Machine Learning

Lecture 5: Optimization

Prof. Dr. Stephan Günnemann

Data Mining and Analytics
Technical University of Munich

19.11.2018

Reading material

Reading material

- Boyd - Convex Optimization: chapters 2.1 – 2.3, 3.1, 3.2, 4.1 – 4.4, 9
  - free pdf version online
- Sebastian Ruder - An overview of gradient descent optimization algorithms
  - https://arxiv.org/abs/1609.04747

Motivation

- Many data mining/machine learning tasks are optimization problems
- Examples we've already seen:
  – Linear Regression $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{2} (\boldsymbol{Xw} - \boldsymbol{y})^{\boldsymbol{T}} (\boldsymbol{Xw} - \boldsymbol{y})$
  – Logistic Regression $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} - \ln p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{X})$
- 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 $\boldsymbol{\theta}$ 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
  - constraints on the parameters!
  - e.g. $\mathcal{X}$ = set of (positive) real numbers
- Let $f(\boldsymbol{\theta})$ denote the objective function
  - e.g. f is the negative log likelihood
- Goal: Find solution $\boldsymbol{\theta}^*$ minimizing function f: $\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta} \in \mathcal{X}} f(\boldsymbol{\theta})$
  - find a global minimum of the function f!
  - similarly, for some problems we are interested in finding the maximum

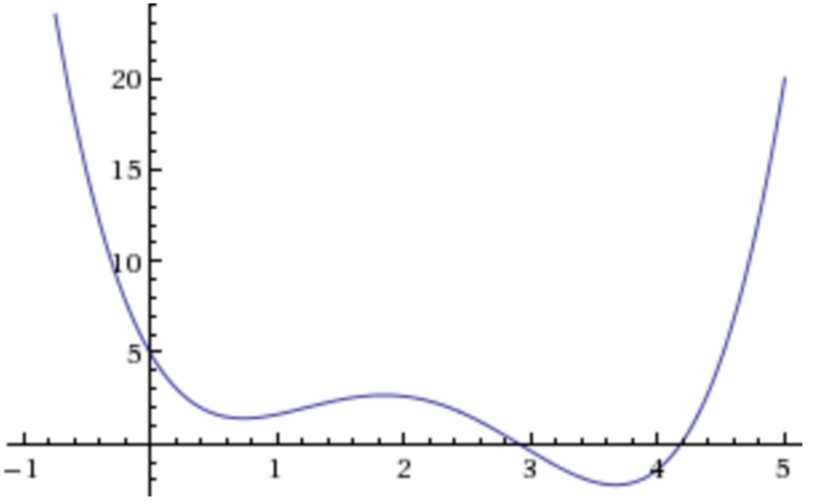
Introductory example

- Goal: Find minimum of function

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

- Unconstrained optimization + differentiable function

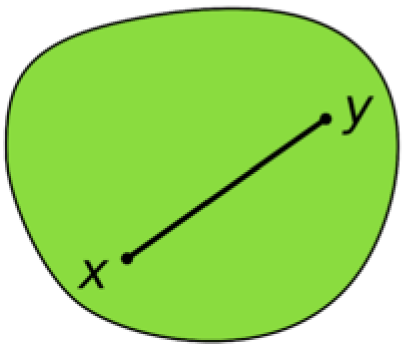
- Necessary condition for minima
  - Gradient = 0
  - Sufficient?

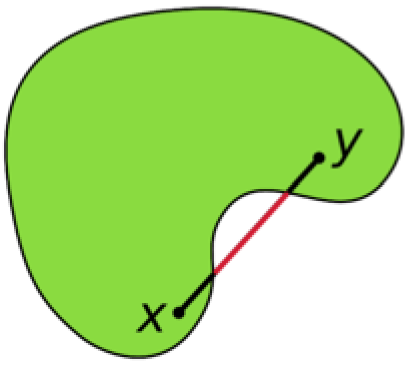


- General challenge: multiple local minima possible

Convexity: Sets

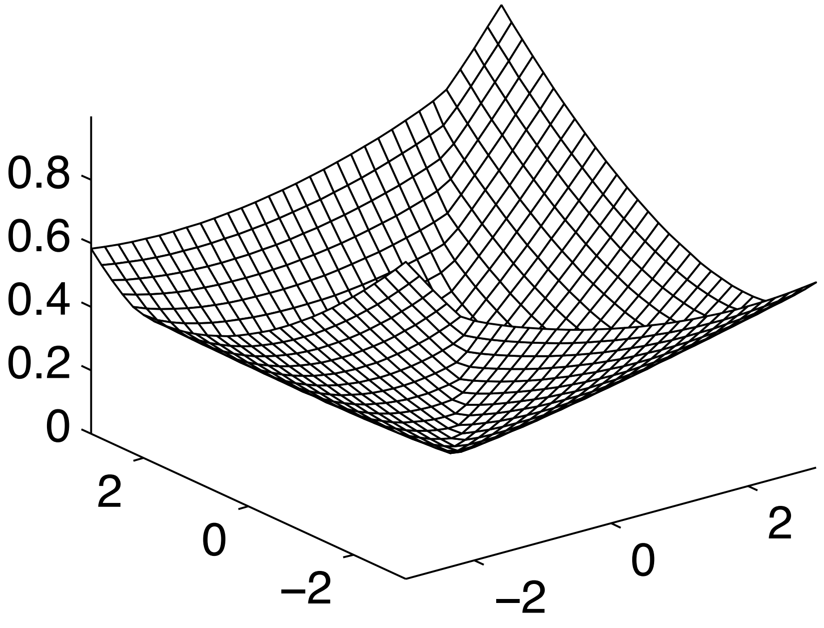
- X is a convex set
  iff
  for all $x, y \in X$ it follows that $\lambda x + (1 - \lambda) y \in X$ for $\lambda \in [0,1]$

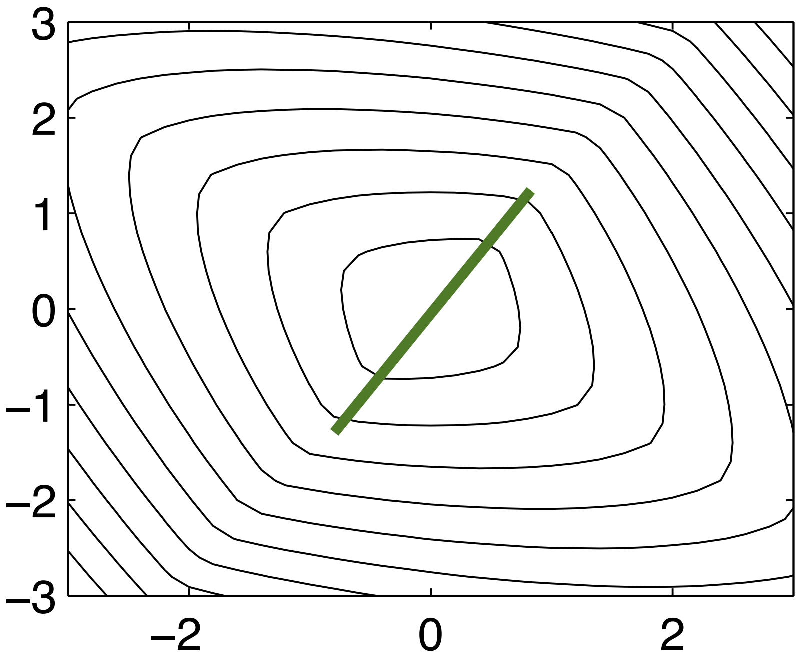




Convexity: Functions

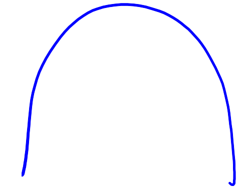
- $f(\boldsymbol{x})$ is a convex function on a convex set X
  iff
  for all $\boldsymbol{x}, \boldsymbol{y} \in X$: $\lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}) \geq f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y})$ for $\lambda \in [0,1]$





Convexity and *minimization* problems

CONCAVE

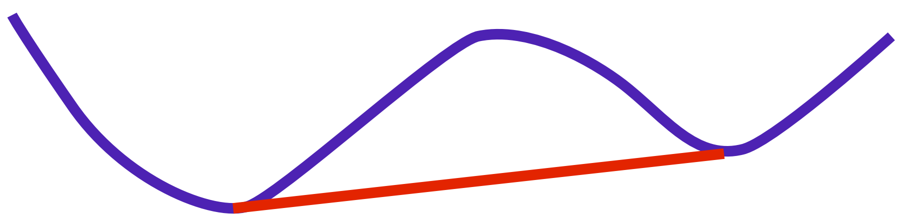


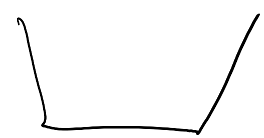
- Region above a convex function is convex

