

Gradient Descent

1

¹based on slides by Eleni Triantafillou, Ladislav Rampasek, Jake Snell, Kevin Swersky, Shenlong Wang and other

Contents

- ▶ Overview
- ▶ Gradient Descent

Overview of Optimization

An informal definition of optimization

Minimize (or maximize) some quantity.

Applications

- ▶ Engineering: Minimize fuel consumption of an automobile
- ▶ Economics: Maximize returns on an investment
- ▶ Supply Chain Logistics: Minimize time taken to fulfill an order
- ▶ Life: Maximize happiness

More formally

Goal: find $\theta^* = \operatorname{argmin}_{\theta} f(\theta)$, (possibly subject to constraints on θ).

- ▶ $\theta \in \mathbb{R}^n$: *optimization variable*
- ▶ $f: \mathbb{R}^n \rightarrow \mathbb{R}$: *objective function*

Maximizing $f(\theta)$ is equivalent to minimizing $-f(\theta)$, so we can treat everything as a minimization problem.

Optimization is a large area of research

The best method for solving the optimization problem depends on which assumptions we want to make:

- ▶ Is θ discrete or continuous?
- ▶ What form do constraints on θ take? (if any)
- ▶ Is f “well-behaved”? (linear, differentiable, convex, submodular, etc.)

Optimization for Machine Learning

Often in machine learning we are interested in learning the parameters θ of a model.

Goal: minimize some loss function

- ▶ For example, if we have some data (x, y) , we may want to maximize $P(y|x, \theta)$.
- ▶ Equivalently, we can minimize $-\log P(y|x, \theta)$.
- ▶ We can also minimize other sorts of loss functions

log can help for numerical reasons

Gradient Descent

Gradient Descent: Motivation

From calculus, we know that the minimum of f must lie at a point where $\frac{\partial f(\theta^*)}{\partial \theta} = 0$.

- ▶ Sometimes, we can solve this equation analytically for θ .
- ▶ Most of the time, we are not so lucky and must resort to iterative methods.

Review

- ▶ Gradient: $\nabla_{\theta} f = (\frac{\partial f}{\partial \theta_1}, \frac{\partial f}{\partial \theta_2}, \dots, \frac{\partial f}{\partial \theta_k})$

Outline of Gradient Descent Algorithm

Where η is the learning rate and T is the number of iterations:

- ▶ Initialize θ_0 randomly
- ▶ for $t = 1 : T$:
 - ▶ $\delta_t \leftarrow -\eta \nabla_{\theta_{t-1}} f$
 - ▶ $\theta_t \leftarrow \theta_{t-1} + \delta_t$

The learning rate shouldn't be too big (objective function will blow up) or too small (will take a long time to converge)

Gradient Descent with Line-Search

Where η is the learning rate and T is the number of iterations:

- ▶ Initialize θ_0 randomly
- ▶ for $t = 1 : T$:
 - ▶ Finding a step size η_t such that $f(\theta_t - \eta_t \nabla_{\theta_{t-1}}) < f(\theta_t)$
 - ▶ $\delta_t \leftarrow -\eta_t \nabla_{\theta_{t-1}} f$
 - ▶ $\theta_t \leftarrow \theta_{t-1} + \delta_t$

Require a line-search step in each iteration.

Gradient Descent with Momentum

We can introduce a momentum coefficient $\alpha \in [0, 1)$ so that the updates have “memory”:

- ▶ Initialize θ_0 randomly
- ▶ Initialize δ_0 to the zero vector
- ▶ for $t = 1 : T$:
 - ▶ $\delta_t \leftarrow -\eta \nabla_{\theta_{t-1}} f + \alpha \delta_{t-1}$
 - ▶ $\theta_t \leftarrow \theta_{t-1} + \delta_t$

Momentum is a nice trick that can help speed up convergence. Generally we choose α between 0.8 and 0.95, but this is problem dependent

Outline of Gradient Descent Algorithm

Where η is the learning rate and T is the number of iterations:

- ▶ Initialize θ_0 randomly
- ▶ Do:
 - ▶ $\delta_t \leftarrow -\eta \nabla_{\theta_{t-1}} f$
 - ▶ $\theta_t \leftarrow \theta_{t-1} + \delta_t$
- ▶ Until convergence

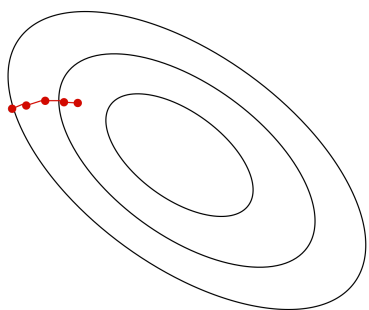
Setting a convergence criteria.

