda_{max}(A^T A)||\nabla f\left(w(t)\right)||^2$$

$$= [-\eta\lambda_{min}(A) + \frac{\beta\eta^2}{2}\lambda_{max}(A^T A)]||\nabla f\left(w(t)\right)||^2$$

as we seen in Q2 $f(w(t))$ has to be monotonic decreasing, so a condition on the step-size η would be :

$$-\eta\lambda_{min}(A) + \frac{\beta\eta^2}{2}\lambda_{max}(A^T A) \leq 0$$

which means:

$$0 < \eta \leq \frac{2\lambda_{min}(A)}{\beta\lambda_{max}(A^T A)}$$

now we will Prove convergence under this condition: we define -c = $-\eta\lambda_{min}(A) + \frac{\beta\eta^2}{2}\lambda_{max}(A^TA)$. under our condition $0 \leq c$ and therefor:

$$c||\nabla f\left(w(t)\right)||^2 \leq f\left(w(t)\right) - f\left(w(t+1)\right)$$

$$c\sum_{t=0}^{\infty}||\nabla f\left(w(t)\right)||^2 \leq \sum_{t=0}^{\infty} f\left(w(t)\right) - f\left(w(t+1)\right) = f\left(w(0)\right) - f_* < \infty$$

so just like we saw in lecture 1:

$$\Rightarrow \lim_{t\to\infty}\nabla f(w(t)) = 0$$

Question 3 - Optimal Convergence Rate

This question relates to slide ~26 in the Optimization lecture slides.

For an objective function $f(w) = \frac{1}{2}W^THW$ and $H = X^TX = U\Lambda U^T$ where Λ is the eigenvalue matrix with eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_d$.

The Gradient Descent step as defined in the lecture:

$$w(t) = w(t-1) - \eta Hw(t-1).$$

For convenience, use $z(0) = U^Tw(0), z(t) = U^Tw(t)$.

Show that

1. $$f(w(t)) = \frac{1}{2}\sum_{i=1}^{d}(1-\eta\lambda_i)^{2t}\lambda_i z_i^2(0)$$

2. $$\text{rate}(\eta) = \max(|1-\eta\lambda_{min}|, |1-\eta\lambda_{max}|)$$

   (you can explain in words why it is true).

3. $$\eta_{\text{optimal}} = arg\min_{\eta}\text{rate}(\eta) = \frac{2}{\lambda_{max}+\lambda_{min}}$$

4. $$R_{\text{optimal}} = \min_{\eta}\text{rate}(\eta) = \frac{\lambda_{max}/\lambda_{min}-1}{\lambda_{max}/\lambda_{min}+1} = \kappa(\text{condition number})$$

Question 3 - Answer

First we will show some assistive claims for our proof:

- Claim 1: Λ Diagonal matrix $\Rightarrow (I-\eta\Lambda)^t$ Diagonal matrix
- Claim 2: $U^TU = UU^T = I$ Orthogonal matrix (eigenvectors)
- Claim 3: $\Lambda_1\Lambda_2 = \Lambda_2\Lambda_1$ for all Diagonal matrix Λ_1 and Λ_2
- Claim 4: H is a semi positive matrix

Proof of Claim 4: Given some vector y we will look at the scalar result of y^THy and we will mark x_i as the i'th row of X:

$$y^THy = y^T(\sum_{i=1}^{n} x_i x_i^T)y = \sum_{i=1}^{n} y^T x_i x_i^T y = \sum_{i=1}^{n}(x_i^T y)^2 \geq 0$$

$\blacksquare$

Part 1:

We will start by showing same as shown in the lecture, $w(t) = U(I-\eta\Lambda)^t z(0)$

$$U^Tw(t) = U^Tw(t-1) - \eta U^T Hw(t-1)$$

Since $U^TU = I$ and $H = U\Lambda U^T$:

$$U^Tw(t) = U^Tw(t-1) - \eta U^TU\Lambda U^Tw(t-1) = U^Tw(t-1) - \eta\Lambda z(t-1) \Rightarrow z(t) =$$

$$w(t) = UU^Tw(t) = Uz(t) = U(I-\eta\Lambda)^t z(0)$$

Now we will show equality 1 we will use the following:

- Claim 1: Λ Diagonal matrix $\Rightarrow (I-\eta\Lambda)^t$ Diagonal matrix
- Claim 2: $U^TU = UU^T = I$ Orthogonal matrix (eigenvectors)
- Claim 3: $\Lambda_1\Lambda_2 = \Lambda_2\Lambda_1$ for all Diagonal matrix Λ_1 and Λ_2

$$f(w(t)) = \frac{1}{2}w^THw = \frac{1}{2}(U(I-\eta\Lambda)^t z(0))^T H(U(I-\eta\Lambda)^t z(0)) = \frac{1}{2}z(0)^T(I-\eta\Lambda)^t$$

The last equality is according to Claim 1, and now we will use $H = U\Lambda U^T$:

$$f(w(t)) = \frac{1}{2}z(0)^T(I-\eta\Lambda)^t U^TU\Lambda U^TU(I-\eta\Lambda)^t z(0) = \frac{1}{2}z(0)^T(I-\eta\Lambda)^t\Lambda(I-\eta\Lambda$$

The last equality is according to Claim 2

$$f(w(t)) = \frac{1}{2}z(0)^T(I-\eta\Lambda)^t\Lambda(I-\eta\Lambda)^t z(0) = \frac{1}{2}z(0)^T(I-\eta\Lambda)^{2t}\Lambda z(0) = f(w(t)) =$$

The first equality is according to Claim 3

$\blacksquare$

Part 2:

The convergence rate according to t is only dependent of the expression $(1-\eta\lambda_i)^{2t}$, under the stability condition which satisfies $|1-\eta\lambda_i| \forall i \in 0, 1, \ldots, d$ The highest value of $|1-\eta\lambda_i|$ will be the one to converge the slowest to 0 when t goes to ∞ and for that reason, the highest element will dictate the convergence rate meaning:

$$\mathrm{rate}(\eta) = \max(|1-\eta\lambda_{min}|, |1-\eta\lambda_{max}|)$$

$$\blacksquare$$

Part 3:

First we will convert this problem to a differentiable one:

$$\eta_{\mathrm{optimal}} = arg\min_{\eta} \mathrm{rate}(\eta) = argmin(max((1-\eta\lambda_{min})^2, (1-\eta\lambda_{max})^2))$$

Now we will represnt $\mathrm{rate}(\eta)$ without using a maximum operator, we will need to find the equality points for the arguments and decide which argument is the maximum one in which portion of the η axis:

$$(1-\eta\lambda_{min})^2 \leq (1-\eta\lambda_{max})^2 \Rightarrow 1 - 2\eta\lambda_{min} + \eta^2\lambda_{min}^2 \leq 1 - 2\eta\lambda_{max} + \eta^2\lambda_{max}^2 \Rightarrow \eta^2$$

$$\eta(2-\eta(\lambda_{min}+\lambda_{max})) \leq 0$$

According to **Claim 4** $\frac{2}{\lambda_{max}+\lambda_{min}} \geq 0$

As the result of this, we will split for $\eta \leq 0$ and $\eta > 0$: $\eta > 0$

```
\mathrm{rate}(\eta)=  \left\{
\begin{array}{ll}
      |1-\eta\lambda_{min}| & \eta \leq \frac{2}{\lambda_{max} + \lambda_{min}} \\
      |1-\eta\lambda_{max}| & \eta \geq \frac{2}{\lambda_{max} + \lambda_{min}} \\
\end{array}
\right,
```

$\eta \leq 0$:

$$\mathrm{rate}(\eta) = |1-\eta\lambda_{max}|$$

This gets us to the final result of $\mathrm{rate}(\eta)$:

```
\mathrm{rate}(\eta)=  \left\{
\begin{array}{ll}
      |1-\eta\lambda_{min}| & 0 \leq \eta \leq \frac{2}{\lambda_{max} + \lambda_{min}} \\
      |1-\eta\lambda_{max}| & \eta > \frac{2}{\lambda_{max} + \lambda_{min}} or \; \eta <
\end{array}
\right,
```

