Machine Learning

Lecture 10: Unconstrained Optimization

Prof. Dr. Aleksandar Bojchevski

28.05.24

Outline

Most machine learning tasks are optimization problems

Examples we've already seen:

- Linear Regression $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{1}{2}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$
- Logistic Regression $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} -\ln p(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X})$

Examples that we'll see in upcoming lectures:

- Support vector machines: find the maximum margin separating hyperplane
- k-means: find clusters and centroids that minimize the squared distances
- Matrix factorization: find matrices that minimize the reconstruction error
- Neural networks: find weights such that the loss is minimized
- ... and many more

General task

Let $\boldsymbol{\theta}$ denote the variables/parameters of our problem we want to learn, e.g., $\boldsymbol{\theta} = \boldsymbol{w}$ in logistic regression.

Let $\mathcal{X}$ denote the domain of $\boldsymbol{\theta}$ – the set of valid instantiations, e.g., $\mathcal{X} = \mathcal{R}^D$, potentially including additional constraints on the parameters.

Let $\mathcal{L}(\boldsymbol{\theta})$ denote the objective function (the loss), e.g., the negative log-likelihood.

Goal: Find a solution $\boldsymbol{\theta}^*$ minimizing the loss $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \mathcal{X}} \mathcal{L}(\boldsymbol{\theta})$

- find a (global) minimum of the function $\mathcal{L}$

Local minima and flat minima

We say $\boldsymbol{\theta}^*$ is a local minimum if

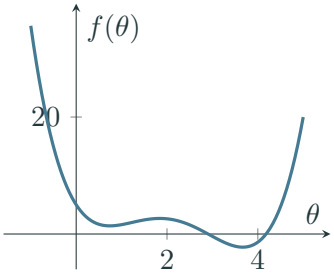
$$\exists \delta > 0, \forall \boldsymbol{\theta} \in \Theta, \text{ such that } \|\boldsymbol{\theta}^* - \boldsymbol{\theta}\| \leq \delta, \mathcal{L}(\boldsymbol{\theta}^*) \leq \mathcal{L}(\boldsymbol{\theta})$$

A local minimum could be surrounded by other local minima with the same objective value, this is known as a **flat** local minimum.[1]

[1] This is still an active area of research, but there is some evidence that flat local minima generalize better.

One dimensional example

Find minimum of $f(\theta) = 0.6 \cdot \theta^4 - 5 \cdot \theta^3 + 13 \cdot \theta^2 - 12 \cdot \theta + 5$.



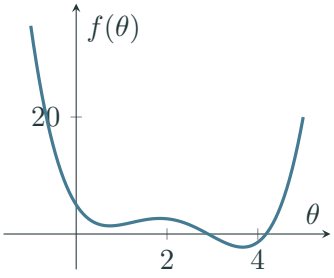
Unconstrained optimization + differentiable function.

Necessary condition for minima: gradient $\nabla_\theta f = 0$. **Sufficient?**

Note: we use interchangably $\mathcal{L}(\boldsymbol{\theta})$ or $f(\boldsymbol{\theta})$ for the function that we optimize.

One dimensional example

Find minimum of $f(\theta) = 0.6 \cdot \theta^4 - 5 \cdot \theta^3 + 13 \cdot \theta^2 - 12 \cdot \theta + 5$.



Unconstrained optimization + differentiable function.

Necessary condition for minima: gradient $\nabla_\theta f = 0$. **Sufficient?**

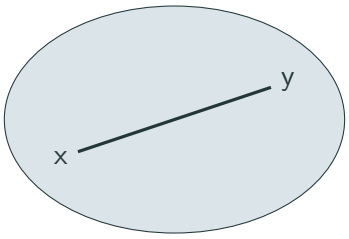
General challenge: **Multiple local minima possible**.

Note: we use interchangably $\mathcal{L}(\boldsymbol{\theta})$ or $f(\boldsymbol{\theta})$ for the function that we optimize.

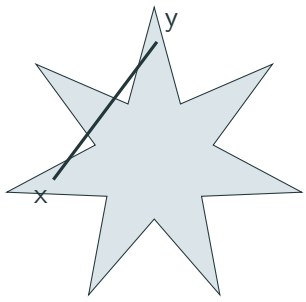
Outline

Convexity for sets

Definition: $\mathcal{X}$ is a convex set iff
for all $x, y \in \mathcal{X}$ it follows that $\lambda x + (1 - \lambda)y \in \mathcal{X}$ for all $\lambda \in [0, 1]$.