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

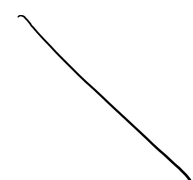
hence $\lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}) \in X$ for $\boldsymbol{x}, \boldsymbol{y} \in X$

- Convex functions don't have local 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 $\boldsymbol{\nabla}\mathrm{f}_0(\boldsymbol{\theta}^*) = 0$ then $\boldsymbol{\theta}^*$ is a global minimum
  - minimization becomes "relatively easy“

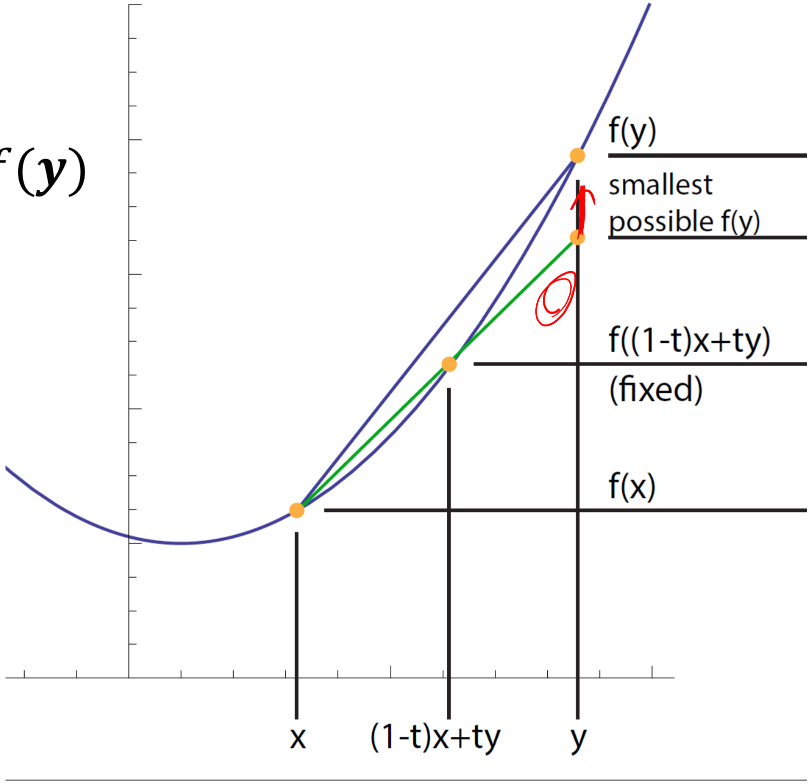


First order convexity conditions (I)

- Convexity imposes a rate of rise on the function

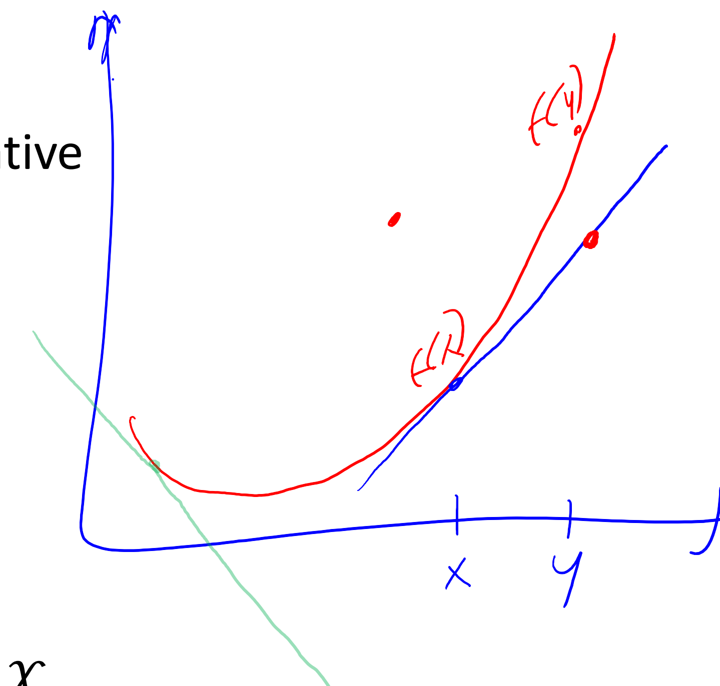
- $f\big((1-t)\boldsymbol{x}+t\boldsymbol{y}\big) \le (1-t)\, f(\boldsymbol{x}) + t\, f(\boldsymbol{y})$
- $f(\boldsymbol{y}) - f(\boldsymbol{x}) \ge \frac{f\big((1-t)\boldsymbol{x}+t\boldsymbol{y}\big)-f(\boldsymbol{x})}{t}$

- Difference between $f(\boldsymbol{y})$ and $f(\boldsymbol{x})$ is bounded by function values between $\boldsymbol{x}$ and $\boldsymbol{y}$



First order convexity conditions (II)

- $f(\boldsymbol{y}) - f(\boldsymbol{x}) \geq \frac{f((1-t)\boldsymbol{x}+t\boldsymbol{y}) - f(\boldsymbol{x})}{t}$
- Let $t \to 0$ and apply the definition of the derivative
- $f(\boldsymbol{y}) - f(\boldsymbol{x}) \geq (\boldsymbol{y}-\boldsymbol{x})^T \nabla f(\boldsymbol{x})$



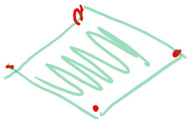
- Theorem:

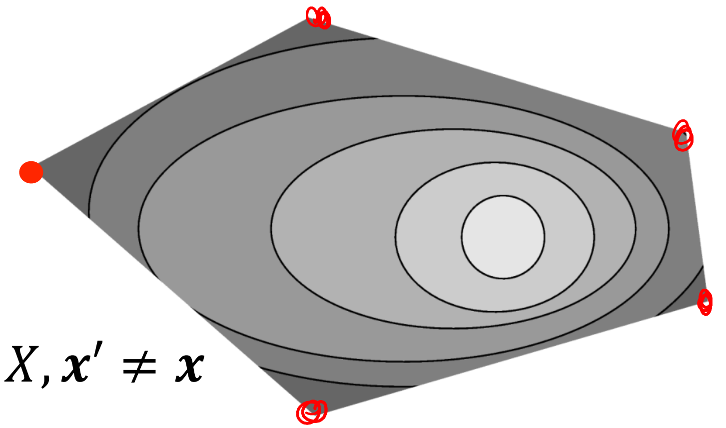
  Suppose $f: \mathcal{X} \to \mathbb{R}$ is a differentiable function and $\mathcal{X}$ is convex. Then f is convex iff for $\mathrm{x}, \mathrm{y} \in \mathcal{X}$

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

- Proof. See Boyd p.70

Convexity: Vertices & Convex Hull





- Given a (closed) convex set X.
  $\boldsymbol{x} \in X$ is a vertex of the convex set if it cannot be extrapolated within the convex set:

  $(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{x}') \notin X$ for $\lambda > 1$ for all $\boldsymbol{x}' \in X, \boldsymbol{x}' \neq \boldsymbol{x}$
- We denote with $\mathrm{Ve}(X)$ the set of all vertices of X

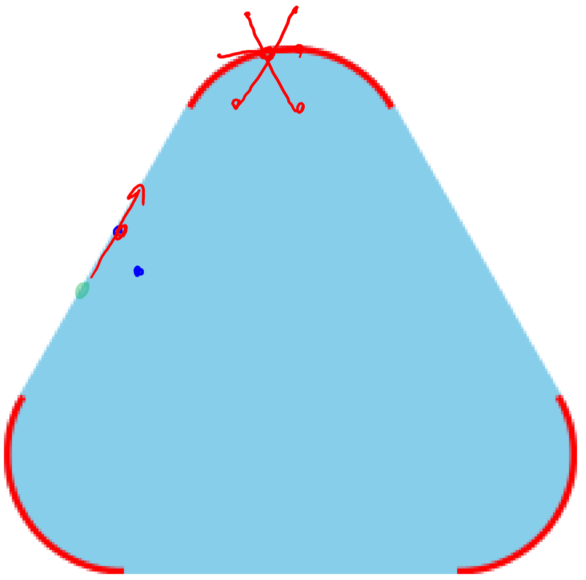
- Convex hull: Given a set of points $X \subseteq \mathbb{R}^d$, the convex hull is defined as
  $\mathrm{Conv}(X) \coloneqq \{\sum_{i=1}^{n} \alpha_i \cdot \boldsymbol{x}_i \,\middle|\, \boldsymbol{x}_i \in X, n \in \mathbb{N}, \sum \alpha_i = 1, \alpha_i \geq 0\}$
- Convex hull of a set is a convex set