Now we will show that for all $0 \leq \eta \leq \frac{2}{\lambda_{max}+\lambda_{min}} \Rightarrow \; 1 - \eta\lambda_{min} \geq 0$: Due to **Claim 4** $\lambda_{min} \geq 0$

$$1 - \eta\lambda_{min} \geq 0 \Leftrightarrow \eta \leq \frac{1}{\lambda_{min}}$$

$$\frac{2}{\lambda_{max} + \lambda_{min}} \leq \frac{1}{\lambda_{min}} \Leftrightarrow 2\lambda_{min} \leq \lambda_{max} + \lambda_{min} \Leftrightarrow \lambda_{min} \leq \lambda_{max}$$

Now we will show that for all $\eta > \frac{2}{\lambda_{max}+\lambda_{min}} or \;\; \eta < 0 \;\;\; \Rightarrow 1 - \eta\lambda_{max} < 0$:

$$1 - \eta\lambda_{max} < 0 \Leftrightarrow \eta > \frac{1}{\lambda_{max}}$$

$$\frac{2}{\lambda_{max} + \lambda_{min}} > \frac{1}{\lambda_{max}} \Leftrightarrow 2\lambda_{max} > \lambda_{max} + \lambda_{min} \Leftrightarrow \lambda_{max} > \lambda_{min}$$

Therefore:

```
\mathrm{rate}(\eta)=  \left\{
\begin{array}{ll}
      1-\eta\lambda_{min} & 0 \leq \eta \leq \frac{2}{\lambda_{max} + \lambda_{min}} \\
      \eta\lambda_{max} - 1 & \eta > \frac{2}{\lambda_{max} + \lambda_{min}} or \; \eta <
\end{array}
\right,
```

And since until the point where $\eta = \frac{2}{\lambda_{max}+\lambda_{min}}$ the rate is decreasing and from that point until ∞ the rate is increasing we get:

$$\eta_{\text{optimal}} = arg\min_{\eta} \text{rate}(\eta) = \frac{2}{\lambda_{max} + \lambda_{min}}$$

$$\blacksquare$$

Part 4:

In this part we will use the result of Part 3 in the formula:

```
\mathrm{rate}(\eta)=  \left\{
\begin{array}{ll}
    1-\eta\lambda_{min} & 0 \leq \eta \leq \frac{2}{\lambda_{max} + \lambda_{min}} \\
    \eta\lambda_{max} - 1 & \eta > \frac{2}{\lambda_{max} + \lambda_{min}} or \; \eta <
\end{array}
\right,
```

$$\mathrm{rate}(\frac{2}{\lambda_{max}+\lambda_{min}}) = 1 - \frac{2\lambda_{min}}{\lambda_{max}+\lambda_{min}} = \frac{\lambda_{max}-\lambda_{min}}{\lambda_{max}+\lambda_{min}} = \frac{\lambda_{max}-\lambda_{min}}{\lambda_{max}+\lambda_{min}} = \frac{\lambda_{max}/\lambda_{m}}{\lambda_{max}/\lambda_{m}}$$

■

Question 4 - Automatic Differentiation

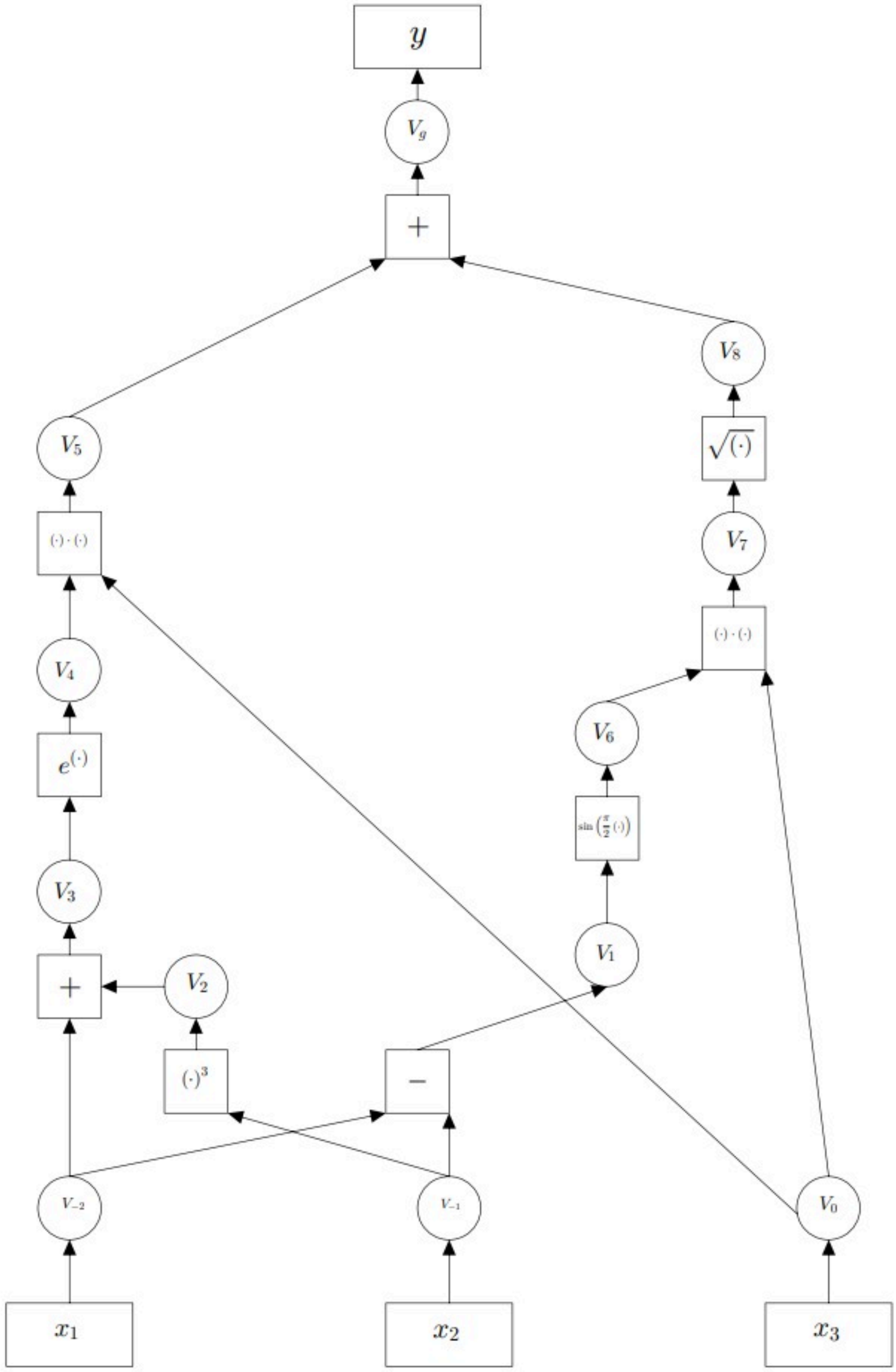
Consider the following function:

$$y = \exp(x_1 + x_2^3)x_3 + \sqrt{x_3 \sin\left(\frac{\pi}{2}(x_1 - x_2)\right)}$$

1. Write this function as a computational graph with *at least* 2 internal variables (you can draw the graph by hand and attach the drawing as an image file).
2. Use **forward mode autodiff** to calculate $\frac{\partial y}{\partial x_1}$ at $(x_1, x_2, x_3) = (2, 1, 1)$.
3. Use **backward mode autodiff** to calculate $\frac{\partial y}{\partial x_2}$ at $(x_1, x_2, x_3) = (2, 1, 1)$.
4. Use **numerical differentiation** to calculate $\frac{\partial y}{\partial x_3}$ at $(x_1, x_2, x_3) = (1, 1, 1)$. Which method for differentiation will you use? What will be the step size (assume the numerical precision $\epsilon = 0.0001$)?
5. Describe the advantages and disadvantages for each method (forward, backward and numerical) for a general function.

