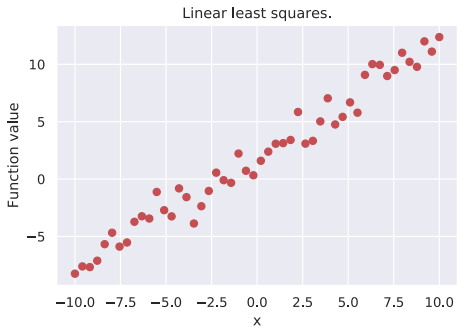
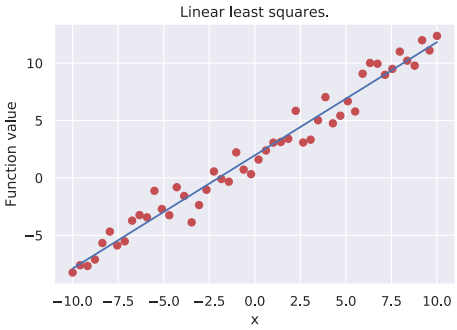
Application of optimization

Linear least squares

Problem





In a least-squares, or linear regression, problem, we have measurements $X \in \mathbb{R}^{m \times n}$ and $y \in \mathbb{R}^m$ and seek a vector $\theta \in \mathbb{R}^n$ such that $X\theta$ is close to y. Closeness is defined as the sum of the squared differences:

$$\sum_{i=1}^{m} (x_i^\top \theta - y_i)^2$$

also known as the l_2-norm squared, $\|X\theta - y\|_2^2$

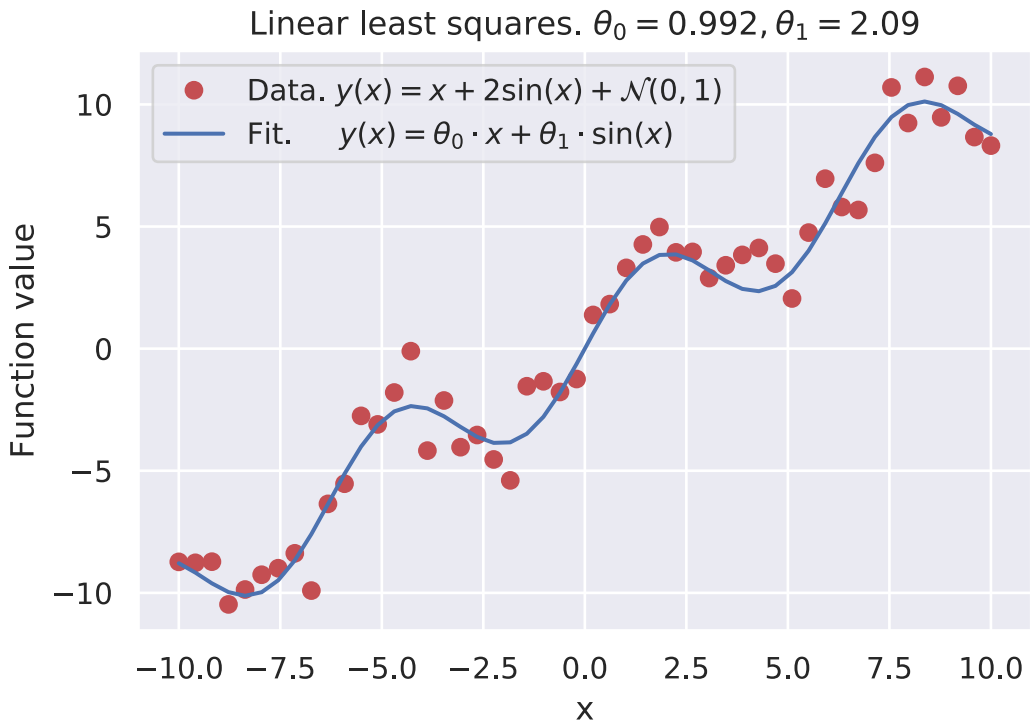
For example, we might have a dataset of m users, each represented by n features. Each row $x_i^\top$ of X is the features for user i, while the corresponding entry y_i of y is the measurement we want to predict from $x_i^\top$, such as ad spending. The prediction is given by $x_i^\top \theta$.

We find the optimal θ by solving the optimization problem

$$\|X\theta - y\|_2^2 \to \min_{\theta \in \mathbb{R}^n}$$

Let θ^* denote the optimal θ. The quantity $r = X\theta^* - y$ is known as the residual. If $\|r\|_2 = 0$, we have a perfect fit.

Note, that the function needn't be linear in the argument x but only in the parameters θ that are to be determined in the best fit.



Approaches

Moore–Penrose inverse

If the matrix X is relatively small, we can write down and calculate exact solution:

$$\theta^* = (X^\top X)^{-1} X^\top y = X^\dagger y,$$

where $X^\dagger$ is called pseudo-inverse matrix. However, this approach squares the condition number of the problem, which could be an obstacle in case of ill-conditioned huge scale problem.

QR decomposition

For any matrix $X \in \mathbb{R}^{m \times n}$ there is exists QR decomposition:

$$X = Q \cdot R,$$

where Q is an orthogonal matrix (its columns are orthogonal unit vectors meaning $Q^\top Q = Q Q^\top = I$ and R is an upper triangular matrix. It is important to notice, that since $Q^{-1} = Q^\top$, we have:

$$Q R \theta = y \quad \longrightarrow \quad R\theta = Q^\top y$$

Now, process of finding theta consists of two steps:

1. Find the QR decomposition of X.
2. Solve triangular system $R\theta = Q^\top y$, which is triangular and, therefore, easy to solve.

