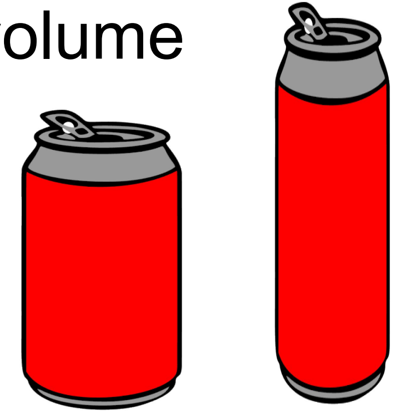
Il problema della lattina?

An optimization problem with pytorch?

**https://www.youtube.com/wat
ch?v=fozU7yKe1xQ**

Soda can optimization: lowest surface at fixed volume



$V_0\ is\ fixed\ at\ 33cl$

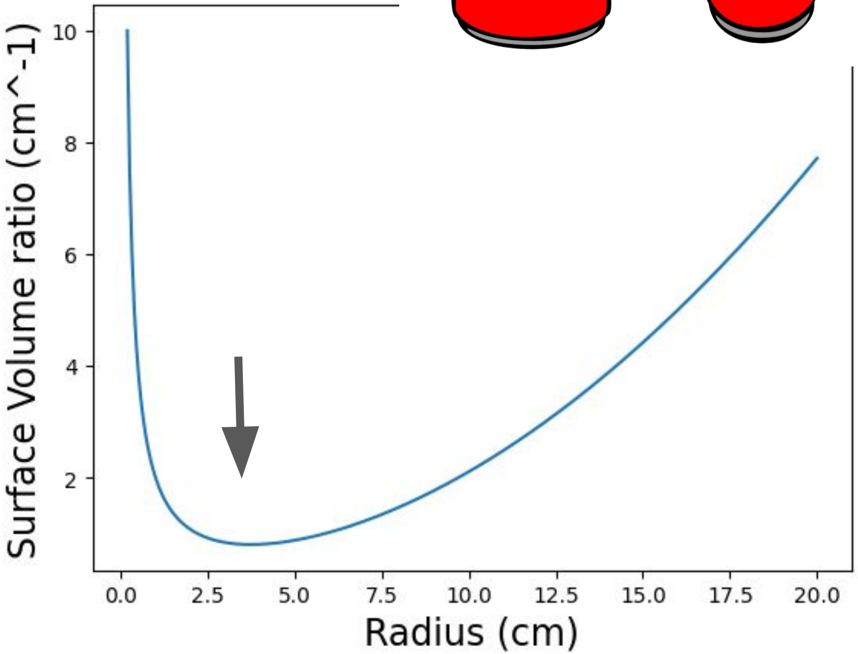
$$\frac{S}{V}(R) = 2R^{-1} + \frac{2\pi}{V_0}R^2$$

We want to find the minimum of the surface to volume ratio.

You know how to do it

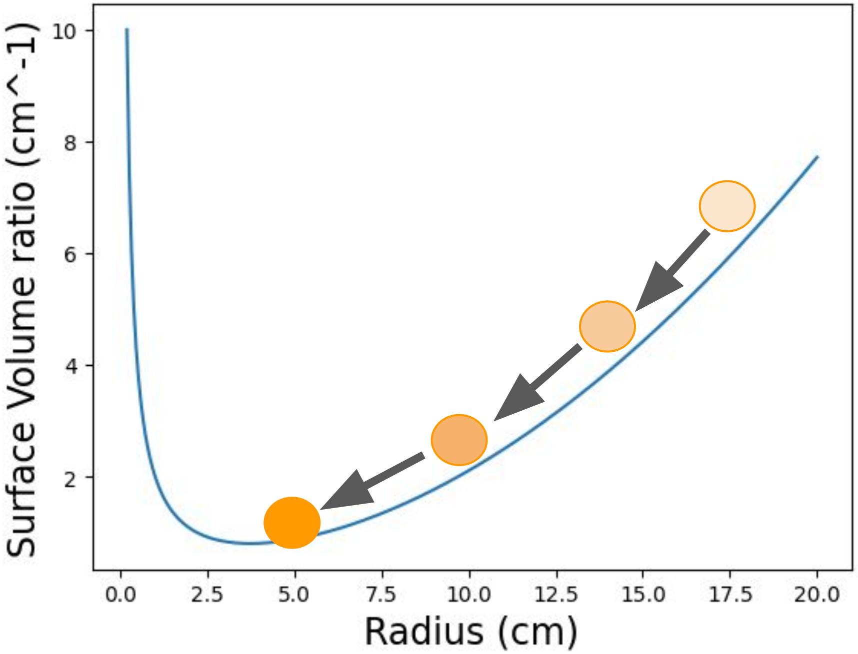
$$\frac{1}{V}\frac{\mathrm{d}S(R)}{\mathrm{d}R} = 0$$

