

MATH3027: Optimization (UK 22/23)

Week 12: Stochastic Gradient Descent

Prof. Richard Wilkinson
School of Mathematical Sciences
University of Nottingham, United Kingdom
Please send any comments or mistakes to
r.d.wilkinson@nottingham.ac.uk

| | |
|---|---|
| The Kaczmarz Algorithm | 2 |
| Stochastic Gradient Descent | 4 |
| Revisiting the Nonlinear Regression Example | 6 |


Dive to deep learning <https://d2l.ai>

Optimization problems of the form

$$\min_x \frac{1}{n} \sum_{i=1}^n Q_i(x) \quad (1)$$

often occur in statistics and machine learning. Usually, i indexes across the different points in the dataset. For example

- *Least squares estimation* as seen in the previous week, when looked at the Gauss

Newton method.  What is $Q_i(x)$ in the nonlinear least squares problem?

- *Maximum likelihood estimation*: Suppose we're given independent identically distributed data y_1, \dots, y_n and that we want to model this with probability density function (pdf) $f(y; \theta)$ where θ is a free parameter we need to estimate. For example, we could assume a normal distribution $y_i \sim N(\mu, \sigma^2)$ and then estimate the free parameters $\theta = (\mu, \sigma^2)$. The likelihood of all n observations is

$$\prod_{i=1}^n f(y_i; \theta).$$

The maximum likelihood principle says that we should estimate θ using

$$\hat{\theta} = \operatorname{argmax}_{\theta} \prod_{i=1}^n f(y_i; \theta)$$



Or equivalently, we can take the log and maximize the log-likelihood or minimize the negative log-likelihood

$$\hat{\theta} = \operatorname{argmin}_{\theta} \sum_{i=1}^n \log f(y_i; \theta)$$

More generally, *M-estimation* is a branch of statistics that looks at estimators that arise as the minimizers of sums.

- *Empirical risk minimization*: Given pairs $\{(x_i, y_i)\}_{i=1}^n$ with $x_i \in X$ and $y_i \in Y$ we want to learn the function that maps from x (input) to y (output), i.e., learn

$$h : X \rightarrow Y.$$

We choose a loss function $L : Y \times Y \rightarrow \mathbb{R}$ to measure distance in Y and then define the empirical risk to be

$$R(h) = \frac{1}{n} \sum_{i=1}^n L(h(x_i), y_i).$$

For example, if $L(h(x), y) = (h(x) - y)^2$ then $R(h)$ is the sum of squared errors. The *empirical risk minimization (ERM)* principle states that a learning algorithm should choose \hat{h} to minimize the empirical risk, i.e.,

$$\hat{h} = \operatorname{argmin}_{h \in \mathcal{H}} R(h)$$

where \mathcal{H} is the space of hypotheses we are willing to consider. For example, we might let \mathcal{H} be all linear functions of x . In which case, if L is squared error ERM is the same as linear regression.

Gradient descent has a computational cost that scales linearly with n , as at each stage we need to evaluate the gradient of the cost function, which is $\sum_{i=1}^n \nabla Q_i(x)$. Therefore, when n is large the gradient descent algorithm may become prohibitively expensive.

Stochastic gradient descent is a way of reducing this cost by not evaluating every term in the sum (1) at each step of the algorithm.

The Kaczmarz Algorithm

We begin our discussion by studying a classical algorithm proposed by the Polish mathematician Stefan Kaczmarz in 1937, and which was later re-discovered in the 1970s in image processing. This technique was implemented in the very first medical scanners. The Kaczmarz algorithm solves the linear system

$$Ax = b$$



by iterating projections along the i -th for of the matrix \mathbf{A} , denoted by \mathbf{a}_i :

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \frac{b_i - \mathbf{a}_i \mathbf{x}^k}{\|\mathbf{a}_i\|^2} \mathbf{a}_i^\top. \quad (\text{Kaczmarz})$$

Note that this algorithm does not require to compute \mathbf{A}^{-1} ! In the original Kaczmarz algorithm, the i -th row that is chosen at the k -th iteration of the algorithm is cycled periodically through all the rows of the matrix \mathbf{A} , i.e.

