Master Mathématiques, Apprentissage, Science et Humanité

Project of Optimization for Machine Learning.

Author :
Emma Kopp

Supervisors :
Vincent Duval
Gabriel Peyré
Clément Royer

2021 - 2022

Table des matières

1 Introduction

The purpose of this project is to illustrate theoretical results studied during the lectures of "Optimization for Machine Learning". These results can be found Peyré [2021], Royer [2021a] Royer [2021b].

I choosed the dataset ”Diamonds” from Kaggle. It contains the prices and other attributes of almost $54,000$ diamonds. I will try during this project to build a function which predict the price of the diamonds from their type properties. The formulation of the problem is inspired from the Numerical Tours of Gabriel Peyre and the Lab session of Clément Royer.

2 Presentation of the problem

2.1 Datas

This dataset contains the prices and other attributes of 53940 diamonds. There are 11 columns. After removing lines where there are missing values it lefts 53940 lines. I choosed to keep 5 qauantitatives features.

I decided to separate my dataframe into 2 parts. The 50% most expensives diamonds and the 50% cheapests ones. We remark that the distribution of the cheapest one have less than 1 carat and the expensives one are grouped around 1 or 2.

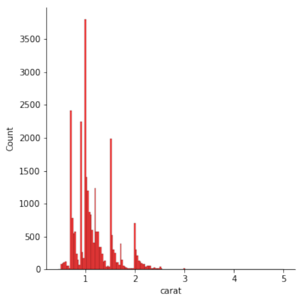


FIGURE 1 – Expensive carat.

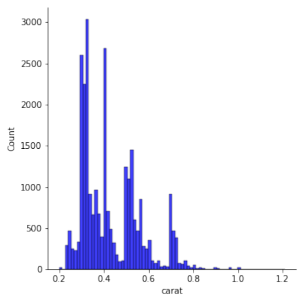
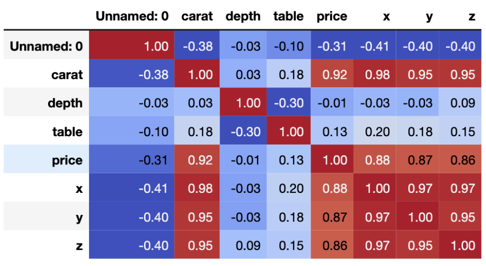


FIGURE 2 – Not expensive carat.

I graphed the correlation matrix of my dataset. We can see that the price variable has a high correlation with all variables except the depth one (0.01 correlation value), meaning that the leaving time of the diamond doesn't have a big impact on the price. Thus, I removed the depth variable.



	Unnamed: 0	carat	depth	table	price	x	y	z
Unnamed: 0	1.00	-0.38	-0.03	-0.10	-0.31	-0.41	-0.40	-0.40
carat	-0.38	1.00	0.03	0.18	0.92	0.98	0.95	0.95
depth	-0.03	0.03	1.00	-0.30	-0.01	-0.03	-0.03	0.09
table	-0.10	0.18	-0.30	1.00	0.13	0.20	0.18	0.15
price	-0.31	0.92	-0.01	0.13	1.00	0.88	0.87	0.86
x	-0.41	0.98	-0.03	0.20	0.88	1.00	0.97	0.97
y	-0.40	0.95	-0.03	0.18	0.87	0.97	1.00	0.95
z	-0.40	0.95	0.09	0.15	0.86	0.97	0.95	1.00

FIGURE 3 – Correlation matrix

2.2 Theory

We will consider a regression problem with a quadratic cost. The problem is of the form :

$$\begin{aligned}\min_{\theta\in\mathbb{R}^d} f(\theta) &= \frac{1}{2n}\|X\theta - y\|^2 \\ &= \frac{1}{2n}\sum_{i=1}^{n}\left(x_i^T\theta - y_i\right)^2 .\end{aligned}$$

We decided to divide the objective function by n, in order to treat the same function in all the project. Also, we can rewrite the problem as a quadratic function :

$$\begin{aligned} f(\theta) &= \frac{1}{2n}\|X\theta - y\|^2 \\ &= \frac{1}{2n}\left(< X^TX\theta, \theta > - < \theta, X^Ty >\right) \\ &= \frac{1}{2} < C, \theta > - < \theta, b > .\end{aligned}$$

with $C := X^TX/n \in \mathbb{R}^{d\times d}$ and $b := X^Ty/n \in \mathbb{R}^d$.