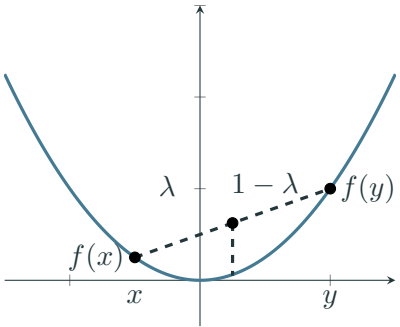
Convex set



Non-convex set

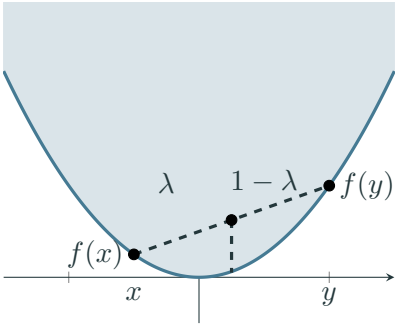
Convexity for functions

Definition: $f(\boldsymbol{x})$ is a convex function on the convex set $\mathcal{X}$ iff
for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X} : f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y})$ for $\lambda \in [0, 1]$.



Convexity for functions

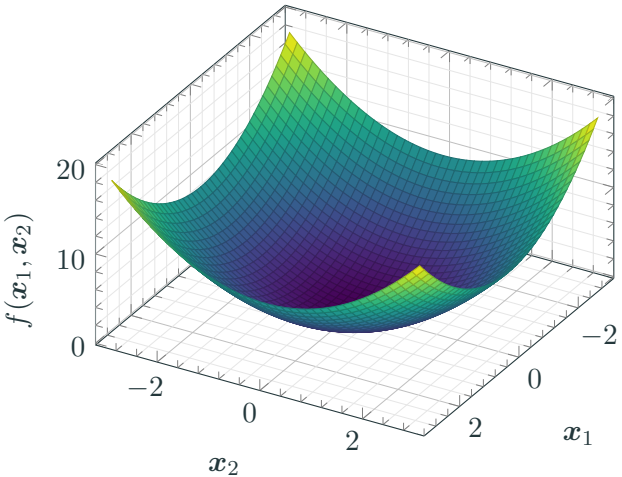
Definition: $f(\boldsymbol{x})$ is a convex function on the convex set $\mathcal{X}$ iff
for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X} : f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y})$ for $\lambda \in [0, 1]$.



Equivalently, a function is convex if its epigraph – the set of points on or above the graph of the function – is a convex set.

Convexity for functions

Definition: $f(\boldsymbol{x})$ is a convex function on the convex set $\mathcal{X}$ iff
for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X} : f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y})$ for $\lambda \in [0, 1]$.



Convexity and minimization problems

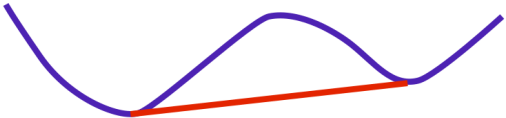
Convex functions have no local minima, which are not global.

Proof by contradiction - linear interpolation breaks local minimum condition:

Convexity and minimization problems

Convex functions have no local minima, which are not global.

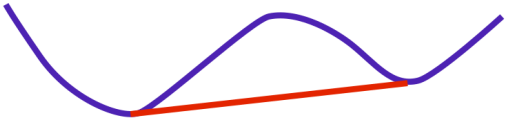
Proof by contradiction - linear interpolation breaks local minimum condition:



Convexity and minimization problems

Convex functions have no local minima, which are not global.

Proof by contradiction - linear interpolation breaks local minimum condition:



Each local minimum is a global minimum:

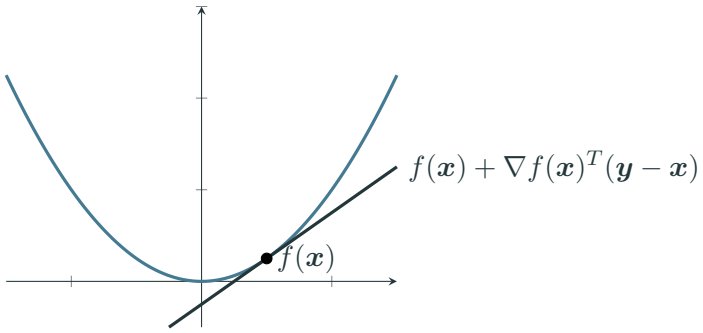
- Zero gradient implies (local) minimum for convex functions
- If f_0 is a convex function and $\nabla f_0(\boldsymbol{\theta}^*) = 0$ then $\boldsymbol{\theta}^*$ is a global minimum

Minimization becomes "relatively easy".

First-order convexity conditions

Theorem: Suppose $f : \mathcal{X} \to \mathbb{R}$ is a differentiable function and $\mathcal{X}$ is convex. Then f is convex iff for $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x})$$



Second-order convexity conditions

If f is twice differentiable, then f is convex iff its domain is convex and its Hessian is positive semidefinite, i.e., for all $\boldsymbol{x}$: $\nabla^2 f(\boldsymbol{x}) \succcurlyeq 0$.

Geometrically, $\nabla^2 f(\boldsymbol{x}) \succcurlyeq 0$ means that the graph of the function has positive (upward) curvature at $\boldsymbol{x}$.

This also implies that all eigenvalues of the Hessian are non-negative.

For a 1D function on $\mathbb{R}$, this reduces to the simple condition $f''(x) \geq 0$.

How to verify whether a function is convex?

