SME3006 Machine Learning – 2025 Fall

# Optimization and gradient descent


INHA UNIVERSITY

# Overview

- Optimization
  - gradient
  - hessian
  - constraint
  - newton's method
- Convex
  - LP, QP
- Gradient descent
  - momentum
  - 2nd order model
  - stochastic optimization

# Optimization problem

> Finding the minimizer of a function subject to constraints:

$$\underset{x}{\text{minimize}}\, J(x)$$

objective function

$$s.t.\, f_i(x) \leq 0,\ \ i = \{1, \dots, k\}$$
$$h_j(x) = 0,\ \ \ j = \{1, \dots, l\}$$

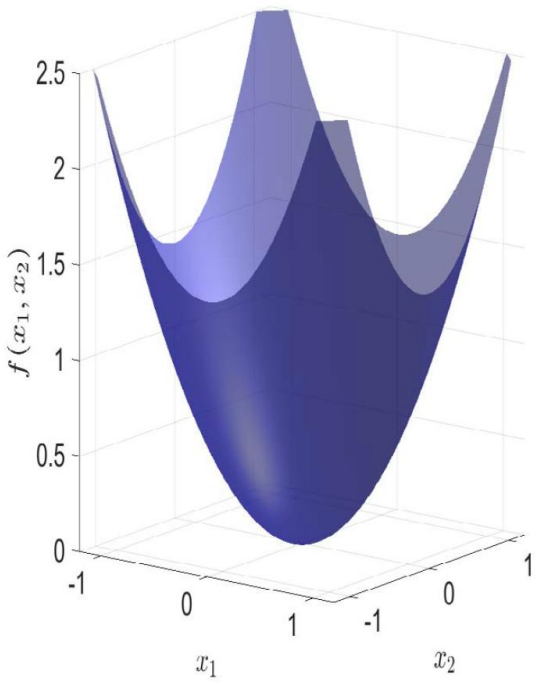
constraints

- Linear regression: $\underset{w}{\text{minimize}} \|Xw - y\|^2$
- Classification (logistic regression): $\underset{w}{\text{minimize}} \sum_{i=1}^{n} \log(1 + \exp(-y_i x_i^\top w))$
- Maximum likelihood estimation: $\underset{\theta}{\text{maximize}} \sum_{i=1}^{n} log p_\theta(x_i)$
- K-means: $\underset{w}{\text{minimize}}\, J(\mu) = \sum_{j=1}^{k} \sum_i \| x_i - \mu_j \|^2$
- …

# Optimization problem

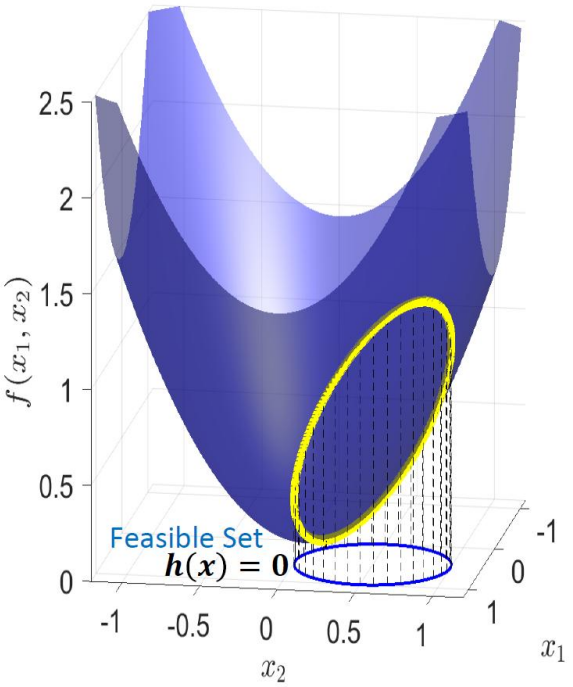
- Constraints = defining feasible sets

**Unconstrained Optimization**




$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x)$$

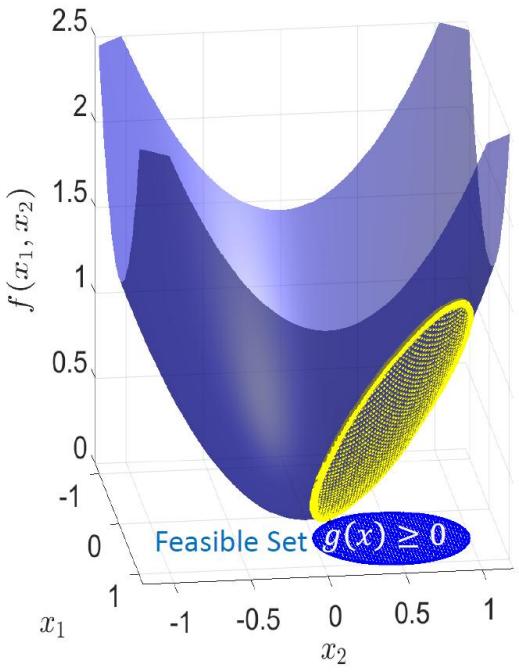
**Optimization with equality Constraints**




$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x)$$

$$\text{subject to} \quad h_i(x) = 0, \; i = 1, .., n_h$$

**Optimization with Inequality Constraints**

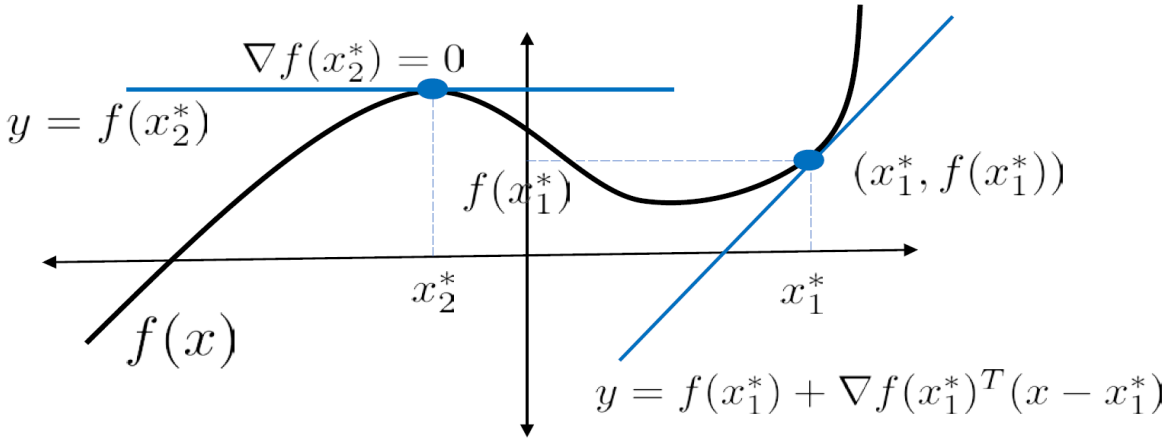



$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x)$$

$$\text{subject to} \quad g_i(x) \geq 0, \; i = 1, ..., n_g$$

# Gradient

- Real-valued scalar function $f(x)$: $\mathbb{R}^n \rightarrow \mathbb{R}$
- Gradient vector: $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \cdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$
- Tangent line (hyperplane) at point $(x^*, f(x^*))$
  - $y = f(x^*) + \nabla f(x^*)^{\mathsf{T}}(x - x^*)$
  - $\nabla f(x^*) = 0 \rightarrow x^*$ : maximum or minimum



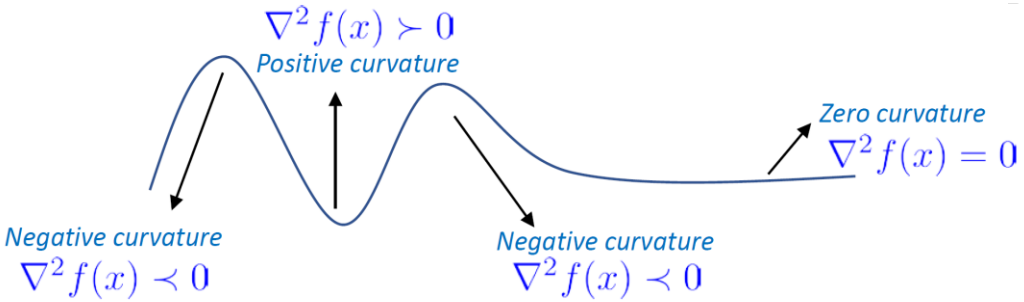
# Hessian

> $f(x) \rightarrow \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \cdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \rightarrow \nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$

Gradient vector

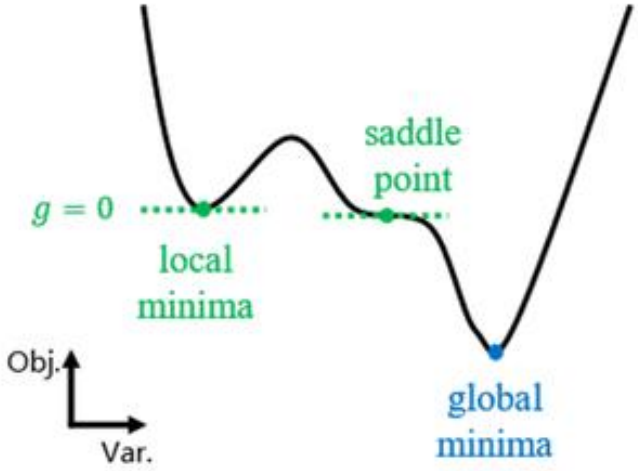
Hessian matrix

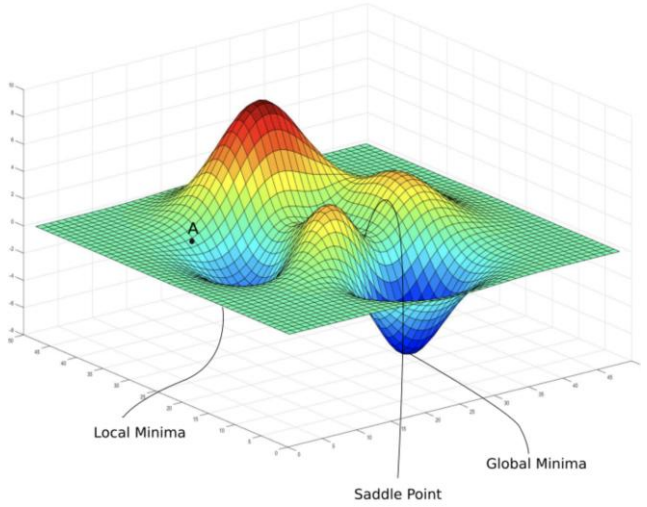
> Hessian matrix describes the local curvature of the function



