Zero order methods. Gradient free optimization. Global optimization

- Шпаргалка по результатам в безградиентной оптимизации
- RL и эволюционные алгоритмы

  Open in Colab
- Global optimization illustration

  Open in Colab
- Nevergrad library

  Open in Colab
- Optuna quickstart
- Демонстрация медленности методов нулевого порядка

  Open in Colab
- Подбор гиперпараметров модели машинного обучения в Keras с помощью Optuna
- A Tutorial on Zero-Order Optimization

Case 1: 2-Point & Multi-Point Estimators

- A naïve approach: $\mathsf{G}_f^{2n}(x;u) = \sum_{i=1}^{n} \frac{f(x+ue_i) - f(x-ue_i)}{2u} e_i$
- When f is L-smooth, we have

$$\|\mathsf{G}_f^{2n}(x;u) - \nabla f(x)\| \le \frac{1}{2} uL\sqrt{n}$$

where $f^u(x) = \mathbb{E}_{y\sim\tilde{\lambda}}[f(x+uy)]$ is a smooth version of f $\lambda = \mathrm{Uni}(\mathbb{B}_n)$

Case 1: 2-Point & Multi-Point Estimators

- 2-point gradient estimator:

  $\mathsf{G}_f^{(2)}(x;u,z) = n\frac{f(x+uz) - f(x-uz)}{2u} z \qquad z \sim \lambda$

  where λ is spherically symmetric with $\mathbb{E}_{z\sim\lambda}\left[\|z\|^2\right] = 1$

- Some facts for L-smooth / convex / μ-strongly convex function f:
  - f^u is L-smooth / convex / μ-strongly convex
  - $|f^u(x) - f(x)| \le \frac{1}{2}u^2 L \cdot \frac{n}{n+2}\mathbb{E}_{z\sim\lambda}\left[\|z\|^4\right]$

    $\|\nabla f^u(x) - \nabla f(x)\| \le uL \cdot \frac{n}{n+1}\mathbb{E}_{z\sim\lambda}\left[\|z\|^3\right]$

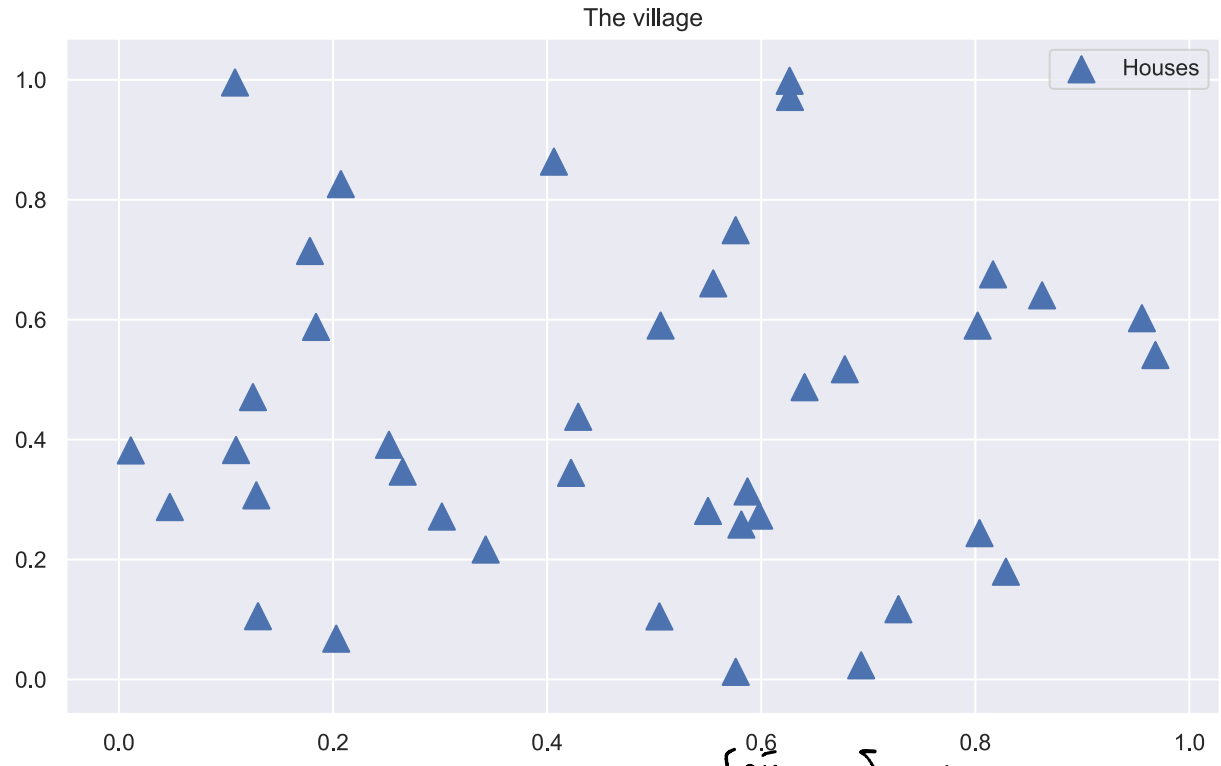
    $\mathbb{E}_{z\sim\lambda}\left[\left\|\mathsf{G}_f^{(2)}(x;u,z)\right\|^2\right]$
    $\le (1+\kappa)\mathbb{E}_{z\sim\lambda}\left[\|z\|^4\right] \cdot n\|\nabla f(x)\|^2$
    $+ \frac{1+\kappa}{4\kappa}\mathbb{E}_{z\sim\lambda}\left[\|z\|^6\right] \cdot n^2u^2L^2$