Example: Vertices & Convex Hull

- Red = set X
- Red + Blue = convex hull of X

- Here: set X equals to
  the vertices of the convex hull

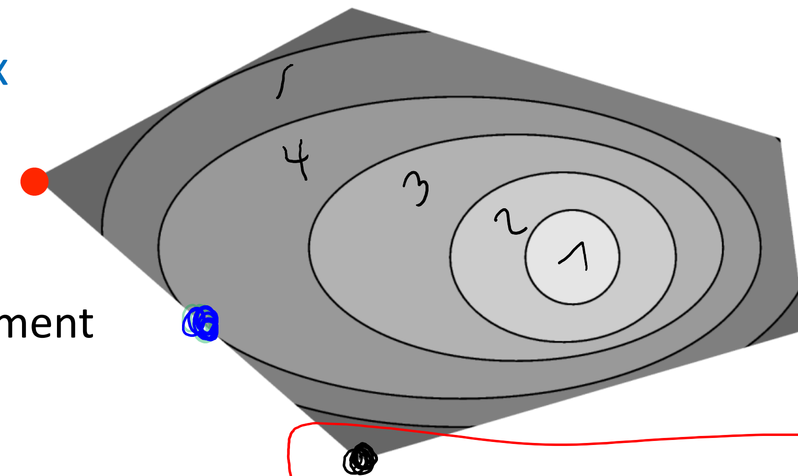
- In general:
  – $\mathrm{Ve}\big(\mathrm{Conv}(X)\big) \subseteq X$



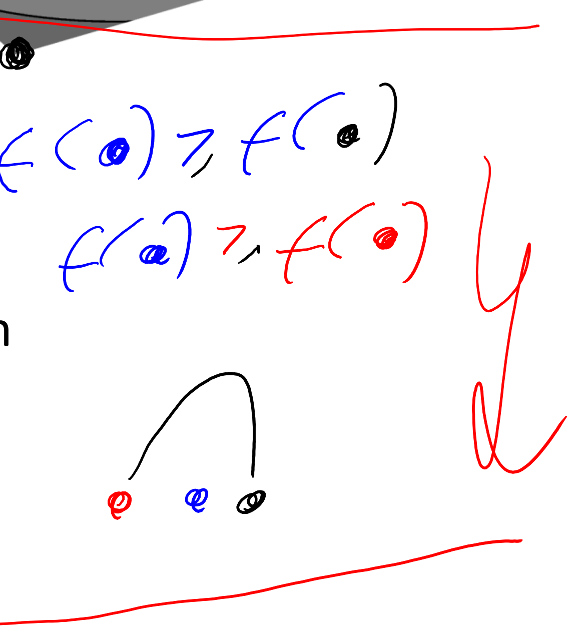
Convexity and *maximization* problems (I)

$$\max_{\theta \in \mathcal{X}} f(\theta)$$

- Maximum over a convex function on a convex set is obtained on a vertex



- Proof:
  - Assume that maximum inside line segment
  - Then function cannot be convex
  - Hence it must be on vertex



- We only need to test the vertices to find the maximum
- In some cases this set is finite (see figure)

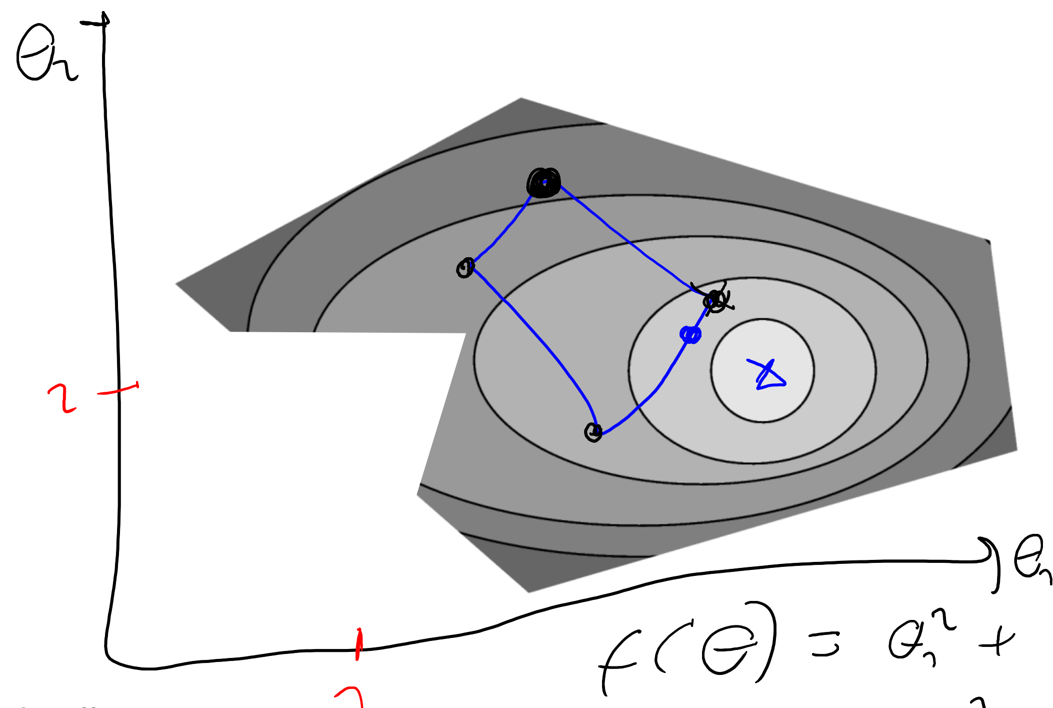
Convexity and *maximization* problems (II)

f is convex ??

- Supremum on convex hull

$$\sup_{\boldsymbol{x}\in X} f(\boldsymbol{x}) = \sup_{\boldsymbol{x}\in \mathrm{Conv}(X)} f(\boldsymbol{x})$$

- Proof: by contradiction



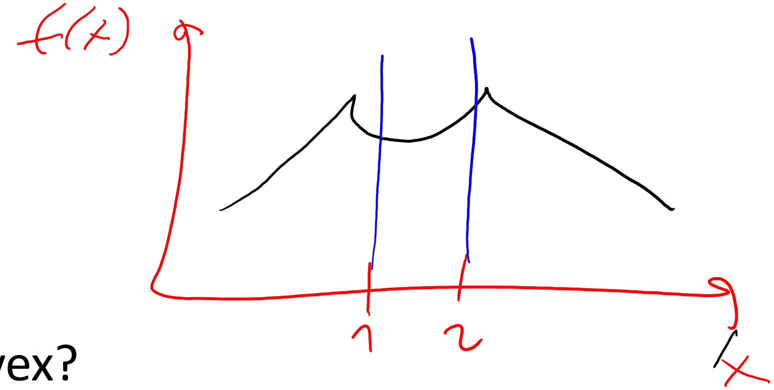
$f(\theta) = \theta_1^2 + 2 \cdot \theta_2^2$

- Instead of working with a "complex" set X, operate with the easier, i.e. convex, set $\mathrm{Conv}(X)$
- One might also simply focus on $\mathrm{Ve}(\mathrm{Conv}(X))$

$\theta_1 + \theta_2 \leq 10$

$\theta_1 \geq 1$ if $\theta_2 \leq 2$

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 7)
2. Exploit special results
   - First order convexity (See slide 10)
   - 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)

$x^T A x \geq 0$

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.