# Optimality conditions

- Unconstrained optimization
  - $\underset{x}{\text{minimize}}\, f(x)$
  - What are the conditions for $x$ to be a minimum point?
  - $\nabla f(x^*) = 0$
  - ~~$\nabla^2 f(x^*) \geq 0$~~
  - $\nabla^2 f(x^*) > 0$



# Optimality conditions

> Unconstrained optimization
  - $\underset{x}{\text{minimize}}\, f(x)$

  - Necessary condition: $\nabla f(x) = 0, \quad \nabla^2 f(x^*) \geq 0$
  - Sufficient condition: $\nabla f(x) = 0, \quad \nabla^2 f(x^*) > 0$

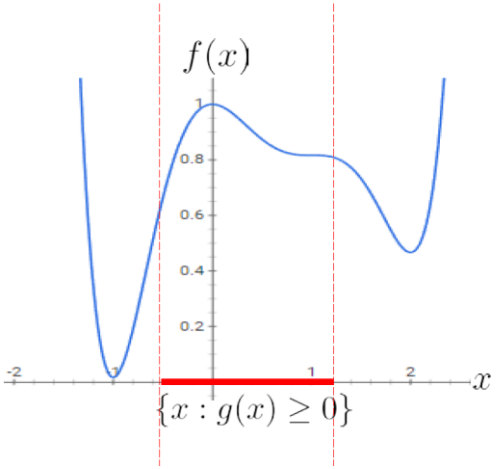
  - Find the roots of $\nabla f(x)$
  - If obtained root satisfies $\nabla^2 f(x^*) > 0$, point $x^*$ is local/global minimum

# Optimality conditions

> Constrained optimization

- $\underset{x}{\text{minimize}}\, f(x)$

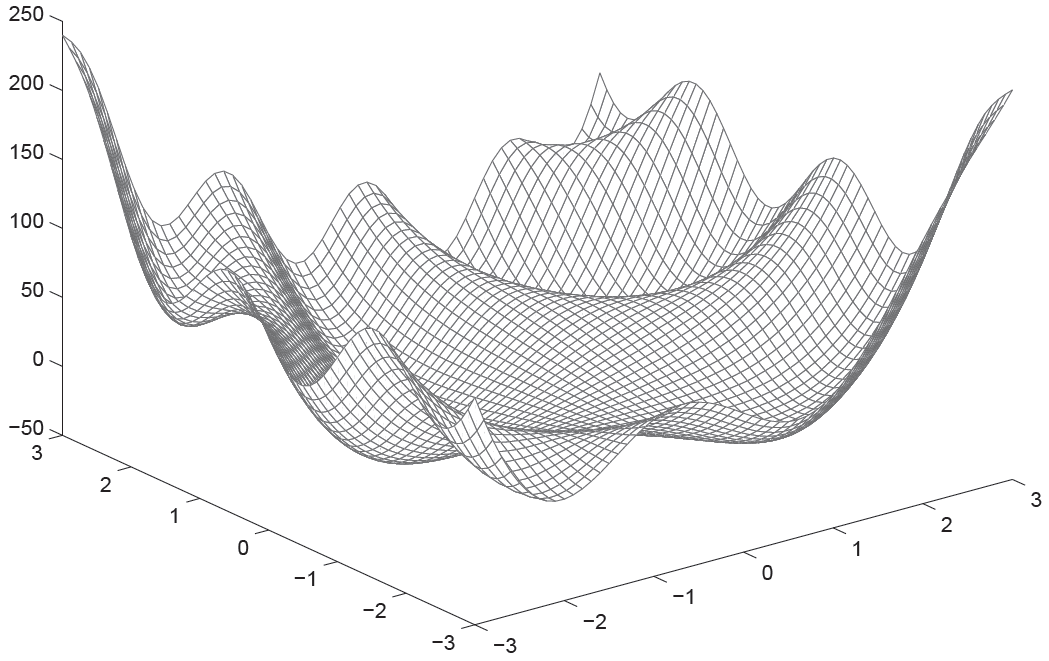
  $subject\ to\ \ g(x) \geq 0$




- Optimality conditions of unconstrained optimization are not valid for constrained optimization
- It will lead to KKT (Karush-Kuhn-Tucker) necessary optimality condition (graduate level)

# Optimality conditions

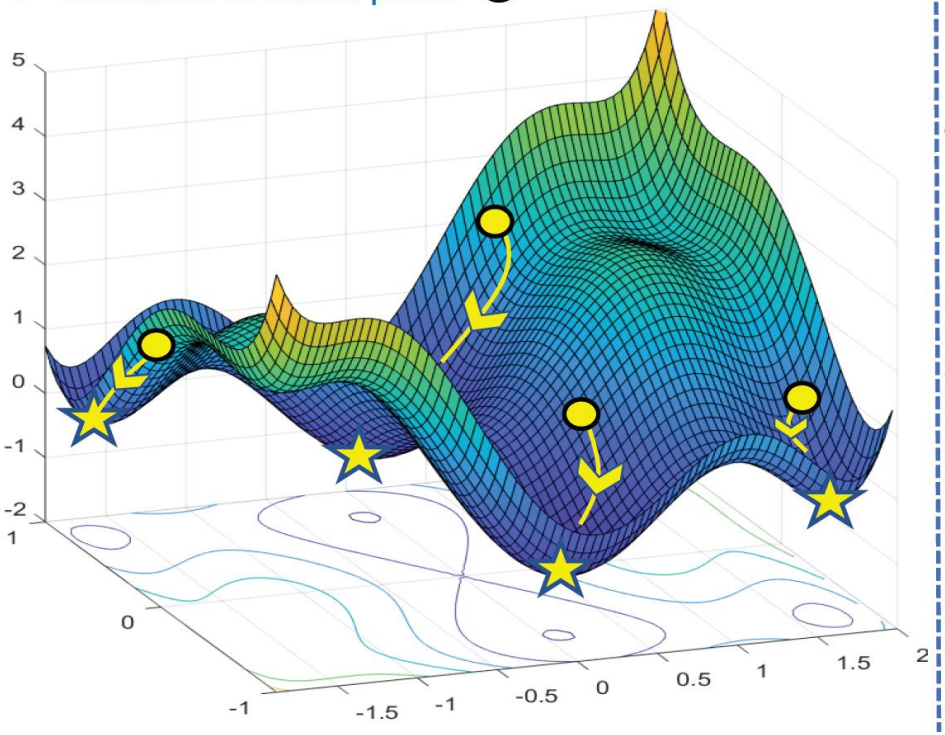
- Even in an unconstrained problem, we don't know whether the minimum point is local or global
- How can we find a global minimum?



# Optimality conditions

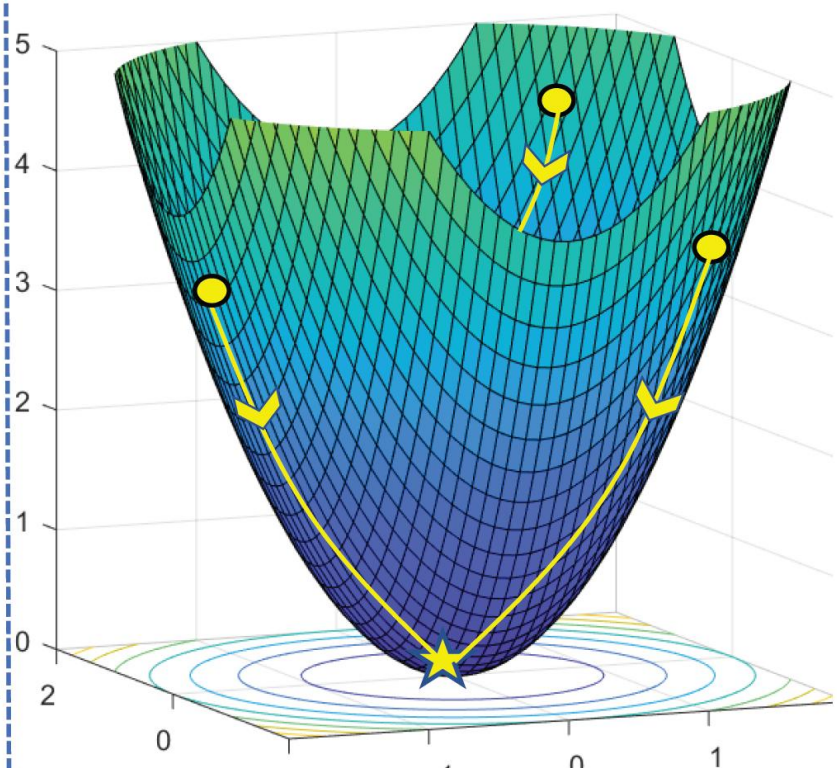
## Nonconvex Optimization

- Multiple local minima
- Sensitive to initial point




## Convex Optimization

- Unique minimum: global/local

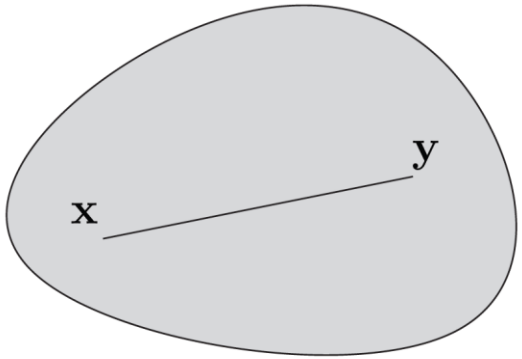


# Convex sets

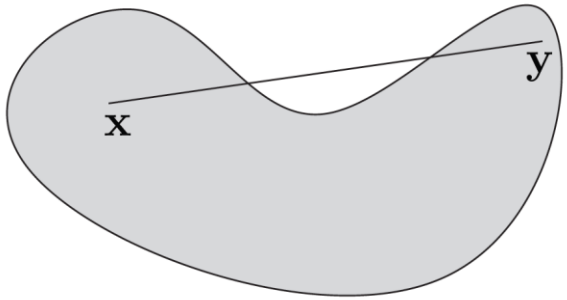
- Definition:

