

Machine Learning

Lecture 6: Optimization

Prof. Dr. Stephan Günnemann

Data Analytics and Machine Learning
Technical University of Munich

Winter term 2022/2023

Motivation

- Many machine learning tasks are optimization problems
- Examples we've already seen:
 - Linear Regression $\mathbf{w}^* = \arg \min_{\mathbf{w}} \frac{1}{2}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$
 - Logistic Regression $\mathbf{w}^* = \arg \min_{\mathbf{w}} -\ln p(\mathbf{y} | \mathbf{w}, \mathbf{X})$
- Other examples:
 - Support Vector Machines: find hyperplane that separates the classes with a maximum margin
 - k-means: find clusters and centroids such that the squared distances is minimized
 - 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 θ denote the variables/parameters of our problem we want to learn
 - e.g. $\theta = w$ in Logistic Regression
- Let \mathcal{X} denote the domain of θ ; the set of valid instantiations
 - constraints on the parameters!
 - e.g. $\mathcal{X} = \text{set of (positive) real numbers}$
- Let $f(\theta)$ denote the **objective function**
 - e.g. f is the negative log likelihood
- Goal: Find solution θ^* minimizing function
$$f : \theta^* = \arg \min_{\theta \in \mathcal{X}} f(\theta)$$
 - find a global minimum of the function f !
 - similarly, for some problems we are interested in finding the maximum

Introductory Example

- Goal: Find minimum of function

$$f(\theta) = 0.6 * \theta^4 - 5 * \theta^3 + 13 * \theta^2 - 12 * \theta + 5$$

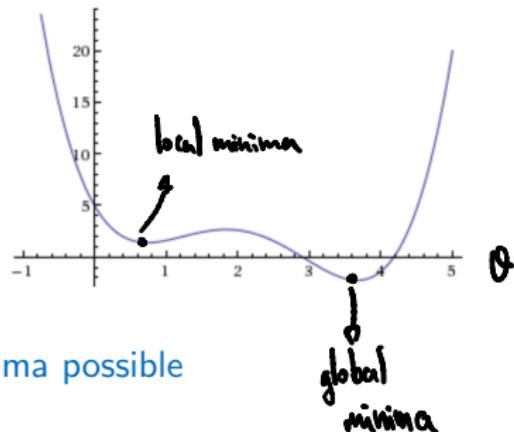
- Unconstrained optimization + differentiable function

- Necessary condition for minima

- Gradient = 0

- Sufficient?

- General challenge: multiple local minima possible

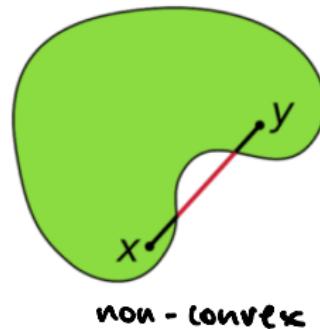
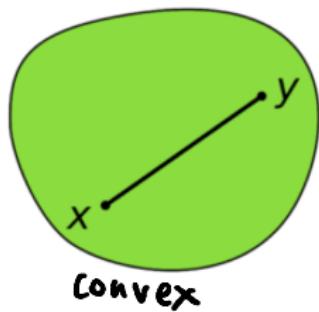


Convexity: Sets

- X is a convex set

iff

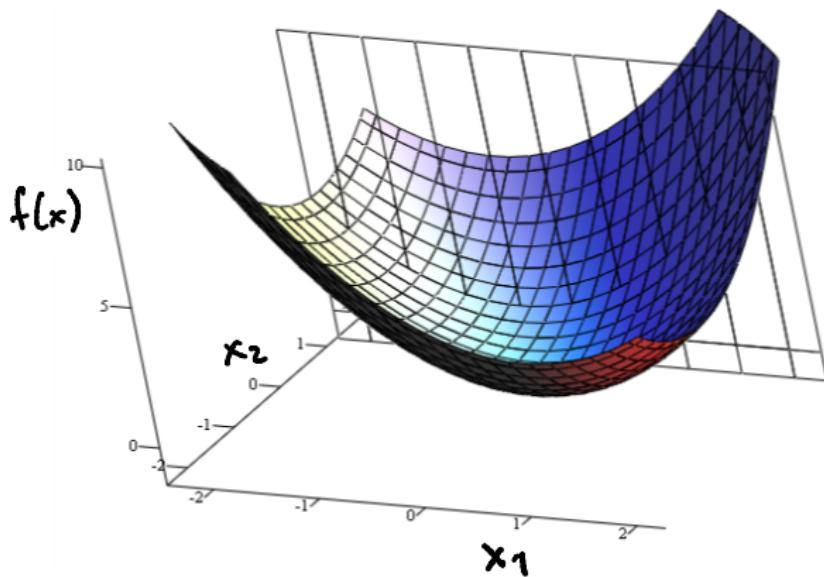
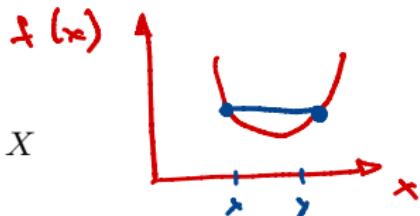
for all $x, y \in X$ it follows that $\lambda x + (1 - \lambda)y \in X$ for $\lambda \in [0, 1]$