Cholesky decomposition

For any positive definite matrix $A \in \mathbb{R}^{n \times n}$ there is exists Cholesky decomposition:

$$X^\top X = A = L^\top \cdot L,$$

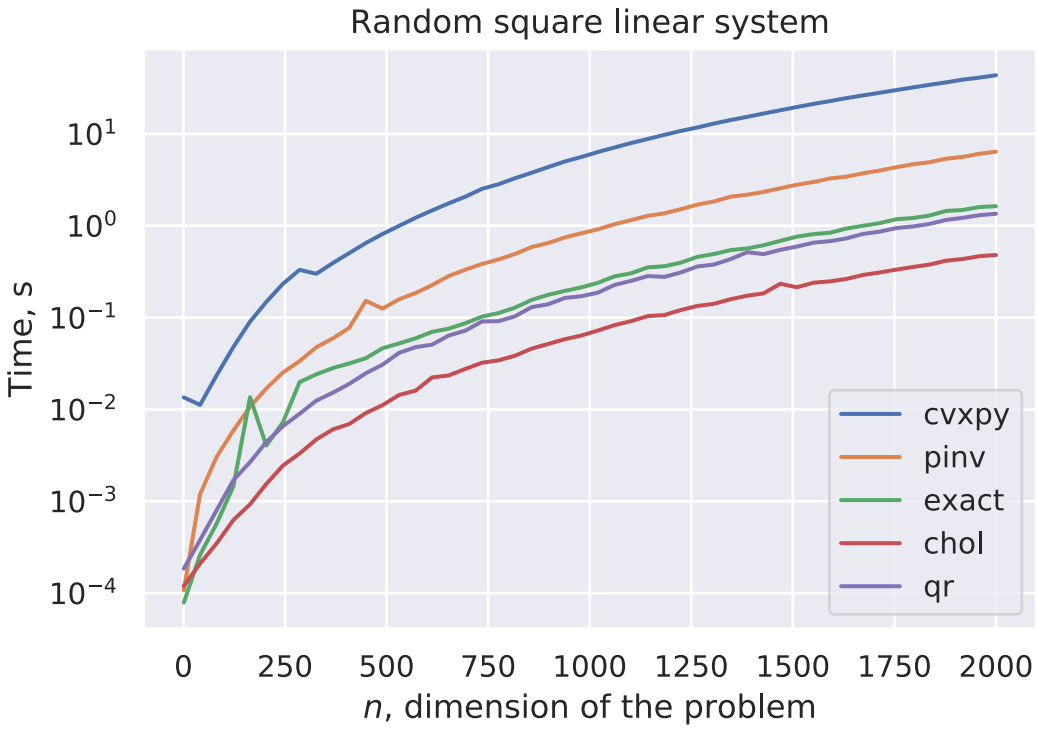
where L is an lower triangular matrix. We have:

$$L^\top L\theta = y \quad \longrightarrow \quad L^\top z_\theta = y$$

Now, process of finding theta consists of two steps:

1. Find the Cholesky decomposition of $X^\top X$.
2. Find the $z_\theta = L\theta$ by solving triangular system $L^\top z_\theta = y$
3. Find the θ by solving triangular system $L\theta = z_\theta$

Note, that in this case the error stil proportional to the squared condition number.



Code

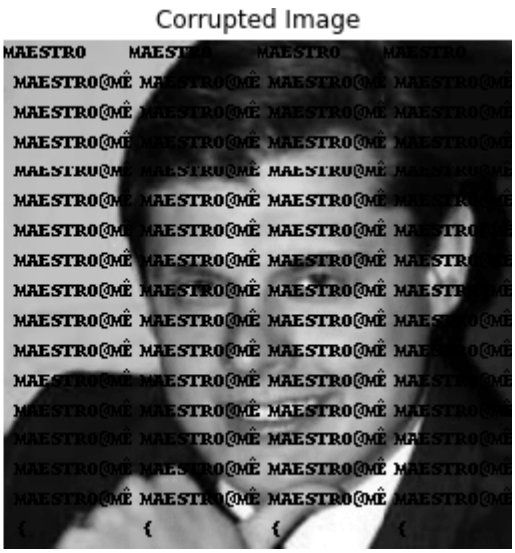
Open in Colab

References

- CVXPY documentation
- Interactive example
- Jupyter notebook by A. Katrutsa

Total variation in-painting

Problem



Grayscale image

A grayscale image is represented as an $m \times n$ matrix of intensities U^{orig} (typically between the values 0 and 255). We are given all the values of corrupted picture, but some of them should be preserved as is through the recovering procedure: U_{ij}^{corr} $\forall (i,j) \in K$, where $K \subset \{1,\dots,m\} \times \{1,\dots,n\}$ is the set of indices corresponding to known pixel values. Our job is to in-paint the image by guessing the missing pixel values, i.e., those with indices not in K. The reconstructed image will be represented by $U \in \mathbb{R}^{m \times n}$, where U matches the known pixels, i.e., $U_{ij} = U_{ij}^{corr}$ for $(i,j) \in K$.

The reconstruction U is found by minimizing the total variation of U, subject to matching the known pixel values. We will use the l_2 total variation, defined as

$$\mathbf{tv}(U) = \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} \left\| \begin{bmatrix} U_{i+1,j} - U_{ij} \\ U_{i,j+1} - U_{ij} \end{bmatrix} \right\|_2.$$

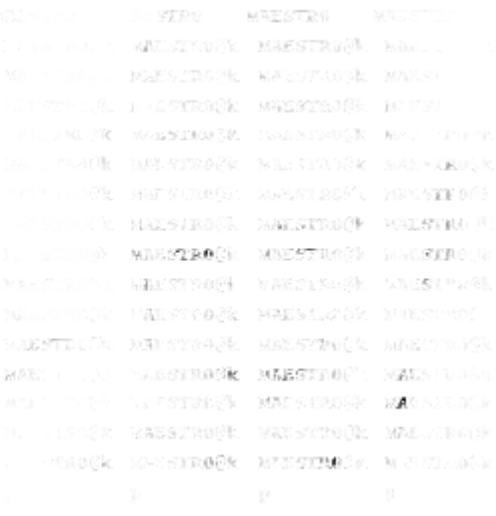
So, the final optimization problem will be written as follows:

$$\mathbf{tv}(U) \to \min_{U \in \mathbb{R}^{m \times n}}$$
$$\text{s.t. } U_{ij} = U_{ij}^{corr},\ (i,j) \in K$$