We have $\nabla^2 f(\theta) = C$. Hence, f is $\lambda_{\min}(C)$-Lipschitz as $\nabla^2 f(\theta) \geq \lambda_{\min}(C) I_d \iff \forall\theta\lambda\left(\nabla^2 f(\theta)\right) \geq \lambda_{\min}(C)$ and f is $\lambda_{\max}(C)$-smoot as $\|\nabla^2 f(\theta)\|_{\text{op}} = \lambda_{\max}(C)$.

Denonting $\mu = \lambda_{\min}(C) = 0.17$ and $L = \lambda_{\max}(C) = 3.88$, we have that f is $C_L^{1,1}$ and μ-strongly convex. We will then applie some results that we studied during the semester.

Since the number of observation is larger than the number of features ($n > p$), the problem is over-determined. We can compute the Ordinalry-Least Squares (OLS) solution of the problem :

$$w^{OLS} = (X^TX)^{-1}X^Ty.$$

But even though we have some theoretical results on the solution of the problem, we aim to use faster algorithm of gradient descent. Indeed, when the design matrice is not well-conditionned, the ordinary-least square solution can be very long to compute.

3 Gradient descent

In this part, we will investigate different algorithms of gradient descent. First we can introduce the classical algorithm. Note that the main objective of the gradient descent algorithm (and it's derivatives), is to construct à sequence $(\theta_k)_{k\in\mathbb{R}}$ such that $\forall k$, $f(\theta_{k+1}) > f(\theta_k)$. We call it the descent property.

Algorithm 1: Gradient Descent

```
Initialization : $\theta_0 \in \mathbb{R}^d$
for $k = 0, 1, ...$ do
    Step 1 : Compute the gradient $\nabla f(\theta_k)$
    Step 2 : Compute a stepsize $\alpha_k > 0$
    Step 3 : Set $\theta_{k+1} = \theta_k - \alpha_k \nabla f(\theta_k)$
end
```

As f is differentiable, $f(\theta - \alpha_k\nabla f(\theta)) = f(\theta) - \alpha_k\|\nabla f(\theta)\|^2 + o(\alpha_k)$. Thus, to choose the direction of the negative gradient.

In this part, the choice of the stepsize sequence $(\alpha_k)_k$ will play a major role. We will present and compare different stepchoices.

3.1 Constant stepsize

We will first consider the simpliest choice of stepsize, the constant stepsize. Hence, $\forall k \in \mathbb{R}, \alpha_k = \alpha \ > 0$. The idea is to select the best α. Theoretical results guarantee the descent property when $0 < \alpha < \frac{2}{L}$. (proposition 2.1.1 of Royer [2021a]). Also, under the assumptions of smoothness and strong convexity, the optimal step size is $\alpha_{opt} = \frac{2}{\mu+L}$ (proposition 4 Peyré [2021]).

I ran the gradient descent algorithm for the optimal value and 4 random value between 0 and $\frac{2}{L}$. Concerning the optimal value, when the design matrix isn't well-conditioned, i.e $\mu \approx 0$, then $\alpha_{opt} \approx \frac{2}{L}$. But for this value, we cannot guarantee the convergance of the algorithm. Hence, the optimal value can diverge depending on the data context.

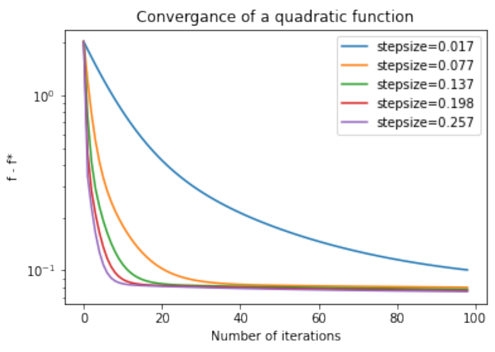


FIGURE 4 – Various tests of gradient descent with constant stepsize.

The different test of the algorithm are relevant. The red curve correponds to the optimal value and is the fastest. It's MSE is of 15%, which is a good accuracy.