1. Prove whether the definition of convexity holds.

2. Exploit special results:

   First-order convexity

   Second-order convexity

3. Show that the function can be obtained from simple convex functions by operations that preserve convexity:

   Start with simple convex functions, e.g. $f(x) = \text{const}, f(\boldsymbol{x}) = \boldsymbol{x}^T \cdot \boldsymbol{b}, f(x) = e^x$.

   Apply "construction rules" (next slide).

Convexity preserving operations

Let $f_1 : \mathbb{R}^d \to \mathbb{R}$ and $f_2 : \mathbb{R}^d \to \mathbb{R}$ be convex functions, and $g : \mathbb{R}^d \to \mathbb{R}$ be a concave function, then

- $h(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$ is convex
- $h(\boldsymbol{x}) = \max\{f_1(\boldsymbol{x}), f_2(\boldsymbol{x})\}$ is convex
- $h(\boldsymbol{x}) = c \cdot f_1(\boldsymbol{x})$ is convex if $c \geq 0$
- $h(\boldsymbol{x}) = c \cdot g(\boldsymbol{x})$ is convex if $c \leq 0$
- $h(\boldsymbol{x}) = f_1(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b})$ is convex ($\boldsymbol{A}$ matrix, $\boldsymbol{b}$ vector)
- $h(\boldsymbol{x}) = m(f_1(\boldsymbol{x}))$ is convex if $m : \mathbb{R} \to \mathbb{R}$ is convex and nondecreasing

Example: $e^{x_1+2\cdot x_2} + x_1 - \log(x_2)$ is convex on, e.g., $[1, \infty) \times [1, \infty)$.

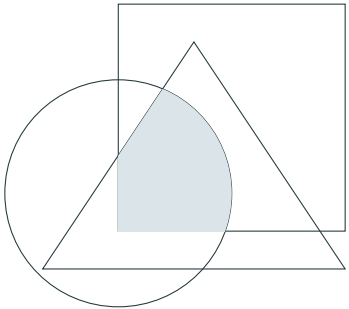
Verifying convexity of sets

1. Prove definition

   Often easier for sets than for functions

2. Apply intersection rule

   Let A and B be convex sets, then $A \cap B$ is a convex set.



Convex objective functions are easy

Objective function differentiable on its whole domain, i.e., we are able to compute gradient f' at every point.

We can solve $f'(\boldsymbol{\theta}) = 0$ for $\boldsymbol{\theta}$ analytically.

Unconstrained minimization, i.e., above computed solution for $\boldsymbol{\theta}$ is valid.

If these conditions are satisfied (e.g., OLS regression), we are done!

Unfortunately, many problems are harder

No analytical solution for $f'(\boldsymbol{\theta}) = 0$, e.g., logistic regression.

- Solution: numerical approaches, e.g., gradient descent.

Constraints on $\boldsymbol{\theta}$, e.g., $f'(\boldsymbol{\theta}) = 0$ only holds for points outside the domain.

- Solution: constrained optimization.

f is not differentiable on the whole domain.

- Potential solution: subgradients. How about discrete optimization problems?

f is not convex.

- Potential solution: convex relaxations. Convex in some variables?

One-dimensional convex differentiable problems

Since the derivative is monotonic, we can do **interval bisection** to find $\nabla f(\theta) = 0$.[2]

Require: interval: $[a, b]$, precision: ϵ

Set $A = a, B = b$

repeat

if $f'(\frac{A+B}{2}) > 0$ **then**

$B = \frac{A+B}{2}$

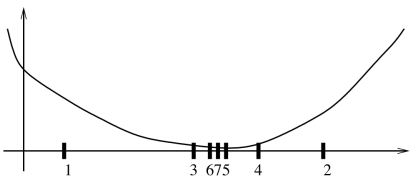
else

$A = \frac{A+B}{2}$

end if

until $(B - A)\min(|f'(A)|, |f'(B)|) \leq \epsilon$

Output: $x = \frac{A+B}{2}$



[2] It can be extended to non-differentiable f: exploit convexity in upper bound.

Outline

Descent direction

A direction $\boldsymbol{d}$ is a descent direction if we can move in direction $\boldsymbol{d}$ by $\eta > 0$ amount and be guaranteed to decrease the function value. Formally, for all $0 < \eta < \eta_{\max}$:

$$\mathcal{L}(\boldsymbol{\theta} + \eta \boldsymbol{d}) < \mathcal{L}(\boldsymbol{\theta})$$

The gradient $\nabla \mathcal{L}(\boldsymbol{\theta}_t)$ at a given iterate $\boldsymbol{\theta}_t$ points in the direction of maximal increase in $\mathcal{L}$. The negative gradient is a descent direction.[3]

[3] It can be shown that any direction $\boldsymbol{d}$ is also a descent direction if the angle between $\boldsymbol{d}$ and $-\nabla \mathcal{L}(\boldsymbol{\theta}_t)$ is less than 90 degrees and satisfies $-\nabla \mathcal{L}(\boldsymbol{\theta}_t)^T \boldsymbol{d} \leq 0$.