– $f(x) = \text{const}$ and $f(\boldsymbol{x}) = \boldsymbol{x}^T \cdot \boldsymbol{b}$ (these are also concave functions)

– $f(x) = e^x$

b) Apply "construction rules" (next slide)

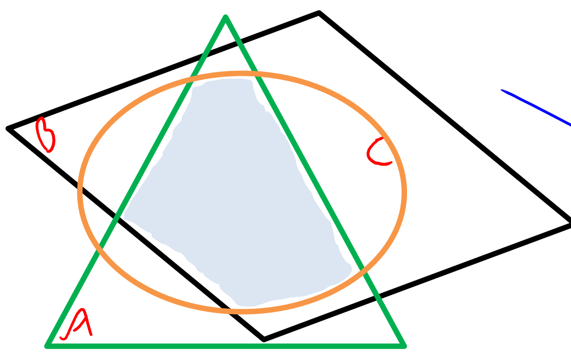
Convexity preserving operations

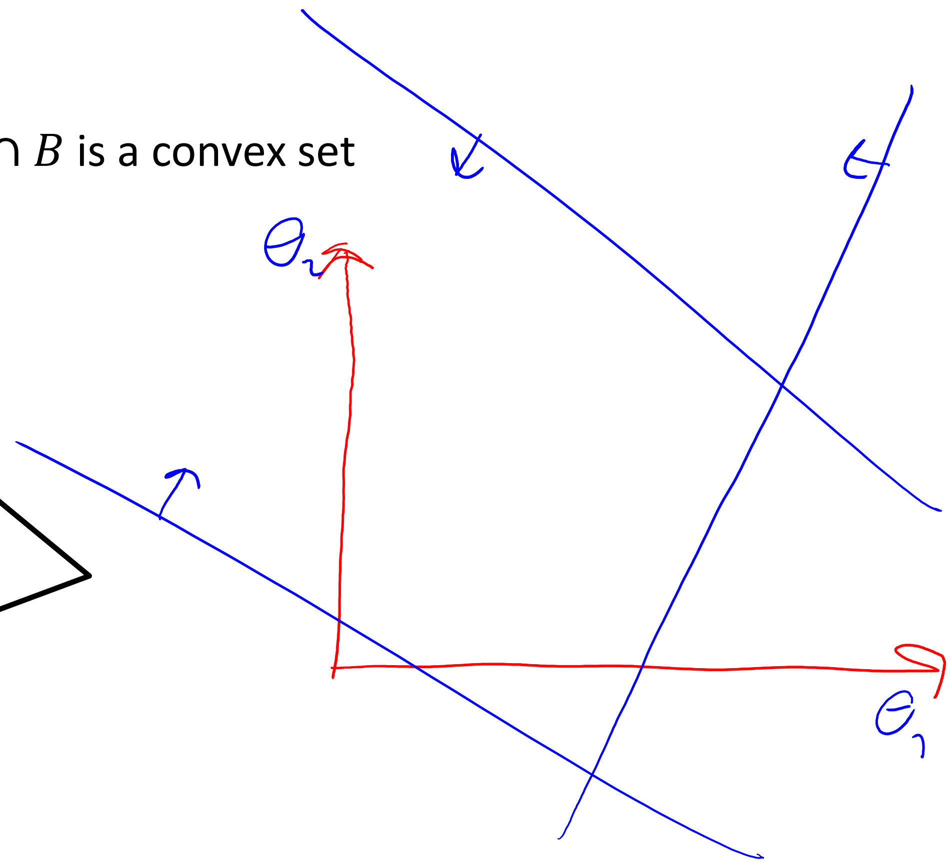
- Let $f_1\colon \mathbb{R}^d \to \mathbb{R}$ and $f_2\colon \mathbb{R}^d \to \mathbb{R}$ be convex functions, and $g\colon \mathbb{R}^d \to \mathbb{R}$ be a concave function, then
  - $h(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$ is convex
  - $h(\boldsymbol{x}) = \max\{f_1(\boldsymbol{x}),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

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

Verifying convexity for 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

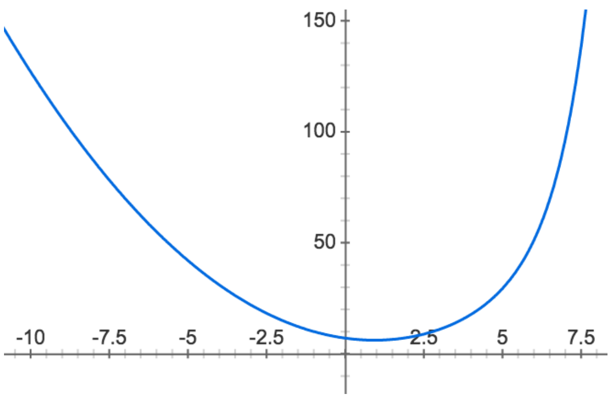
$\theta = \dots$

Convex objective function f

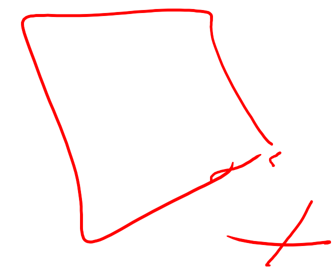
- Objective function differentiable on its whole domain
  - i.e. we are able to compute gradient f' at every point
- We can solve $f'(\boldsymbol{\theta}) = 0$ for $\boldsymbol{\theta}$ analytically
  - i.e. solution for $\boldsymbol{\theta}$ where gradient = 0 is known
- Unconstrained minimization
  - i.e. above computed solution for $\boldsymbol{\theta}$ is valid
- We are done!

- Example: Ordinary Least Squares Regression

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

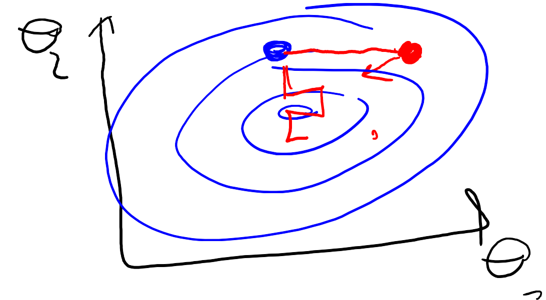


Outlook

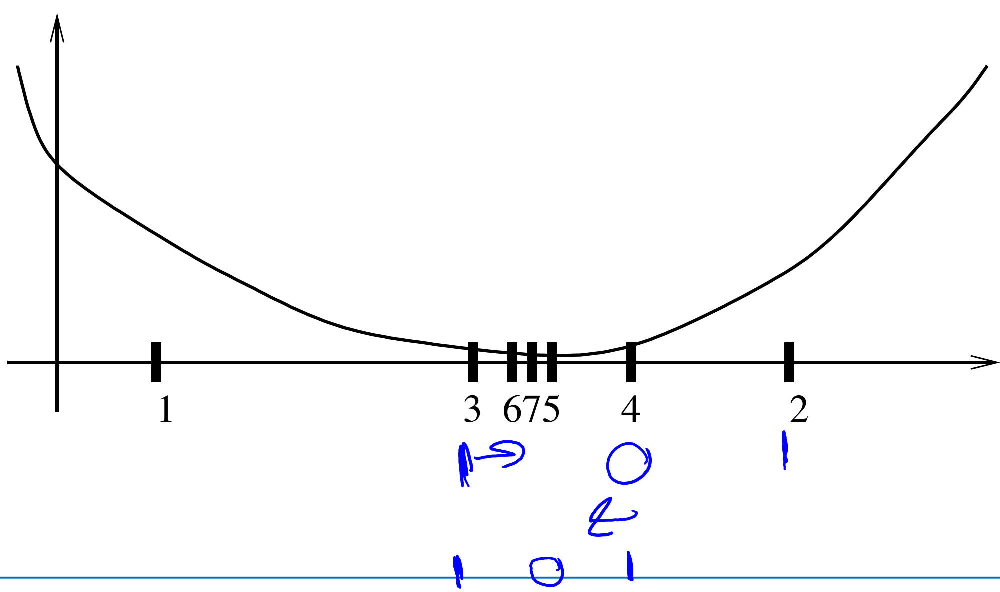


- Unfortunately, many problems are harder…
- No analytical solution for $f'(\boldsymbol{\theta}) = 0$
  - e.g. Logistic Regression
  - Solution: try numerical approaches, e.g. gradient descent
- Constraints on $\boldsymbol{\theta}$
  - e.g. $f'(\boldsymbol{\theta}) = 0$ only holds for points outside the domain
  - Solution: constrained optimization
- f not differentiable on 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



- 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

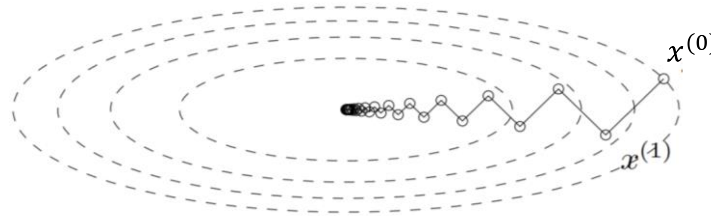


Require: a, b, Precision ϵ
  Set $A = a, B = b$
  repeat
    if $f'\left(\frac{A+B}{2}\right) > 0$ **then**
      $B = \frac{A+B}{2}$
    else
      $A = \frac{A+B}{2}$
    end if
  until $(B - A)\min(|f'(A)|, |f'(B)|) \leq \epsilon$
Output: $x = \frac{A+B}{2}$

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

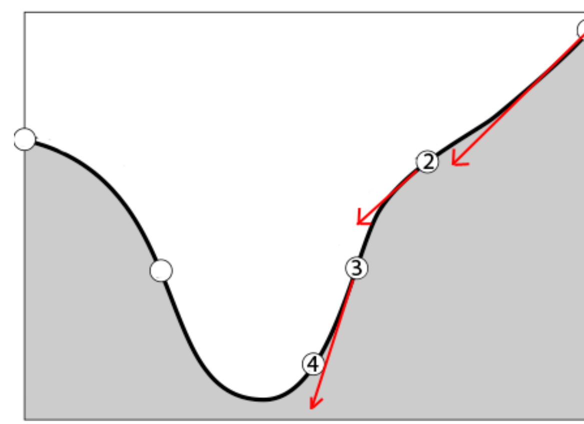


- GD with Line Search
  - Get descent direction, then unconstrained line search
  - ! Turn a multidimensional problem into a one-dimensional problem that we already know how to solve

1-D

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

until stopping criterion is satisfied.