$$i = \text{mod}(k, m) + 1,$$

where m is the number of rows of \mathbf{A} . Provided that the system is consistent, that is, there exists at least one solution of the system, the iteration \mathbf{x}^k converges to the minimum norm solution of the problem, assuming that $\mathbf{x}^0 = \mathbf{0}$. The convergence analysis of this algorithm remained an open problem until probabilistic methods were introduced by 2009. Nowadays, we can show that the Kaczmarz algorithm converges exponential (and independently of the number of rows) if at the k -th iteration, the i -th row is chosen randomly. This algorithm is known as *Randomized Kaczmarz Algorithm*. We can sample uniformly among the rows, or with a probability that is proportional to the squared row norm $\|\mathbf{a}_i\|^2$, known as *importance sampling*. Figure 1 shows a comparison between cycling, uniform, and importance sampling among the rows of a 20 by 20 linear system. In each case, the method converges to the solution of $\mathbf{Ax} = \mathbf{b}$, however the method is very slow. For large-scale systems, avoiding the inversion of \mathbf{A} is desirable, so this method can be applied in this case. We recall that solving the linear system $\mathbf{Ax} = \mathbf{b}$ can be cast as the

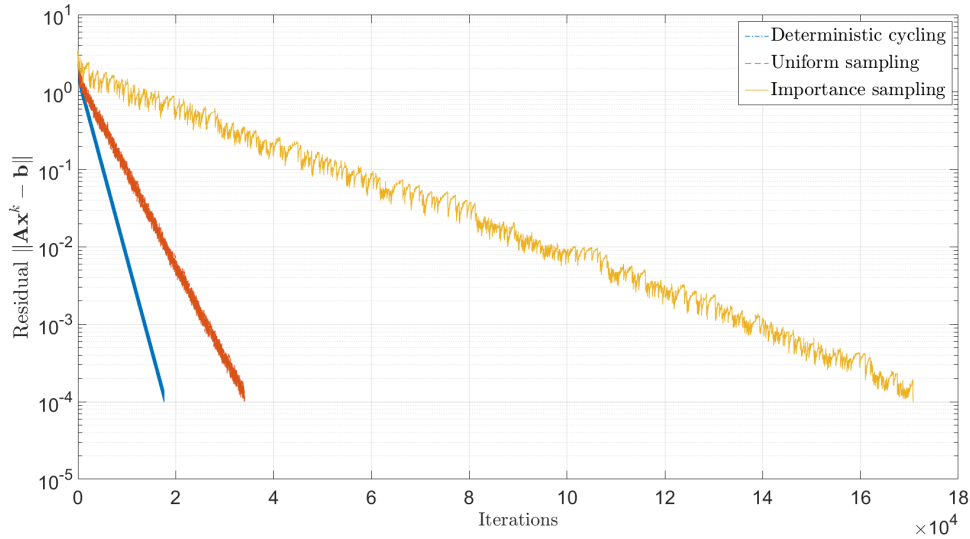


Figure 1: Iteration of the Kaczmarz algorithm for a 20 by 20 linear system of equations, under three different sampling procedures for the rows of \mathbf{A} .

optimization problem

$$\min_{\mathbf{x}} \frac{1}{2m} \|\mathbf{Ax} - \mathbf{b}\|_2^2 = \frac{1}{2m} \sum_{i=1}^m (\mathbf{a}_i \mathbf{x} - b_i)^2,$$



for which a gradient descent method can be constructed as

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \frac{t}{m} \mathbf{A}^\top (\mathbf{A} \mathbf{x} - \mathbf{b}),$$

and comparing against (Kaczmarz) we can interpret it as a gradient descent where at each iteration, instead of computing the full gradient of

$$\frac{1}{2m} \sum_{i=1}^m (\mathbf{a}_i \mathbf{x} - b_i)^2,$$

we sample a single element of the cost and work with the gradient of

$$\frac{1}{2m} (\mathbf{a}_i \mathbf{x} - b_i)^2.$$

Note that in the case of a uniform sampling among the rows, we can write

$$\frac{1}{2m} \sum_{i=1}^m (\mathbf{a}_i \mathbf{x} - b_i)^2 = \frac{1}{2} \mathbb{E}_i [\mathbf{a}_i \mathbf{x} - b_i],$$

where the expected value is among a uniform distribution when choosing the vector $[\mathbf{a}_i | b_i]$ uniformly from the rows of the augmented matrix $[\mathbf{A} | \mathbf{b}]$. The generalization of this idea for nonlinear regression problems is what we will discuss in the next section as stochastic gradient descent.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a fundamental algorithm in machine learning. It is naturally meant for solving nonlinear regression/estimation problems where the cost is of the type

$$\min_{\mathbf{x}} \frac{1}{m} \sum_{i=1}^m Q_i(\mathbf{x}). \quad (2)$$

For example, in week 6 we studied the following nonlinear regression problem.