The crucial thing about this problem is defining set of known pixels K. There are some heuristics: for example, we could state, that each pixel with color similar (or exactly equal) to the color of text is unknown. The results for such approach are presented below:

In-Painted Image

Difference Image



Color image

For the color case we consider in-painting problem in a slightly different setting: destroying some random part of all pixels. In this case the image itself is 3d tensor (we convert all others color chemes to the RGB). As it was in the grayscale case, we construct the mask K of known pixels for all color channels uniformly, based on the principle of similarity of particular 3d pixel to the vector $[0, 0, 0]$ (black pixel). The results are quite promising - note, that we have no information about the original picture, but assumption, that corrupted pixels are black. For the color picture we just sum all tv's on the each channel:

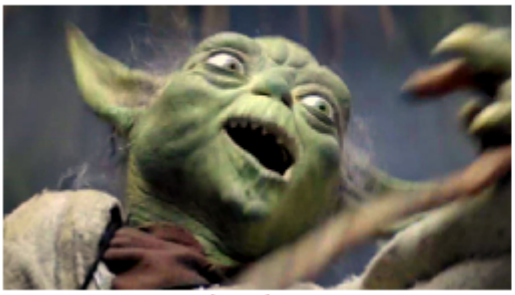
$$\mathbf{tv}(U) = \sum_{k=1}^{3}\sum_{i=1}^{m-1}\sum_{j=1}^{n-1}\left\|\begin{bmatrix} U_{i+1,j}^k - U_{ij}^k \\ U_{i,j+1}^k - U_{ij}^k \end{bmatrix}\right\|_2.$$

Then, we need to write down optimization problem to be solved:

$$\mathbf{tv}(U) \to \min_{U \in \mathbb{R}^{m\times n\times 3}}$$
$$\text{s.t. } U_{ij}^k = U_{ij}^{corr,k},\ (i,j) \in K,\ k = 1,2,3$$

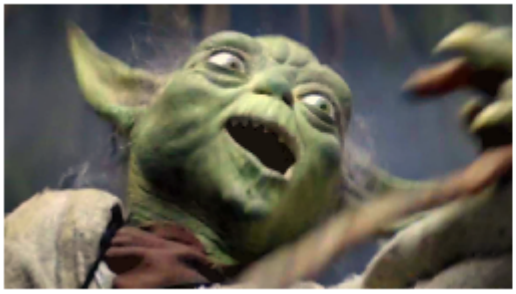
Results are presented below (these computations are really take time):

Original Image

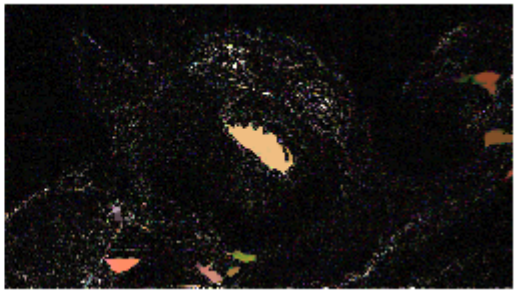


Corrupted Image. Corruption level 0.707

In-Painted Image

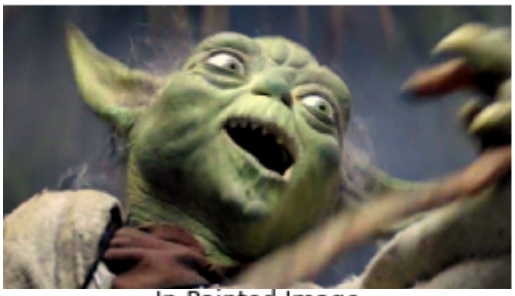


Difference Image

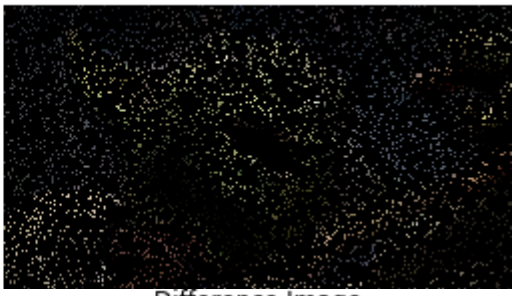


It is not that easy, right?