PART 1

$$y = \exp(x_1 + x_2^3)x_3 + \sqrt{x_3 \sin\left(\frac{\pi}{2}(x_1 - x_2)\right)}$$



PART 2

Let us do the forward step:

$$v_{-2} = 2, v_{-1} = 1, v_0 = 1$$

$$v_2 = 1, v_3 = 3, v_4 = e^3, v_3 = e^3$$

$$v_1 = 1, v_6 = 1, v_7 = 1, v_8 = 1$$

$$y = 1 + e^3$$

$$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial v_5}\frac{\partial v_5}{\partial v_4}\frac{\partial v_4}{\partial v_3}\frac{\partial v_3}{\partial x_1} + \frac{\partial y}{\partial v_8}\frac{\partial v_8}{\partial v_7}\frac{\partial v_7}{\partial v_6}\frac{\partial v_6}{\partial v_1}\frac{\partial v_1}{\partial x_1}$$

To simplify that, we will define - $\dot{v_i} = \frac{\partial v_i}{\partial x_1}$ therefor we want to calculate : $\frac{\partial y}{\partial x_1} = \dot{v_8}\frac{\partial y}{\partial v_8} + \dot{v_5}\frac{\partial y}{\partial v_5}$

$$\dot{v_3} = \frac{\partial v_3}{\partial x_1} = 1$$

$$\dot{v_4} = \frac{\partial v_4}{\partial v_3}\dot{v_3} = e^3$$

$$\dot{v_5} = \frac{\partial v_5}{\partial v_4}\dot{v_4} = e^3$$

$$\dot{v_5} = \frac{\partial v_5}{\partial v_4}\dot{v_4} = e^3$$

$$\dot{v_5} = \frac{\partial v_5}{\partial v_4}\dot{v_4} = e^3$$

$$\frac{\partial y}{\partial v_5} = \frac{\partial y}{\partial v_8} = 1$$

$$\dot{v_1} = \frac{\partial v_1}{\partial x_1} = 1$$

$$\dot{v_6} = \frac{\partial v_6}{\partial v_1}\dot{v_1} = \frac{\pi}{2}cos(\frac{\pi}{2} \cdot 1) = 0$$

Therefor :

$$\frac{\partial y}{\partial v_8}\frac{\partial v_8}{\partial v_7}\frac{\partial v_7}{\partial v_6}\frac{\partial v_6}{\partial v_1}\frac{\partial v_1}{\partial x_1} = 0$$

And:

$$\frac{\partial y}{\partial x_1} = \dot{v_5}\frac{\partial y}{\partial v_5} = e^3$$

PART 3

To calculate : $\frac{\partial y}{\partial x_2}$ we can calculate

$$\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial v_5}\frac{\partial v_5}{\partial v_4}\frac{\partial v_4}{\partial v_3}\frac{\partial v_3}{\partial x_1} + \frac{\partial y}{\partial v_8}\frac{\partial v_8}{\partial v_7}\frac{\partial v_7}{\partial v_6}\frac{\partial v_6}{\partial v_1}\frac{\partial v_1}{\partial x_1}$$

To simplify that, we will define - $\bar{v_i} = \frac{\partial y}{\partial v_i}$ Therefor we want to calculate :
$\frac{\partial y}{\partial x_2} = \bar{v_2}\frac{\partial v_2}{\partial x_2} + \bar{v_1}\frac{\partial v_1}{\partial x_2}$

$$\bar{v_9} = \frac{\partial y}{\partial v_9} = 1$$

$$\bar{v_5} = \frac{\partial y}{\partial v_5} = \bar{v_8} = \frac{\partial y}{\partial v_8} = 1$$

$$\bar{v_4} = \frac{\partial y}{\partial v_4} = \bar{v_5}\frac{\partial v_5}{\partial v_4} = 1 \cdot e^3$$

$$\bar{v_3} = \frac{\partial y}{\partial v_3} = \bar{v_4}\frac{\partial v_4}{\partial v_3} = e^3 \cdot 1$$

$$\bar{v_2} = \frac{\partial y}{\partial v_2} = \bar{v_3}\frac{\partial v_3}{\partial v_2} = e^3 \cdot 1$$

$$\bar{v_7} = \frac{\partial y}{\partial v_7} = \bar{v_8}\frac{\partial v_8}{\partial v_7} = 1/2$$

$$\bar{v_6} = \frac{\partial y}{\partial v_6} = \bar{v_7}\frac{\partial v_7}{\partial v_6} = 1/2$$

$$\bar{v_1} = \frac{\partial y}{\partial v_1} = \bar{v_6}\frac{\partial v_6}{\partial v_1} = 0$$

$$\bar{x_2} = \frac{\partial y}{\partial x_2} = \bar{v_2}\frac{\partial v_2}{\partial x_2} + \bar{v_1}\frac{\partial v_1}{\partial x_2} = e^3 \cdot 3 + 0$$

So we get that :

$$\frac{\partial y}{\partial x_2} = 3e^3 \approx 60.25$$

PART 4

For numerical differentiation calculation we need to choose a step size, we would choose the same step size learned in lecture 3 which is: We chose centered difference therefor:

$$h = \sqrt{\epsilon}(1 + \sqrt{1^2 + 1^2 + 1^2}) = 10^{-2}(1 + \sqrt{3})$$

Now we will define $f(x_1, x_2, x_3) = \exp(x_1 + x_2^3)x_3 + \sqrt{x_3 \sin\left(\frac{\pi}{2}(x_1 - x_2)\right)}$ And we will calculate :

$$\frac{f(x_1, x_2, x_3 + h) - f(x_1, x_2, x_3 - h)}{2h} = \frac{f(1, 1, 1 + h) - f(1, 1, 1 - h)}{2h} = \frac{(1 + h)e^2 -}{2h}$$

As we can see the size of h is not important (make sense since the numerator is linear in x_3).

PART 5

Forward Mode: Advantage: it is easy to add inputs to our function. Disadvantage: cost a lot of memory.

Reverse mode: Advantage: it is easy to add outputs to our function. Disadvantage: cost a lot of memory.

Numerical Differentiation: Advantage: very simple to calculate. Disadvantage: depends on the machine we may have a round off error for small steps.

Part 2 - Code Assignments

- You must write your code in this notebook and save it with the output of aall of the code cells.
- Additional text can be added in Markdown cells.
- You can use any other IDE you like (PyCharm, VSCode...) to write/debug your code, but for the submission you must copy it to this notebook, run the code and save the notebook with the output.

In [28]:
```
# imports for the practice (you can add more if you need)
import os
import numpy as np
import pandas as pd
import torch
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import LogNorm
from sklearn.datasets import load_iris
import math
seed = 211
np.random.seed(seed)
torch.manual_seed(seed)
# %matplotlib notebook
%matplotlib inline
```

Task 1 - The Beale Function

The Beale function is defined as follows:

$$f(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$$

1. What is the global minima of this function?
2. Implement the Beale function: `beale_f(x,y)` .
3. Implement a function, `beale_grads(x,y)` that returns the gradients of the Beale function.
4. 3D plot the Beale function wit the global minima you found. Use Matplotlib's `ax.plot_surface(x_mesh, y_mesh, z, norm=LogNorm(), rstride=1, cstride=1, edgecolor='none', alpha=.8, cmap=plt.cm.jet)` for the function, and `ax.plot(x, y, f(x, y), 'r*', markersize=20)` for the minima.
5. 2D plot the contours with `ax.contour(x_mesh, y_mesh, z, levels=np.logspace(-.5, 5, 35), norm=LogNorm(), cmap=plt.cm.jet)` and the minima with `ax.plot(x, y, 'r*', markersize=20)` .