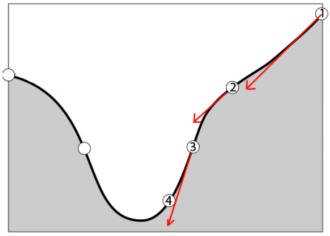
Gradient descent

Key idea: repeatedly move in the direction of the negative gradient.[4]

Descent with **line search**: get a descent direction, then unconstrained line search.
From a multi-dimensional to a one-dimensional problem that we can solve.

given a starting point $\boldsymbol{\theta} \in \mathrm{Dom}(\mathcal{L})$
repeat
1. Direction: $\Delta\boldsymbol{\theta} := -\nabla\mathcal{L}(\boldsymbol{\theta})$
2. Line search: $t^* = \arg\min_{t>0} \mathcal{L}(\boldsymbol{\theta} + t \cdot \Delta\boldsymbol{\theta})$
3. Update: $\boldsymbol{\theta} := \boldsymbol{\theta} + t^*\Delta\boldsymbol{\theta}$

until stopping criterion is satisfied.



[4] Locally, the gradient is a good (first oder Taylor) approximation of the objective function.

Convergence of gradient descent

Let p^* be the optimal value, $\boldsymbol{\theta}^*$ be the minimizer - the point where the minimum is obtained, and $\boldsymbol{\theta}^{(0)}$ be the starting point.

For strongly[5] convex f, the residual error ρ for the t-th iteration is:

$$\rho = f(\boldsymbol{\theta}^{(t)}) - p^* \leq c^t (f(\boldsymbol{\theta}^{(0)}) - p^*), \quad c < 1$$

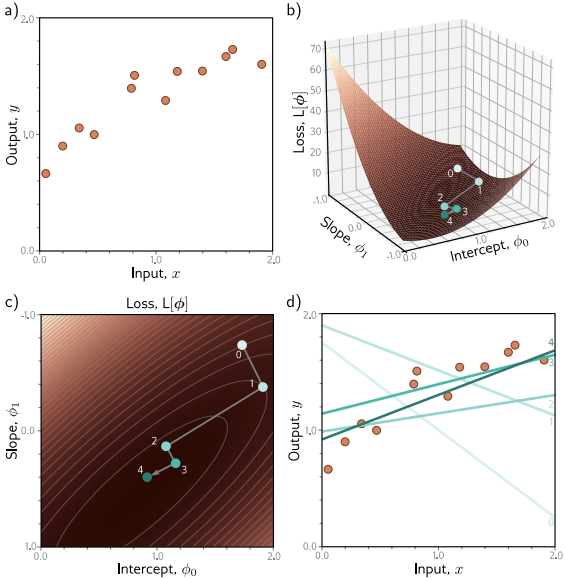
Thus $f(\boldsymbol{\theta}^{(t)})$ converges to p^* as $t \to \infty$.

We must have $f(\boldsymbol{\theta}^{(t)}) - p^* \leq \epsilon$ after at most $\frac{\log((f(\boldsymbol{\theta}^{(0)}) - p^*)/\epsilon)}{\log(1/c)}$ iterations.

Linear convergence for strongly convex objective $t \sim \log(\rho^{-1})$, i.e., linear when plotting on a log scale (old statistics terminology).

[5] A function $f(\boldsymbol{x})$ is strongly convex with parameter $m > 0$ if $\nabla^2 f(\boldsymbol{x}) \succcurlyeq m\boldsymbol{I}$.

Linear regression example



Finite sum problems

Often problems are of the form: $\mathcal{L}(\boldsymbol{\theta}) = \sum_i l_i(\boldsymbol{\theta}) + g(\boldsymbol{\theta})$, i.e., sum over instances i.

For example, OLS regression with regularization:

$$l_i(\boldsymbol{w}) = (\boldsymbol{x}_i^T \boldsymbol{w} - y_i)^2 \qquad g(\boldsymbol{w}) = \lambda \cdot \|w\|_2^2$$

The total gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ can be decomposed into a sum of per-instance gradients $\nabla_{\boldsymbol{\theta}} l_i(\boldsymbol{\theta})$ based on the sum rule.

Finite sum problems

Often problems are of the form: $\mathcal{L}(\boldsymbol{\theta}) = \sum_i l_i(\boldsymbol{\theta}) + g(\boldsymbol{\theta})$, i.e., sum over instances i.

For example, OLS regression with regularization:

$$l_i(\boldsymbol{w}) = (\boldsymbol{x}_i^T \boldsymbol{w} - y_i)^2 \qquad g(\boldsymbol{w}) = \lambda \cdot \|w\|_2^2$$

The total gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ can be decomposed into a sum of per-instance gradients $\nabla_{\boldsymbol{\theta}} l_i(\boldsymbol{\theta})$ based on the sum rule.