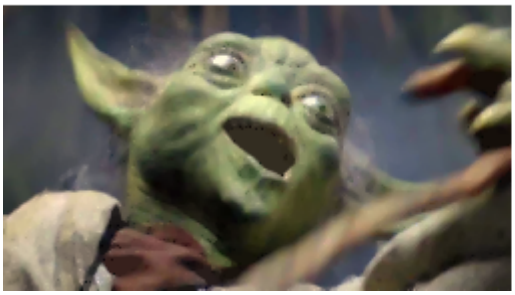
Original Image



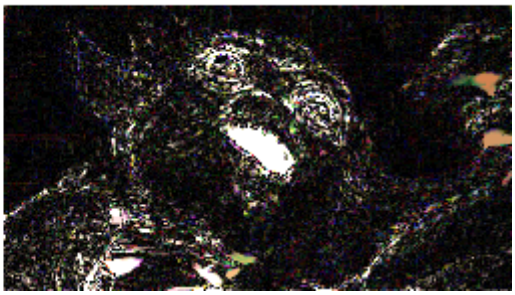
Corrupted Image. Corruption level 0.9



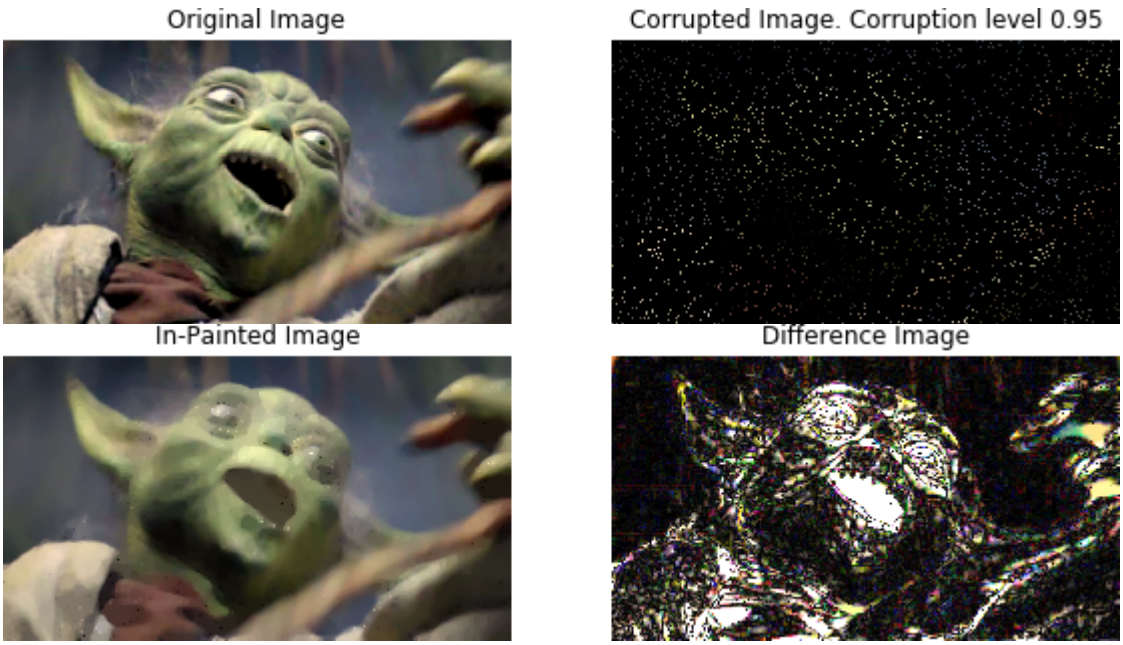
In-Painted Image



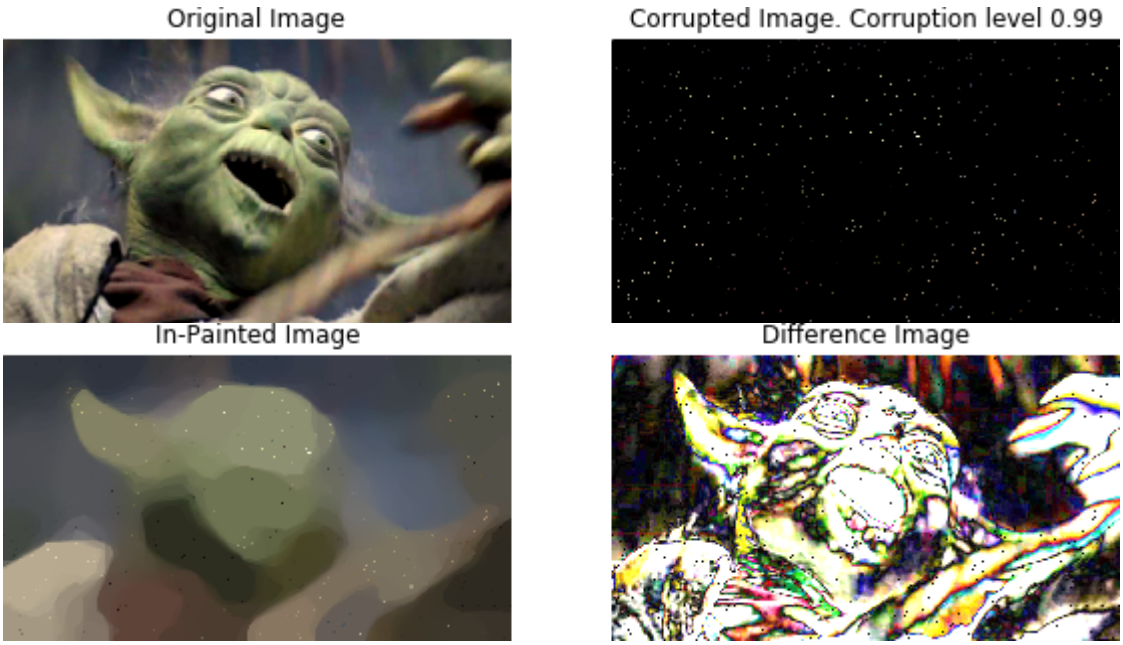
Difference Image



Only 5% of all pixels are left:



What about 1% of all pixels?