1. f is a sum of quadric expressions, so if there is a combination of x, y that achieves $f(x,y) = 0$ than this is the minima:

$$\begin{array}{ccccccc} 1.5 & - & x & + & xy & = & 0 \\ 2.25 & - & x & + & xy^2 & = & 0 \\ 2.625 & - & x & + & xy^3 & = & 0 \end{array}$$

- There is a solution to this set of equations which is $(x, y) = (3, 0.5)$

In [29]:

```
# Set the manually calculated minima
min_x = 3
min_y = 0.5

def beale_f(x, y):
    value = None
    value = (1.5 - x + x*y)**2 + (2.25 - x + x*y**2)**2 + (2.625 - x + x*y
    return value

def beale_grads(x, y):
    dx, dy = None, None
    dx = 2*(y-1)*(1.5 - x + x*y) + 2*(y**2 - 1)*(2.25 - x + x*y**2) + 2*(y
    dy = 2*(1.5 - x + x*y) + 4*x*y*(2.25 - x + x*y**2) + 6*x*y**2*(2.625 -
    grads = np.array([dx, dy])
    return grads
```

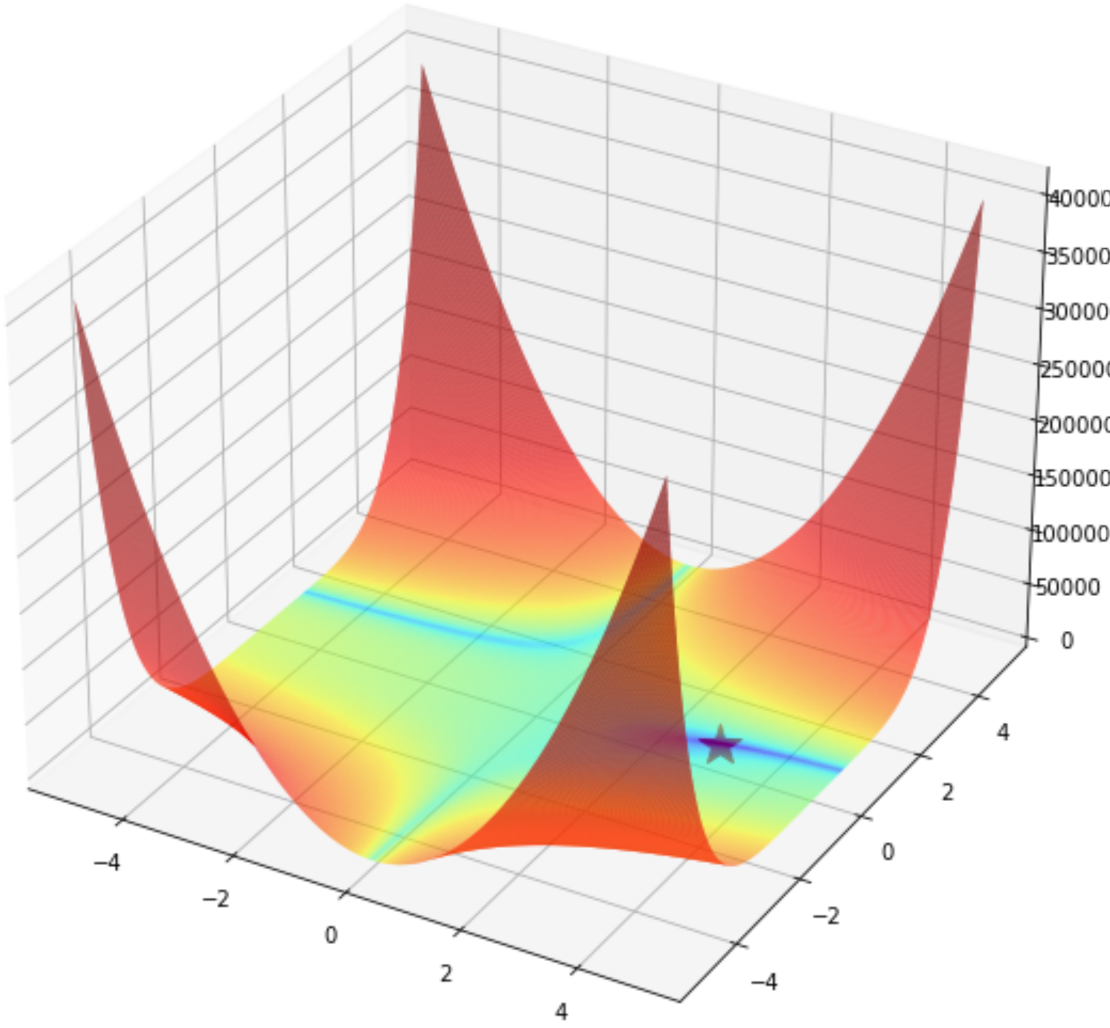
In [30]:

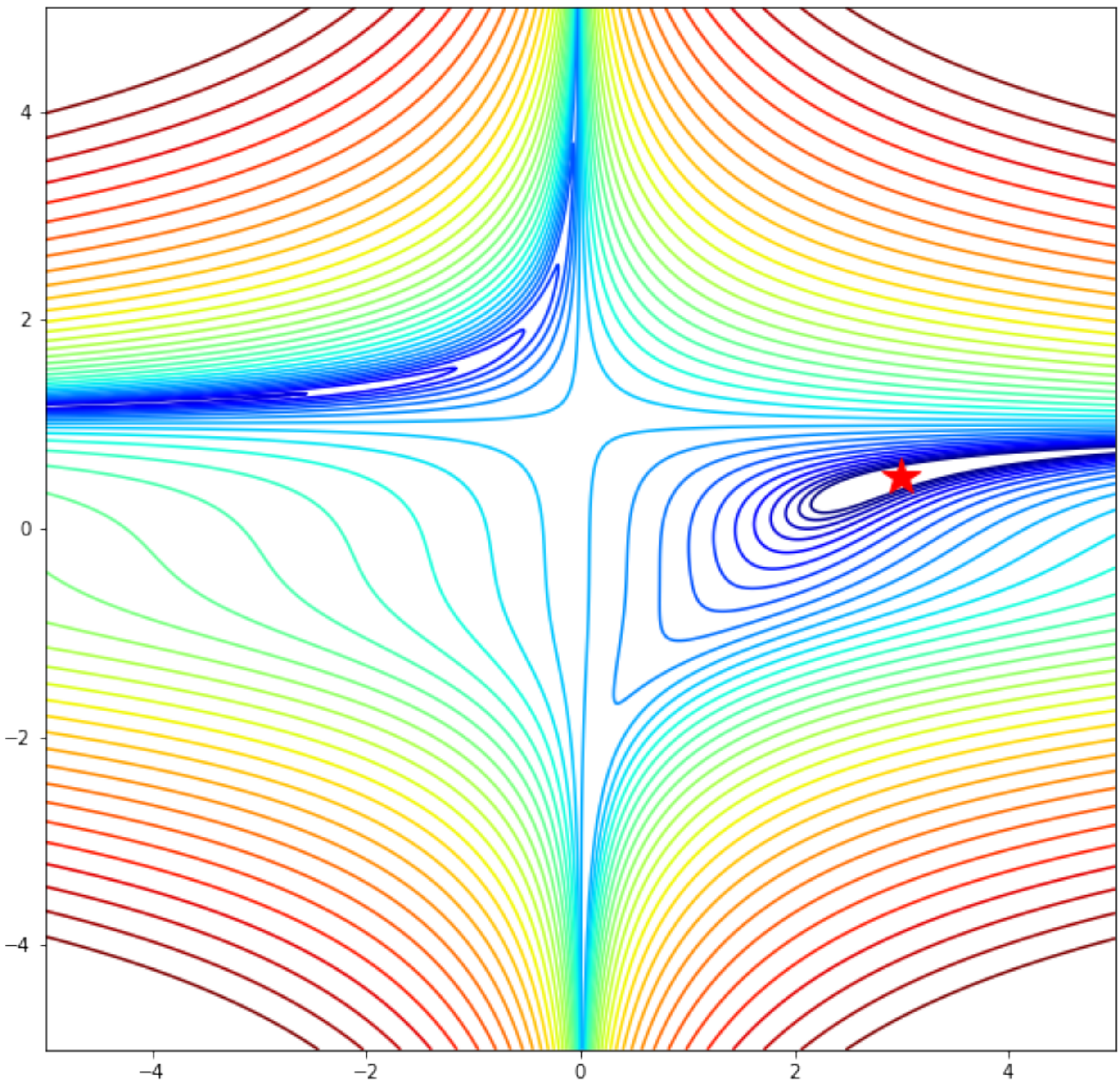
```
minima = np.array([min_x, min_y])
beale_res = beale_f(*minima)
grads_res = beale_grads(*minima)
print(f"minima (1x2 row vector shape): {minima}")
print(f"beale_f output: {beale_res}")
print(f"beale_grad output: {grads_res}")
x = np.linspace(-5, 5, 500)
y = np.linspace(-5, 5, 500)
x_mesh, y_mesh = np.meshgrid(x, y)
z = beale_f(x_mesh, y_mesh)
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x_mesh, y_mesh, z, norm=LogNorm(), rstride=1, cstride=1, ec
ax.plot(min_x, min_y, beale_f(min_x, min_y), 'r*', markersize=20)
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(1, 1, 1)
ax.contour(x_mesh, y_mesh, z, levels=np.logspace(-.5, 5, 35), norm=LogNorm(
ax.plot(min_x, min_y, 'r*', markersize=20)
```

```
minima (1x2 row vector shape): [3.  0.5]
beale_f output: 0.0
beale_grad output: [-0.  0.]
```

Out[30]:

```
[<matplotlib.lines.Line2D at 0x7fb96b5a32e8>]
```


400000
350000
300000
250000
200000
150000
100000
50000
0
−4
−2
0
2
4
4
2
0
−2
−4


4
2
0
−2
−4
−4
−2
0
2
4

Task 2 - Building an Optimizer - Adam

In this task, you are going to implement the Adam optimizer. We are giving the skeleton of the code and the description of the methods, and you need to implement the optimizer.