Finite sum problems are easy to parallelize and distribute:

1. Distribute data over several machines
2. Compute partial gradients (on each machine in parallel)
3. Aggregate partial gradients to obtain the final total gradient

Scalability analysis of gradient descent with line search

+ Linear time in number of instances
+ Linear memory in problem size (i.e., number of parameters, not data)
+ Logarithmic time to get ϵ-optimal solution
+ "Perfect" scalability

– Multiple passes through a dataset for each iteration due to the line search

A faster algorithm: avoid the line search

Simply update $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \cdot \nabla \mathcal{L}(\boldsymbol{\theta}_t)$ where η is a hyper-parameter often called the learning rate.

Only a single pass through data per iteration.

Logarithmic iteration bound (as before) if the learning rate is chosen "correctly".

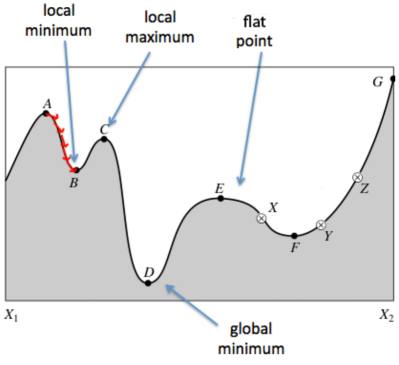
How to pick the learning rate?

- too small: slow convergence
- too high: algorithm might oscillate, no convergence

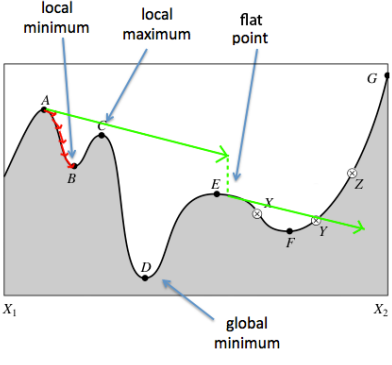
Interactive tutorial: http://www.benfrederickson.com/numerical-optimization/.

The value of the learning rate η

A too small value for η: find the minimum more slowly, end up in local minima or saddle/flat points.



A too large value for η: may never find a minimum, oscillations usually occur, only need 1 step to overshoot.



Learning rate schedule

Adapt the learning rate η_t as a (e.g., decreasing) function of the iteration number t.

First iterations cause larger changes in the parameters; later do fine-tuning.

Convergence if Robbins-Monro conditions $\eta_t \to 0$, $\frac{\sum_t \eta_t^2}{\sum_t \eta_t} \to 0$ are satisfied.

Simple: $\eta_{t+1} \leftarrow \alpha \cdot \eta_t$ for $0 < \alpha < 1$.
Piecewise constant: $\eta_t = \eta_i$ if $t_i \leq t \leq t_{i+1}$.
Exponential decay: $\eta_t = \eta_0 \exp^{-\lambda t}$
Polynomial decay: $\eta_t = \eta_0(\beta t + 1)^{-1\alpha}$

Learning rate warmup: quickly increase and then gradually decrease.
Cyclical learning rate: increase and decrease multiple times.

Other types of learning rate adaptation

As long as gradients point to the same direction, the search accelerates.

Momentum (heavy ball) incorporates the "history" of previous gradients:

$$\boldsymbol{m}_{t+1} \leftarrow \beta \cdot \boldsymbol{m}_t + \nabla \mathcal{L}(\boldsymbol{\theta}_t)$$

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \cdot \boldsymbol{m}_{t+1}$$

where $\boldsymbol{m}_t$ is the momentum and $0 < \beta < 1$, often $\beta = 0.9$.

Other types of learning rate adaptation

As long as gradients point to the same direction, the search accelerates.

Momentum (heavy ball) incorporates the "history" of previous gradients:

$$\boldsymbol{m}_{t+1} \leftarrow \beta \cdot \boldsymbol{m}_t + \nabla \mathcal{L}(\boldsymbol{\theta}_t)$$

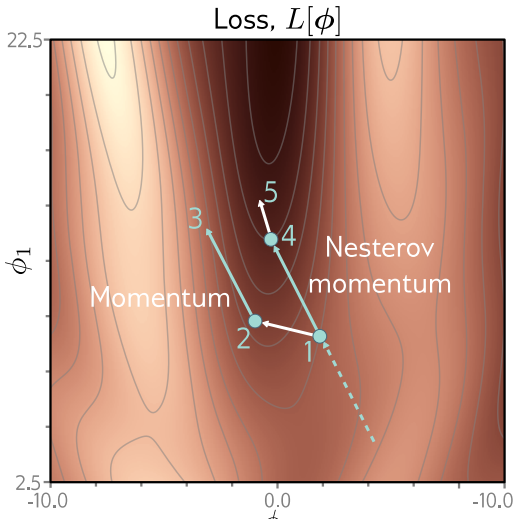
$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \cdot \boldsymbol{m}_{t+1}$$

where $\boldsymbol{m}_t$ is the momentum and $0 < \beta < 1$, often $\beta = 0.9$.