3.2 Backtracking linesearch

The backtracking line search is a method where the stepsize is non-constant. It aims to calculate at each iteration the new stepsize, such that $f(\theta_k - \alpha_k \nabla f(\theta_k))$ is "sufficiently smaller than $f(\theta_k)$". The algorithm is defined as :

Algorithm 2: Backtracking line search in direction $-\nabla f(\theta_k)$

```
Input : $\theta_k \in \mathbb{R}^d$, $\nabla f(\theta_k) \in \mathbb{R}^d$, $\alpha^0 \in \mathbb{R}$ , $r < 1$
Initialization : Set $\alpha = \alpha^0$ and $j = 0$
while $f(\theta_k - \alpha \nabla f(\theta_k)) > f(\theta_k)$ do
    Set $\alpha = r\alpha$
end
Output : $\alpha_j$.
```

We choosed to set $r = \frac{1}{2}$ and obtain the following results.

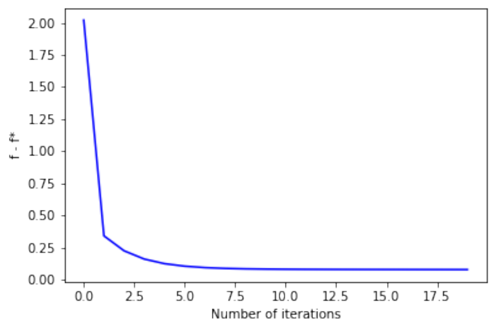


FIGURE 5 – Gradient descent with backtracking line search for $\alpha = \frac{1}{L}$.

The convergence is faster in terms of iteration (around 5) and the MSE is of 15%. However, one iteration requires to evaluate the objective function, which can be expensive sometimes.

3.3 Accelerated Methods

We will derive accelerated method. The accelerated method use the momentum step, which permits to consider a different evaluation of the gradient. These techniques are adapted to the structure of our objective function. We will focus on two methods. The Heavy-Ball algorithm and the Nesterov algorithm.

The Heavy Ball method is adapted to μ-strongly convex and L-smooth functions. The algorithm can be defined as :

Algorithm 3: Heavy Ball Method

```
Initialization : $\theta_0 \in \mathbb{R}^d$, $\theta_{-1} = \theta_0$
for $k = 0, 1, ...$ do
    Step 1 : Compute the gradient $\nabla f(\theta_k)$.
    Step 2 : Compute a stepsize $\alpha > 0$ and a parameter $\beta > 0$.
    Step 3 : Set $\theta_{k+1} = \theta_k - \alpha_k \nabla f(\theta_k) + \beta_k(\theta_k - \theta_{k-1})$.
end
```

In this algorithm, we shall play with two sequences of parameters. I ran the algorithm for :

$$\begin{cases} \alpha_k = \frac{4}{(\sqrt{L}+\sqrt{\mu})^2}, & \forall k \in \mathbb{N} \\ \beta_k = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}, & \forall k \in \mathbb{N} \end{cases}$$

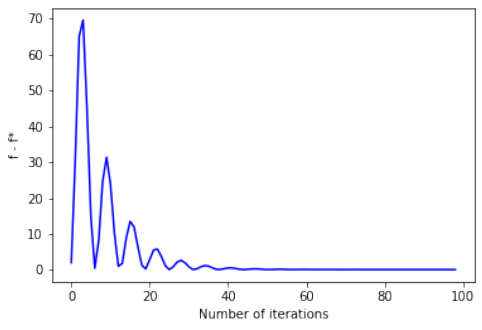


FIGURE 6 – Gradient descent with Heavy-Ball method.

The algorithm converge in 40 iterations with an MSE of 15%. It is twice more than gradient descent with constant step size and backtracking line search. The theoreticals results aren't confirmed. The Heavy-Ball algorithm should converge faster than a simple gradient descent with constant step size. Indeed we confront a linear convergence to an exponential one.

We observe a non monotone sequence of objective values $(f(\theta_k))_k$, which is in accordance with the course notes. However, note that

The Nesterov algorithm is a generalization of the Heavy-Ball algorithm to convex functions.

```
Algorithm 4: Nesterov Accelerated Gradient Method
  Initialization : $\theta_0 \in \mathbb{R}^d$, $\theta_{-1} = \theta_0$
  for $k = 0, 1, ...$ do
      Step 1 : Compute a stepsize $\alpha_k > 0$ and a parameter $\beta_k > 0$.
      Step 2 : Set $\theta_{k+1} = \theta_k - \alpha_k \nabla f(\theta_k + \beta_k(\theta_k - \theta_{k-1})) + \beta_k(\theta_k - \theta_{k-1})$.
  end
```

For this algorithm, I simply changed the sequence $\alpha_k = \frac{1}{L}$.