Convexity: Functions

- $f(\mathbf{x})$ is a convex function on the convex set X
iff

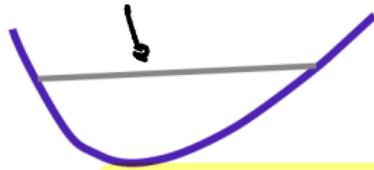
for all $\mathbf{x}, \mathbf{y} \in X : \lambda f(\mathbf{x}) + (1 - \lambda)f(\mathbf{y}) \geq f(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y})$ for $\lambda \in [0, 1]$



Convexity and *minimization problems*

- Region **above** a convex function is convex

the connection is above the function



$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$

hence $\lambda f(x) + (1 - \lambda)f(y) \in X$ for $x, y \in X$

- Convex functions have no local minima which are not global minima
 - 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 $\nabla f_0(\theta^*) = 0$ then θ is a global minimum
 - minimization becomes "relatively easy"

First order convexity conditions (I)

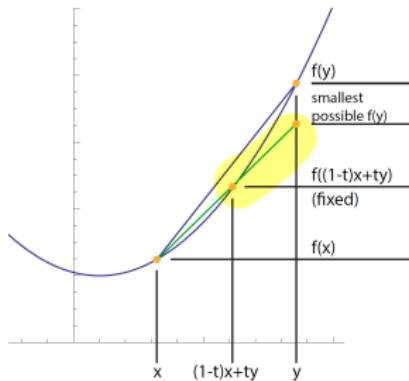
Properties

- Convexity imposes a rate of rise on the function

(function →) (connection →)

- $f((1-t)x+ty) \leq (1-t)f(x)+tf(y)$
- $f(y) - f(x) \geq \frac{f((1-t)x+ty)-f(x)}{t}$

- Difference between $f(y)$ and $f(x)$ is bounded by function values between x and y



First order convexity conditions (II)

$$\bullet f(\mathbf{y}) - f(\mathbf{x}) \geq \frac{f((1-t)\mathbf{x} + t\mathbf{y}) - f(\mathbf{x})}{t}$$

- Let $t \rightarrow 0$ and apply the definition of the derivative

$$\bullet f(\mathbf{y}) - f(\mathbf{x}) \geq (\mathbf{y} - \mathbf{x})^T \nabla f(\mathbf{x})$$

- Theorem:

Suppose $f : X \rightarrow \mathbb{R}$ is a differentiable function

and X is convex. Then f is convex iff for $\mathbf{x}, \mathbf{y} \in X$

$$\left[\underbrace{\frac{f(\mathbf{x} + \epsilon) - f(\mathbf{x})}{\epsilon}}_{\epsilon \rightarrow 0} \right] \cdot (\mathbf{y} - \mathbf{x})$$

$\epsilon \rightarrow 0 \Rightarrow$ gradient
 $\nabla f(\mathbf{x})$

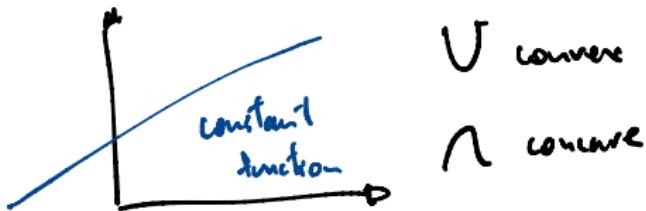
$$f(\mathbf{y}) \geq f(\mathbf{x}) + (\mathbf{y} - \mathbf{x})^T \nabla f(\mathbf{x})$$

- Proof. See Boyd p.70

Verifying convexity (I)

- Convexity makes optimization "easier"
 - How to verify whether a function is convex?
 - For example: $e^{x_1+2*x_2} + x_1 - \log(x_2)$ convex on $[1, \infty) \times [1, \infty)$?
1. Prove whether the definition of convexity holds (See slide 6)
 2. Exploit special results
 - First order convexity (See slide 9)
 - Example: A twice differentiable function of one variable is convex on an interval if and only if its **second-derivative is non-negative** on this interval
 - More general: a twice differentiable function of several variables is convex (on a convex set) if and only if its **Hessian matrix is positive semidefinite** (on the set)

Verifying convexity (II)



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

- a) Start with simple convex functions, e.g.