Code

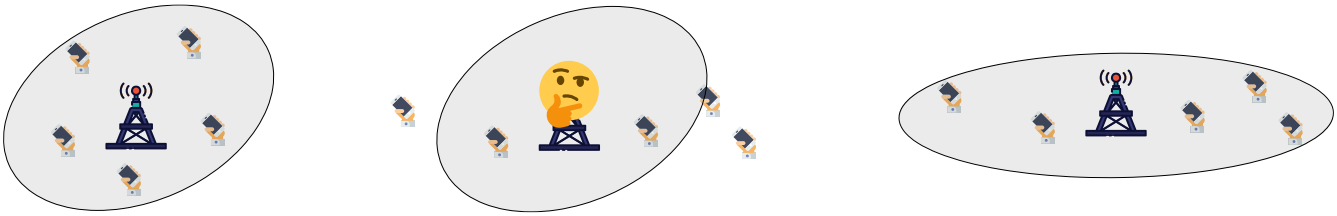
Open in Colab

References

- CVXPY documentation
- Interactive demo

Minimum volume ellipsoid

Problem



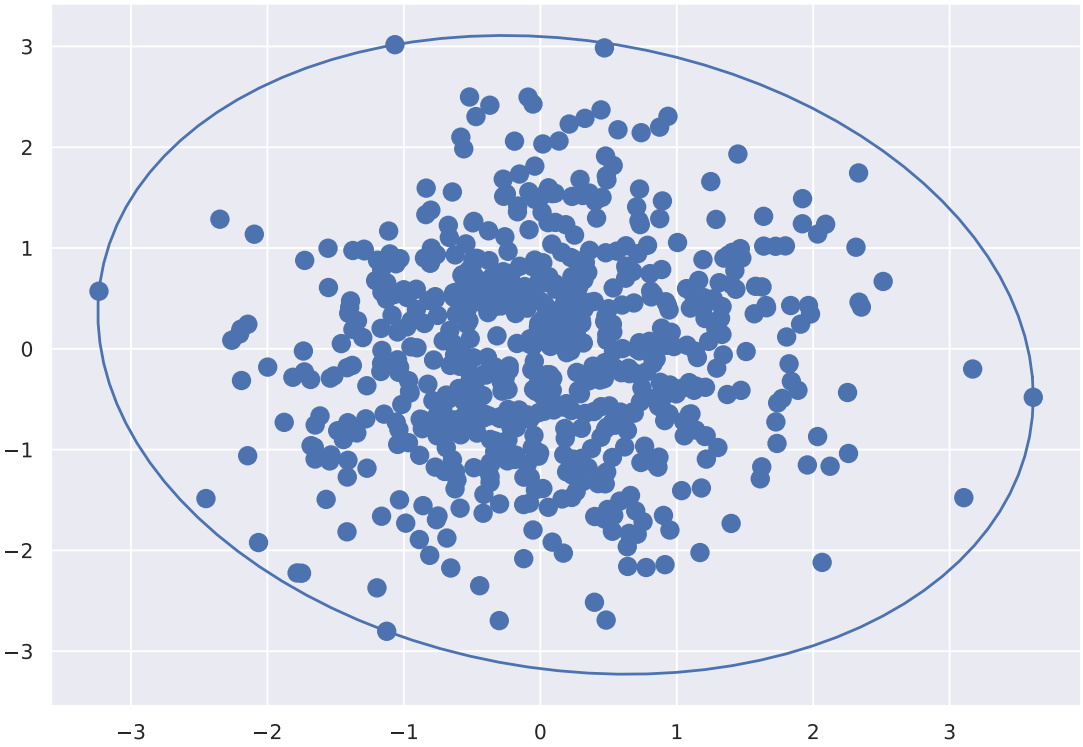
Let $x_1, \dots, x_n$ be the points in $\mathbb{R}^2$. Given these points we need to find an ellipsoid, that contains all points with the minimum volume (in 2d case volume of an ellipsoin is just the square).

An invertible linear transformation applied to a unit sphere produces an ellipsoid with the square, that is $\det A^{-1}$ times bigger, than the unit sphere square, that's why we parametrize the interior of ellipsoid in the following way:

$$S = \{x \in \mathbb{R}^2 \mid u = Ax + b, \|u\|_2^2 \leq 1\}$$

Sadly, the determinant is the function, which is relatively hard to minimize explicitly. However, the function $\log \det A^{-1} = -\log \det A$ is actually convex, which provides a great opportunity to work with it. As soon as we need to cover all the points with ellipsoid of minimum volume, we pose an optimization problem on the convex function with convex restrictions:

$$\begin{aligned} \min_{A \in \mathbb{R}^{2\times 2}, b \in \mathbb{R}^2} & -\log \det(A) \\ \text{s.t. } & \|Ax_i + b\| \leq 1, i = 1, \dots, n \\ & A \succ 0 \end{aligned}$$



Code

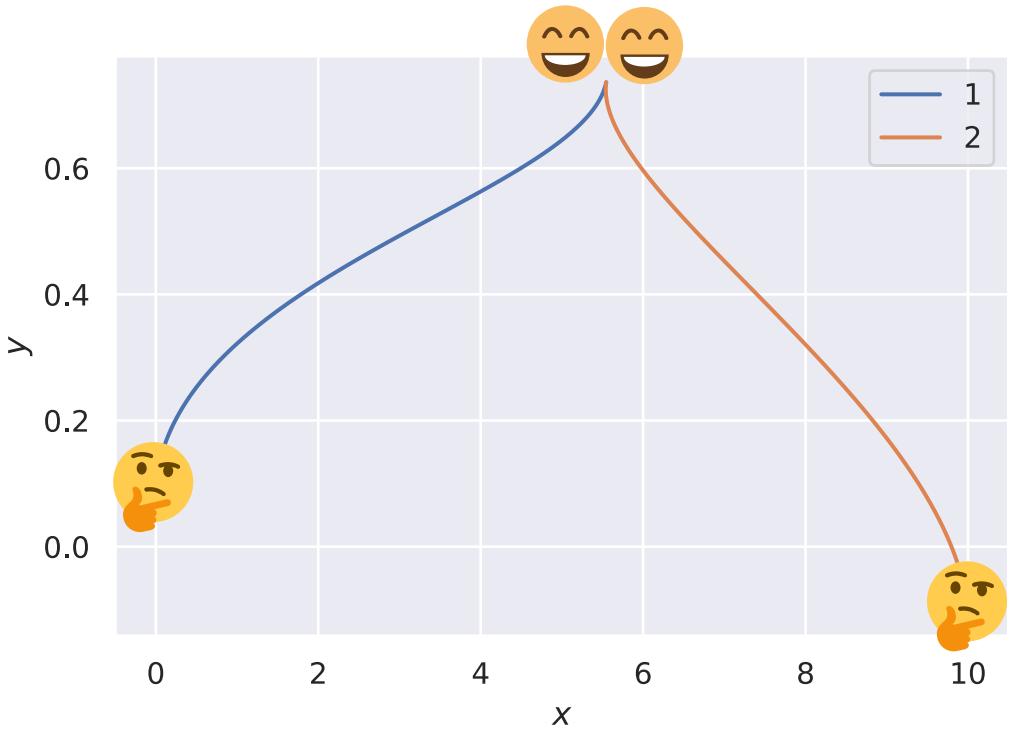
Open in Colab

References

- Jupyter notebook by A. Katrutsa
- https://cvxopt.org/examples/book/ellipsoids.html

Rendezvous problem

Problem



We have two bodies in discrete time: the first is described by its coordinate x_i and its speed v_i, the second has coordinate z_i and speed u_i. Each body has its own dynamics, which we denote as linear systems with matrices A, B, C, D:

$$x_{i+1} = Ax_i + Bu_i$$
$$z_{i+1} = Cz_i + Dv_i$$

We want these bodies to meet in future at some point T in such a way, that preserve minimum energy through the path. We will consider only kinetic energy, which is proportional to the squared speed at each point of time, that's why optimization problem takes the following form:

$$\min \sum_{i=1}^{T} \|u_i\|_2^2 + \|v_i\|_2^2$$
$$\text{s.t. } x_{t+1} = Ax_t + Bu_t,\ t = 1, \ldots, T-1$$
$$z_{t+1} = Cz_t + Dv_t,\ t = 1, \ldots, T-1$$
$$x_T = z_T$$

Problem of this type arise in space engeneering - just imagine, that the first body is the spaceship, while the second, say, Mars.