A set $C \subseteq \mathbb{R}^n$ is convex if for $x, y \in C$ and any $\alpha \in [0,1]$,

$$\alpha x + (1-\alpha)y \in C$$



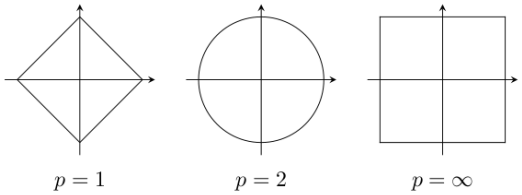

convex

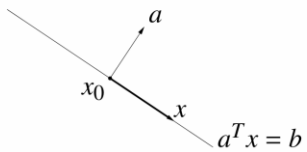


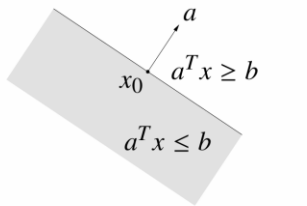

non-convex

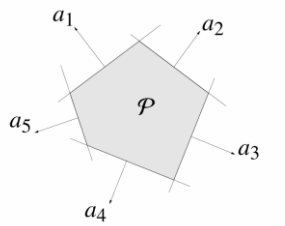
# Convex sets

- Examples
  - norm ball $\{x: \|x\| \leq r\}$
  - hyperplane $\{x: a^\top x = b\}$
  - halfspace $\{x: a^\top x \leq b\}$
  - affine space $\{x: Ax = b\}$
  - polyhedron $\{x: Ax \leq b\}$



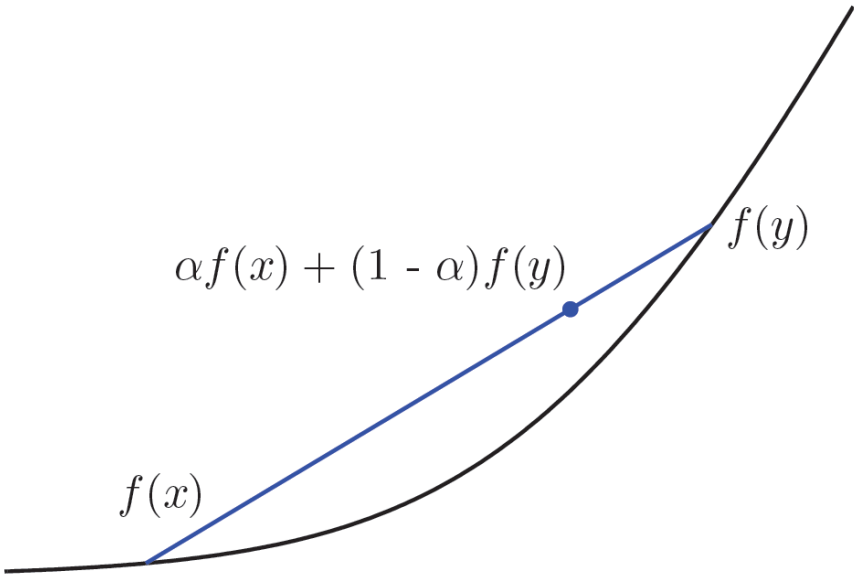

- Arbitrary intersection of convex sets is a convex set

# Convex function

> Definition:

A function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is convex if for $x, y \in dom\ f$ and any $\alpha \in [0,1]$,

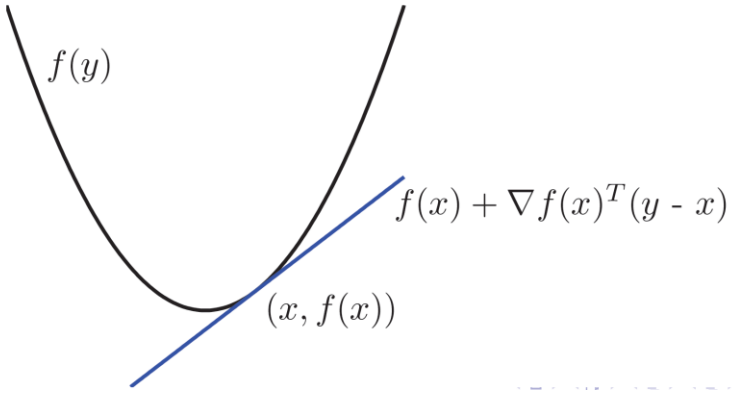
$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$$



# Convex function

- First order convexity condition:
  - Suppose $f: \mathbb{R}^n \to \mathbb{R}$ is differentiable
    $f$ is convex iff for all $x, y \in dom\ f$

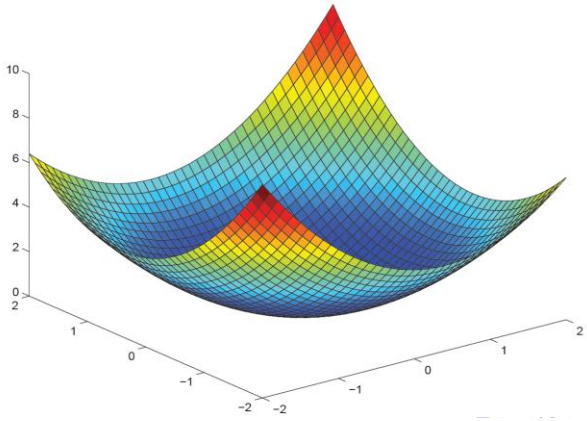
$$f(y) \geq f(x) + \nabla f(x)^{\mathsf{T}}(y - x)$$




- Second order convexity condition:
  - Suppose $f: \mathbb{R}^n \to \mathbb{R}$ is twice differentiable
    $f$ is convex iff for all $x \in dom\ f$

$$\nabla^2 f(x) \geq 0$$

$$\nabla^2 f(x) \succeq 0.$$

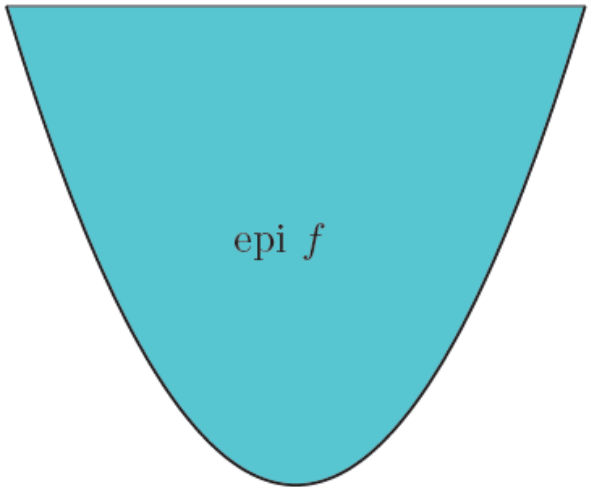


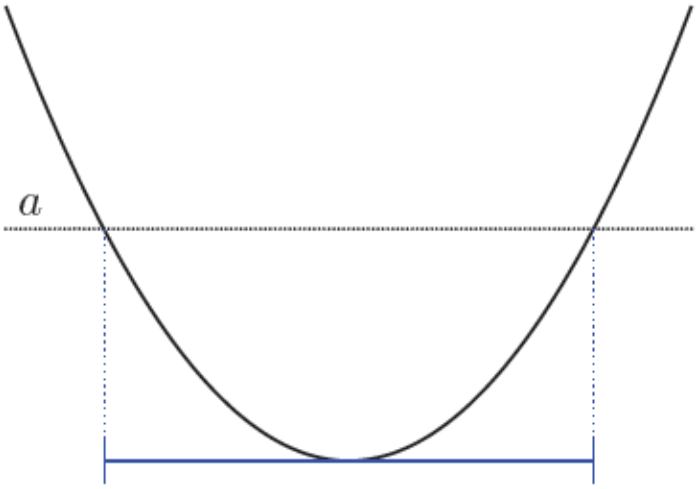
# Convex sets and convex functions

- Epigraph of a function $f$ is the set of points

$$\text{epi } f = \{(x, t): f(x) \leq t\}$$

- epi $f$ is convex if and only if f is convex
- Sublevel sets, $\{x: f(x) \leq a\}$ are convex for convex $f$



# Convex optimization

- An optimization problem is convex
  - if its objective is a convex function
  - inequality constraints $f_i(x)$ are convex
  - equality constraints $h_j(x)$ are affine ($h_j(x) = b^\top x + c$)