Some convergence criteria

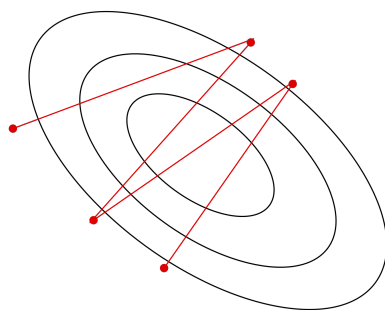
- ▶ Change in objective function value is close to zero:
 $|f(\theta_{t+1}) - f(\theta_t)| < \epsilon$
- ▶ Gradient norm is close to zero: $\|\nabla_{\theta} f\| < \epsilon$
- ▶ Validation error starts to increase (this is called *early stopping*)

Learning Rate (Step Size)

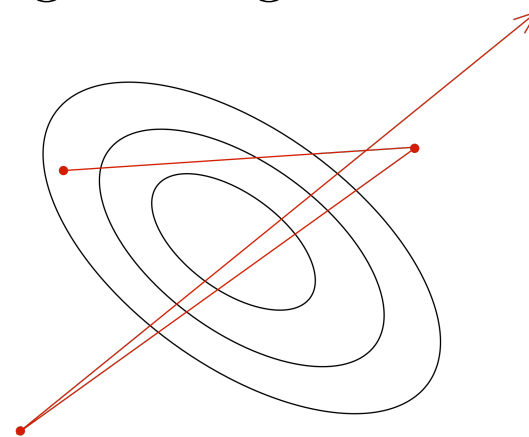
- In gradient descent, the learning rate α is a hyperparameter we need to tune. Here are some things that can go wrong:



α too small:
slow progress



α too large:
oscillations



α much too large:
instability

- Good values are typically between 0.001 and 0.1. You should do a grid search if you want good performance (i.e. try 0.1, 0.03, 0.01, ...).

Checkgrad

- ▶ When implementing the gradient computation for machine learning models, it's often difficult to know if our implementation of f and ∇f is correct.
- ▶ We can use finite-differences approximation to the gradient to help:

$$\frac{\partial f}{\partial \theta_i} \approx \frac{f((\theta_1, \dots, \theta_i + \epsilon, \dots, \theta_n)) - f((\theta_1, \dots, \theta_i - \epsilon, \dots, \theta_n))}{2\epsilon}$$

Why don't we always just use the finite differences approximation?

- ▶ slow: we need to recompute f twice for each parameter in our model.
- ▶ numerical issues

Stochastic Gradient Descent

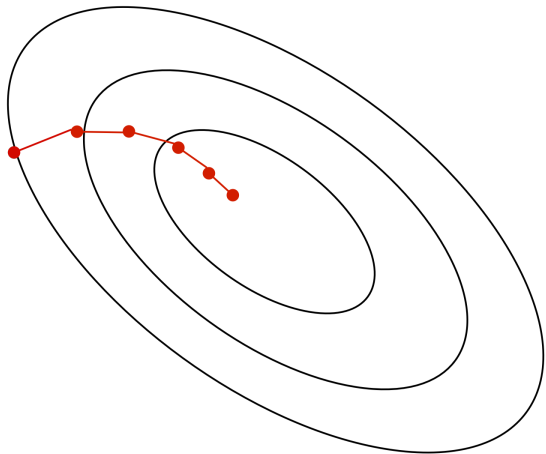
- ▶ Any iteration of a gradient descent (or quasi-Newton) method requires that we sum over the entire dataset to compute the gradient.
- ▶ SGD idea: at each iteration, sub-sample a small amount of data (even just 1 point can work) and use that to estimate the gradient.
- ▶ Each update is noisy, but very fast!
- ▶ It can be shown that this method produces an unbiased estimator of the true gradient.
- ▶ This is the basis of optimizing ML algorithms with huge datasets (e.g., recent deep learning).
- ▶ Computing gradients using the full dataset is called batch learning, using subsets of data is called mini-batch learning.