Code

Open in Colab

References

- Jupyter notebook by A. Katrutsa

Travelling salesman problem

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.

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:

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

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.

Genetic (evolution) algorithm

Our approach is based on the famous global optimization algorithm, known as evolution algorithm.

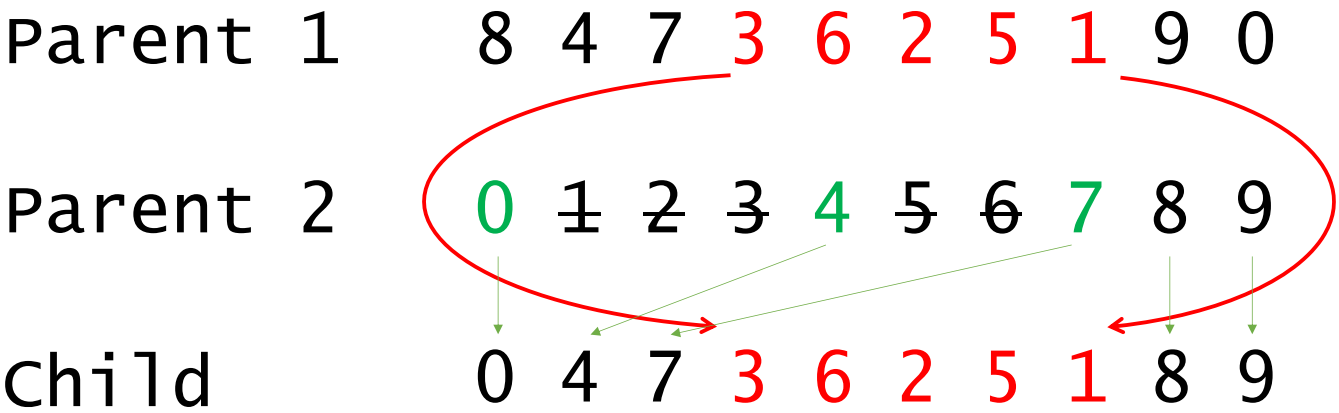
Population and individuals

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 $1, \dots, N$, 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).

Crossing procedure

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.