Travelling salesman problem

1 Problem

Suppose, we have N points in $\mathbb{R}^d$ Euclidian space (for simplicity, we'll consider and plot case with $d = 2$). Let's imagine, that these points are nothing else but houses in some 2d village. Salesman should find the shortest way to go through the all houses only once.

TSP задача коммивояжера



N домов

любой обход:

y = [1, 7, 100500, 2, ... 3]

↑ inp.range(N) + 1

Illustration

That is, very simple formulation, however, implies NP - hard problem with the factorial growth of possible combinations. The goal is to minimize the following cumulative distance:

$$d = \sum_{i=1}^{N-1} \|x_{y(i+1)} - x_{y(i)}\|_2^2 \to \min_y,$$

y = ? N!

where x_k is the k-th point from N and y stands for the N- dimensional vector of indicies, which describes the order of path. Actually, the problem could be formulated as an LP problem, which is easier to solve.

2 🧬 Genetic (evolution) algorithm

Our approach is based on the famous global optimization algorithm, known as evolution algorithm. ### Population and individuals Firstly, we need to generate the set of random solutions as an initialization. We will call a set of solutions $\{y_k\}_{k=1}^n$ as *population*, while each solution is called *individual* (or creature).

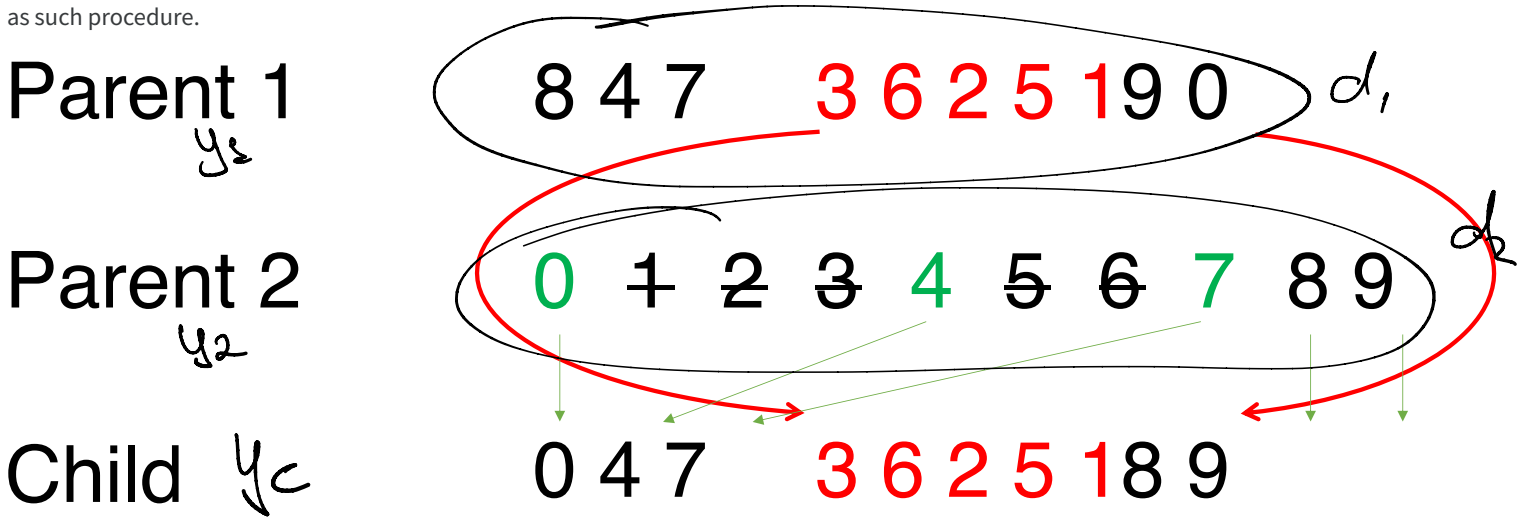
Each creature contains integer numbers $1, \ldots, N$, which indicates the order of bypassing all the houses. The creature, that reflects the shortest path length among the others will be used as an output of an algorithm at the current iteration (generation).

2.1 Crossing procedure

y - особь

популяция $\begin{pmatrix} y_1 \\ \vdots \\ y_P \end{pmatrix}$ ← $\frac{P}{2}$ потомков

Each iteration of the algorithm starts with the crossing (breed) procedure. Formally speaking, we should formulate the mapping, that takes two creature vectors as an input and returns its offspring, which inherits parents properties, while remaining consistent. We will use ordered crossover as such procedure.