$$\underset{x}{\text{minimize}}\, J(x)$$

convex function

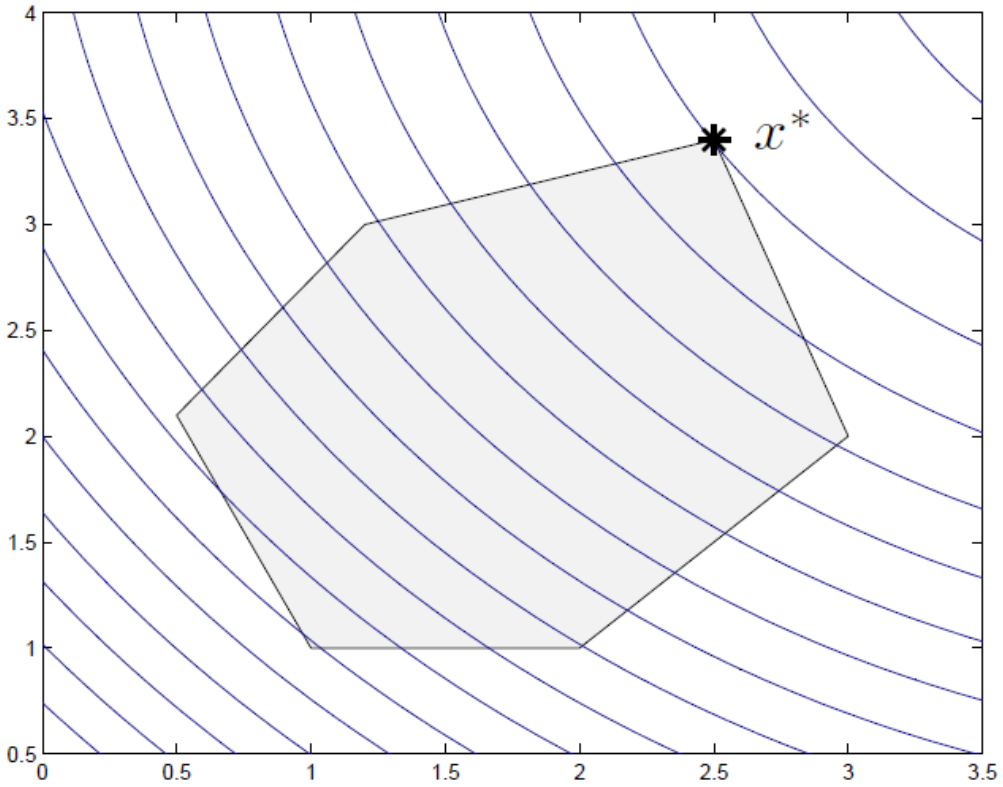
$$s.t.\, f_i(x) \leq 0,\ \ i = \{1, \dots, k\}$$
$$h_j(x) = 0,\ \ j = \{1, \dots, l\}$$

convex sets

affine

# Convex optimization

- If $x^*$ is a local minimizer of a convex optimization problem, then it is a global minimizer
- $\nabla f(x) = 0$ if and only if $x$ is a global minimizer of $f(x)$



# Convex optimization

- If $x^*$ is a local minimizer of a convex optimization problem, then it is a global minimizer
- $\nabla f(x) = 0$ if and only if $x$ is a global minimizer of $f(x)$
  - Proof.
  - $\nabla f(x) = 0$
    - we have $f(y) \geq f(x) + \nabla f(x)^{\mathsf{T}}(y - x) = f(x)$
  - $\nabla f(x) \neq 0$
    - There is a direction of descent

# Class of convex optimizations

- Linear program (LP)
  - $\underset{x}{\text{minimize}}\, c^\top x$

    $s.t. x \geq 0,$
    $Ax = b$
- Quadratic program (QP)
  - $\underset{x}{\text{minimize}} \frac{1}{2} x^\top P_0 x + q_0^\top x + c_0$

    $s.t. Ax = b$
    $\mathrm{x} \geq 0$
- Second-order cone program (SOCP)
- Semidefinite program (SDP)
- Cone program
- …

# Gradient descent

- The simplest optimization algorithm

$$\text{Optimization goal: } \underset{x}{\text{minimize}}\, J(x)$$

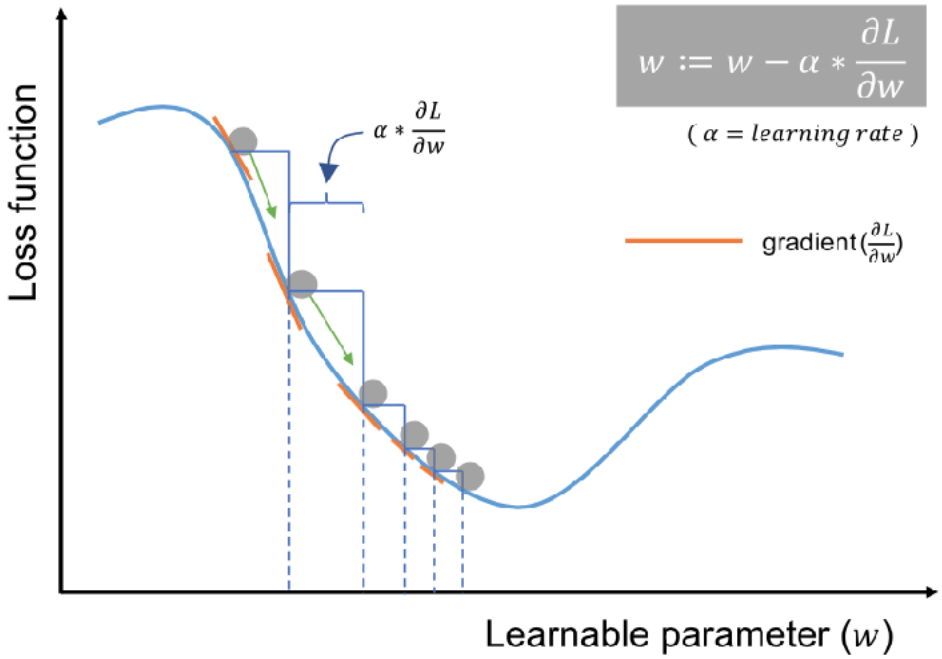
- Just iterate

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t)$$

where $\alpha_t$ is step size (or learning rate)

# Gradient descent

- $x_{t+1} = x_t - \alpha_t \nabla f(x_t)$
- Gradient descent is steepest descent method
  - direction $\nabla f(x)$ gives greatest reduction in $f(x_t)$ per unit change in $x$
  - learning rate determines how far we move in that direction



# Gradient descent

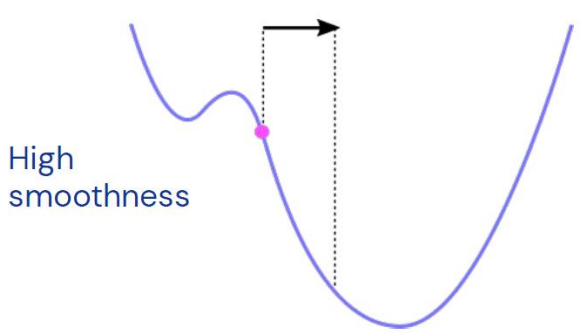
- 1st order Tayler series for $f(x)$ around current $x$ is

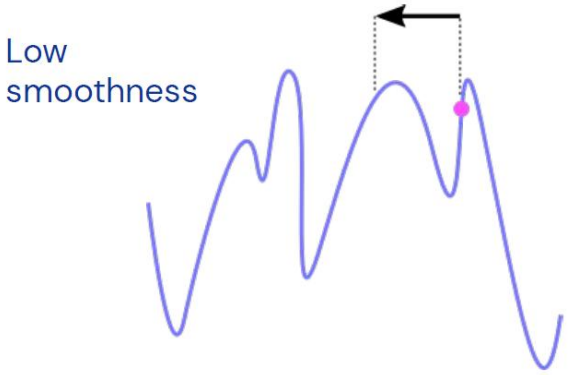
$$f(x+d) \approx f(x) + \nabla f(x)^{\mathsf{T}} d$$

  - for small enough $d$, this will be a reasonable approximation
  - GD update computed by minimizing this within a sphere of radius $r$

$$-\alpha_t \nabla f(x_t) = \arg \min_{d:\|d\| \le r} f(x) + \nabla f(x)^{\mathsf{T}} d$$

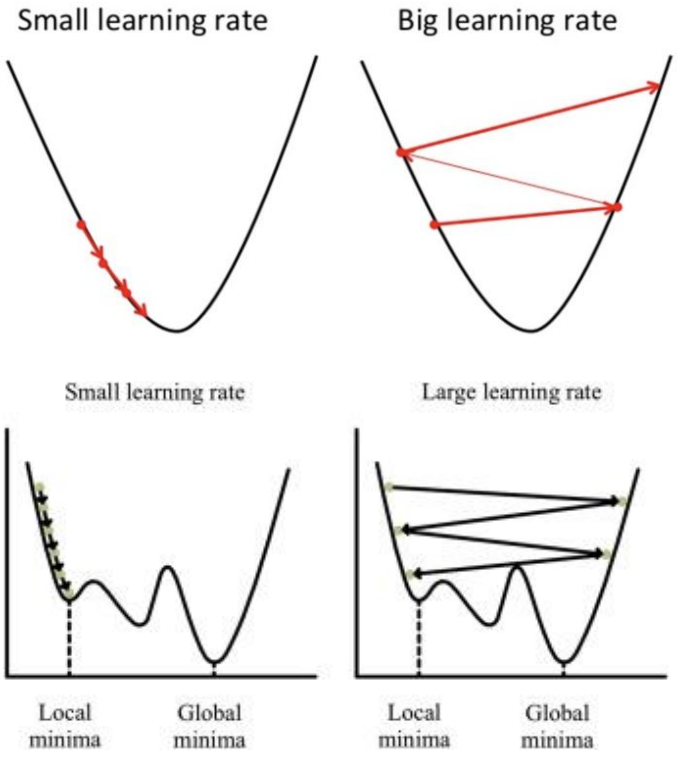
- If $f(x)$ is sufficiently smooth, and learning rate is small, gradient will keep pointing down-hill over the region