Nesterov momentum with one step "look ahead" can prevent oscillation:

$$\boldsymbol{m}_{t+1} \leftarrow \beta \cdot \boldsymbol{m}_t - \eta \nabla \mathcal{L}(\boldsymbol{\theta}_t + \beta \boldsymbol{m}_t)$$

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \boldsymbol{m}_{t+1}$$



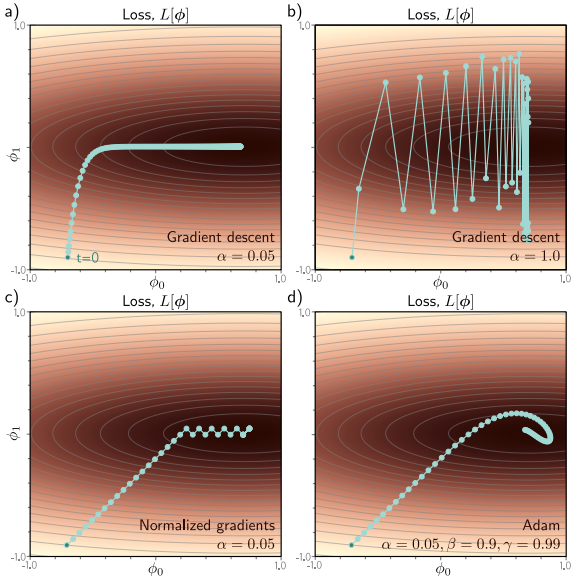
Adaptive moment estimation (Adam)

Low learning rate ($\alpha = 0.05$) we make progress in vertical direction but stall in horizontal direction.

High learning rate ($\alpha = 1.0$) overshoots in the vertical direction and becomes unstable.

Solution: Move a fixed distance along each axis, by normalizing (retaining the sign). However, this can oscillate around the minimum.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta} - \alpha \frac{\nabla \mathcal{L}(\boldsymbol{\theta}_t)}{\sqrt{\nabla \mathcal{L}(\boldsymbol{\theta}_t)^2 + \epsilon}}$$



Adaptive moment estimation (Adam)

Adam update rule: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\hat{\boldsymbol{v}}_t + \epsilon}} \hat{\boldsymbol{m}}_t$ where:

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \nabla \mathcal{L}(\boldsymbol{\theta}_t)$$

- Estimate of the first moment (mean) of the gradient
- Exponentially decay average of past gradients m_t (similar to momentum)

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)(\nabla \mathcal{L}(\boldsymbol{\theta}_t))^2$$

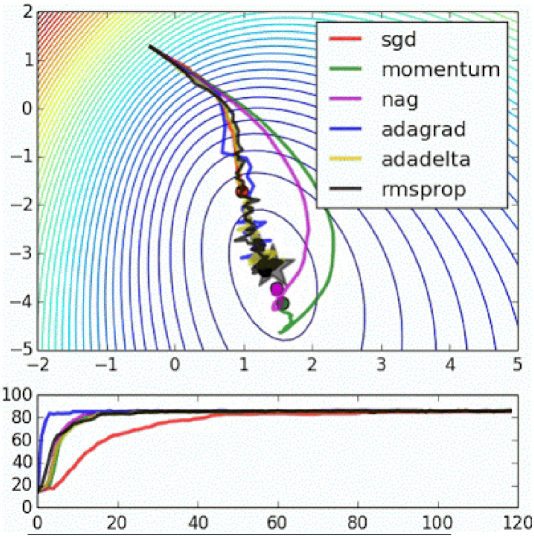
- Estimate of the second moment (uncentered variance) of the gradient
- Exponentially decaying average of past squared gradients v_t

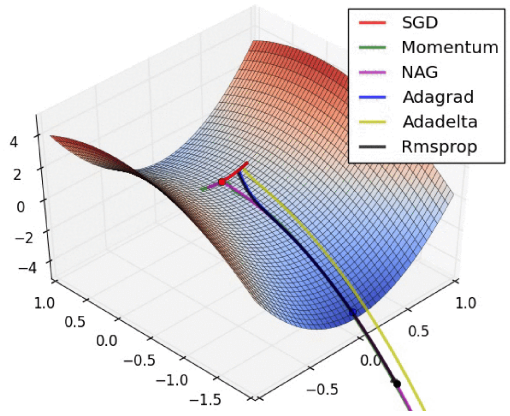
$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1 - \beta_1^t} \qquad \hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1 - \beta_2^t}$$

- Bias correction to avoid bias towards zero (due to 0's initialization)

Default values: $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$.

Visualizing gradient descent variants





https://www.ruder.io/optimizing-gradient-descent/

Newton method

Gradient descent (and its variants) are called **first-order** optimization techniques since they only exploit gradient information (i.e., first-order derivative).

Higher-order techniques use higher-order derivatives, e.g., the Hessian matrix.

How many steps will it take to converge if the Newton method is applied to linear regression?

Newton method

Gradient descent (and its variants) are called **first-order** optimization techniques since they only exploit gradient information (i.e., first-order derivative).

Higher-order techniques use higher-order derivatives, e.g., the Hessian matrix.

Taylor expansion of f at point θ_t

$$f(\boldsymbol{\theta}_t + \boldsymbol{\delta}) = f(\boldsymbol{\theta}_t) + \boldsymbol{\delta}^T \nabla f(\boldsymbol{\theta}_t) + \frac{1}{2}\boldsymbol{\delta}^T \nabla^2 f(\boldsymbol{\theta}_t)\boldsymbol{\delta} + O(\boldsymbol{\delta}^3)$$

How many steps will it take to converge if the Newton method is applied to linear regression?