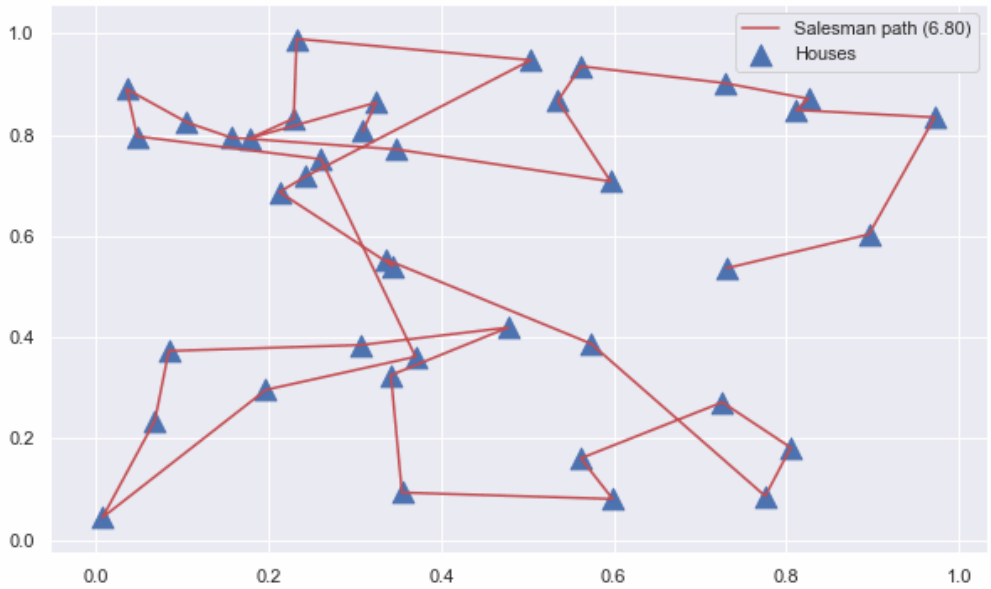
Illustration

2.2 Mutation

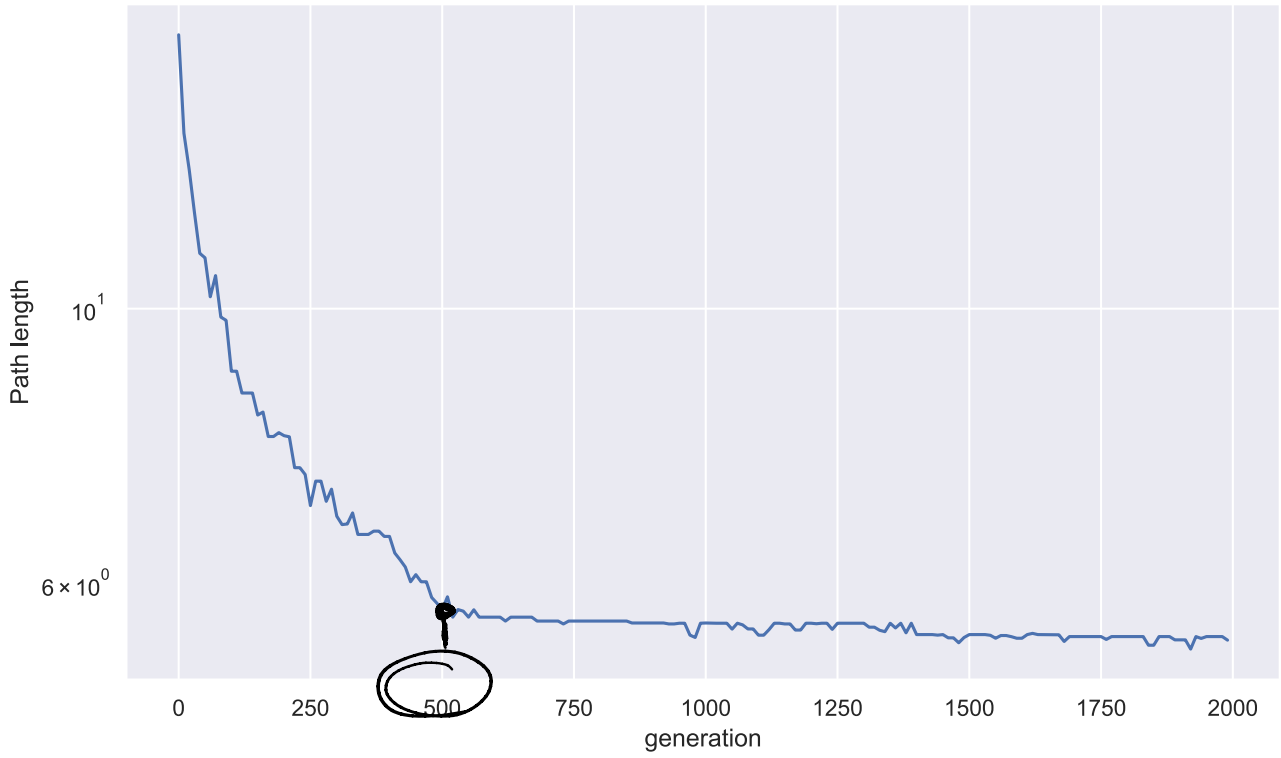
In order to give our algorithm some ability to escape local minima, we provide it with mutation procedure. We simply swap some houses in an individual vector. To be more accurate, we define mutation rate (say, 0.05). On the one hand, the higher the rate, the less stable the population is, on the other, the smaller the rate, the more often algorithm gets stuck in the local minima. We choose $\text{mutation rate} \cdot n$ individuals and in each case swap random $\text{mutation rate} \cdot N$ digits.