Week 6 revisited. Consider the nonlinear model in $\theta \in \mathbb{R}$

$$f(\theta; \mathbf{x}) = x_1 e^{x_2 \theta} \cos(x_3 \theta + x_4),$$

with parameters $\mathbf{x} \in \mathbb{R}^4$ for which we want to find the optimal value \mathbf{x}^* minimizing the norm of the ℓ_2 -error with respect to m observations of the *true* model

$$f_i := f(\theta_i), \quad i = 1, \dots, m.$$



We formulate this problem as a nonlinear least squares problem

$$\min_{\mathbf{x}} g(\mathbf{x}) := \frac{1}{m} \sum_{i=1}^m (f(\theta_i; \mathbf{x}) - \hat{f}_i)^2. \quad (\text{NLS})$$

Note that the $\frac{1}{m}$ scaling does not affect the minimizer. Setting

$$Q_i(\mathbf{x}) = (f(\theta_i; \mathbf{x}) - \hat{f}_i)^2$$

we can see how nonlinear regression problems lead to costs or **loss functions** similar to eq. (2). In general, while setting a gradient descent iteration for this problem

$$\mathbf{x}^{k+1} = \mathbf{x}^k - t^k \nabla g(\mathbf{x}^k) = \mathbf{x}^k - \frac{t^k}{m} \sum_{i=1}^m \nabla Q_i(\mathbf{x}^k)$$

is straightforward (following exactly what we discussed in week 5 and 6), when the number of observations m is large¹, and the parameter space is high-dimensional ($\mathbf{x} \in \mathbb{R}^n$, $n \gg 1$), the computation of $\sum_{i=1}^m \nabla Q_i(\mathbf{x}^k)$ is overwhelmingly expensive. In the spirit of the Kaczmarz algorithm, stochastic gradient descent circumvents this limitation. Stochastic Gradient Descent dates back to 1951 to the paper by Robbins and Munro “A stochastic approximation method”, and instead of computing the gradient with all m points, only a single data point is randomly selected and used. At the next iteration, another randomly selected point is used to compute the gradient and update the solution

$$\mathbf{x}^{k+1} = \mathbf{x}^k - t^k \nabla Q_i(\mathbf{x}^k),$$

where the index i is sampled in each gradient iteration. In machine learning applications, the parameter t^k is known as the **learning rate**. Note that if the samples are drawn uniformly, then

$$g(\mathbf{x}) := \frac{1}{m} \sum_{i=1}^m Q_i(\mathbf{x}) = \mathbb{E}_i[Q_i(\mathbf{x})].$$

and that a random sample constitutes an unbiased estimator of $\nabla g(\mathbf{x})$. The convergence of SGD is stated in the following theorem.

Theorem (Convergence of SGD). *Assume:*

- The cost $g(\mathbf{x})$ is such that

$$\|\nabla g(\mathbf{x}) - \nabla g(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|, \quad \text{and} \quad \nabla^2 g(\mathbf{x}) \succeq \mu \mathbf{I}.$$

- The sample gradient $\nabla Q_i(\mathbf{x}^k)$ is an unbiased estimate of $\nabla g(\mathbf{x}^k)$.
- For all \mathbf{x} ,

$$\mathbb{E}_i[\|Q_i(\mathbf{x})\|^2] \leq \sigma^2 + c\|\nabla g(\mathbf{x})\|^2.$$

¹ as in a *big data* framework -think about image datasets, Spotify songs-



Then, if $t^k \equiv t \leq \frac{1}{Lc}$, then SGD achieves

$$\mathbb{E}[g(\mathbf{x}^k) - g(\mathbf{x}^*)] \leq \frac{tL\sigma^2}{2\mu} + (1 - t\mu)^k (g(\mathbf{x}^0) - g(\mathbf{x}^*)).$$

The result above implies:

1. Fast (linear) convergence during the first iterations.
2. Convergence to a neighbourhood of \mathbf{x}^* , without further progress.
3. If gradient computation is noiseless, that is $\sigma = 0$, then linear convergence to optimal points.
4. A smaller stepsize t yield better converging points.