Recall the Adam update rule:

$$m_{k+1} = \beta_1 m_k + (1-\beta_1)\nabla f(w^k) = \beta_1 m_k + (1-\beta_1) g_k$$

$$v_{k+1} = \beta_2 v_k + (1-\beta_2)(\nabla f(w^k))^2 = \beta_2 v_k + (1-\beta_2) g_k^2$$

Then, they use an **unbiased** estimation:

$$\hat{m}_{k+1} = \frac{m_{k+1}}{1-\beta_1^{k+1}}$$

$$\hat{v}_{k+1} = \frac{v_{k+1}}{1-\beta_2^{k+1}}$$

(the β's are taken with the power of the current iteration)

$$w_{k+1} = w_k - \frac{\alpha}{\sqrt{\hat{v}_{k+1}} + \epsilon}\hat{m}_{k+1}$$

- ϵ deafult's is 10^{-8}

1. Implement `class AdamOptimizer()` .
    - `function` is the Python function you want to optimize.
    - `gradients` is the Python function that returns the gradients of `function` .
    - `x_init` and `y_init` are the initialization points for the optimizer.
    - Save the `path` of the optimizer (the minima points the optimizer visits during the optimization).
    - Stopping criterion: change in minima `<1e-7` .
    - **You can change the class however you wish, you can remove/add variables and methods as you wish**
2. For `x_init=0.7, y_init=1.4, learning_rate=0.1, beta1=0.9, beta2=0.999` , optimize the Beale function. Plot the results **with the path taken** (better do it on the 2D contour plot).
3. Choose different initialization and learning rate and show the results as in 2.

In [31]:

```
class AdamOptimizer():
    def __init__(self, function, gradients, x_init=None, y_init=None,
                 learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8)
```

```
        self.f = function
        self.g = gradients
        scale = 3.0
        self.current_val = np.zeros([2])
        if x_init is not None:
            self.current_val[0] = x_init
        else:
            self.current_val[0] = np.random.uniform(low=-scale, high=scale)
        if y_init is not None:
            self.current_val[1] = y_init
        else:
            self.current_val[1] = np.random.uniform(low=-scale, high=scale)
        print("x_init: {:.3f}".format(self.current_val[0]))
        print("y_init: {:.3f}".format(self.current_val[1]))

        self.lr = learning_rate
        self.grads_first_moment = np.zeros([2])
        self.grads_second_moment = np.zeros([2])
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon

        # for accumulation of loss and path (w, b)
        self.z_history = []
        self.x_history = []
        self.y_history = []


    def func(self, variables):
        """Beale function.

        Args:
          variables: input data, shape: 1-rank Tensor (vector) np.array
            x: x-dimension of inputs
            y: y-dimension of inputs

        Returns:
          z: Beale function value at (x, y)
        """
        return self.f(*variables)

    def gradients(self, variables):
        """Gradient of Beale function.

        Args:
          variables: input data, shape: 1-rank Tensor (vector) np.array
            x: x-dimension of inputs
            y: y-dimension of inputs

        Returns:
          grads: [dx, dy], shape: 1-rank Tensor (vector) np.array
            dx: gradient of Beale function with respect to x-dimension of i
            dy: gradient of Beale function with respect to y-dimension of i
        """
        return self.g(*variables)
```

```
    def weights_update(self, grads, time):
        """Weights update using Adam.

          g1 = beta1 * g1 + (1 - beta1) * grads
          g2 = beta2 * g2 + (1 - beta2) * grads ** 2
          g1_unbiased = g1 / (1 - beta1**time)
          g2_unbiased = g2 / (1 - beta2**time)
          w = w - lr * g1_unbiased / (sqrt(g2_unbiased) + epsilon)
        """
        self.grads_first_moment = self.beta1 * self.grads_first_moment + (1
        self.grads_second_moment = self.beta2 * self.grads_second_moment +
        grads_first_moment_unbiased = self.grads_first_moment / (1 - self.b
        grads_second_moment_unbiased = self.grads_second_moment / (1 - self
        self.current_val = self.current_val - self.lr * grads_first_moment_

    def history_update(self, z, x, y):
        """Accumulate all interesting variables
        """
        #print(f"Updating history x = {x}, y = {y}, z = {z}")
        self.x_history.append(x)
        self.y_history.append(y)
        self.z_history.append(z)

    def train(self, max_steps):
        """


        :param max_steps:
        :return:
        """
        delta = math.inf
        self.history_update(self.func(self.current_val), *self.current_val)
        k = 1
        while delta >  10 ** (-7) and k <= max_steps:
            curr_minima = self.func(self.current_val)
            grads = self.gradients(self.current_val)
            self.weights_update(grads, k)
            new_minima = self.func(self.current_val)
            self.history_update(self.func(self.current_val), *self.current_
            delta = abs(new_minima - curr_minima)
            k += 1
```

In [32]:

```
opt = AdamOptimizer(beale_f, beale_grads, x_init=0.7, y_init=1.4, learning_
```

```
x_init: 0.700
y_init: 1.400
```

In [33]:

```
%time
opt.train(1000)
print("Global minima")
print("x*: {:.2f}  y*: {:.2f}".format(minima[0], minima[1]))
print("Solution using the gradient descent")
print("x: {:.4f}  y: {:.4f}".format(opt.current_val[0], opt.current_val[1])
```

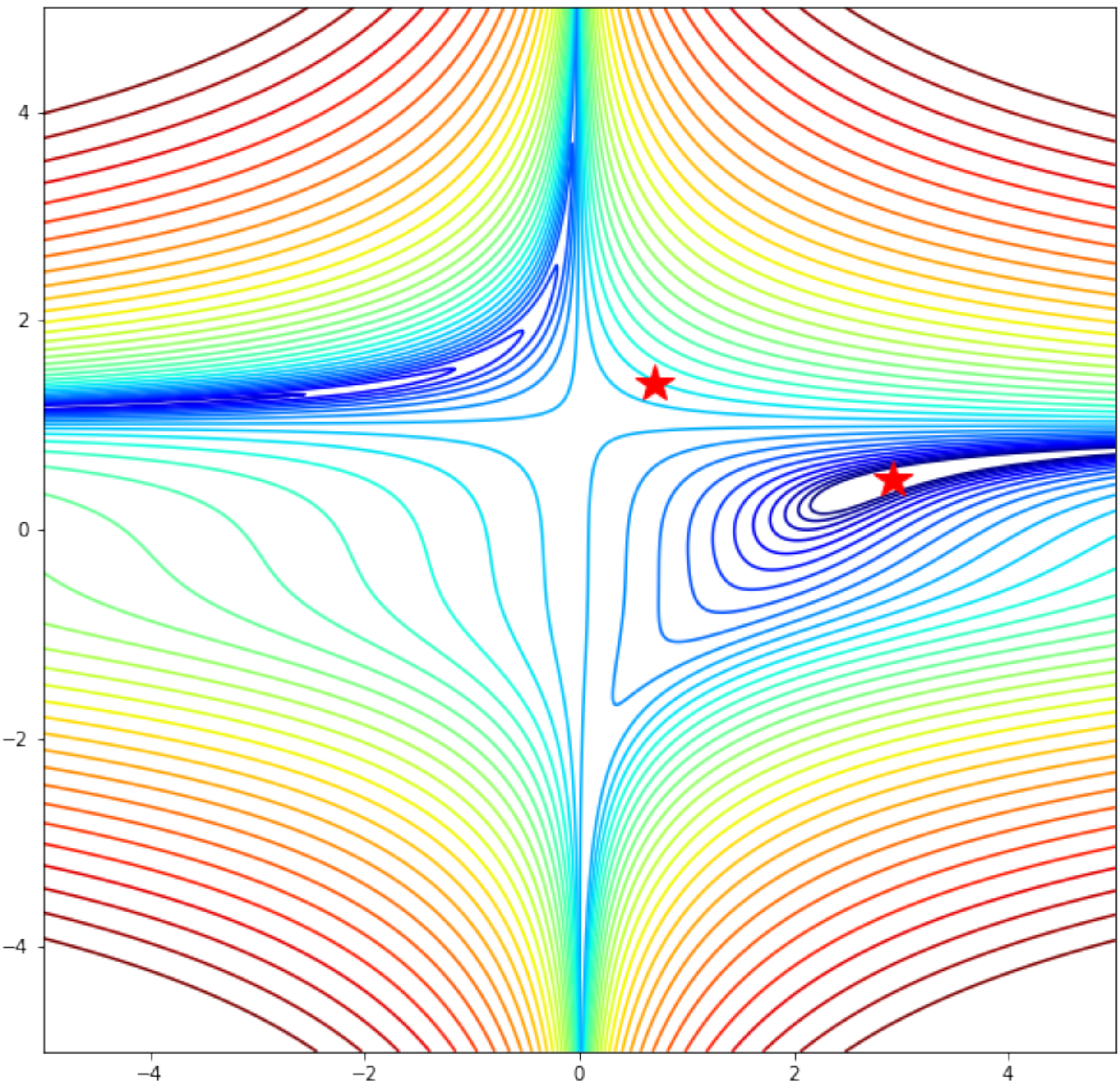
```
CPU times: user 1 µs, sys: 0 ns, total: 1 µs
Wall time: 3.1 µs
Global minima
x*: 3.00  y*: 0.50
Solution using the gradient descent
x: 2.9308  y: 0.4802
```