2.3 Selection

At the end of the iteration we have increased population (due to crossing results), then we just calculate total path distance to each individual and select top n of them.



Illustration



Illustration

In general, for any $c > 0$, where d is the number of dimensions in the Euclidean space, there is a polynomial-time algorithm that finds a tour of length at most $\left(1 + \frac{1}{c}\right)$ times the optimal for geometric instances of TSP in

$$\mathcal{O}\left(N(\log N)^{(\mathcal{O}(c\sqrt{d}))^{d-1}}\right)$$

3 Code

Open In Colab

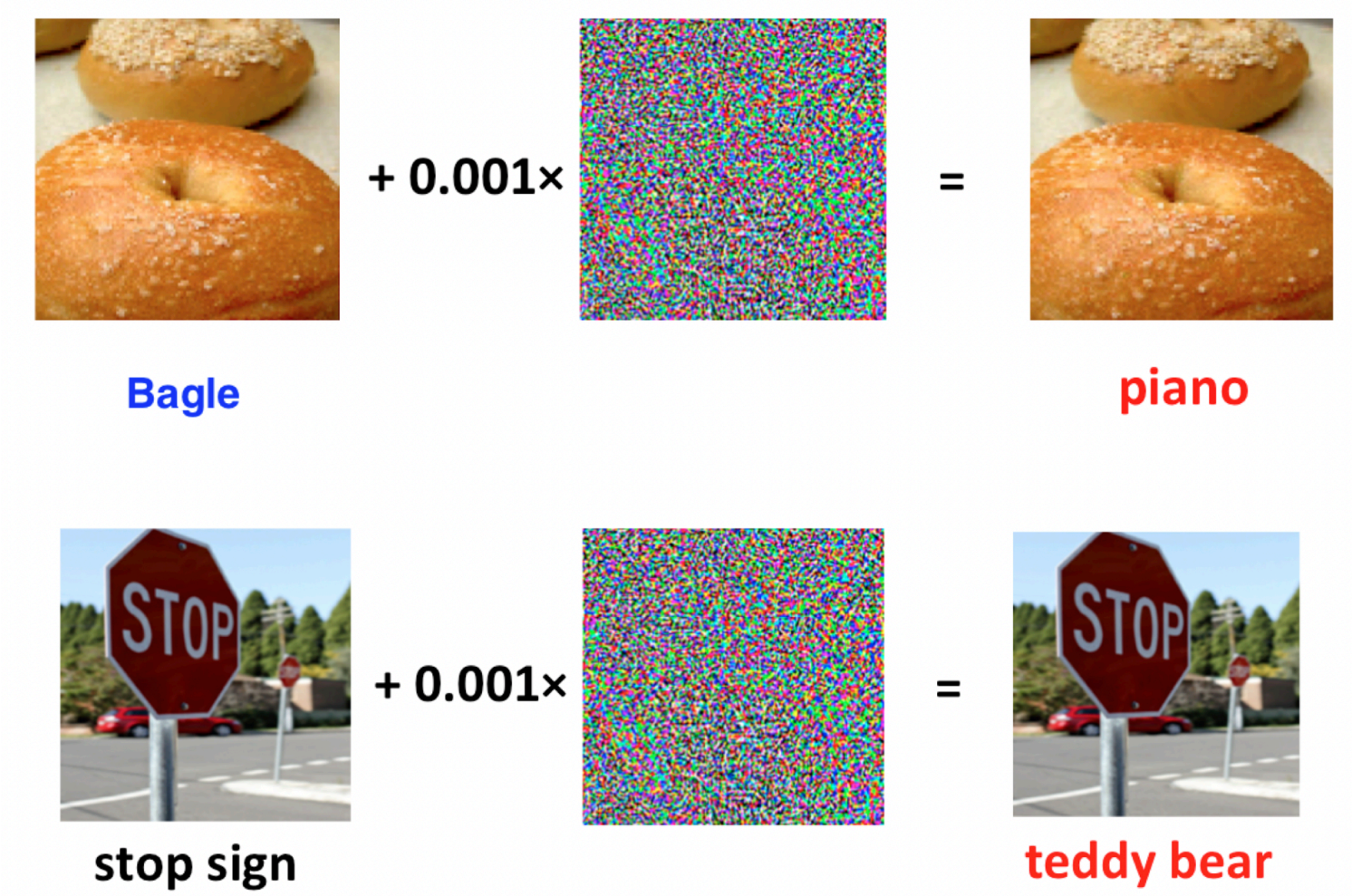
4 References

- General information about genetic algorithms
- Wiki

Neural network Lipschitz constant

1 Lipschitz constant and robustness to a small perturbation

It was observed, that small perturbation in Neural Network input could lead to significant errors, i.e. misclassifications.



Typical illustration of adversarial attacks in image domain. Source

Lipschitz constant bounds the magnitude of the output of a function, so it cannot change drastically with a slight change in the input