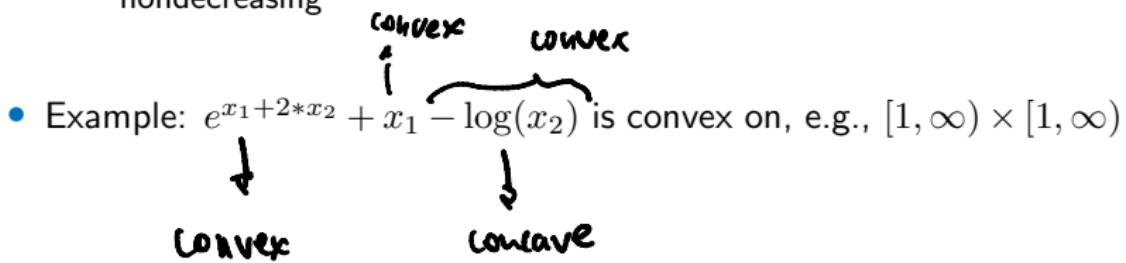
↳ concave and convex at the same time

- $f(x) = \text{const}$ and $f(x) = x^T \cdot b$ (there are also concave functions)
 - $f(x) = e^x$

- b) Apply "construction rules" (next slide)

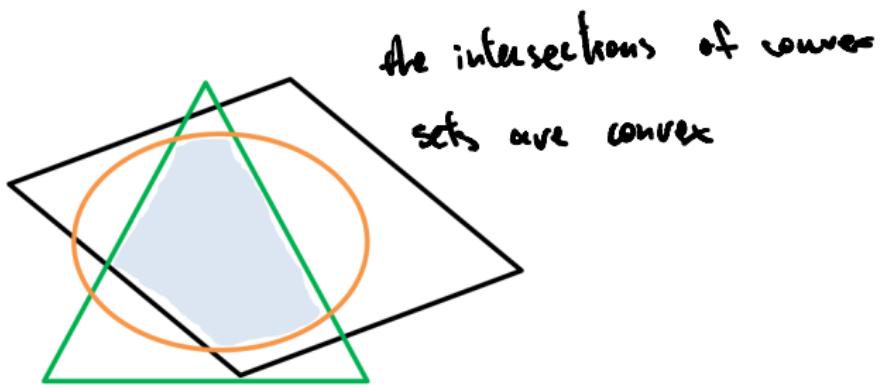
Convexity preserving operations

- Let $f_1 : \mathbb{R}^d \rightarrow \mathbb{R}$ and $f_2 : \mathbb{R}^d \rightarrow \mathbb{R}$ be convex functions, and $g : \mathbb{R}^d \rightarrow \mathbb{R}$ be a concave function, then
 - $h(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x})$ is convex
 - $h(\mathbf{x}) = \max\{f_1(\mathbf{x}), f_2(\mathbf{x})\}$ is convex
 - $h(\mathbf{x}) = c \cdot f_1(\mathbf{x})$ is convex if $c \geq 0$
 - $h(\mathbf{x}) = c \cdot g(\mathbf{x})$ is convex if $c \leq 0$
 - $h(\mathbf{x}) = f_1(\mathbf{A}\mathbf{x} + \mathbf{b})$ is convex (\mathbf{A} matrix, \mathbf{b} vector)
 - $h(\mathbf{x}) = m(f_1(\mathbf{x}))$ is convex if $m : \mathbb{R} \rightarrow \mathbb{R}$ is convex and nondecreasing



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

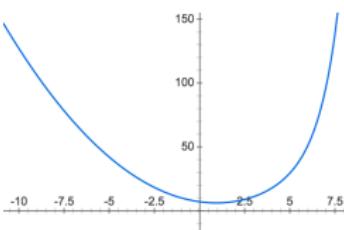


An easy problem

Convex objective function f

- Objective function differentiable on its whole domain
 - i.e. we are able to compute gradient f' at every point
 - We can solve $f'(\theta) = 0$ for θ analytically
 - i.e. solution for θ where gradient = 0 is known
 - Unconstrained minimization (~~constrain domain~~)
 - i.e. above computed solution for θ is valid
 - We are done!
-
- Example: Ordinary Least Squares Regression