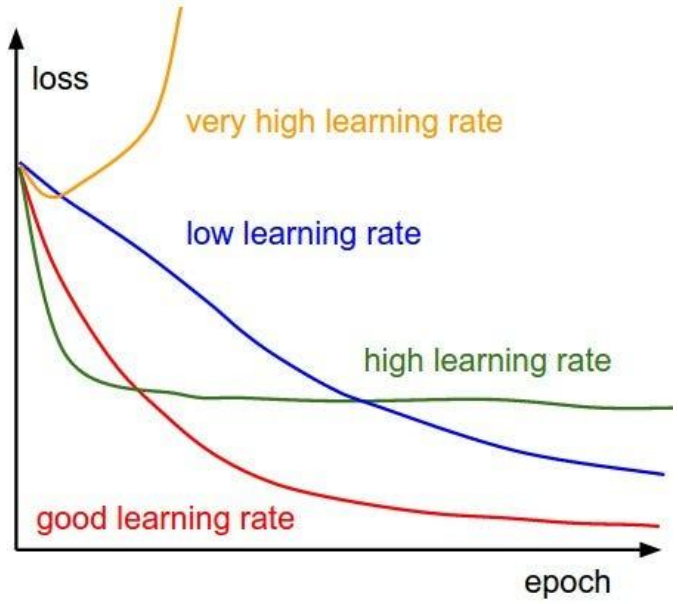


# Challenges

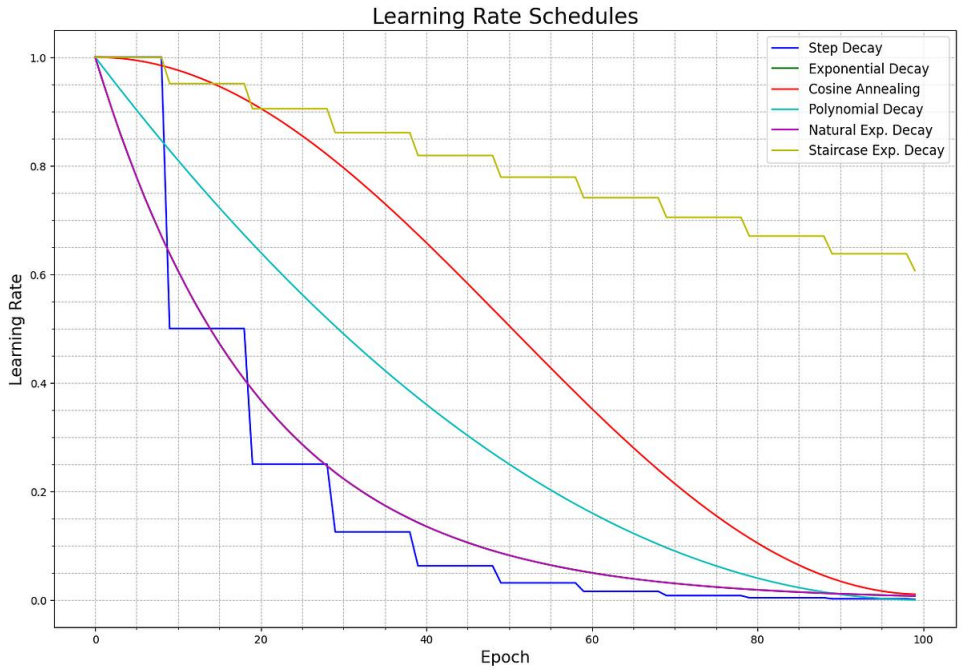
- Choosing a proper learning rate can be difficult
  - a small learning rate lead slow convergence, while a large learning rate hinder convergence
  - a small learning rate may get stuck in local minima



# Challenges

- Choosing a proper learning rate can be difficult
  - simple solution is scheduling the learning rate
  - $\alpha_t = \alpha_0 \beta^t, \beta \in [0,1]$
  - reducing the learning rate according to a predefined schedule (have to be defined in advance, thus unable to adapt to a dataset)



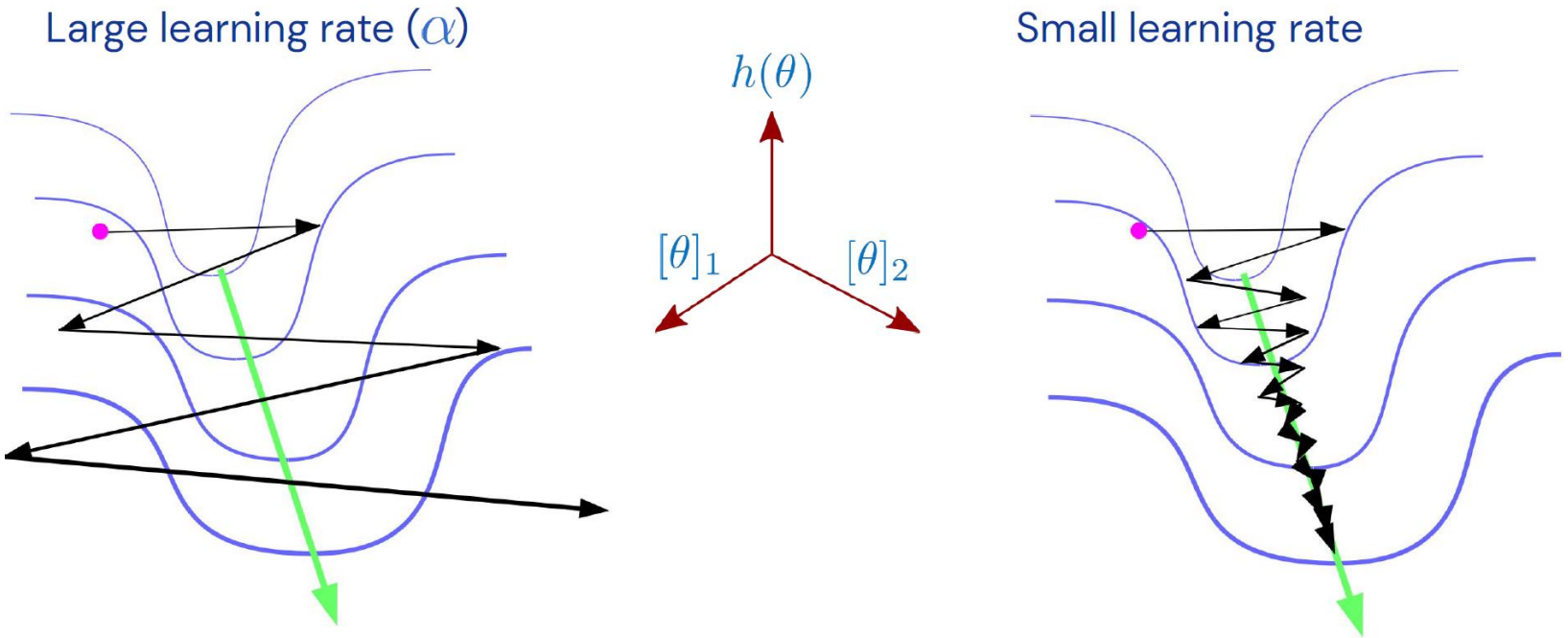
# Challenges

- Choosing a proper learning rate can be difficult
  - the same learning rate applies to all parameter updates
  - if our data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features

  - $x \leftarrow x - \alpha \nabla f(x) \quad \Rightarrow \quad x^i \leftarrow x^i - \alpha^i \nabla f\left(x^i\right)$

# Challenges

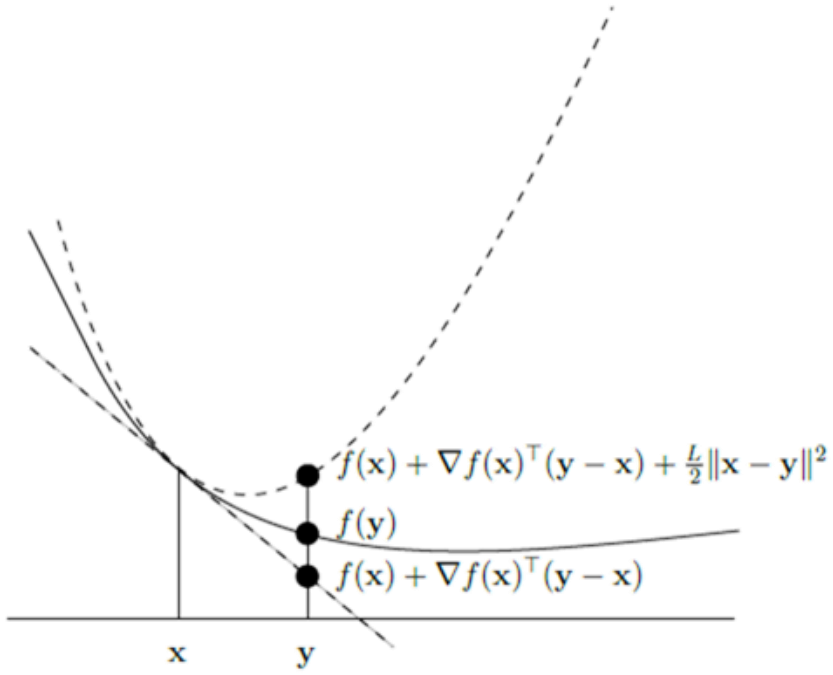
- Gradient direction is not optimal
  - Steepest gradient is not the most efficient way to the minimum
  - e.g., 2D narrow valley example



# Challenges

- Gradient direction is not optimal
  - GD minimizes the following primitive local 2nd-order approximation to $h(\theta)$
  - smooth convex function

$$f(y) \leq f(x) + \nabla f(x)^{\top}(y - x) + \frac{L}{2}\|x - y\|^2$$



# Challenges

- Gradient direction is not optimal
  - GD minimizes the following primitive local 2nd-order approximation to $h(\theta)$

$$f(\theta + d) \approx f(\theta) + \nabla f(\theta)^\top d + \frac{1}{2} d^\top F(\theta) d$$
$$\approx f(\theta) + \nabla f(\theta)^\top d + \frac{1}{2} d^\top L I d = f(\theta) + \nabla f(\theta)^\top d + \frac{L}{2} \|d\|^2$$

  - $\arg\min_d h(\theta) + \nabla h(\theta)^\top d + \frac{L}{2} \|d\|^2 = -\frac{1}{L} \nabla f(\theta)$

  - LI is a very conservative / pessimistic approximation to $F(\theta)$ that treats all directions as having the same (very high) curvature
    - this explains why it struggles on problems where curvature varies a lot