Newton method

Gradient descent (and its variants) are called **first-order** optimization techniques since they only exploit gradient information (i.e., first-order derivative).

Higher-order techniques use higher-order derivatives, e.g., the Hessian matrix.

Taylor expansion of f at point θ_t

$$f(\boldsymbol{\theta}_t + \boldsymbol{\delta}) = f(\boldsymbol{\theta}_t) + \boldsymbol{\delta}^T \nabla f(\boldsymbol{\theta}_t) + \frac{1}{2}\boldsymbol{\delta}^T \nabla^2 f(\boldsymbol{\theta}_t)\boldsymbol{\delta} + O(\boldsymbol{\delta}^3)$$

Newton method: Minimizing the approximation leads to the update:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - [\nabla^2 f(\boldsymbol{\theta}_t)]^{-1} \nabla f(\boldsymbol{\theta}_t)$$

How many steps will it take to converge if the Newton method is applied to linear regression?

Scalability analysis of the Newton method

+ Few passes through data needed
+ Parallel aggregation of gradient and Hessian possible
+ Gradient requires $O(d)$ data
+ Good rate of convergence

- Hessian requires $O(d^2)$ data
- Update step is $O(d^3)$ & nontrivial to parallelize

Usually, we can only use it only for low dimensional problems!

Outline

Large-scale optimization

Higher-order techniques have nice properties (e.g., convergence), but they are prohibitively expensive for high-dimensional problems.

For large-scale data or high-dimensional problems we usually use first-order techniques (gradient descent variants).

But for real-world large-scale data even first-order methods are too costly.

Solution: stochastic optimization!

Stochastic gradient descent: motivation

Goal: minimize $\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} l_i(\boldsymbol{\theta})$ (+ potential constraints).

For large-scale data: even a single pass through the data is very costly.
Lots of time is required to even compute the very first gradient.

Is it possible to update the parameters more frequently/faster?

Stochastic gradient descent

Consider the task as empirical risk minimization

$$\mathcal{L}(\boldsymbol{\theta}) = n\frac{1}{n}\sum_{i=1} l_i(\boldsymbol{\theta}) = n\mathbb{E}_{i\sim\{1,\dots,n\}}[l_i(\boldsymbol{\theta})]$$

The expectation can be approximated by a smaller amount of samples:
$\mathbb{E}_{i\sim\{1,\dots,n\}}[l_i(\boldsymbol{\theta})] \approx \frac{1}{|S|}\sum_{j\in S}(L_j(\boldsymbol{\theta}))$ // with $S \subseteq \{1,\dots,n\}$.

Equivalently: $\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} l_i(\boldsymbol{\theta}) \approx \frac{n}{|S|}\sum_{j\in S} l_j(\boldsymbol{\theta})$.

Usually, it's smarter to directly define $\mathcal{L}$ as an average instead of a sum since the learning rate will be independent of the batch size. In this case, we don't need the extra n term.

Stochastic gradient descent

Replace exact gradient with a noisy but unbiased estimate from fewer samples.

SGD algorithm:

1. randomly pick a (small) subset S of the points (so called mini-batch)
2. compute gradient based on the mini-batch
3. update: $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \tau \cdot \frac{n}{|S|} \cdot \sum_{j \in S} \boldsymbol{\nabla} L_j(\boldsymbol{\theta}_t)$
4. pick a new subset and repeat with 2

"Original" SGD uses mini-batches of size 1, but larger mini-batches lead to more stable gradients (i.e., smaller variance in the estimated gradient.)

In many cases, the data is sampled so that we don't see any data point twice. Then, each full iteration over the complete data set is called one "epoch".

Perceptron

Let $y_i \in \{-1, 1\}$ and $\hat{(y)}_i = \text{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$ and $l(\hat{y}, y) = \max(0, -\hat{y}y) = l_{01}(\hat{y}, y)$.

If sample i is classified incorrectly, the gradients are

$$\nabla_{\boldsymbol{w}} l(\hat{y}_i, y_i) = \nabla_{\boldsymbol{w}}[(\boldsymbol{w}^T \boldsymbol{x}_i + b) \cdot y_i] = y_i \cdot \boldsymbol{x}_i$$
$$\nabla_b l(\hat{y}_i, y_i) = \nabla_b[(\boldsymbol{w}^T \boldsymbol{x}_i + b) \cdot y_i] = y_i \cdot b$$

This leads to the update

$$\boldsymbol{w} =\leftarrow \begin{cases} \boldsymbol{w} + \boldsymbol{x}_i & \text{if } y_i = 1, \\ \boldsymbol{w} - \boldsymbol{x}_i & \text{if } y_i = -1. \end{cases} \qquad w_0 \leftarrow \begin{cases} w_0 + 1 & \text{if } y_i = 1, \\ w_0 - 1 & \text{if } y_i = -1. \end{cases}$$

If sample i is classified incorrectly, the gradients are zero.