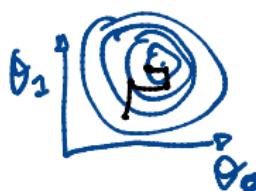
$$x^2 + e^{x-3} - 2x + 7$$



Outlook

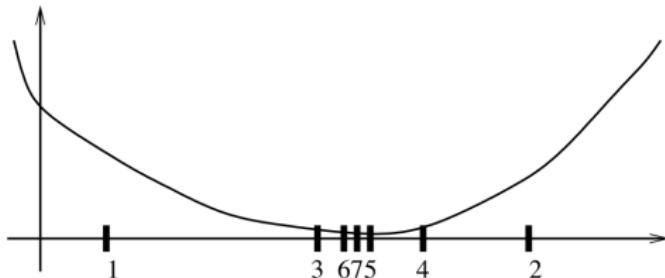
- Unfortunately, many problems are harder...
- No analytical solution for $f'(\theta) = 0$
 - e.g. Logistic Regression
 - Solution: try numerical approaches, e.g. gradient descent
- Constraint on θ
 - e.g. $f'(\theta) = 0$ only holds for points outside the domain
 - Solution: constrained optimization
- f not differentiable on the whole domain
 - Potential solution: subgradients; or is it a discrete optimization problem?
- f not convex
 - Potential solution: convex relaxations; convex in some variables?

One-dimensional problems



coordinate
descent /
coordinate
interval
bisection

- Key Idea
 - For differentiable f search for θ with $\nabla f(\theta) = 0$
 - Interval bisection (derivative is monotonic)
- Can be extended to nondifferentiable problems
 - exploit convexity in upper bound and keep 5 points



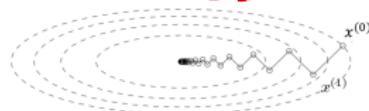
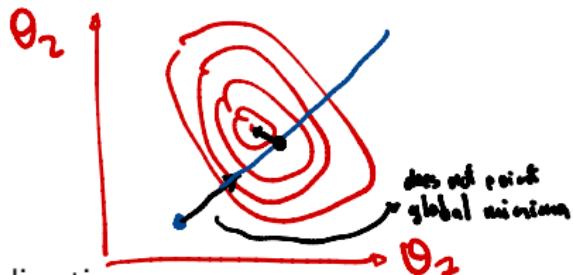
Require: a, b , Precision ϵ
Set $A = a, B = b$
repeat
 if $f'(\frac{A+B}{2}) > 0$ then
 $B = \frac{A+B}{2}$
 else
 $A = \frac{A+B}{2}$
 end if
until
$$(B - A) \min(|f'(A)|, |f'(B)|) \leq \epsilon$$

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

solution on the left

Gradient Descent

- Key Idea
 - Gradient points into steepest ascent direction
 - Locally, the gradient is a good approximation of the objective function



- GD with Line Search
 - Get descent direction, then unconstrained line search
 - Turn a multidimensional problem into a one-dimensional problem that we already know how to solve

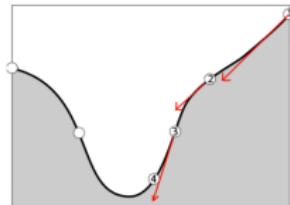
given a starting point $\theta \in \text{Dom}(f)$

repeat

1. $\Delta\theta := -\nabla f(\theta)$
2. Line search. $t^* = \arg \min_{t>0} f(\theta + t \cdot \Delta\theta)$
3. Update. $\theta := \theta + t^* \Delta\theta$

until stopping criterion is satisfied.

how much goes through blue line
(1D problem)



Gradient Descent convergence

- Let p^* be the optimal value, θ^* be the minimizer - the point where the minimum is obtained, and $\theta^{(0)}$ be the starting point
- For strongly convex f (replace \geq with $>$ in the definition of convexity) the residual error ρ , for the k -th iteration is:

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

$f(\theta^{(k)})$ converges to p^* as $k \rightarrow \infty$

- We must have $f(\theta^{(k)}) - p^* \leq \epsilon$ after at most $\frac{\log((f(\theta^{(0)}) - p^*)\epsilon)}{\log(1/c)}$ iterations
- Linear convergence for strongly convex objective
 - $k \sim \log(\rho^{-1})$ // k = number of iterations, ρ
- Linear convergence for strongly convex objective
 - i.e. linear when plotting on a log scale - old statistics terminology

Distributed/Parallel implementation

- Often problems are of the form
 - $f(\theta) = \sum_i L_i(\theta) + g(\theta)$
 - where i iterates over, e.g., each data instance
- Example OLS regression: // with regularization
 - $L_i(\mathbf{w}) = (\mathbf{x}_i^T \mathbf{w} - y_i)^2$ $g(\mathbf{w}) = \lambda \cdot \|\mathbf{w}\|_2^2$
- Gradient can simply be decomposed based on the sum rule
- Easy to parallelize/distribute

Basic steps

given a starting point $\theta \in \text{Dom}(f)$
repeat

1. $\Delta\theta := -\nabla f(\theta)$
2. Line search. $t^* = \arg \min_{t>0} f(\theta + t \cdot \Delta\theta)$
3. Update. $\theta := \theta + t^* \Delta\theta$

until stopping criterion is satisfied.

easy parallel computation

evaluating function might be
done multiple times: expensive!