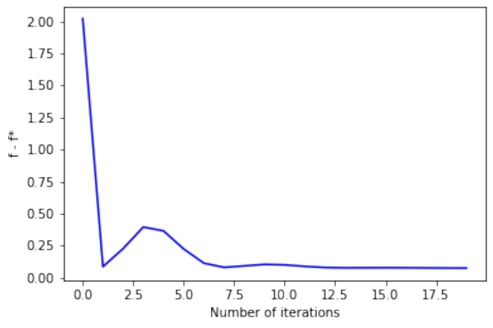


FIGURE 7 – Gradient descent with Nesterov method.

The algorithm converges after 8 iterations with an MSE of 15%. It is faster than the Heavy-Ball algorithm. This may be due to the smallness of α in the Heavy-Ball algorithm.

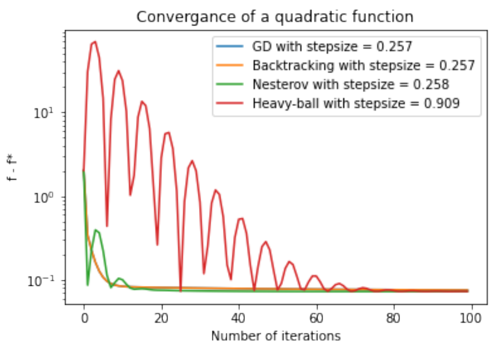


FIGURE 8 – Gradient descent for different methods.

To conclude this part, we may opt for the classical GD with constant step size. The blue and the orange curve are overlayed, meaning that the backstracking line search doesn't give better results. But we shall remain vigilent on Backstracking line search function which may be expensive in some contexts. Note that comparing the MSE's isn't relevant as they all are equal.

4 Stochastic Gradient Descents

In this section, we will derive different Stochastic Gradient (SG). The difference between gradient descent (GD) and stochastic gradient lies in the calculation of the gradient at each iteration. For gradient descent, we compute a full gradient at each iteration. For stochastic gradient, we pick at each iteration one index at random and compute partial gradient in that direction. Thus, the SG is cheaper in terms of epoch.

The SG use the structure of the objective function, being a finite sum. We can write :

$$\begin{aligned} f(\theta) &= \frac{1}{2n}\|X\theta - y\|^2 \\ &= \frac{1}{n}\sum_{i=1}^{n} f_i(\theta). \end{aligned}$$

With $f_i(\theta) = \frac{1}{2}\left(x_i^T\theta - y_i\right)^2$.

4.1 Stochastic gradient descent vs Gradient Descent

We can write the SGD as below :

Algorithm 5: Stochastic Gradient Descent

```
Initialization : $\theta_0 \in \mathbb{R}^d$
for $k = 0, 1, ...$ do
    Step 1 : Compute a stepsize $\alpha_k > 0$
    Step 2 : Draw a random index $i_k \in \{1, ..., n\}$
    Step 3 : Set $\theta_{k+1} = \theta_k - \alpha_k \nabla f_{i_k}(\theta_k)$
end
```

This algorithm is not a descent, meaning it doesn't necessarily lead to convergence. The sequence $(\theta_k)_k$ is thus random and can oscillate.

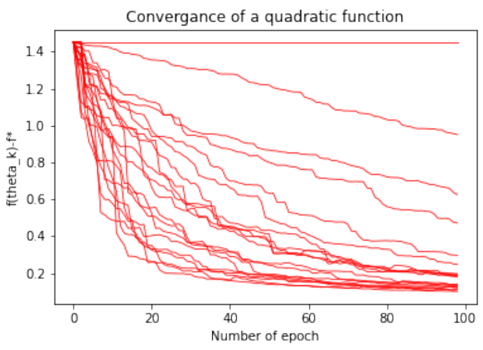


FIGURE 9 – Display of a large number of trajectories $k \mapsto \theta_k \in \mathbb{R}$ generated by several runs of SG with step size between 0 and μ excluded.