Stochastic Gradient Descent

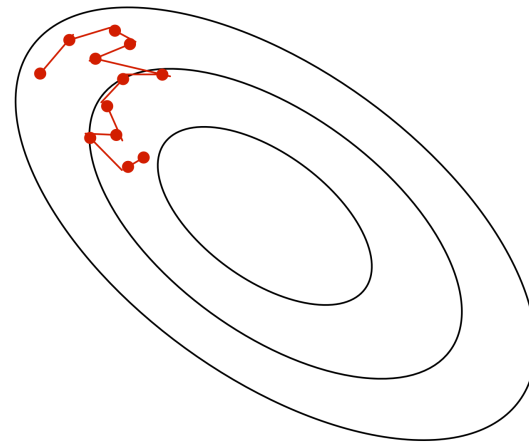
- ▶ The reason SGD works is because similar data yields similar gradients, so if there is enough redundancy in the data, the noise from subsampling won't be so bad.
- ▶ SGD is very easy to implement compared to other methods, but the step sizes need to be tuned to different problems, whereas batch learning typically “just works”.
- ▶ Tip 1: divide the log-likelihood estimate by the size of your mini-batches. This makes the learning rate invariant to mini-batch size.
- ▶ Tip 2: subsample without replacement so that you visit each point on each pass through the dataset (this is known as an epoch).

Stochastic Gradient Descent

- Batch gradient descent moves directly downhill. SGD takes steps in a noisy direction, but moves downhill on average.



batch gradient descent

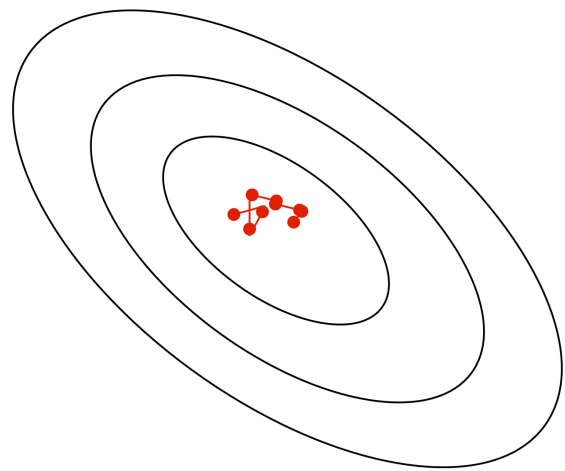


stochastic gradient descent

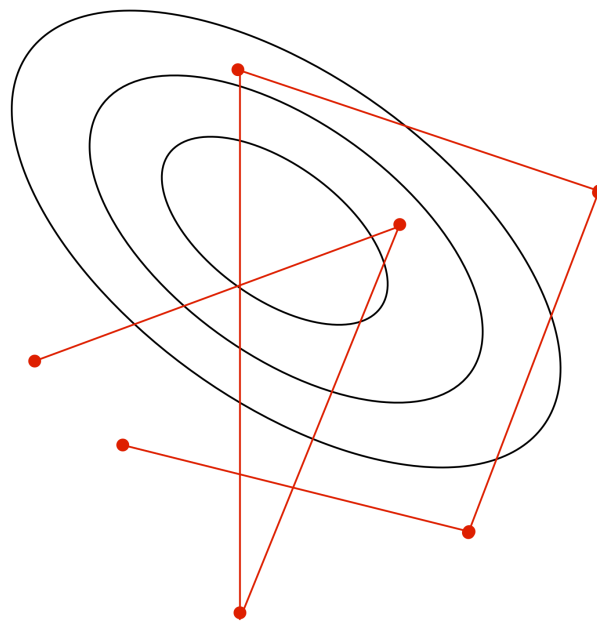
SGD Learning Rate

- In stochastic training, the learning rate also influences the **fluctuations** due to the stochasticity of the gradients.

small learning rate



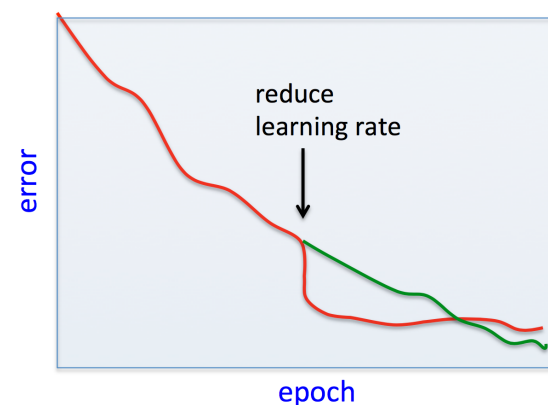
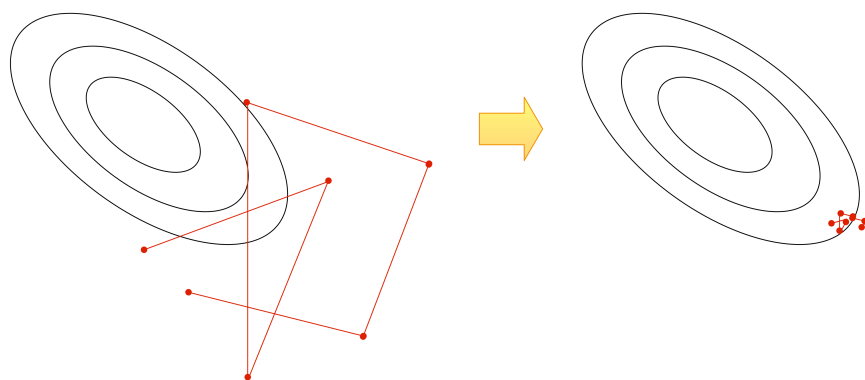
large learning rate