Gradient Descent convergence

- 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
- The residual error ρ, for the k-th iteration is (for strongly convex f):

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

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

  k

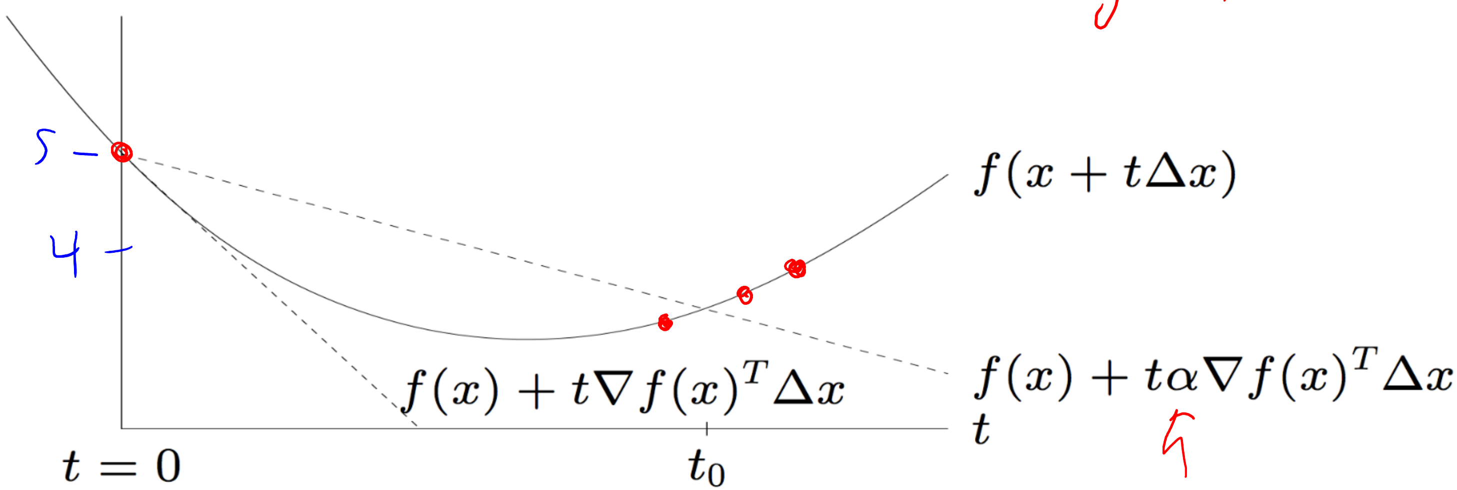
- We must have $f\left(\boldsymbol{\theta}^{(k)}\right) - p^* \leq \epsilon$ after at most $\frac{\log\left(\left(f\left(\boldsymbol{\theta}^{(0)}\right) - p^*\right)/\epsilon\right)}{\log(1/c)}$ iterations
- Linear convergence for strongly convex objective
  - $k \sim \log(\rho^{-1})$ // k = number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

Line search types

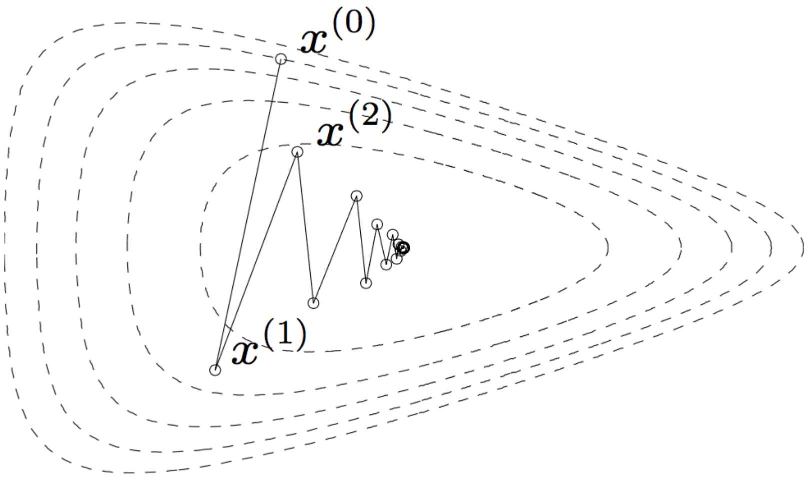
$f(\theta_{t+1}) < f(\theta_t)$

- Exact line search: $t = \arg\min_{t>0} f(\boldsymbol{x} + t \cdot \Delta\boldsymbol{x})$

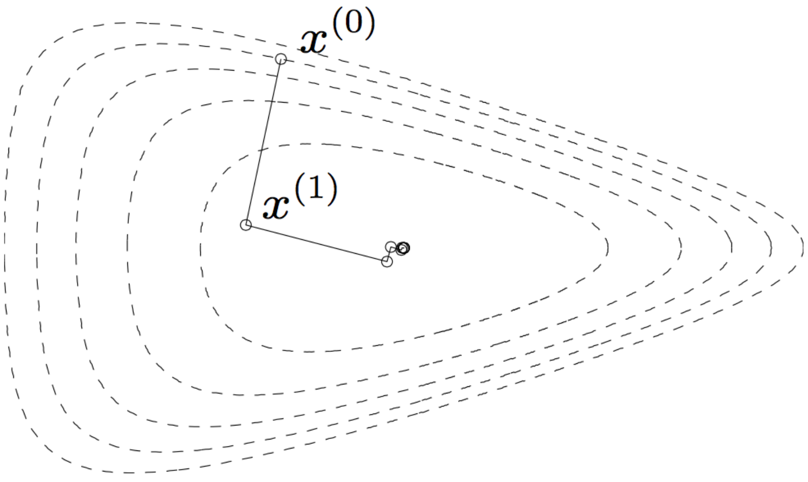
- Backtracking line search: (with parameters $\alpha \in (0, 1/2), \beta \in (0,1)$)
  - starting at $t = 1$, repeat $t \coloneqq \beta t$ until
    $$\underbrace{f(\boldsymbol{x} + t \cdot \Delta\boldsymbol{x})}_{h(t)} < \underbrace{f(\boldsymbol{x}) + t \cdot \alpha \cdot \nabla f(\boldsymbol{x})^T \Delta\boldsymbol{x}}_{g(t)}$$
  - graphical interpretation: backtrack until $t \leq t_0$



Backtracking vs. exact line search



backtracking line search



exact line search

from Boyd & Vandenberghe

Distributed/Parallel implementation

- Often problems are of the form
  - $f(\boldsymbol{\theta}) = \sum_i L_i(\boldsymbol{\theta}) \; + \; g(\boldsymbol{\theta})$
  - where i iterates over, e.g., each data instance
- Example OLS regression: // with regularization
  - $L_i(\boldsymbol{w}) = \left(\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{T}} \, \boldsymbol{w} - y_i\right)^2$ $\qquad g(\boldsymbol{w}) = \lambda \cdot \|w\|_2^2$

- Gradient can simply be decomposed based on the sum rule
- Easy to parallelize/distribute

Basic steps (I)

given a starting point $\boldsymbol{\theta} \in \mathrm{Dom}(f)$.
repeat
1. $\boldsymbol{\Delta\theta} \coloneqq -\boldsymbol{\nabla} f(\boldsymbol{\theta})$
2. Line search. $t = argmin_{t>0}\, f(\boldsymbol{\theta} + t \cdot \boldsymbol{\Delta\theta})$
3. Update. $\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} + t\boldsymbol{\Delta\theta}$

until stopping criterion is satisfied.

easy parallel computation

- Distribute data over several machines
- Compute partial gradients (on each machine in parallel)
- Aggregate the partial gradients to the final one
- Communicate the final gradient back to all machines