But we will solve it with pytorch



Gradient Descent Optimization

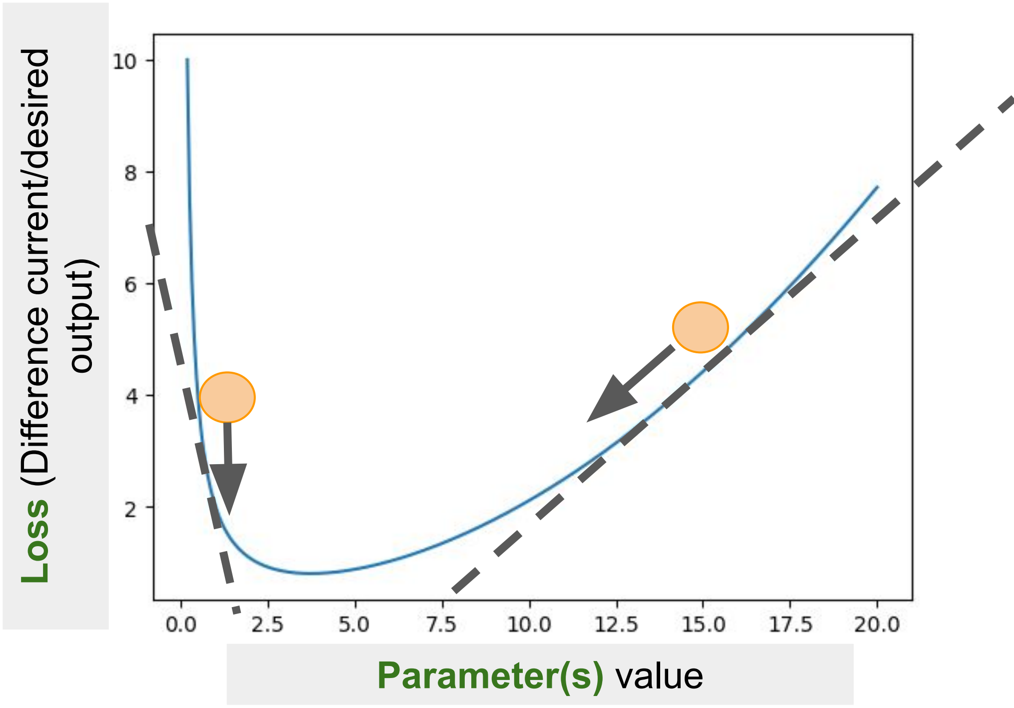
1 - Initialize R0
2 - Compute the value S(R0) (FORWARD)
3 - Compute the gradient S'(R0) (BACKWARD)
4 - Make an optimization Step
5 - Iterate 1-4 until convergence



Gradient Descent Optimization (For a Neural network)

The strategy is the same if :
FUNCTION → ERROR FUNCTION (computed on data)
R → PARAMETERS OF THE MODEL
COST OPTIMIZATION

OUTPUT depends on parameters



Classes with Python

```
[17] class Parent():# this is a Class
       def __init__(self):
         self.name = None
         self.introduction= "Hi,son . "
       def give_name(self,name):# this is a method of Parent class
         self.name =name
       def speak(self):
         if self.name is not None:
           print(self.introduction+"I am {}".format(self.name))
         else:
           print("give name first")

     class Child(Parent):# this is a child class. It has the same methods of Parent
       def __init__(self):# Init is replaced. Everthing else is the same
         self.name = None
         self.introduction= "Hi,Mama . "
```

```
[18] mama=Parent()# create a instance of class Parent
     mama.give_name("Marta")# use a method (give name is a "function" inside the class)
     son=Child()# create a instance of class Child
     son.give_name("Andrea")# we can use methods from the parent class

     mama.speak()# He can speak!
     son.speak()# He also can speak!
```

```
Hi,son . I am Marta
Hi,Mama . I am Andrea
```

This is an example with classes. When we create the dataset and the generator, we create a sub class that behaves like the “standard” pytorch generator but we adapted it to our needs.

This part is more complex that it seems and the also “terminology” is a bit simplified.

You will need only the basics.

Training a NN

$$input\ space : X$$

$$label\ space : Y$$

***n* training points**

$$z_i = (x_i, y_i) \in X \times Y$$

Network output depends on the parameters of the model

$$\tilde{y}_i = NN_w(x_i) \qquad w \in W \subseteq \mathbb{R}^p \leftarrow p{\sim}10^{7-10}$$

Training → minimize the Loss function (average of the element wise loss over the training set)

$$\mathbf{L}_w = \sum_{i=1}^{n} \frac{1}{n} L_i(z_i, w) = \mathbf{L}(\tilde{y})$$

We hope that, for a point not included in the training set, the loss is small

The solution is the set of weights

$$\hat{w} = \underset{w \in W}{argmin} \sum_i L(z_i, w)$$

$$\Rightarrow L(z_{test}, \hat{w})\ is\ small$$

Training a NN

***n* training points**

$$\tilde{y}_i = NN_w(x_i) \qquad z_i = (x_i, y_i) \in X \times Y$$

$$w \in W \subseteq \mathbb{R}^p \leftarrow p{\sim}10^{7-10}$$

Training → The best set of parameters is usually found using Stochastic Gradient Descent (SDG) optimization (or something similar)

$$\mathbf{L}_w = \sum_{i=1}^{n} \frac{1}{n} L_i(z_i, w) = \mathbf{L}(\tilde{y})$$

Loss function as the average of the element wise error

$$\boxed{w(e+1) = w(e) - \eta \nabla_w \mathbf{L}}$$