Orignial rules: stochastic gradient descent with mini-batch size 1 and $\eta = 1$.

Convergence in expectation

Subject to relatively mild assumptions, stochastic gradient descent converges almost surely to a global minimum when the objective function is convex.

And almost surely to a local minimum for non-convex functions.

The expectation of the residual error ρ decreases with speed:

$$\mathbb{E}[\rho] \sim t^{-1} \quad \Leftrightarrow \quad t \sim \mathbb{E}[\rho]^{-1}$$

Compare to standard gradient descent that has speed $t \sim \log \rho^{-1}$ which has faster convergence speed, but each iteration takes longer.

SGD can also act as a regularized and avoid getting stuck in "bad" local minima.

Optimizing logistic regression

Recall we wanted to solve $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$

$$\mathcal{L}(\boldsymbol{w}) = -\ln p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{X}) = -\sum_{i=1}^{N} y_i \ln \sigma(\boldsymbol{w}^T \boldsymbol{x}_i) + (1 - y_i) \ln(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i))$$

The closed-form solution does not exist, but we can find $\boldsymbol{w}^*$ using gradient descent.

Is $\mathcal{L}(\boldsymbol{w})$ convex?
Can you use SGD?
How can you choose the learning rate?
What changes if we add regularization, i.e. $\mathcal{L}_{\text{reg}}(\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2$?

Outline

Distributed learning

Assuming we have a single machine, SGD speeds up by using a subset of the data.

Might still be too slow when operating with really large data and large models.

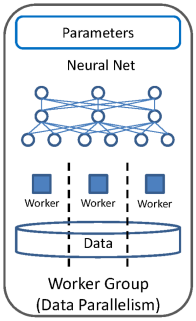
In practice, we often have multiple machines available $\Rightarrow$ distributed learning.

Distribute computation across multiple machines.

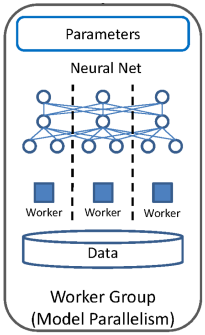
Core challenge: distribute work so that communication doesn't kill you.

Distributed learning: data vs. model parallelism

Multiple model replicas simultaneously process different subsets. All collaborate to update model state (parameters) in shared parameter server(s).

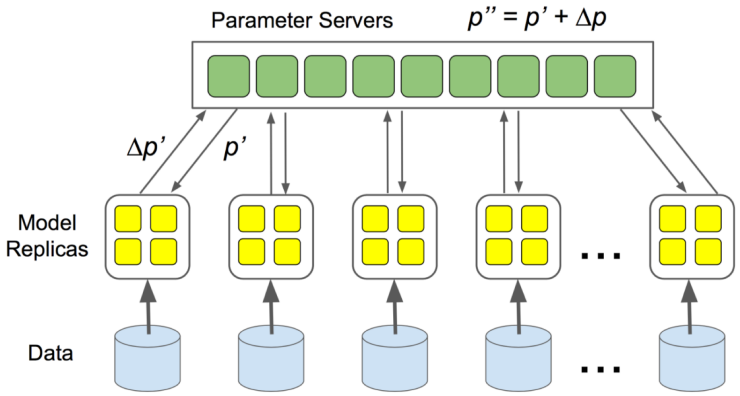


Many models have lots of inherent parallelism, e.g., local connectivity in CNNs or matrix factorization where some parameters are instance specific.



Parameter server

General goal: Keep the time to send/receive parameters over the network small, compared to the actual time used for computation.



Distributed learning in practice

Distributed optimization/learning is essential when operating with very large data (and large models).

Many modern ML frameworks (e.g., Tensorflow, PyTorch, MXNet, ...) provide support for distributed learning.

Many further aspects/challenges:

- Desired synchronization
- Fault tolerance, recovery
- Automatic placement (of data/model) to reduce communication

Summary

General task: Find solution $\boldsymbol{\theta}^*$ minimizing function $\mathcal{L}(\boldsymbol{\theta})$

Verifying convexity: definition, special results (first- or second-order convexity), convexity-preserving operations.

Gradient descent: $\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t - \eta \nabla \mathcal{L}(\boldsymbol{\theta}_t)$. Choosing the learning rate η:

- Line search, fixed, scheduled, adaptive (momentum, AdaGrad, Adam).

Stochastic gradient descent (SGD): subsets of data (mini-batches) at each step.

Distributed learning exploits multiple machines: data/model parallelism.

Reading material

Main reading

- "Convex Optimization" by Boyd
  [ch. 2.1 – 2.3, 3.1, 3.2, 4.1, 4.2, 9]
- "Probabilistic Machine Learning: An Introduction" by Murphy
  [ch. 8.1–8.4]
- "Understanding Deep Learning" by Simon J.D. Prince
  [ch. 6]
- "An overview of gradient descent optimization algorithms" by S. Ruder

Some figures are from the "Understanding Deep Learning" book.