Basic Steps (II)

given a starting point $\boldsymbol{\theta} \in \mathrm{Dom}(f)$.
repeat
1. $\boldsymbol{\Delta\theta} \coloneqq -\boldsymbol{\nabla} f(\boldsymbol{\theta})$
2. Choose t via exact or backtracking line search.
3. Update. $\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} + t\boldsymbol{\Delta\theta}$

until stopping criterion is satisfied.

update value in search direction and feed back (might be done multiple times: expensive!)

communicate final step size to each machine

- 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
$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \tau \cdot \boldsymbol{\nabla} f(\boldsymbol{\theta}_t)$$
  – τ is often called the learning rate

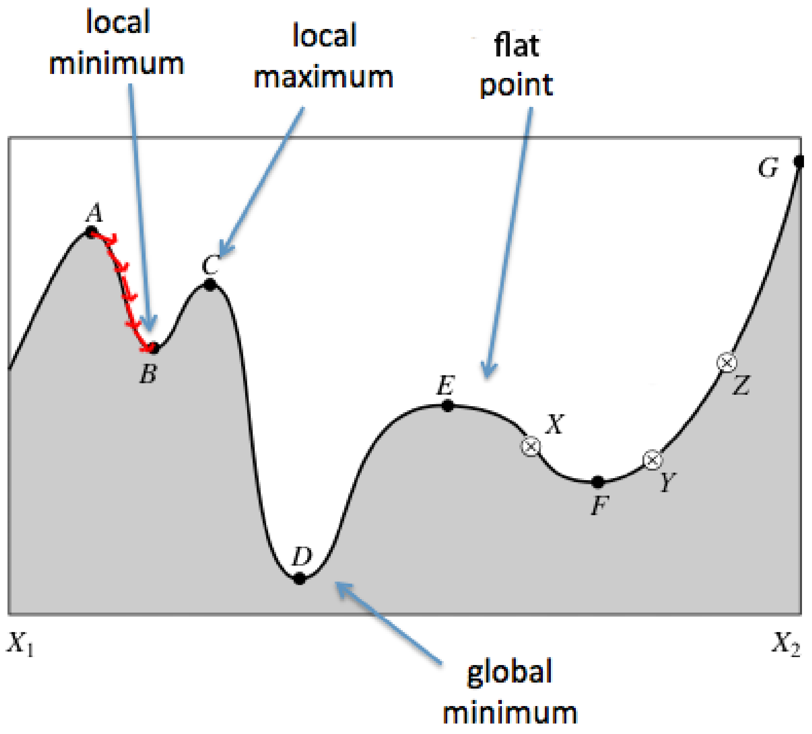
- Only single pass through data per iteration
- Logarithmic iteration bound (as before)
  – if learning rate is chosen "correctly"

- How to pick the learning rate?
  – too small: slow convergence
  – too high: algorithm might oscillate, no convergence

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

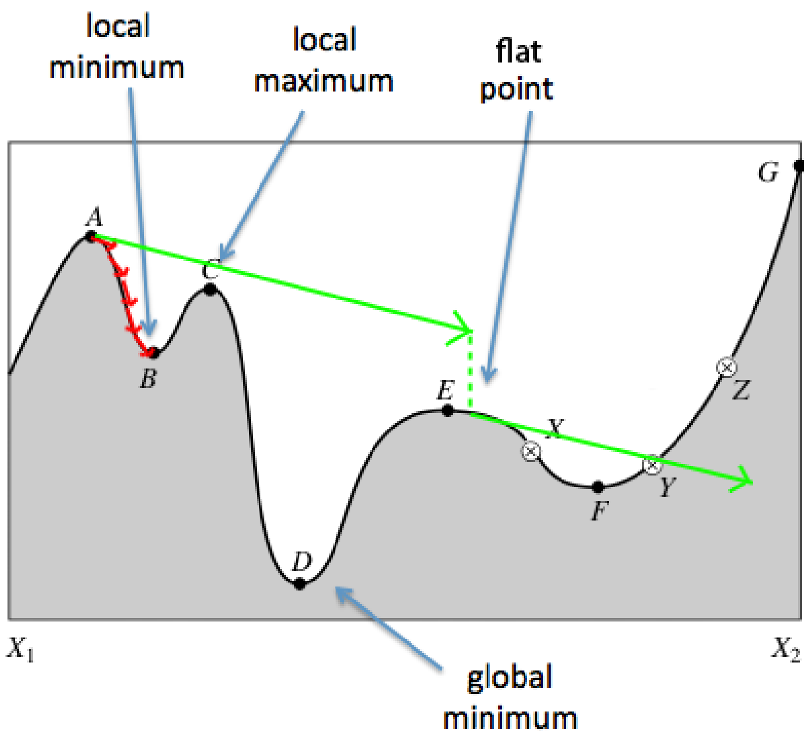
The value of τ

- 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 steps to overshoot



Learning rate adaptation

- 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 $\lim_{t \to \infty} \tau_t = 0$
  - example: $\tau_{t+1} \leftarrow \alpha \cdot \tau_t$ for $0 < \alpha < 1$

Learning rate adaptation

- Other solutions: Incorporate "history" of previous gradients
- Momentum:
  - $\boldsymbol{m}_t \leftarrow \tau \cdot \boldsymbol{\nabla} f(\boldsymbol{\theta}_t) + \gamma \cdot \boldsymbol{m}_{t-1}$ // often $\gamma = 0.5$
  - $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \boldsymbol{m}_t$
  - As long as gradients point to the same direction, the search accelerates
- 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

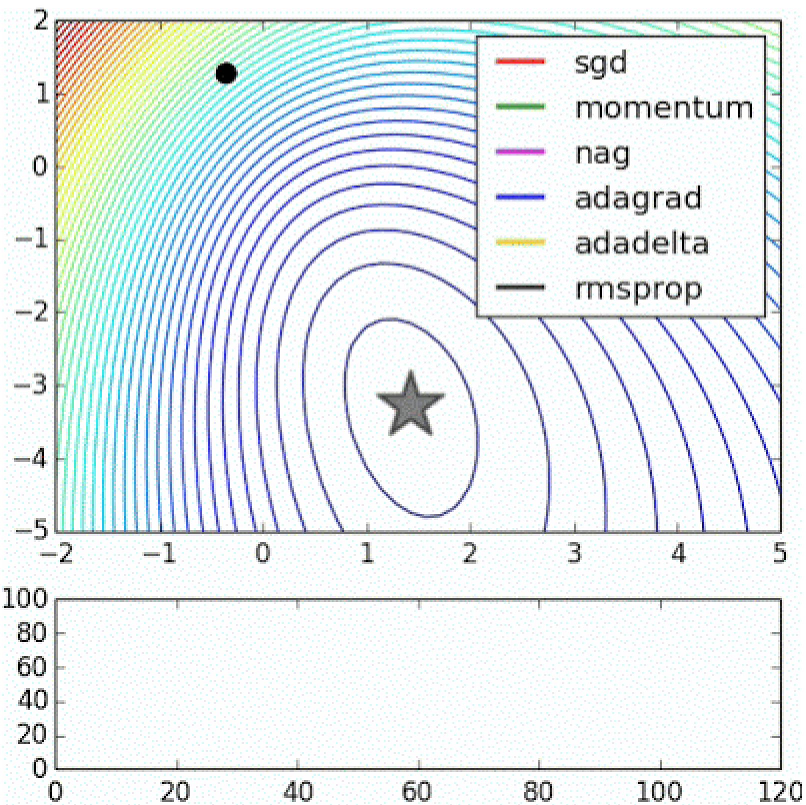
Adaptive moment estimation (Adam)