In [34]:

```
# plot the Beale function values durin
x = np.linspace(-5, 5, 500)
y = np.linspace(-5, 5, 500)
x_mesh, y_mesh = np.meshgrid(x, y)
z = opt.f(x_mesh, y_mesh)
fig = plt.figure(figsize=(10, 10))
ax1 = fig.add_subplot(1, 1, 1)
ax1.contour(x_mesh, y_mesh, z, levels=np.logspace(-.5, 5, 35), norm=LogNorr
ax1.plot(opt.current_val[0], opt.current_val[1], 'r*', markersize=20)
ax1.plot(0.7, 1.4, 'r*', markersize=20)
```

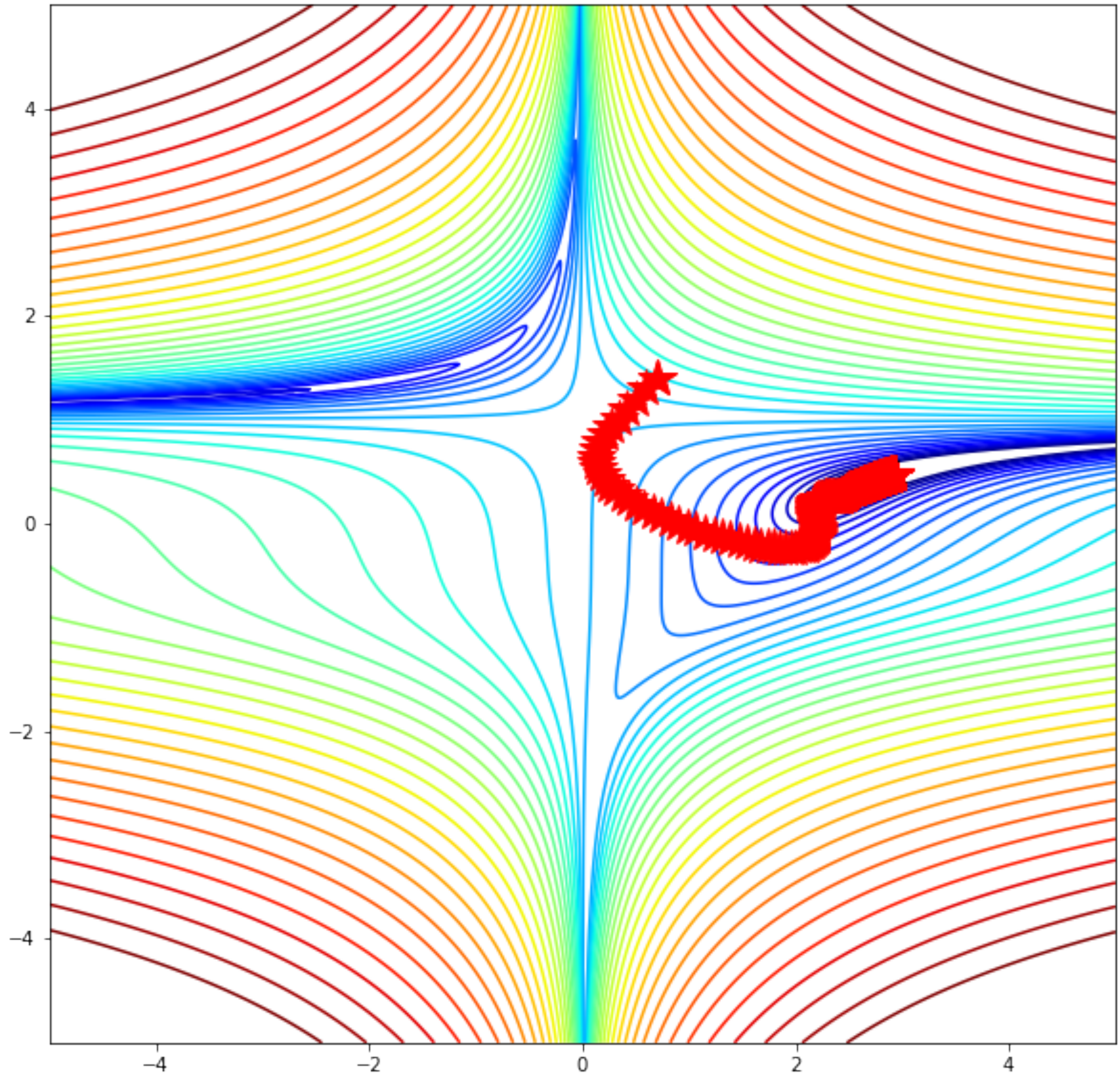
Out[34]:

```
[<matplotlib.lines.Line2D at 0x7fb94b102240>]
```



In [35]:
```
# plot the optimization path
fig = plt.figure(figsize=(10, 10))
ax1 = fig.add_subplot(1, 1, 1)
ax1.contour(x_mesh, y_mesh, z, levels=np.logspace(-.5, 5, 35), norm=LogNorr
ax1.plot(opt.x_history, opt.y_history, 'r*', markersize=20)
ax1.plot(0.7, 1.4, 'r*', markersize=20)
```

Out[35]: [<matplotlib.lines.Line2D at 0x7fb8d9407080>]