The algorithm may require multiple passes through all the data to converge, but each step is now easy to evaluate versus the full computation of the gradient. If instead of a single point, a different subset of points is sampled at each iteration, then we have a **batch gradient descent** algorithm:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - t^k \nabla g(\mathbf{x}^k) = \mathbf{x}^k - \frac{t^k}{|K|} \sum_{i \in K} \nabla Q_i(\mathbf{x}^k),$$

where K denotes a set of p randomly selected datapoints.

Revisiting the Nonlinear Regression Example

If we go back to the model

$$f(\theta; \mathbf{x}) = x_1 e^{x_2 \theta} \cos(x_3 \theta + x_4),$$

during week 6 we explored the Gauss-Newton method for nonlinear regression. We shall modify this setting and implement SGD. Results shown in Figure 2 show the different behaviours observed for different learning rates. Notice the large number of iterations required.

Model:

```
1 function [val]=model(t,x)
2 val= x(1).*exp(x(2).*t).*cos(x(3).*t+x(4)); %%model
```

Gradient of the model with respect to \mathbf{x} :



```

1 function [val]=modelgrad(t,x)
2 val(1,1)=exp(x(2).*t).*cos(x(3).*t+x(4));
3 val(2,1)=x(1).*exp(x(2).*t).*t.*cos(x(3).*t+x(4));
4 val(3,1)=-x(1).*exp(x(2).*t).*sin(x(3).*t+x(4)).*t;
5 val(4,1)=-x(1).*exp(x(2).*t).*sin(x(3).*t+x(4));

```

Cost function:

```

1 function [val]=g1(x,tm,fn)
2 m=length(tm);
3 val=norm(model(tm,x)-fn)^2/m;

```

Gradient of the cost function:

```

1 function[val]=gradg1(x,tm,fn)
2 m=length(tm);
3 ind=randi([1 m]); %% sampling among the dataset
4 val=2*(model(tm(ind),x)-fn(ind))*modelgrad(tm(ind),x);

```

Main:

```

1 clear all
2
3 xt=[1;2;pi;0]; %% true parameters
4 tm=[-1:0.001:1]'; %% measurements of the independent variable
5 randn('seed',666);
6 ft=model(tm,xt); %% true model measurements
7 m=length(tm);
8 maxiter=10^4;
9 x0=[1;1;1;1];
10 xm=x0;
11 t=0.01; %% learning rate
12 hist=[g1(xm,tm,fn)];
13 for i=1:maxiter
14 xp=xm-t*gradg1(xm,tm,ft); %%the random sampling is inside gradg1
15 hist=[hist g1(xp,tm,ft)];
16 xm=xp;
17 end

```



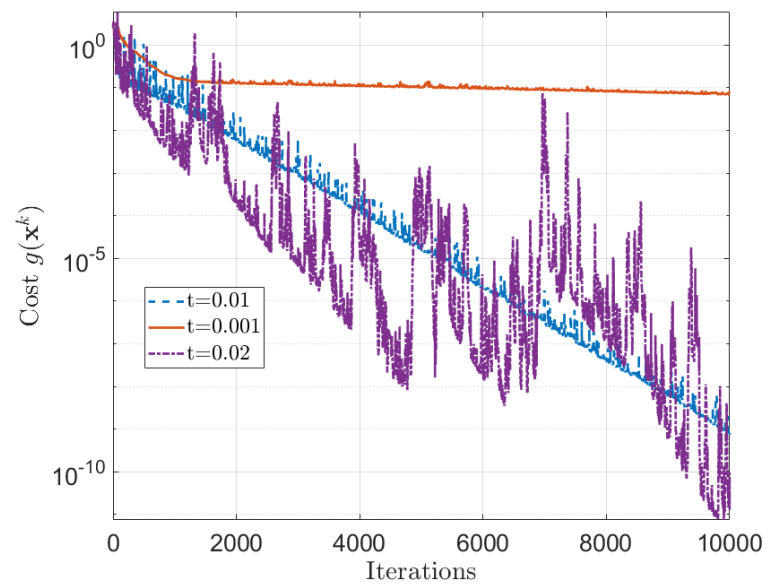


Figure 2: Convergence of the SGD iteration for different learning rates.