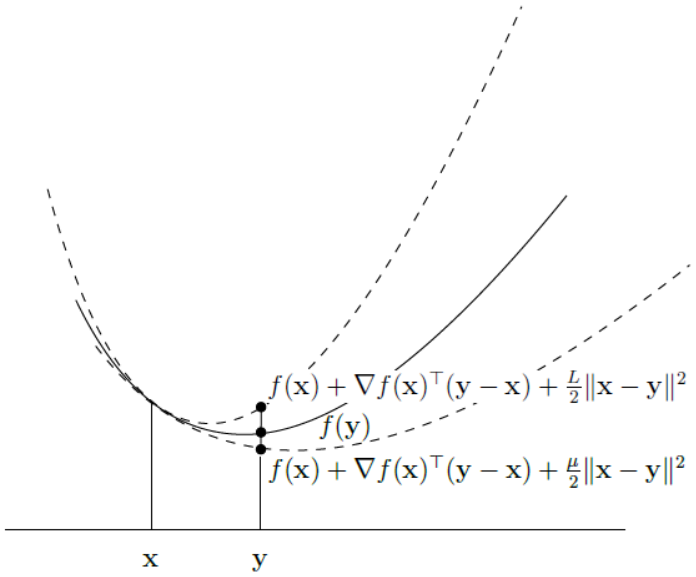
# Challenges

- Gradient direction is not optimal
  - GD minimizes the following primitive local 2nd-order approximation to $h(\theta)$
  - smooth convex function

$$f(y) \leq f(x) + \nabla f(x)^{\mathsf{T}}(y - x) + \frac{L}{2}\|x - y\|^2$$

  - strongly convex function

$$f(y) \geq f(x) + \nabla f(x)^{\mathsf{T}}(y - x) + \frac{\mu}{2}\|x - y\|^2$$



# Convergence theory

- $f(x)$ has Lipschitz continuous derivatives (Lipschitz smooth)
  - $\|\nabla f(x) - \nabla f(x')\| \leq L\|x - x'\|$ (upper bound on the curvature)

- $f(x)$ is strongly convex (perhaps only near minimum)
  - $f(x + d) \geq f(x) + \nabla f(x)^{\mathsf{T}} d + \frac{\mu}{2}\|d\|^2$ (lower bound con the curvature)

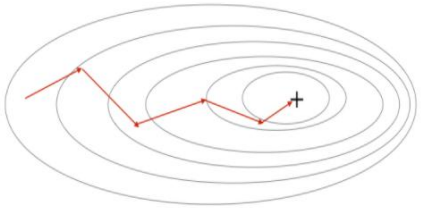
- We take $\alpha = \frac{2}{L+\mu}$, $\quad f(x_t) - f(x^*) \leq \frac{L}{2}\left(\frac{\kappa-1}{\kappa+1}\right)^{2t} \|x_0 - x^*\|^2$ where $\kappa = \frac{L}{\mu}$

- Number of iterations to achieve $f(x_t) - f(x^*) \leq \epsilon$ is

$$t \in \mathcal{O}\left(\kappa \log \frac{1}{\epsilon}\right)$$

# Momentum method

> Motivation: gradient has a tendency to flip back and forth as we take steps when the learning rate is large



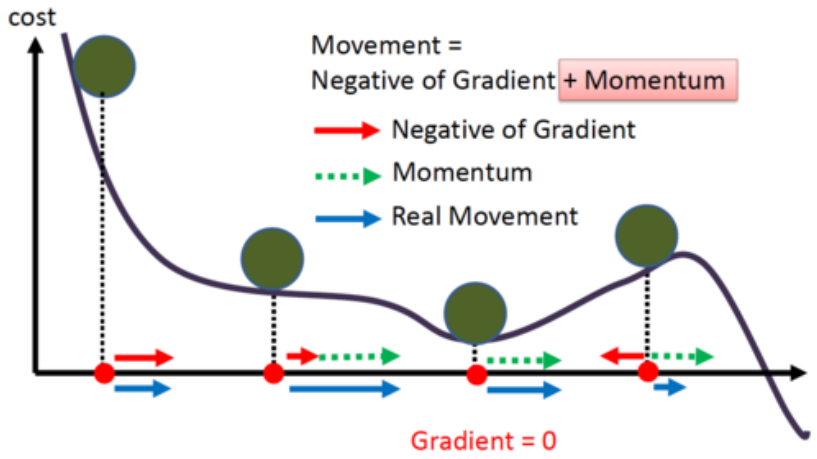
> Key idea: accelerate movement along directions that point consistently down-hill across many consecutive iterations

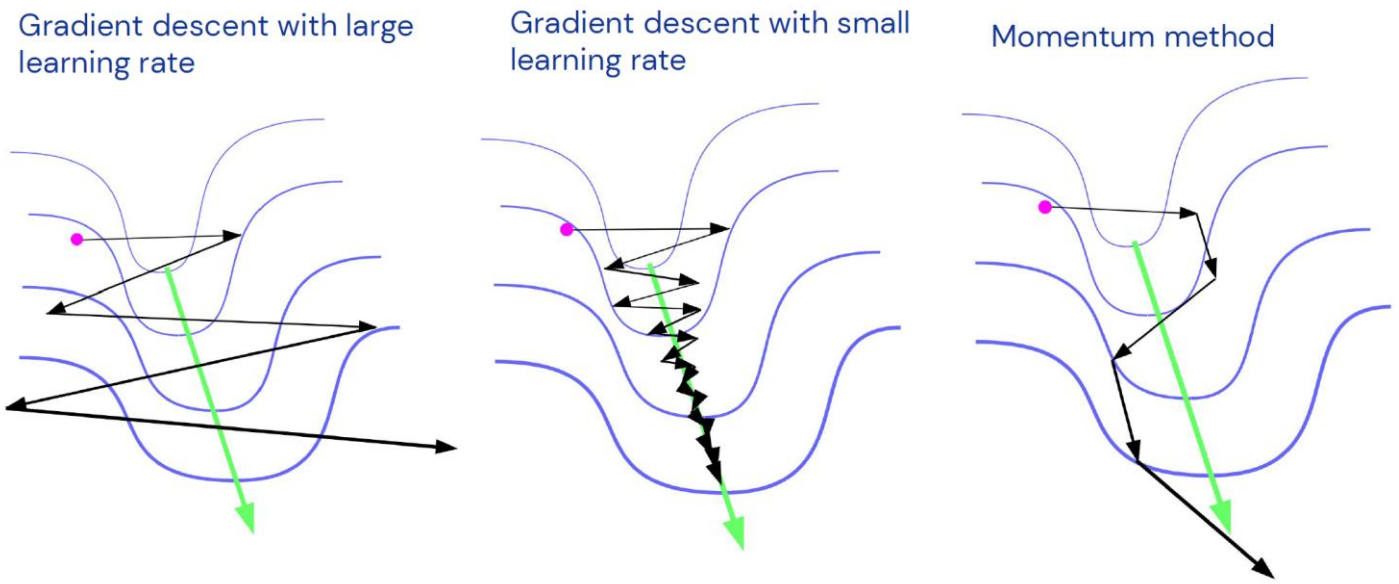
$$x_{t+1} = x_t + \alpha_t v_{t+1}$$

$$v_{t+1} = \gamma v_t - \nabla f(x_t), \quad v_0 = 0$$

$\gamma$ is momentum constant (usually 0.9)

# Momentum method

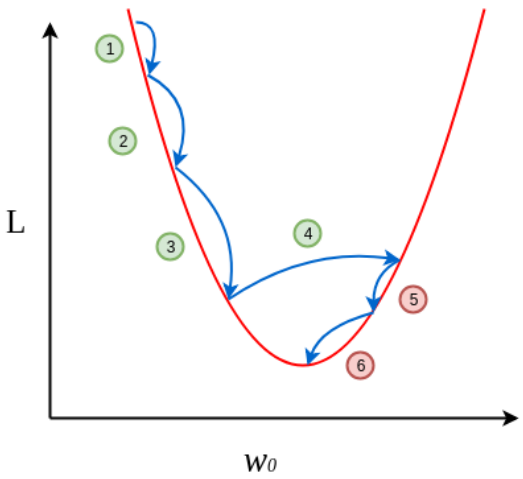


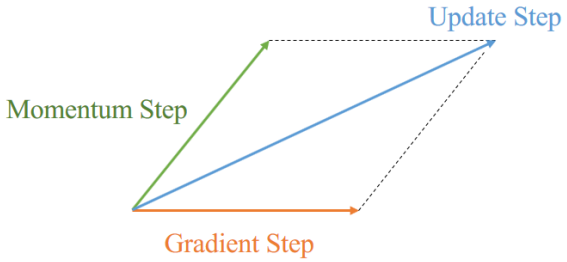



# Nesterov's momentum

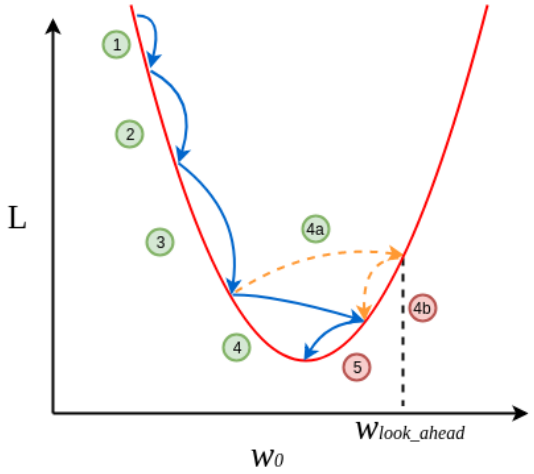
> Intuition: look ahead before you leap!