- Distribute data over several machines
- Compute partial gradients (on each machine in parallel)
- Aggregate the partial gradients to the final one
- BUT: Line search is expensive
 - for each tested step size: scan through all datapoints

Scalability analysis

- + Linear time in number of instances
- + Linear memory consumption in problem size (not data)
- + Logarithmic time in accuracy
- + 'Perfect' scalability

- Multiple passes through dataset for each iteration

A faster algorithm

- Avoid the line search; simply pick update

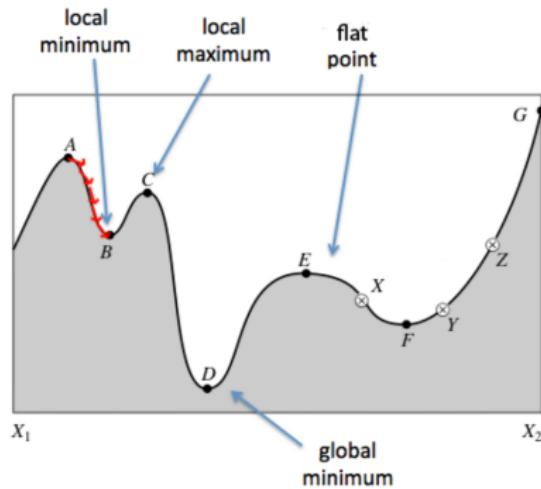
$$\theta_{t+1} \leftarrow \theta_t - \tau \cdot \nabla f(\theta_t)$$

— τ is often called the learning rate

- Only a single pass through data per iteration
- Logarithmic iteration bound (as before)
 - if learning rate is chosen "correctly"
- How to pick the learning rate?
 - too small: slow convergence (*very slow*)
 - too high: algorithm might oscillate, no convergence (*overshoot*)
- Interactive tutorial on optimization
 - <http://www.benfrederickson.com/numerical-optimization/>

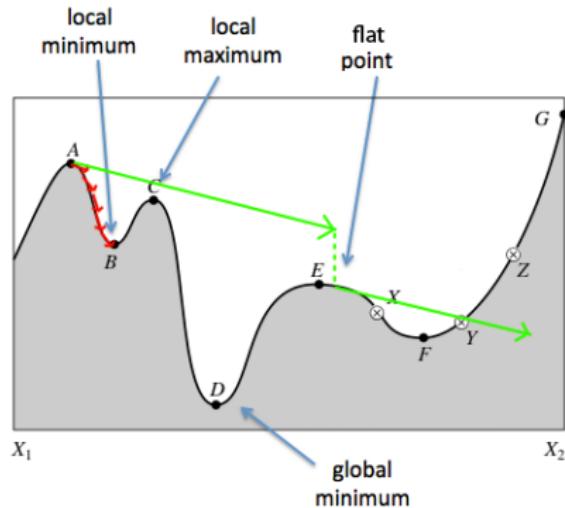
The value of τ

- A too **small** value for τ has two drawbacks
 - We find the minimum more slowly
 - We end up in local minima or saddle/flat points



The value of τ

- A too large value for τ has one drawback
 - You may never find a minimum; oscillations usually occur
- We only need 1 step to overshoot



Learning rate adaptation

Starts with 1 big learning rate and in each iteration, it decreases a little bit

- Simple solution: let the learning rate be a decreasing function τ_t of the iteration number t
 - so called **learning rate schedule**
 - first iterations cause large changes in the parameters; later do fine-tuning
 - convergence easily guaranteed if $\lim_{t \rightarrow \infty} \tau_t = 0$
 - example: $\tau_{t+1} \leftarrow \alpha \cdot \tau_t$ for $0 < \alpha < 1$