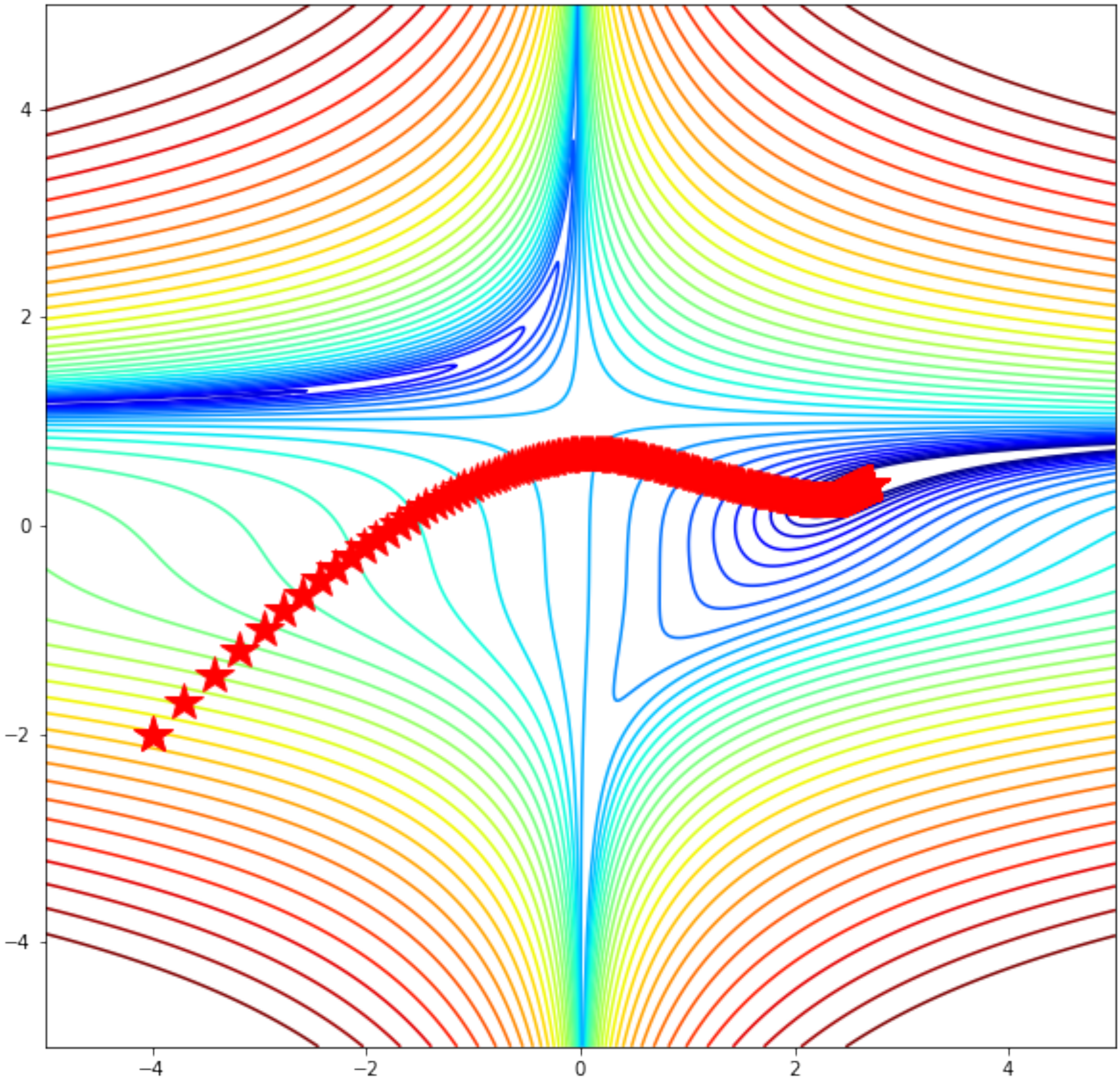


In [36]:
```
opt = AdamOptimizer(beale_f, beale_grads, x_init=-4, y_init=-2, learning_ra
opt.train(1000)
z = opt.f(x_mesh, y_mesh)
fig = plt.figure(figsize=(10, 10))
ax1 = fig.add_subplot(1, 1, 1)
ax1.contour(x_mesh, y_mesh, z, levels=np.logspace(-.5, 5, 35), norm=LogNorr
ax1.plot(opt.x_history, opt.y_history, 'r*', markersize=20)
ax1.plot(-4, -2, 'r*', markersize=20)
```

```
x_init: -4.000
y_init: -2.000
```

Out[36]:

```
[<matplotlib.lines.Line2D at 0x7fb928137ef0>]
```



Task 3 - PyTorch Autograd

For the function from the theory practice:

$$f = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$$

1. Implement it and its dervative (explicitly) using `torch` .
2. Define a scalar tensor `x` and use `autograd` to calculate the derivative w.r.t x. Does the result correspond to the output of the function the calculates the derivative explicitly?

In [37]:

```
def f(x):
    f_val = torch.exp(torch.exp(x) + (torch.exp(x))**2) + torch.sin(torch.e
    return f_val

def derv_f(x):
    derv_val = (torch.exp(torch.exp(x) + torch.exp(2*x)) + torch.cos(torch.
    return derv_val
```

In [38]:

```
x = torch.tensor(0.5, requires_grad=True)
print(x)
f_res = f(x)
f_manual_grad = derv_f(x.detach())

# Calculate with torch autograd
f_autograd = None
f_res.backward()
f_autograd = x.grad

print(f_manual_grad)
print(f_autograd)
```

```
tensor(0.5000, requires_grad=True)
tensor(555.9719)
tensor(555.9719)
```

Task 3 - Answer

Yes the result correspond to the output of the function that calculates the derivative explicitly, As expected from what we have seen in the class regarding Autograd

Task 4 - Low Rank Matrix Factorization

Consider the following optimization problem:

$$\min_{\hat{U},\hat{V}} ||A - \hat{U}\hat{V}||_F^2$$

Where $A \in \mathcal{R}^{m\times n}, \hat{U} \in \mathcal{R}^{m\times r}, \hat{V} \in \mathcal{R}^{r\times n}$ and $r < min(m, n)$ (r is the rank of the matrix). $||\cdot||_F^2$ denotes the Frobenius norm.

1. Implement a function, `gd_factorize_ad(A, rank, num_epochs=1000, lr=0.01)` , that given a 2D tensor `A` and a `rank` , will calculate the low-rank factorization of `A` using **gradient decsent**. Compute and apply all the gradients of $\hat{U}$ and of $\hat{V}$ once per epoch. $\hat{U}$ and $\hat{V}$ should be initially created with uniform random values. Use PyTorch's `autograd` for the gradients.
   - To compute the squared Frobenius norm loss (reconstruction loss), use `torch.nn.functional.mse loss with reduction='sum'` .
2. Use the provided `data` of the Iris dataset of 150 instances and 4 features. Apply `gd_factorize_ad` to compute the 2-rank matrix factorization of `data` . What is the reconstruction loss?

In [39]:
```
df = load_iris(as_frame=True).data # option 1
# df = pd.read_csv('./iris.data', header=None) # option 2
data = torch.tensor(df.iloc[:, [0, 1, 2, 3]].values)
data = data - data.mean(dim=0)
```

In [40]:

```
def gd_factorize_ad(A, rank, num_epochs=1000, lr=0.01):
    # initialize
    m = A.shape[0]
    n = A.shape[1]
    U = torch.rand((m, rank), requires_grad=True)
    V = torch.rand((rank, n), requires_grad=True)
    # implement gradient descent
    for epoch in range(num_epochs):
        loss = torch.nn.functional.mse_loss(torch.mm(U,V), A, reduction='su
        loss.backward()
        with torch.no_grad():
            U -= lr * U.grad
            V -= lr * V.grad
            U.grad = None
            V.grad = None
        if epoch % 5 == 0:
            print(f'epoch: {epoch}, loss: {loss}')
    return U, V
```