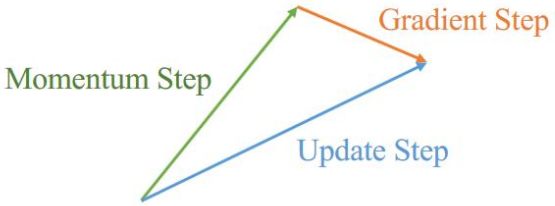
$$x_{t+1} = x_t + \alpha_t v_{t+1}$$
$$v_{t+1} = \gamma v_t - \nabla f(x_t)$$



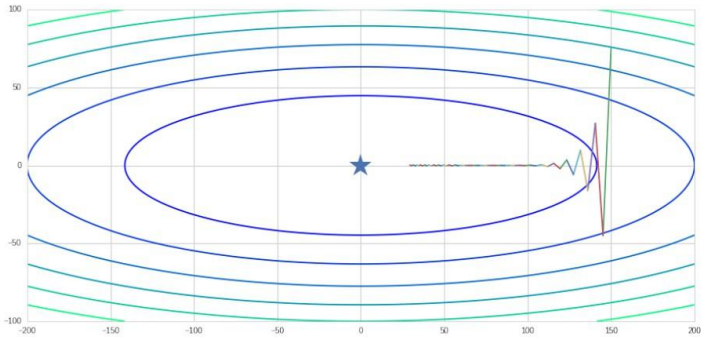

$$x_{t+1} = x_t + \alpha_t v_{t+1}$$
$$v_{t+1} = \gamma v_t - \nabla f(x_t + \alpha_t \gamma v_t)$$



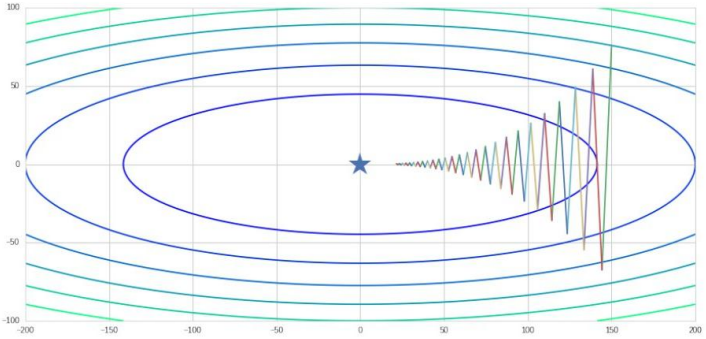



# Simple convex function performance

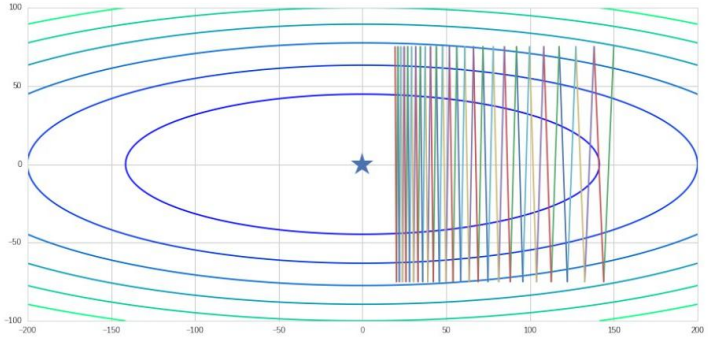
> $f(x, y) = x^2 + 10y^2$



(a) SGD with learning rate 0.08



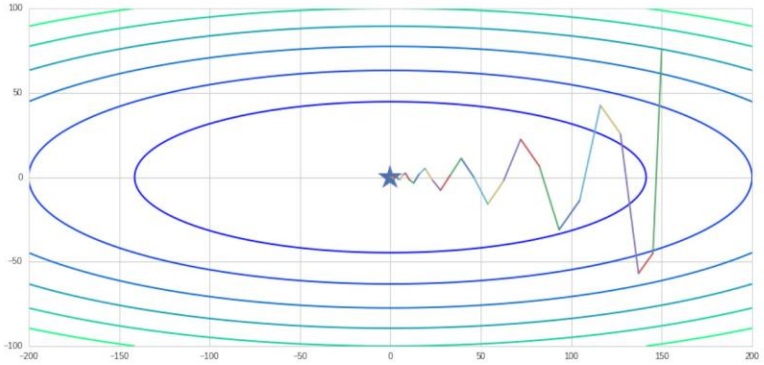
(b) SGD with learning rate 0.09



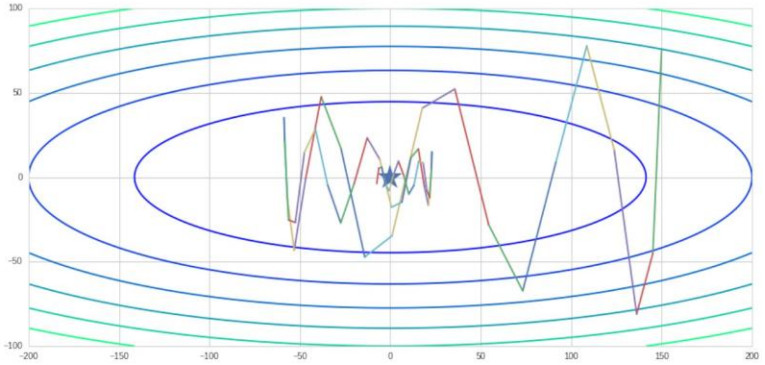
(c) SGD with learning rate 0.10

# Simple convex function performance

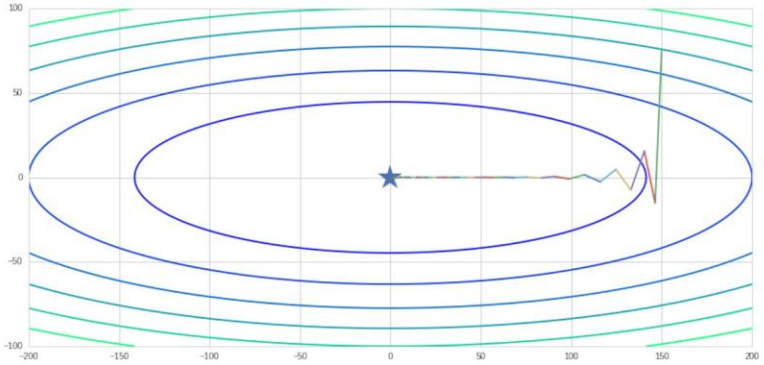
> $f(x, y) = x^2 + 10y^2$



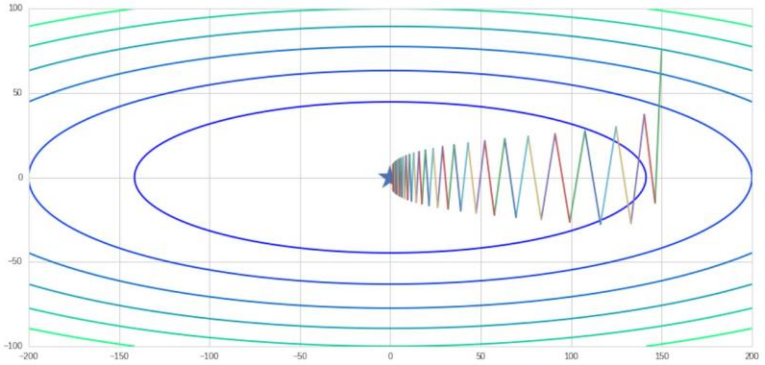
(a) Momentum with momentum factor 0.7



(b) Momentum with momentum factor 0.9



(c) NAG with momentum factor 0.7



(d) NAG with momentum factor 0.9

# Convergence theory

- A first-order method is one where updates are linear combinations of observed gradients.
  - i.e., $x_{k+1} - x_k = d \in span\{\nabla f(x_0), \nabla f(x_1), \ldots, \nabla f(x_k)\}$
  - gradient descent
  - momentum methods
  - conjugate gradients
- To achieve $f(x_k) - f(x^*) \leq \epsilon$, the number of iterations $k$ satisfies:
  - (worst-case) lower bound for 1st-order methods: $k \in \Omega\left(\sqrt{\kappa}\log\frac{1}{\epsilon}\right)$
  - upper bound for gradient descent: $k \in \mathcal{O}\left(\kappa\log\frac{1}{\epsilon}\right)$
  - upper bound for GD w/ Nesterov's momentum: $k \in \mathcal{O}\left(\sqrt{\kappa}\log\frac{1}{\epsilon}\right)$

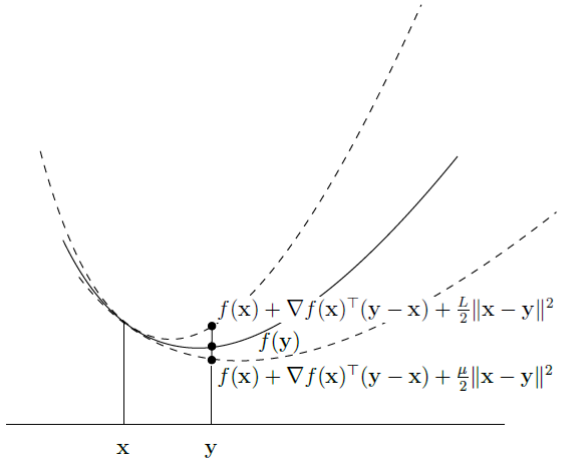
# 2nd-order method

> For any 1st-order method, the number of steps needed to converge grows with condition number

$$\kappa = \frac{L}{\mu}$$

Max curvature

Min curvature



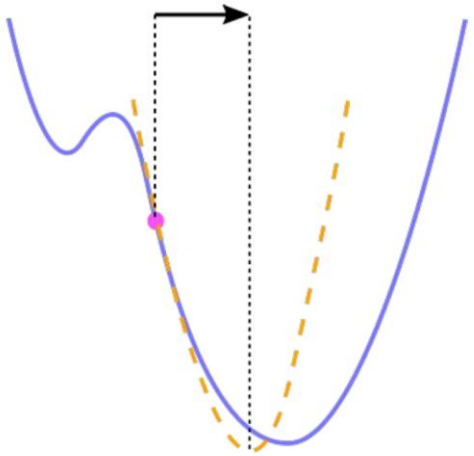

> 2nd –order methods can improve this dependency

- approximate $f(x)$ by its 2nd order Taylor series around current $x$

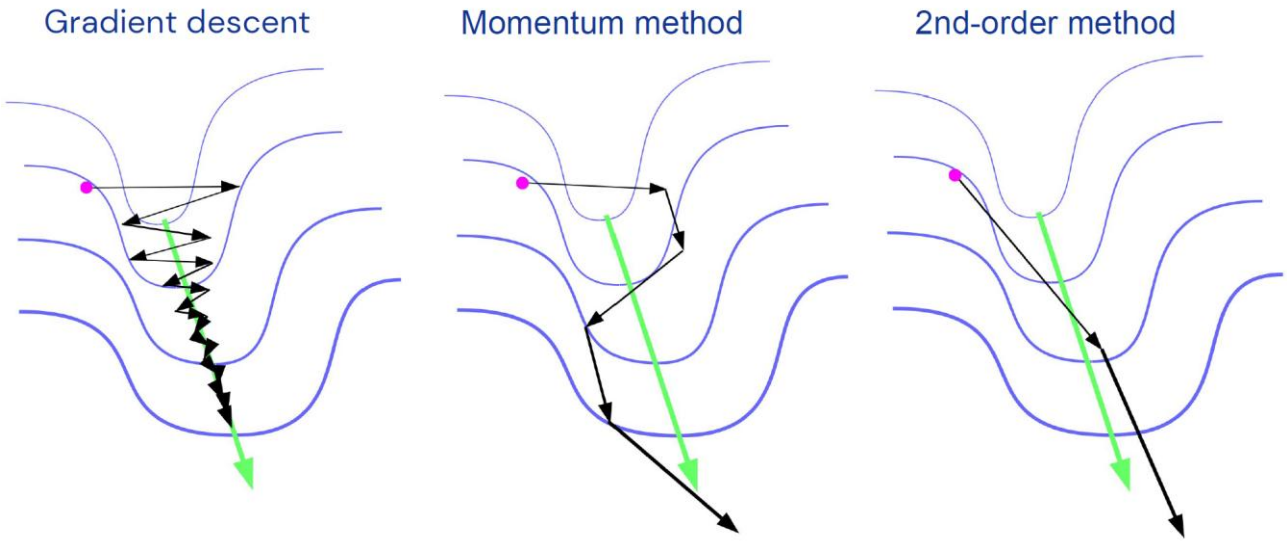
$$f(x + d) \approx f(x) + \nabla f(x)^\top d + \frac{1}{2} d^\top H(x) d$$