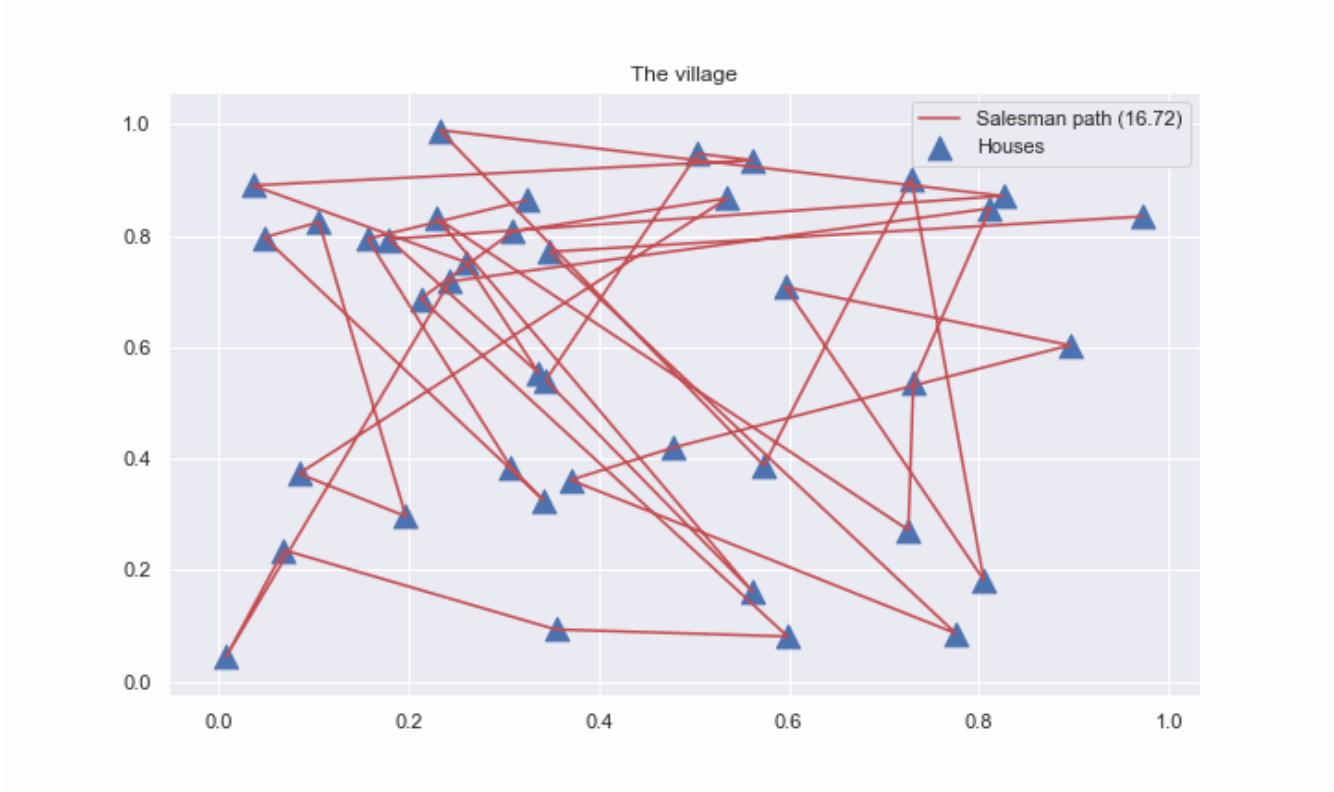


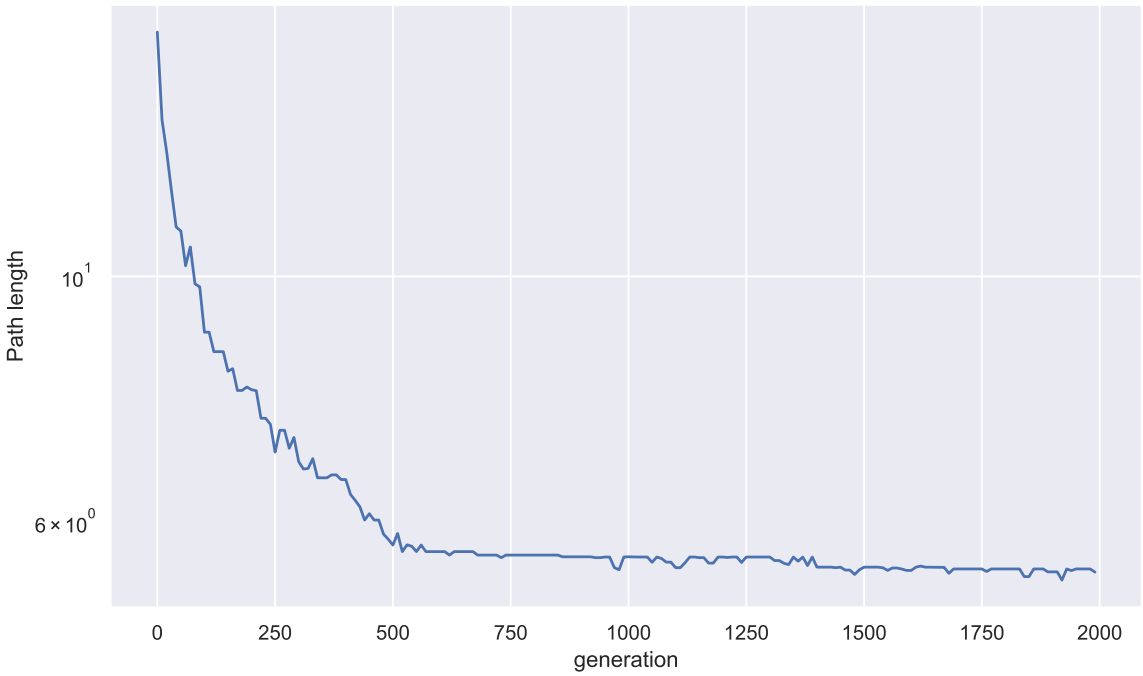
Mutation

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.

Selection

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





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 $(1 + \frac{1}{c})$ times the optimal for geometric instances of TSP in