Learning rate adaptation

- Other solutions: Incorporate "history" of previous gradients
- Momentum:
 - $m_t \leftarrow \tau \cdot \nabla f(\theta_t) + \gamma \cdot m_{t-1}$ // often $\gamma = 0.5$
 - $\theta_{t+1} \leftarrow \theta_t - m_t$
 - As long as gradients point to the same direction, the search accelerates
takes larger steps if the previous one and the current one goes in the same direction
- AdaGrad:
 - different learning rate per parameter
 - learning rate depends inversely on accumulated "strength" of all previously computed gradients
 - large parameter updates ("large" gradients) lead to small learning rates

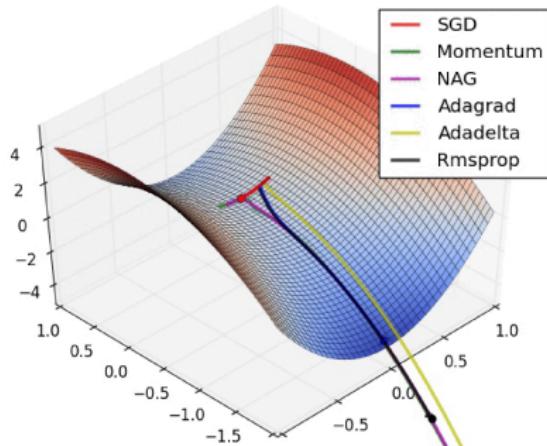
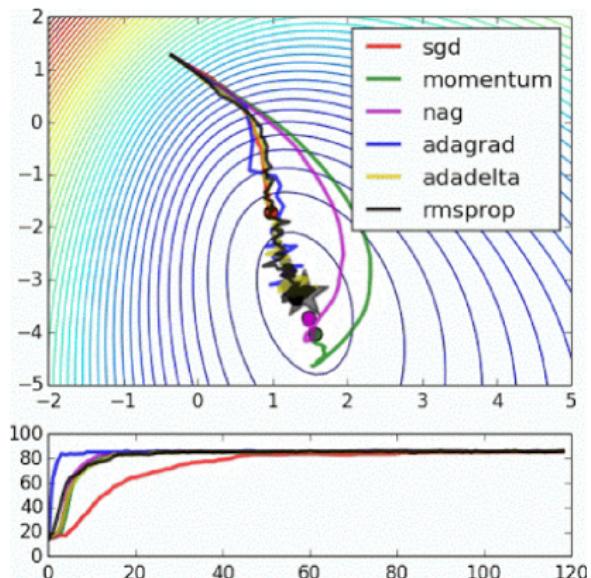
Adaptive moment estimation (Adam)

- $m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla f(\theta_t)$
 - estimate of the first moment (mean) of the gradient
 - Exponentially decaying average of past gradients m_t (similar to momentum)
- $v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla f(\theta_t))^2$
 - estimate of the second moment (uncentered variance) of the gradient
 - exponentially decaying average of past squared gradients v_t
- To avoid bias towards zero (due to 0's initialization) use bias-corrected version instead:
 - $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$ $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$
- Finally, the Adam update rule for parameters θ :
 - $\theta_{t+1} = \theta_t - \frac{\tau}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$
- Default values: $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$

Visualizing gradient descent variants

- AdaGrad and variants
 - often have faster convergence
 - might help to escape saddlepoints

<http://sebastianruder.com/optimizing-gradient-descent/>

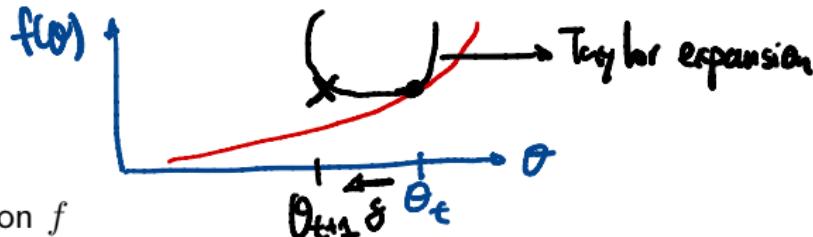


Discussion

- Gradient descent and similar techniques are called first-order optimization techniques
 - they only exploit information of the gradients (i.e. first order derivative)
- Higher-order techniques use higher-order derivatives
 - e.g. second-order = Hessian matrix
 - Example: Newton Method

Newton method

$$x^T A x \geq 0$$



- Convex objective function f
- Nonnegative second derivative: $\nabla^2 f(\theta) \succeq 0$ // Hessian matrix
 - $\nabla^2 f(\theta) \succeq 0$ means that the Hessian is positive semidefinite
- Taylor expansion of f at point θ_t (looks at the 1st and 2nd derivative)

$$f(\theta_t + \delta) = f(\theta_t) + \delta^T \nabla f(\theta_t) + \frac{1}{2} \delta^T \nabla^2 f(\theta_t) \delta + O(\delta^3) = g(\delta)$$

- Minimize approximation: leads to

$$\nabla g(\delta) = 0$$

$$\theta_{t+1} \leftarrow \theta_t - [\nabla^2 f(\theta_t)]^{-1} \nabla f(\theta_t)$$

$\underbrace{\quad}_{\gamma}$

- Repeat until convergence

Parallel Newton method

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

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

- Use it only for low dimensional problems!