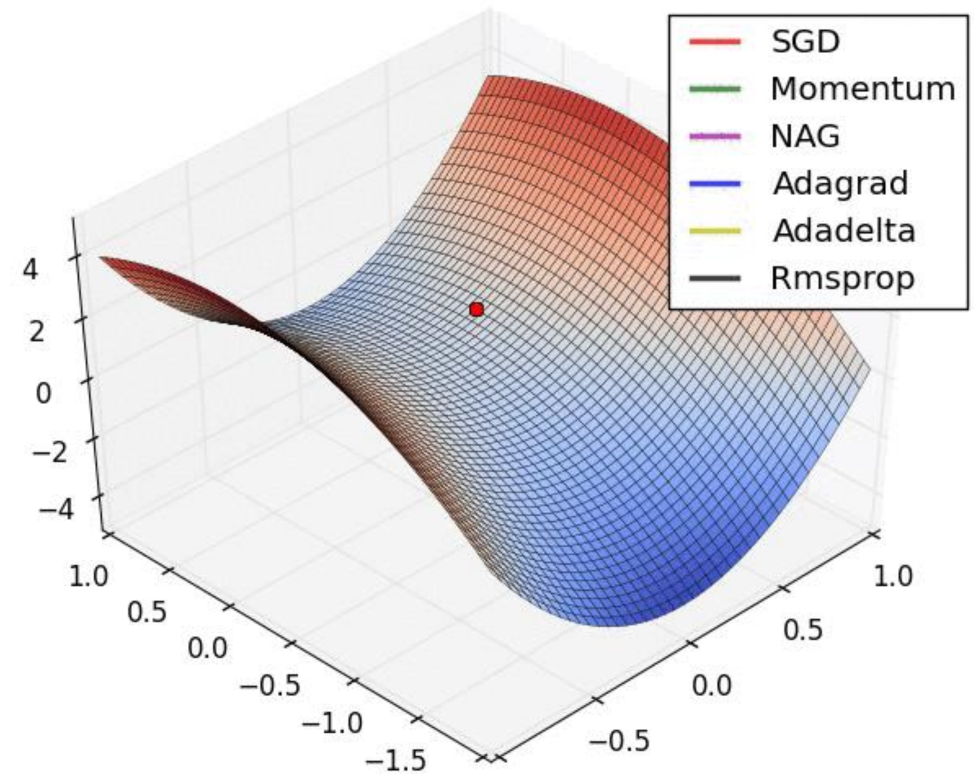
- $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \nabla f(\boldsymbol{\theta}_t)$
  - estimate of the first moment (mean) of the gradient
  - Exponentially decaying average of past gradients m_t (similar to momentum)
- $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) (\nabla f(\boldsymbol{\theta}_t))^2$
  - 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:
  - $\widehat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}$ $\quad\quad$ $\widehat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t}$
- Finally, the Adam update rule for parameters θ:
  - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\tau}{\sqrt{\widehat{\mathbf{v}}_t} + \epsilon} \widehat{\mathbf{m}}_t$
- 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/



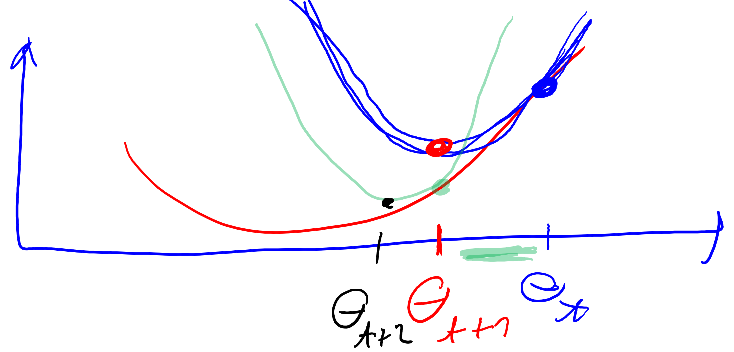


Discussion

- 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



- Convex objective function f
- Nonnegative second derivative: $\boldsymbol{\nabla}^2 f(\boldsymbol{\theta}) \succcurlyeq 0$ // Hessian matrix
- Taylor expansion of f at point θ_t

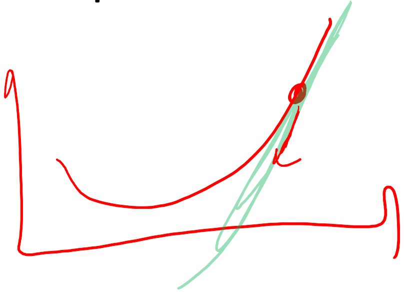
$$\underbrace{f(\boldsymbol{\theta}_t + \boldsymbol{\delta})}_{h(\delta)} = f(\boldsymbol{\theta}_t) + \boldsymbol{\delta}^T \cdot \boldsymbol{\nabla} f(\boldsymbol{\theta}_t) + \frac{1}{2}\boldsymbol{\delta}^T \boldsymbol{\nabla}^2 f(\boldsymbol{\theta}_t)\boldsymbol{\delta} + O(\boldsymbol{\delta}^3)$$