$$\|\mathcal{NN}(image) - \mathcal{NN}(image + \varepsilon)\| \leq L_{\mathcal{NN}}\|\varepsilon\|$$

Note, that a variety of feed-forward neural networks could be represented as a series of linear transformations, followed by some nonlinear function (say, $\mathrm{ReLU}\ (x)$):

$$\mathcal{NN}(x) = f_L \circ w_L \circ \ldots \circ f_1 \circ w_1 \circ x,$$

where L is the number of layers, f_i - non-linear activation function, $w_i = W_i x + b_i$ - linear layer.

2 Estimating Lipschitz constant of a neural network

Theorem

An everywhere differentiable function $f : \mathbb{R} \to \mathbb{R}$ is Lipschitz continuous with $L = \sup |\nabla f(x)|$ if and only if it has bounded first derivative.

Theorem

If $f = g_1 \circ g_2$, then $L_f \leq L_{g_1} L_{g_2}$

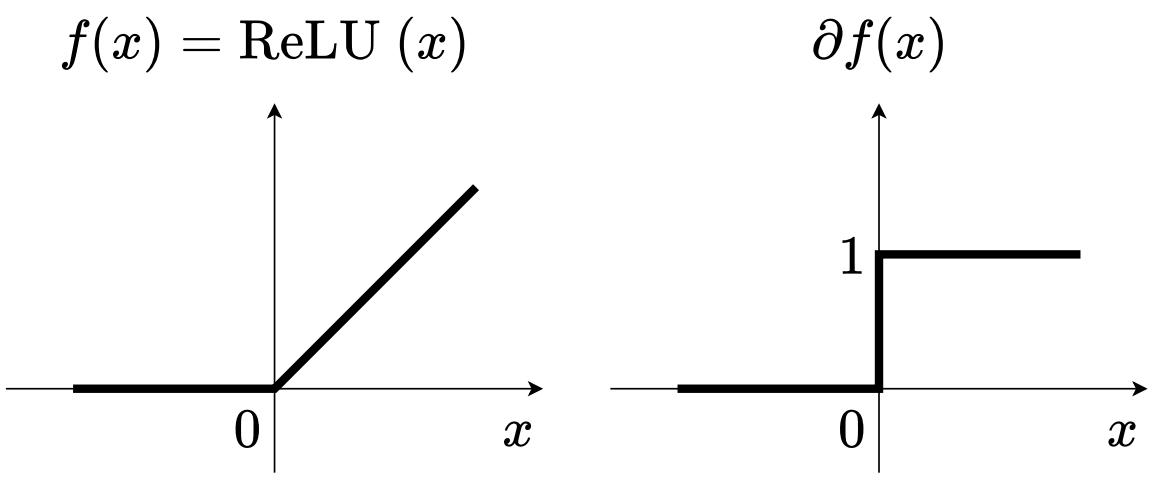
Therefore, we can bound the Lipschitz constant of a neural network:

$$L_{\mathcal{NN}} \leq L_{f_1} \dots L_{f_L} L_{w_1} \dots L_{w_L}$$

Example

Let's consider one the simplest non-liear activation function.