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 / high dimensional problems use first-order techniques
 - i.e. variants of gradient descent
- But for real-world large scale data even first-order methods are too costly
- Solution: **Stochastic optimization!**

Motivation: Stochastic Gradient Descent

- Goal: $f(\theta) = \sum_{i=1}^n L_i(\theta)$ + potential constraints
- For very large data: even a single pass through the data is very costly
- Lots of time 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

$$\frac{1}{n} \left(\sum_{i=1}^n L_i(\boldsymbol{\theta}) \right) = \mathbb{E}_{i \sim \{1, \dots, n\}} [L_i(\boldsymbol{\theta})]$$

- (Exact) expectation can be approximated by smaller sample:
 $\mathbb{E}_{i \sim \{1, \dots, n\}} [L_i(\boldsymbol{\theta})] \approx \frac{1}{|S|} \sum_{j \in S} (L_j(\boldsymbol{\theta}))$ // with $S \subseteq \{1, \dots, n\}$

or equivalently: $\sum_{i=1}^n L_i(\boldsymbol{\theta}) \approx \underbrace{\frac{n}{|S|} \sum_{j \in S} L_j(\boldsymbol{\theta})}_{\text{smaller subset of data}}$

Stochastic Gradient Descent

- Intuition: Instead of using "exact" gradient, compute only a **noisy (but still unbiased) estimate** based on smaller sample
- Stochastic gradient decent:
 1. randomly pick a (small) subset S of the points → so called mini-batch
 2. compute gradient based on mini-batch
 3. update: $\theta_{t+1} \leftarrow \theta_t - \tau \cdot \frac{n}{|S|} \cdot \sum_{j \in S} \nabla L_j(\theta_t)$
 4. pick a new subset and repeat with 2
- "Original" SGD uses mini-batches of size 1
 - 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**".

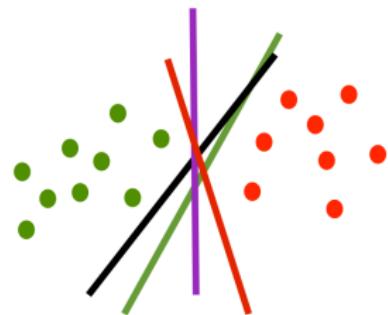
Example: Perceptron

- Simple linear binary classifier:

$$\delta(\mathbf{x}) \begin{cases} 1 & \text{if } \mathbf{w}^T \mathbf{x} + b > 0 \\ -1 & \text{else} \end{cases}$$

- Learning task:

Given $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ with $y_i \in \{-1, 1\}$
Find $\min_{\mathbf{w}, b} \sum_i L(y_i, \mathbf{w}^T \mathbf{x}_i + b)$



- L is the loss function, with $\epsilon > 0$

- e.g. $L(u, v) = \max(0, \epsilon - u \cdot v) = \begin{cases} \epsilon - uv & \text{if } uv < \epsilon \\ 0 & \text{else} \end{cases}$ \leftarrow incorrect prediction
 \leftarrow correct prediction

Example: Perceptron

$$\theta_{t+1} \leftarrow \theta_t - \gamma \cdot \frac{m}{n} \nabla L(\dots)$$

!
batch
size

- Let's solve this problem via SGD
- Result:

initialize $w = 0$ and $b = 0$

repeat

if $y_i \cdot (w^T x_i + b) < \epsilon$ then

$w \leftarrow w + \tau \cdot n \cdot y_i \cdot x_i$ and $b \leftarrow b + \tau \cdot n \cdot y_i$

end if

until all classified correctly

missclassified

τ

$$\tau = \frac{1}{N}$$

- Note: Nothing happens if classified correctly
 - gradient is zero

- Does this remind you of the original learning rules for perceptron?