This first experiment reveals the random side of the stochastic gradient. It shows that the descent property isn't verified. However, the lecture notes provide convergence in law, which is sufficient in this project to approach a good solution. We don't have a lot of theoretical on the step size as in this part the batch size plays a major role.

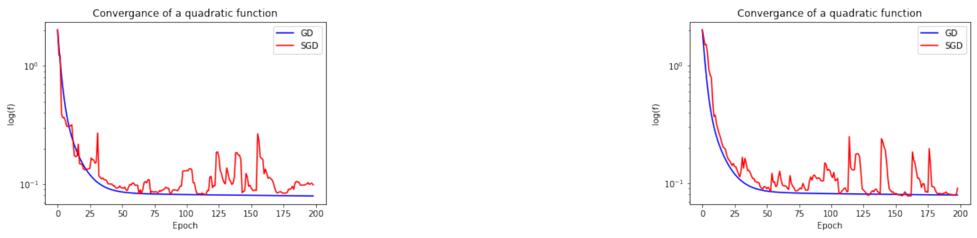


FIGURE 10 – Gradient descent vs Stochastic gradient.

In this study, the GD is a better choice as the algorithms are fast. However, a graph of the evolution of the objective value is required as the convergence isn't guaranteed. Indeed, we have convergence at the right and divergence at the left. Note that the MSE is a bit less than for previous algorithm, around 17%.

In the next algorithm, we aim to reduce the oscillation, i.e the randomness that is a little bit excessive (but cheaper in cost). That is why we introduce now the Batch Gradient.

4.2 Batch gradient descent

In this part, we introduce the batch size. The Batch gradient is a trade off between GD and SG. An iteration of a given batch method with batch index set Sk consists in $|S_k|$ calls to gradients ∇f_i, or, equivalently, $|S_k|$ accesses to data points : The iteration is derived below :

$$\theta_{k+1} = \theta_k - \alpha_k \frac{1}{|S_k|} \sum_{i \in S_k} \nabla f_i(\theta_k).$$

We don't focus anymore on the step size so we fix it to $\frac{2}{L} - 0.1$. I volontary choosed a "bad" stepsize to highlight the effects of the Batch size. Otherwise, I would have choosen the optimal one.

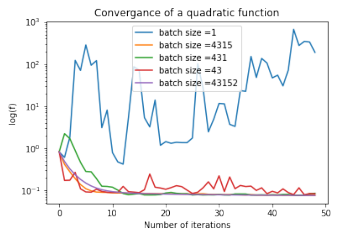


FIGURE 11 – Batch gradient for different batch size and a step size $\alpha \approx \frac{2}{L}$.

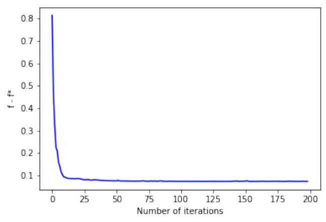


FIGURE 12 – Batch gradient for $S_k = 433$.

The choice of the batch size is then a trade off between GD and SG. Here we can observe that the optimal choice for $|S_k|$ is around 4315, corresponding to $S_k = \frac{n}{10}$ (i.e the orange curve). Once I choosed the best batch size, I plotted the best one. The algorithm converges really fast and the MSE is 14 %. This graph confirms the course notes. Indeed choosing a batch size near n reduced oscillatory.

4.3 Averaging gradient

In this part we modify the stochastic gradient algorithm so that it maintains an average of all SG iterates. It reduces the oscillations and allows a quicker convergence. This algorithm aims to reduce the oscillation. An iteration is described as below :

$$\begin{cases} \theta_{k+1} = \theta_k - \alpha_k \nabla f_{i_k}(\theta_k) \\ \hat{\theta}_{k+1} = \frac{1}{k+1} \sum_{j=0}^{k} \theta_j \end{cases}$$

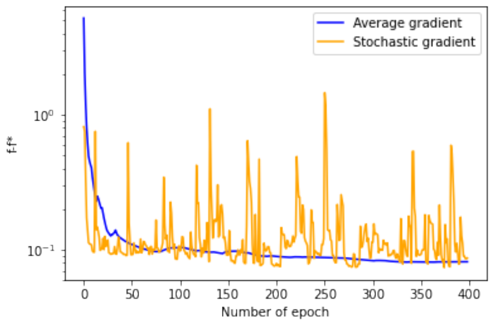


FIGURE 13 – Averaging gradient

This graphic highlights how averaging methods reduce the randomness of the sequence solution. The algorithm seems to converge fastly for an MSE of 15%.