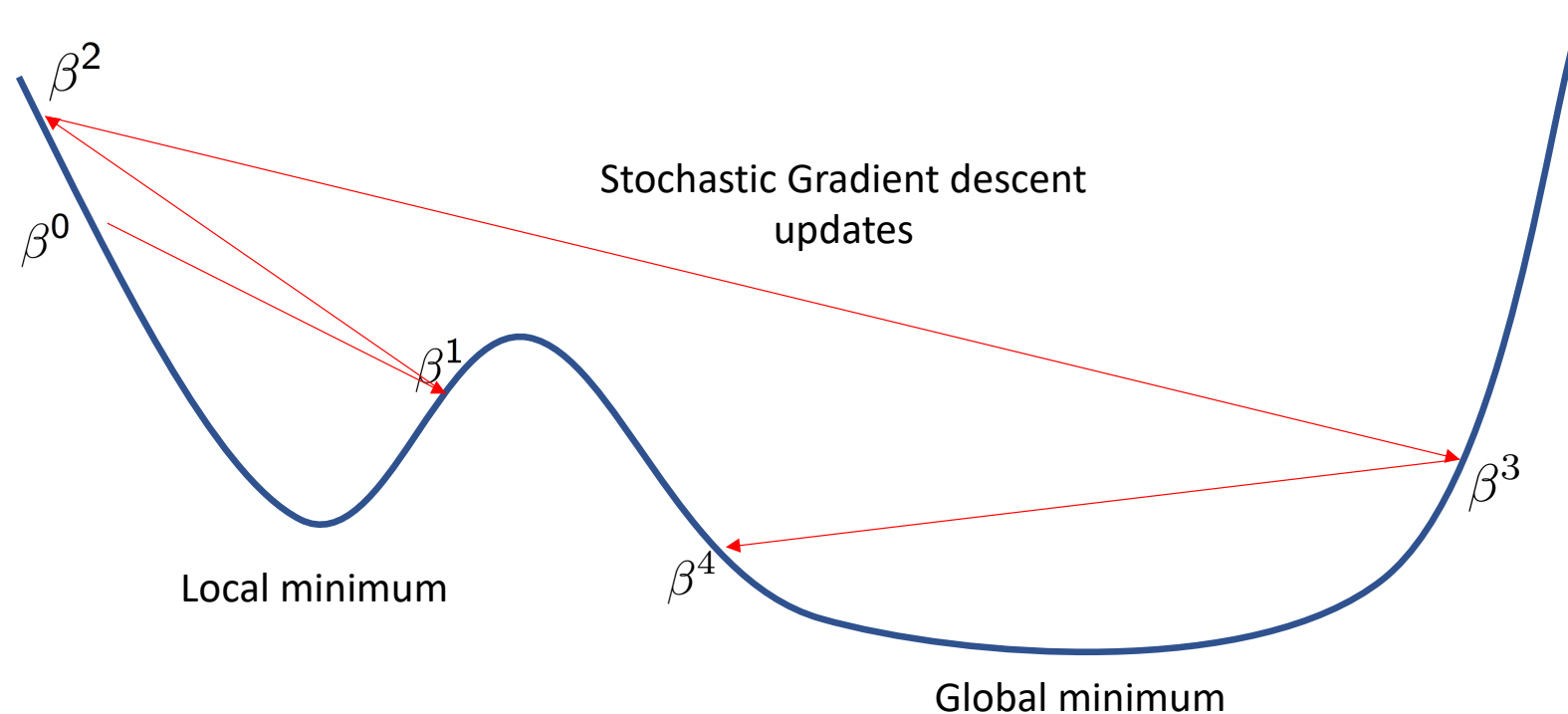
- Typical strategy:
 - ▶ Use a large learning rate early in training so you can get close to the optimum
 - ▶ Gradually decay the learning rate to reduce the fluctuations

SGD Learning Rate

- Warning: by reducing the learning rate, you reduce the fluctuations, which can appear to make the loss drop suddenly. But this can come at the expense of long-run performance.



SGD and Non-convex optimization



- Stochastic methods have a chance of escaping from bad minima.
- Gradient descent with small step-size converges to first minimum it finds.