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

Code

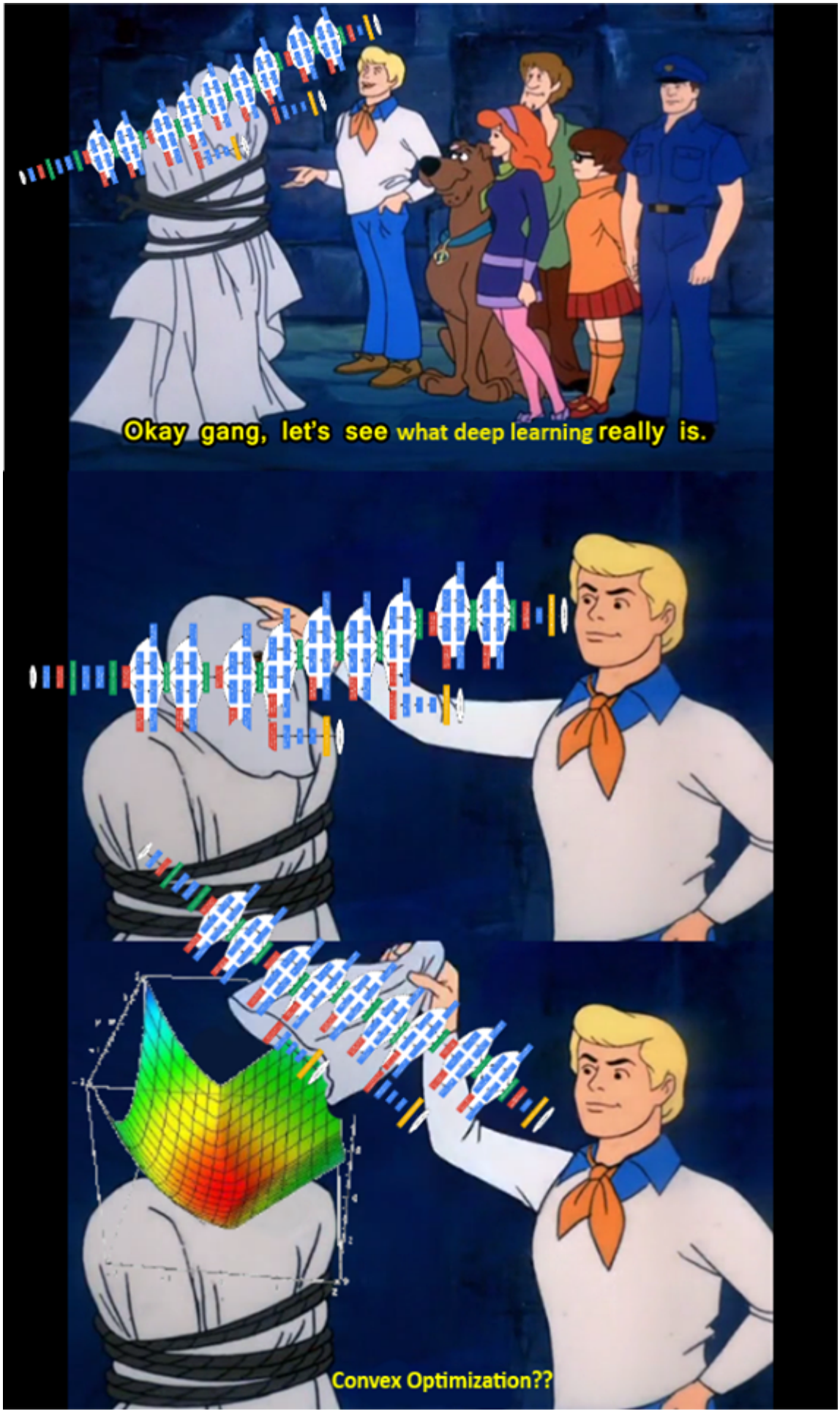
Open in Colab

References

- General information about genetic algorithms
- Wiki

Deep learning

Problem


Okay gang, let's see what deep learning really is.
Convex Optimization??

A lot of practical task nowadays are being solved by the deep learning approach, which is usually implies finding local minimum of a non - convex function, that generalizes well (enough 😉). The goal of this short text is to provide you an importance of the optimization behind neural network training.

Cross entropy

One of the most commonly used loss functions in classification tasks is the normalized categorical cross entropy in K class problem:

$$L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}(y_i^\top \log(h_\theta(x_i)) + (1-y_i)^\top \log(1-h_\theta(x_i))), \qquad h_\theta^k(x_i) = \frac{e^{\theta_k^\top x_i}}{\sum_{j=1}^{K} e^{\theta_j^\top x_i}}$$

Since in Deep Learning tasks the number of points in a dataset could be really huge, we usually use {%include link.html title='Stochastic gradient descent based approaches as a workhorse.

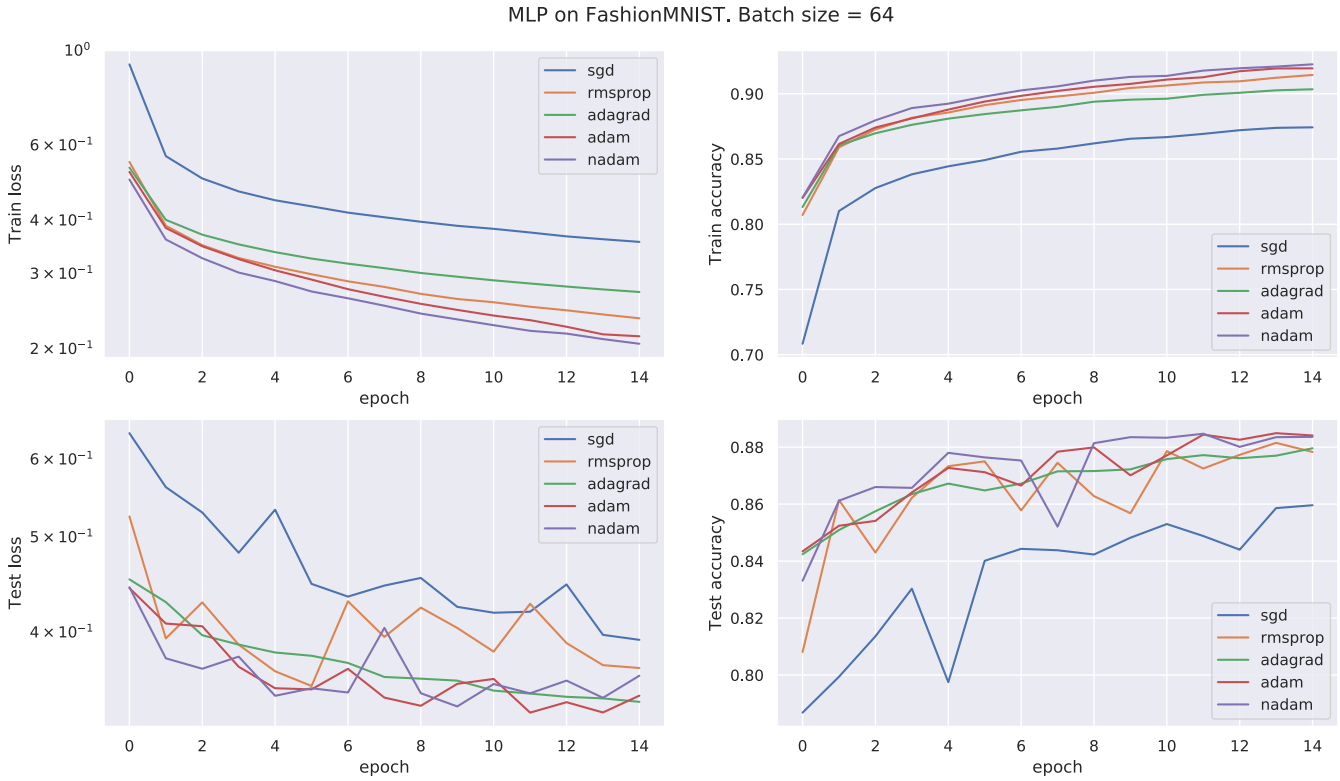
In such algorithms one uses the estimation of a gradient at each step instead of the full gradient vector, for example, in cross entropy we have:

$$\nabla_\theta L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(h_\theta(x_i) - y_i)\, x_i^\top$$

The simplest approximation is statistically judged unbiased estimation of a gradient:

$$g(\theta) = \frac{1}{b}\sum_{i=1}^{b}(h_\theta(x_i) - y_i)\, x_i^\top \approx \nabla_\theta L(\theta)$$

where we initially sample randomly only $b \ll n$ points and calculate sample average. It can be also considered as a noisy version of the full gradient approach.



Code

Open in Colab

References

- Optimization for Deep Learning Highlights in 2017
- An overview of gradient descent optimization algorithms