5 Regularization

In this section, we test the effects of regularization. The general form for regularization is :

$$\min f(\theta) + \text{pen}(\lambda).$$

Pen is a function to define. In particular, Ridge and Lasso penalizations make use of the norm (l1 or l2) of the vector w : $\text{pen}(\lambda) = \lambda \|\theta\|$, for a parameter $\lambda > 0$. That way, when λ increases, the norm of the minimizer decreases. This effect is illustrated in the regularization paths plotted bellow.

5.1 Ridge regularization

Ridge regularization is obtained by introducing an l2 penalty. It is defined as :

$$\min f(\theta) + \lambda\|\theta\|_2^2.$$

The explicit solution of this problem exists and is of the form :

$$\theta^R = \left(X^T X + \lambda I\right)^{-1} X^T y.$$

The regularization introduce a bias and avoid problem when the smallest eigenvalue of the design matrix is approximatively 0. The algoritmh derive relatievly good coefficiant, with an MSE of 15 % as usual. The regularization parameter λ choosen is the one which minimizes the error. Following the results, we choose = 167.3 corresponding to an MSE equal to 0.38.

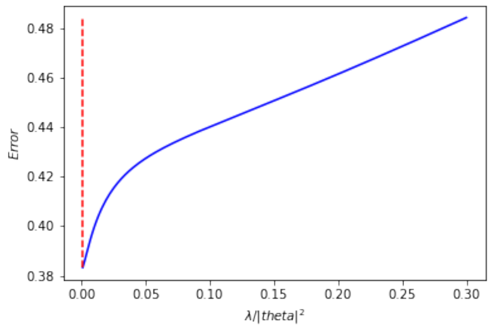


Figure 14 – Ridge regularization

It is not surprising to find an optimal λ that slightly improves the relative prediction error compared to our previous methods.

5.2 LASSO regularization

LASSO regularization is obtained by introducing an l1 penalty. It is defined as :

$$\min f(\theta) + \lambda\|\theta\|_1.$$

This class of methods, which can be viewed as an extension of the classical gradient algorithm, is attractive due to its simplicity and thus is adequate for solving large-scale problems even with dense matrix data. However, such methods are also known to converge quite slowly
The ISTA algorithm reads :

$$\begin{cases} S(\theta_k) = sign(\theta_k) \max\left(\theta_k - \lambda, 0\right) \\ \theta_{k+1} = S_{\lambda_\tau}\{\theta_k - \tau X^T\left(X\theta_k - y\right)\} \end{cases}$$

We execute the algorithm and obtain the following results.

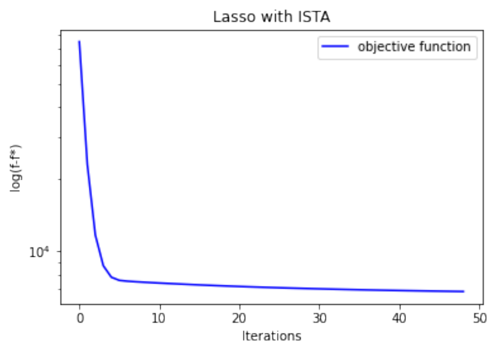


FIGURE 15 – ISTA algorithm

The algorithm provide a fast convergence, up 10 iterations and a MSE of 16%. The algorithm select the variables carat, x and z.

6 Conclusion

This project illustrated many different methods for solving a linear regression problem. We have seen that numerical methods like gradient descent and stochastic gradient can provide solutions with efficiency very close to exact solutions. Considering our linear regression problem, numerical methods don't seem necessary as we can get directly the solution through ordinary least squares (or Ridge estimator if the covariance matrix is ill conditioned). However, it gives us a good insight into methods like stochastic gradient which are very useful in other situations where we don't have an exact solution, like neural networks.

Références

Gabriel Peyré. *Course notes on Optimization for Machine Learning.* 2021.

Clément Royer. *Advanced gradient descent and acceleration.* 2021a.

Clément Royer. *Lecture notes on stochastic gradient methods.* 2021b.