$$f(x) = \text{ReLU}(x)$$



Its Lipschitz L_f constant equals to 1, because:

$$\|f(x) - f(y)\| \leq 1 \|x - y\|$$

Question

What is the Lipschitz constant of a linear layer of neural network

$$w(x) = Wx + b$$

Answer

3 How to compute $\|W\|_2$?

Let $W = U \Sigma V^T$ – SVD of the matrix $W \in \mathbb{R}^{m \times n}$. Then

$$\|W\|_2 = \sup_{x \neq 0} \frac{\|Wx\|_2}{\|x\|_2} = \sigma_1(W) = \sqrt{\lambda_{\max}(W^* W)}$$

For $m = n$, computing SVD is $\mathcal{O}(n^3)$.

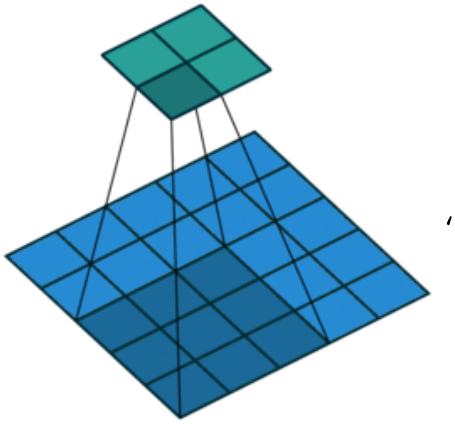
Time measurements with jax and Google colab CPU.

n	10	100	1000	5000
Time	38.7 µs	3.04 ms	717 ms	1min 21s
Memory for W in fp32	0.4 KB	40 KB	4 MB	95 MB

Works only for small linear layers.

In this notebook we will try to estimate Lipschitz constant of some convolutional layer of a Neural Network.

4 Convolutional layer



Animation of Convolution operation. Source

Suppose, that we have an input X and the convolutional layer C with the filter size $k \times k$. Here we assume, that p_1, p_2 - are the indices of pixels of the kernel, while q_1, q_2 are the indices of pixels of the output.

$$C \circ X = \sum_{p_1=0}^{k-1} \sum_{p_2=0}^{k-1} C_{p_1,p_2} X_{p_1+q_1,p_2+q_2}$$

While multichannel convolution could be written in the following form:

$$Y_j = \sum_{i=1}^{m_{in}} C_{:,:,i,j} \circ X_{:,:,i},$$

where

- $C \in \mathbb{R}^{k \times k \times m_{in} \times m_{out}}$ – convolution kernel
- k – filter size
- m_{in} – number of input channels (e.g., 3 for RGB)
- m_{out} – number of output channels.

It is easy to see, that the output of the multichannel convolution operation is linear w.r.t. the input X, which actually means, that one can write it as matvec:

$$\text{vec}\,(Y) = W \text{vec}\,(X)$$

Question

What is the size of the matrix W in this case, if the input image is square of size n?

Answer

Example

If the image size is $n = 224$, number if input and output channels are $m_{in} = m_{out} = 64$, then $W \in \mathbb{R}^{3\,211\,264 \times 3\,211\,264}$. Storing this matrix in `fp32` requW^T W x_kires approximately 40 Gb of RAM.

It seems, that computing $\|W\|_2$ is almost impossible, isn't it?

5 Power method for computing the largest singular value

Since we are interested in the largest singular value and

$$\|W\|_2 = \sigma_1(W) = \sqrt{\lambda_{\max}(W^*W)}$$

we can apply the power method

$$x_{k+1} = \frac{W^T W x_k}{\|W^T W x_k\|_2}$$

$$\sigma_{k+1} = \frac{W x_{k+1}}{\|x_{k+1}\|_2}$$

6 Code

Open In Colab{: .btn }

7 References

- Maxim Rakhuba's lecture on Matrix and tensor methods in ML.