$h(\delta) \approx h'(\delta) \leftarrow$ Approx

- Minimize approximation: leads to

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

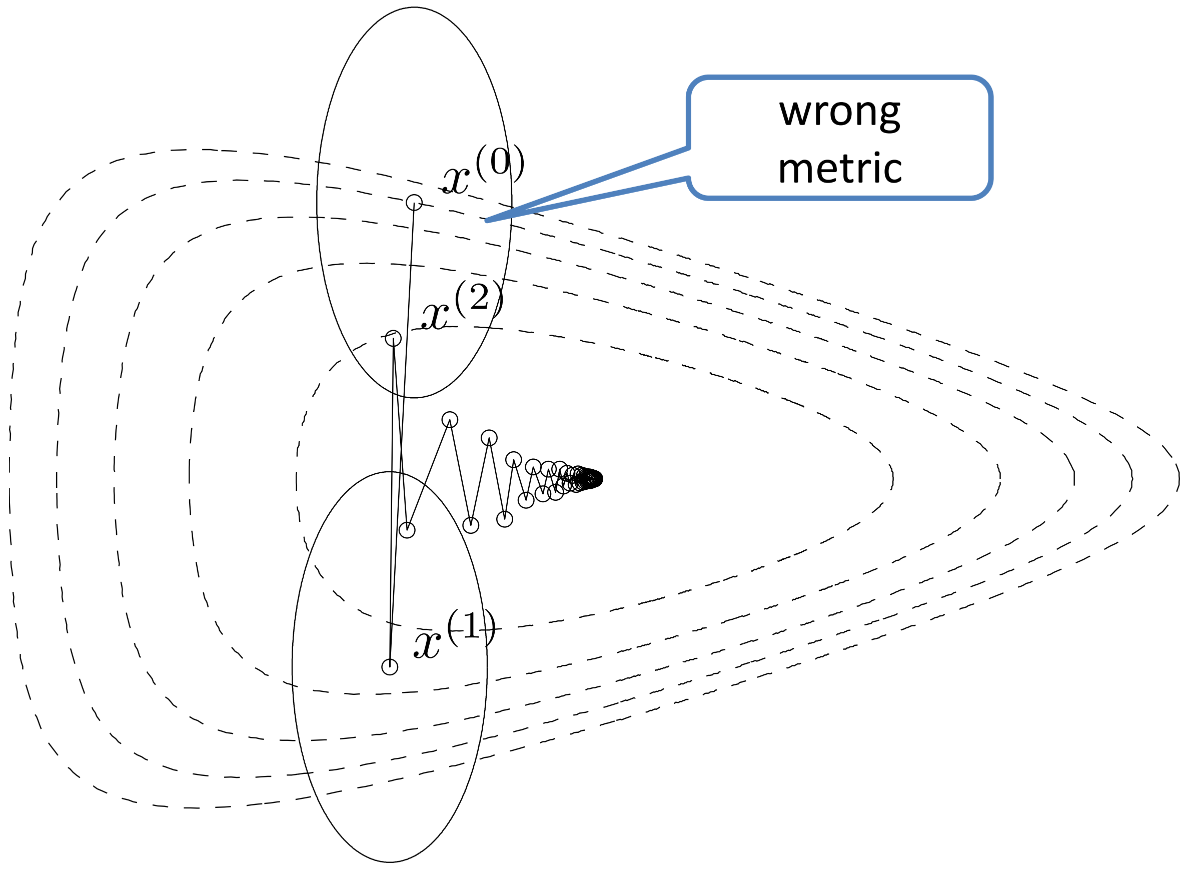
- Repeat until convergence



$\nabla h'(\delta) \overset{!}{=} 0 \iff \delta =$

Rescaling of space

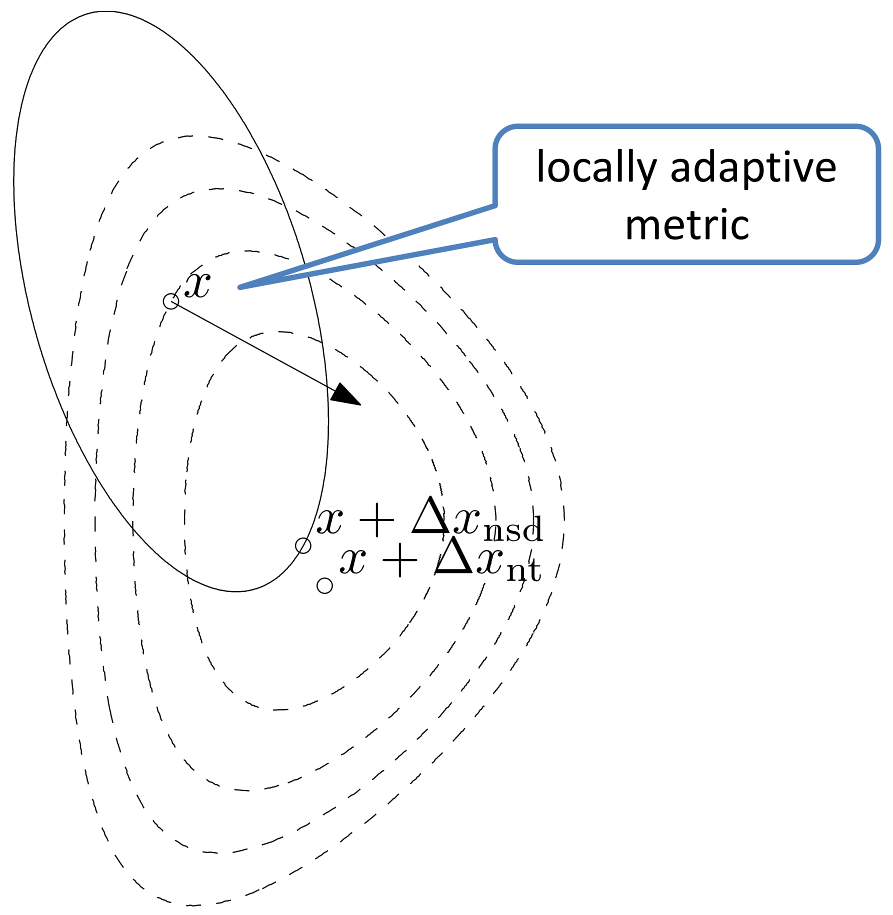
- Newton method rescales the space



from Boyd & Vandenberghe

Rescaling of space

- Newton method rescales the space



Parallel Newton method

+ Good rate of 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

$f(\theta) = \sum_{i=1}^{N} L_i(\theta)$

- 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!

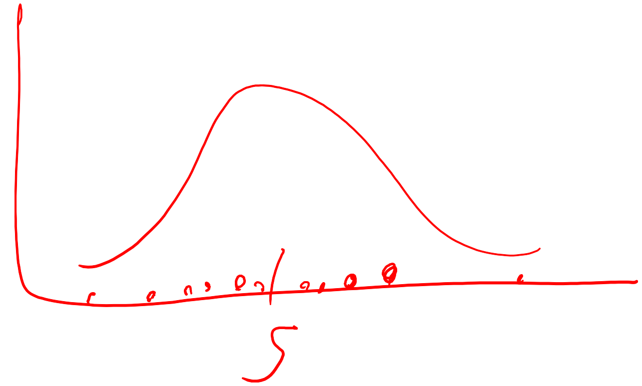
Motivation: Stochastic Gradient Descent

- Goal: minimize $f(\boldsymbol{\theta}) = \sum_{i=1}^{n} L_i(\boldsymbol{\theta})$ + 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:
- $\mathbb{E}_{i\sim\{1,\dots,n\}}[L_i(\boldsymbol{\theta})] \approx \frac{1}{|S|}\sum_{j\in S}\left(L_j(\boldsymbol{\theta})\right)$ // with $S \subseteq \{1,\dots,n\}$

  or equivalently: $\sum_{i=1}^{n} L_i(\boldsymbol{\theta}) \approx \frac{n}{|S|}\sum_{j\in S} L_j(\boldsymbol{\theta})$

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: $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \tau \cdot \frac{n}{|S|} \cdot \sum_{j \in S} \boldsymbol{\nabla} L_j(\boldsymbol{\theta}_t)$
  4. pick a new subset and repeat with 2

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

Example: Perceptron

$w_{t+1} \leftarrow w_t - 0$

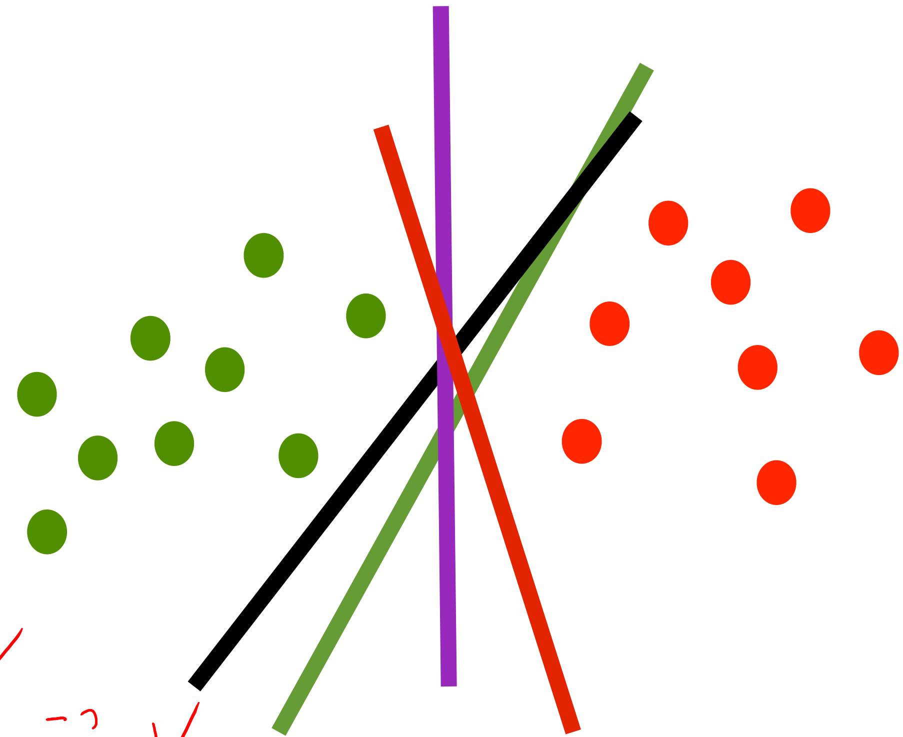
- Simple linear binary classifier:

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

- Learning task:

  Given $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ find

  $$\min_{\mathbf{w},\mathrm{b}} \sum_i \mathrm{L}(y_i, \boldsymbol{w}^T\boldsymbol{x}_i + b)$$



+1 +1 ✓

-1 -1 ✓

- L is the loss function

  – e.g. $L(u, v) = \max(0, -u \cdot v) = \begin{cases} -uv & if\ uv < 0 \\ 0 & else \end{cases}$ ← *incorrect classification* ← *correct classification*

$-(y_i \cdot (w^T x_i + b)) =$

Example: Perceptron

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

initialize $\boldsymbol{w} = \mathbf{0}$ and $b = 0$
repeat
  if $\mathrm{y}_i \cdot (\boldsymbol{w}^T \boldsymbol{x}_i + b) \leq 0$ **then**
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \tau \cdot n \cdot y_i \cdot \boldsymbol{x}_i$ and $b \leftarrow b + \tau \cdot n \cdot \ y_i$
  end if
until all classified correctly

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

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

Optimizing Logistic Regression

- Recall we wanted to solve $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\arg\min}\, E(\boldsymbol{w})$
- $E(\boldsymbol{w}) = -\ln p(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X})$
  $$= -\sum_{\{i=1\}}^{N} y_i \ln \sigma(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x}_{\boldsymbol{i}}) + (1 - y_i) \ln(1 - \sigma(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x}_i))$$
- Closed form solution does not exist
- Solution:
  - Computed the gradient $\boldsymbol{\nabla} E(\boldsymbol{w})$
  - Find $\boldsymbol{w}^*$ using gradient descent
- Is $E(\boldsymbol{w})$ convex?
- Can you use SGD?
- How can you choose the learning rate?
- What changes if we add regularization, i.e. $E_{reg}(\boldsymbol{w}) = E(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2$?

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

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

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

Summary

- General task: Find solution $\boldsymbol{\theta}^*$ minimizing function f
- Convex sets & functions
  - Global vs. local minimum
  - Convex hull $\mathrm{Conv}(X)$
  - Verifying convexity: Definition, special results (first-order convexity, 2nd derivative), convexity-preserving operations
- Gradient descent: $\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} - t\,\boldsymbol{\nabla} f(\boldsymbol{\theta})$
  - Line search: How to choose t? E.g. backtracking, exact
  - Learning rate: Fix $t = \tau$, change via learning rate adaptation (momentum, AdaGrad)
  - Faster methods: Adam, Newton method
  - Stochastic gradient descent (SGD): Only use part of data (mini-batches) at each step