In [41]:

```
U, V = gd_factorize_ad(data.float(), rank=2, num_epochs=1000, lr=0.01)
```

```
epoch: 0, loss: 888.7950439453125
epoch: 5, loss: 531.0763549804688
epoch: 10, loss: 72.64854431152344
epoch: 15, loss: 51.544254302978516
epoch: 20, loss: 51.36109924316406
epoch: 25, loss: 51.35871124267578
epoch: 30, loss: 51.358497619628906
epoch: 35, loss: 51.358375549316406
epoch: 40, loss: 51.3582649230957
epoch: 45, loss: 51.35813522338867
epoch: 50, loss: 51.358001708984375
epoch: 55, loss: 51.35786437988281
epoch: 60, loss: 51.35771560668945
epoch: 65, loss: 51.3575439453125
epoch: 70, loss: 51.35737228393555
epoch: 75, loss: 51.357181549072266
epoch: 80, loss: 51.35697937011719
epoch: 85, loss: 51.356754302978516
epoch: 90, loss: 51.35651397705078
epoch: 95, loss: 51.356258392333984
epoch: 100, loss: 51.35597229003906
epoch: 105, loss: 51.35566329956055
epoch: 110, loss: 51.35532760620117
epoch: 115, loss: 51.35496520996094
epoch: 120, loss: 51.35456085205078
epoch: 125, loss: 51.354129791259766
epoch: 130, loss: 51.35364532470703
epoch: 135, loss: 51.35313415527344
epoch: 140, loss: 51.352561950683594
epoch: 145, loss: 51.35193634033203
epoch: 150, loss: 51.35125732421875
epoch: 155, loss: 51.35050964355469
epoch: 160, loss: 51.34967803955078
epoch: 165, loss: 51.348777770996094
epoch: 170, loss: 51.34778594970703
```

```
epoch: 175, loss: 51.3466911315918
epoch: 180, loss: 51.345489501953125
epoch: 185, loss: 51.344173431396484
epoch: 190, loss: 51.34272003173828
epoch: 195, loss: 51.34111785888672
epoch: 200, loss: 51.339359283447266
epoch: 205, loss: 51.33742141723633
epoch: 210, loss: 51.33527374267578
epoch: 215, loss: 51.33292007446289
epoch: 220, loss: 51.330326080322266
epoch: 225, loss: 51.32745361328125
epoch: 230, loss: 51.32429885864258
epoch: 235, loss: 51.32081604003906
epoch: 240, loss: 51.31696701049805
epoch: 245, loss: 51.312721252441406
epoch: 250, loss: 51.30804443359375
epoch: 255, loss: 51.302879333496094
epoch: 260, loss: 51.29718017578125
epoch: 265, loss: 51.29087829589844
epoch: 270, loss: 51.2839241027832
epoch: 275, loss: 51.27625274658203
epoch: 280, loss: 51.26778030395508
epoch: 285, loss: 51.2584228515625
epoch: 290, loss: 51.24809265136719
epoch: 295, loss: 51.236671447753906
epoch: 300, loss: 51.224082946777344
epoch: 305, loss: 51.21016311645508
epoch: 310, loss: 51.194801330566406
epoch: 315, loss: 51.17783737182617
epoch: 320, loss: 51.15909957885742
epoch: 325, loss: 51.13842010498047
epoch: 330, loss: 51.11558151245117
epoch: 335, loss: 51.09037780761719
epoch: 340, loss: 51.06254196166992
epoch: 345, loss: 51.031829833984375
epoch: 350, loss: 50.9979362487793
epoch: 355, loss: 50.96053695678711
epoch: 360, loss: 50.91927719116211
epoch: 365, loss: 50.87377166748047
epoch: 370, loss: 50.82358932495117
epoch: 375, loss: 50.76826095581055
epoch: 380, loss: 50.7072868347168
epoch: 385, loss: 50.640106201171875
epoch: 390, loss: 50.56611633300781
epoch: 395, loss: 50.48465347290039
epoch: 400, loss: 50.3950080871582
epoch: 405, loss: 50.29640197753906
epoch: 410, loss: 50.1879997253418
epoch: 415, loss: 50.06890106201172
epoch: 420, loss: 49.93812561035156
epoch: 425, loss: 49.79463577270508
epoch: 430, loss: 49.63733673095703
epoch: 435, loss: 49.465023040771484
epoch: 440, loss: 49.27644729614258
epoch: 445, loss: 49.070289611816406
epoch: 450, loss: 48.845176696777344
epoch: 455, loss: 48.59966278076172
epoch: 460, loss: 48.33226776123047
epoch: 465, loss: 48.041465759277344
epoch: 470, loss: 47.7257194519043
epoch: 475, loss: 47.38349151611328
```

```
epoch: 480, loss: 47.01325607299805
epoch: 485, loss: 46.61355209350586
epoch: 490, loss: 46.1829833984375
epoch: 495, loss: 45.72027587890625
epoch: 500, loss: 45.22431945800781
epoch: 505, loss: 44.69416427612305
epoch: 510, loss: 44.12913131713867
epoch: 515, loss: 43.52882385253906
epoch: 520, loss: 42.89315414428711
epoch: 525, loss: 42.222408294677734
epoch: 530, loss: 41.517295837402344
epoch: 535, loss: 40.778961181640625
epoch: 540, loss: 40.009002685546875
epoch: 545, loss: 39.20949935913086
epoch: 550, loss: 38.383018493652344
epoch: 555, loss: 37.532554626464844
epoch: 560, loss: 36.66156768798828
epoch: 565, loss: 35.773868560791016
epoch: 570, loss: 34.87360763549805
epoch: 575, loss: 33.965187072753906
epoch: 580, loss: 33.05316162109375
epoch: 585, loss: 32.14216232299805
epoch: 590, loss: 31.236812591552734
epoch: 595, loss: 30.34162139892578
epoch: 600, loss: 29.460878372192383
epoch: 605, loss: 28.598615646362305
epoch: 610, loss: 27.758508682250977
epoch: 615, loss: 26.94382095336914
epoch: 620, loss: 26.157373428344727
epoch: 625, loss: 25.40153694152832
epoch: 630, loss: 24.678178787231445
epoch: 635, loss: 23.98870849609375
epoch: 640, loss: 23.334070205688477
epoch: 645, loss: 22.71478271484375
epoch: 650, loss: 22.13097381591797
epoch: 655, loss: 21.5824031829834
epoch: 660, loss: 21.068532943725586
epoch: 665, loss: 20.58855628967285
epoch: 670, loss: 20.141443252563477
epoch: 675, loss: 19.725984573364258
epoch: 680, loss: 19.340845108032227
epoch: 685, loss: 18.98457908630371
epoch: 690, loss: 18.655685424804688
epoch: 695, loss: 18.352617263793945
epoch: 700, loss: 18.073823928833008
epoch: 705, loss: 17.817764282226562
epoch: 710, loss: 17.582923889160156
epoch: 715, loss: 17.36782455444336
epoch: 720, loss: 17.17104721069336
epoch: 725, loss: 16.991230010986328
epoch: 730, loss: 16.82707405090332
epoch: 735, loss: 16.67736053466797
epoch: 740, loss: 16.540925979614258
epoch: 745, loss: 16.416690826416016
epoch: 750, loss: 16.30364418029785
epoch: 755, loss: 16.200843811035156
epoch: 760, loss: 16.107412338256836
epoch: 765, loss: 16.02254295349121
epoch: 770, loss: 15.945486068725586
epoch: 775, loss: 15.875553131103516
epoch: 780, loss: 15.812110900878906
```

```
epoch: 785, loss: 15.75457763671875
epoch: 790, loss: 15.702415466308594
epoch: 795, loss: 15.655145645141602
epoch: 800, loss: 15.612312316894531
epoch: 805, loss: 15.573509216308594
epoch: 810, loss: 15.538369178771973
epoch: 815, loss: 15.506549835205078
epoch: 820, loss: 15.477744102478027
epoch: 825, loss: 15.451665878295898
epoch: 830, loss: 15.42806625366211
epoch: 835, loss: 15.40670776367l875
epoch: 840, loss: 15.387384414672852
epoch: 845, loss: 15.369898796081543
epoch: 850, loss: 15.354082107543945
epoch: 855, loss: 15.339773178100586
epoch: 860, loss: 15.326831817626953
epoch: 865, loss: 15.315126419067383
epoch: 870, loss: 15.304540634155273
epoch: 875, loss: 15.294964790344238
epoch: 880, loss: 15.286310195922852
epoch: 885, loss: 15.278478622436523
epoch: 890, loss: 15.271400451660156
epoch: 895, loss: 15.265000343322754
epoch: 900, loss: 15.259212493896484
epoch: 905, loss: 15.253978729248047
epoch: 910, loss: 15.249248504638672
epoch: 915, loss: 15.244970321655273
epoch: 920, loss: 15.241104125976562
epoch: 925, loss: 15.237606048583984
epoch: 930, loss: 15.234445571899414
epoch: 935, loss: 15.231586456298828
epoch: 940, loss: 15.22900390625
epoch: 945, loss: 15.226667404174805
epoch: 950, loss: 15.224556922912598
epoch: 955, loss: 15.22264575958252
epoch: 960, loss: 15.220922470092773
epoch: 965, loss: 15.219362258911133
epoch: 970, loss: 15.217950820922852
epoch: 975, loss: 15.2166748046875
epoch: 980, loss: 15.215520858764648
epoch: 985, loss: 15.21448040008545
epoch: 990, loss: 15.21353816986084
epoch: 995, loss: 15.212684631347656
```

Credits

- Icons made by Becris from www.flaticon.com
- Icons from Icons8.com - https://icons8.com
- Datasets from Kaggle - https://www.kaggle.com/