Convergence in expectation

- Subject to relatively mild assumptions, stochastic gradient descent converges **almost surely** to a global minimum when the objective function is convex
 - almost surely to a local minimum for non-convex functions

- The expectation of the residual error decreases with speed

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

$$/\!/ \text{ i.e. } t \sim \mathbb{E}[\rho]^{-1}$$

residual

$$t f(\theta_t) - f(\theta^*)$$

- Note: Standard GD has speed $t \sim \log \rho^{-1}$
 - faster convergence speed; but each iteration takes longer

Optimizing Logistic Regression

- Recall we wanted to solve $\mathbf{w}^* = \arg \min_{\mathbf{w}} E(\mathbf{w})$

- $E(\mathbf{w}) = -\ln p(\mathbf{y}|\mathbf{w}, \mathbf{X})$

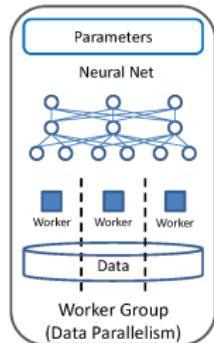
$$= - \sum_{i=1}^N y_i \ln \sigma(\mathbf{w}^T \mathbf{x}_i) + (1 - y_i) \ln(1 - \sigma(\mathbf{w}^T \mathbf{x}_i))$$

- Closed form solution does not exist
- Solution:
 - Compute the gradient $\nabla E(\mathbf{w})$
 - Find \mathbf{w}^* using gradient descent
- Is $E(\mathbf{w})$ convex?
- Can you use SGD?
- How can you choose the learning rate?
- What changes if we add regularization, i.e. $E_{reg}(\mathbf{w}) = E(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2$?

Large-Scale Learning - Distributed Learning

- So far, we (mainly) assumed a single machine
 - SGD achieves speed-up by only operating on a subset of the data
 - Might still be too slow when operating with really large data and large models
 - In practice: We have often multiple machines available
- ⇒ Distributed learning
- Distribute computation across multiple machines
 - Core challenge: distribute work so that communication doesn't kill you

Distributed Learning: Data vs. Model Parallelism



Use multiple model replicas to process different examples at the same time

- all collaborate to update model state (parameters) in shared parameter server(s)

Many models have lots of inherent parallelism

- local connectivity (as found in CNNs)
- specialized parts of model active only for some examples (see, e.g., Matrix Factorization)

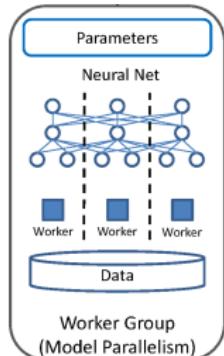


figure based on <https://svn.apache.org/repos/infra/websites/production/singa/content/v0.1.0/architecture.html>

Parameter Server

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

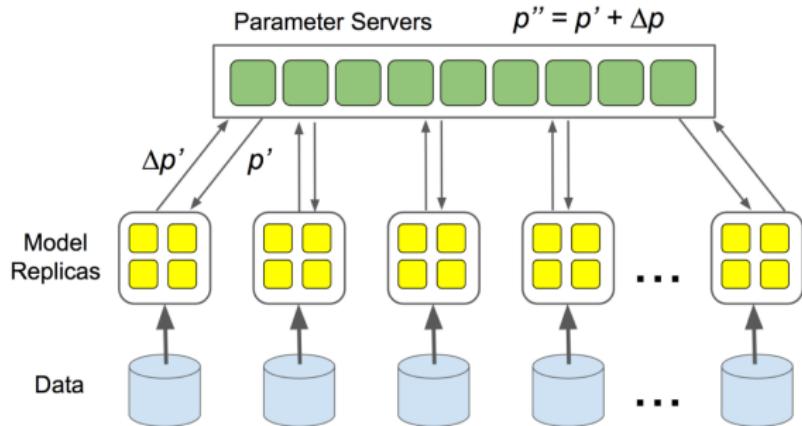


figure from Large Scale Distributed Systems for Training Neural Networks, Jeff Dean

Distributed Learning in Practice

- Distributed optimization/learning is essential when operating with very large data (and large models)
 - Default for training ML models in today's production systems
- 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 θ^* minimizing function f
- Convex sets & functions
 - Global vs. local minimum
 - Verifying convexity: Definition, special results (first-order convexity, 2nd derivative), convexity-preserving operations
- Gradient descent: $\theta := \theta - t \nabla f(\theta)$
 - How to choose t ? Line search, fixed
 - Learning rate: Fix $t = \tau$; or use an adaptive learning rate (momentum, AdaGrad, Adam)
 - Stochastic gradient descent (SGD): Only use part of data (mini-batches) at each step
- Distributed Learning: exploit multiple machines
 - data parallelism, model parallelism

Reading material

Reading material

- Boyd - Convex Optimization: chapters 2.1-2.3, 3.1, 3.2, 4.1, 4.2, 9
 - free PDF version online
- Sebastian Ruder - An overview of gradient descent optimization algorithms
 - <https://arxiv.org/abs/1609.04747>