# 2nd-order method

- Minimize the 2nd-order approximation
  - $f(x+d) \approx f(x) + \nabla f(x)^{\mathsf{T}} d + \frac{1}{2} d^{\mathsf{T}} H(x) d$
  - $d = -H(x)^{-1} \nabla h(x)$

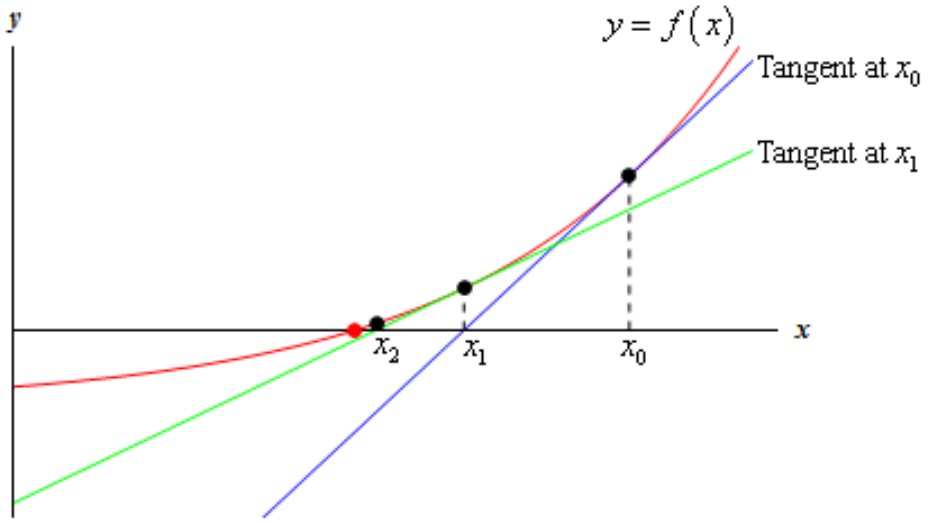


- Update current iterate with this:
  - $x_{t+1} = x_t - H(x)^{-1} \nabla f(x_t)$



# Newton's method

- Newton's method: finding a zero of a function (numerical optimization)
  - To find $f(x) = 0$, $x \leftarrow x - \frac{f(x)}{f\prime(x)}$




  - The maxima of $l$ corresponds to points where its first derivate $l'(\theta)$ is 0
  - To find $l'(\theta) = 0$, $\theta \leftarrow \theta - \frac{l'(x)}{l''(x)}$
  - In multi-dimensional setting $\theta \leftarrow \theta - H^{-1}\nabla_\theta l(\theta)$,

# Comparison to gradient descent

- GD: $x_{t+1} = x_t - \alpha \nabla f(x_t)$
  - To avoid divergence, $\alpha \leq 1/L$
    $L$ is maximum curvature (Lipschitz constant)
- Newton's method: $x_{t+1} = x_t - H(x)^{-1} \nabla f(x_t)$

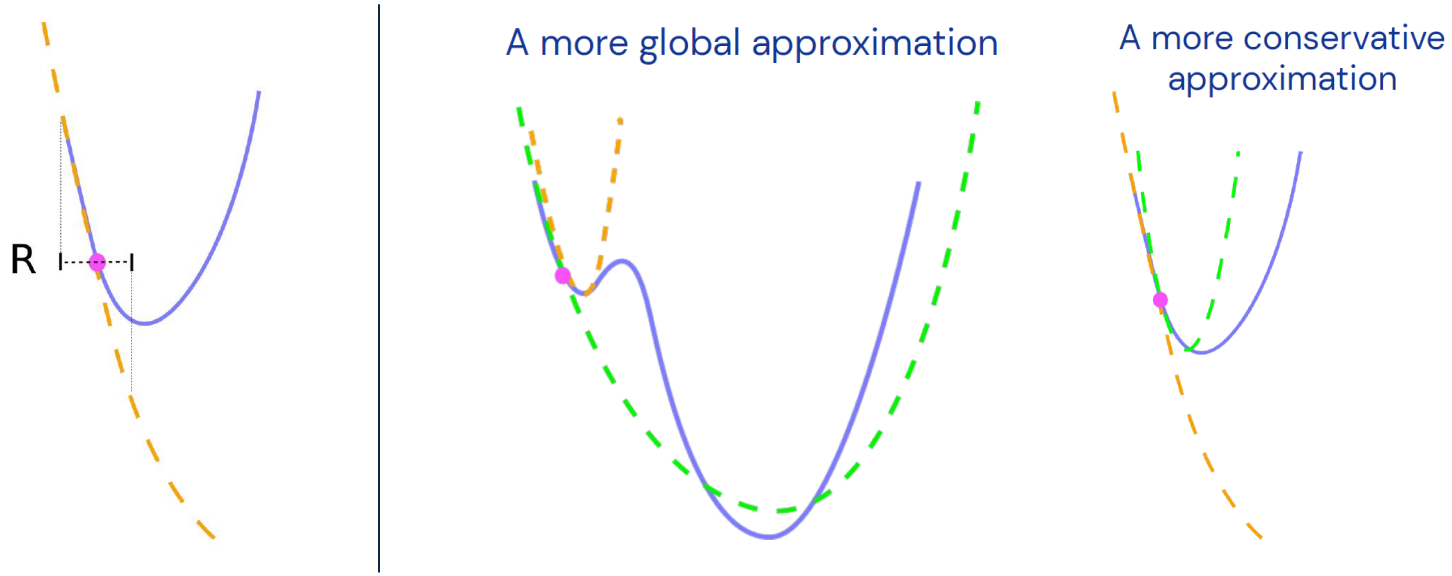
- $H(x)$ corresponds to $LI$
  - $f(x+d) \approx f(x) + \nabla f(x)^\top d + \frac{1}{2} d^\top H(x) d$

    $\approx f(x) + \nabla f(x)^\top d + \frac{1}{2} d^\top L I d$

  - GD implicitly minimizes a bad approximation of 2nd-order Taylor series
  - GD treats all directions as having max curvature

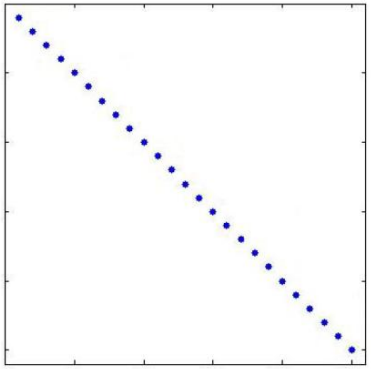
# 2nd-order method

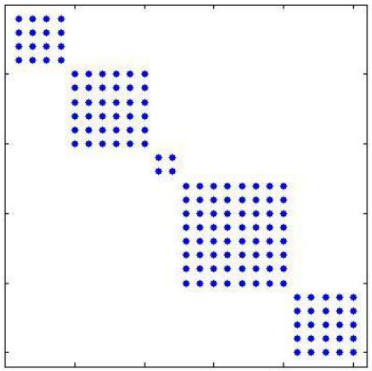
- Quadratic approximation of objective is only trustworthy in a local region around current $x$
  - Constrain update $d$ to lie in a trust region $R$ around where approximation remains good enough
- Or we use a better quadratic approximation instead of using $H(x)$
  - Generalized Gauss-Newton matrix (GCN), Fisher information matrix, …



# 2nd-order method

- Disadvantages
  - For neural networks, $x \in \mathbb{R}^n$ can have millions or billions of dimensions
  - We can't compute and store $n \times n$ matrix
  - To use 2nd-order methods, we must simplify the curvature matrix's
    - computation
    - storage
    - and inversion
  - This can be done by approximating the matrix with a simpler form
    - such as using only the diagonal entries
    - or using only the diagonal blocks





# $2^{nd}$-order method

- Many optimizers retain some benefits of curvature-aware optimization without the full computational burden
  - We may call it quasi-$2^{nd}$ methods
  - use first-order information (gradients) only, but implicitly incorporate curvature information (typically through the averages)

- Most modern optimizers (used in deep learning) are quasi-$2^{nd}$ order
  - Adagrad, Adadelta, RMSProp, Adam, AdamW, Nadam, AMSGrad

# Optimizer: Adagrad

- It adapts the learning rate to the parameters
  - smaller update to frequently occurring features
  - larger update to infrequent features
  - well-suited for dealing with sparse data

  - gradient: $g_{t,i} = \nabla J(x_{t,i})$
  - update: $x_{t+1,i} = x_{t,i} - \frac{\alpha}{\sqrt{G_{t,ii}+\epsilon}} g_{t,i}$
  - $G_t$ is a diagonal matrix where its diagonal entry $i$ is the sum of the squares of the gradients w.r.t. $x_i$ up to time step $t$ (contains the past gradients)

  - Learning rate is autonomously reduced for each parameter
  - Weakness: $G_t$ is keeps accumulating, causing the learning rate to keep shirinking

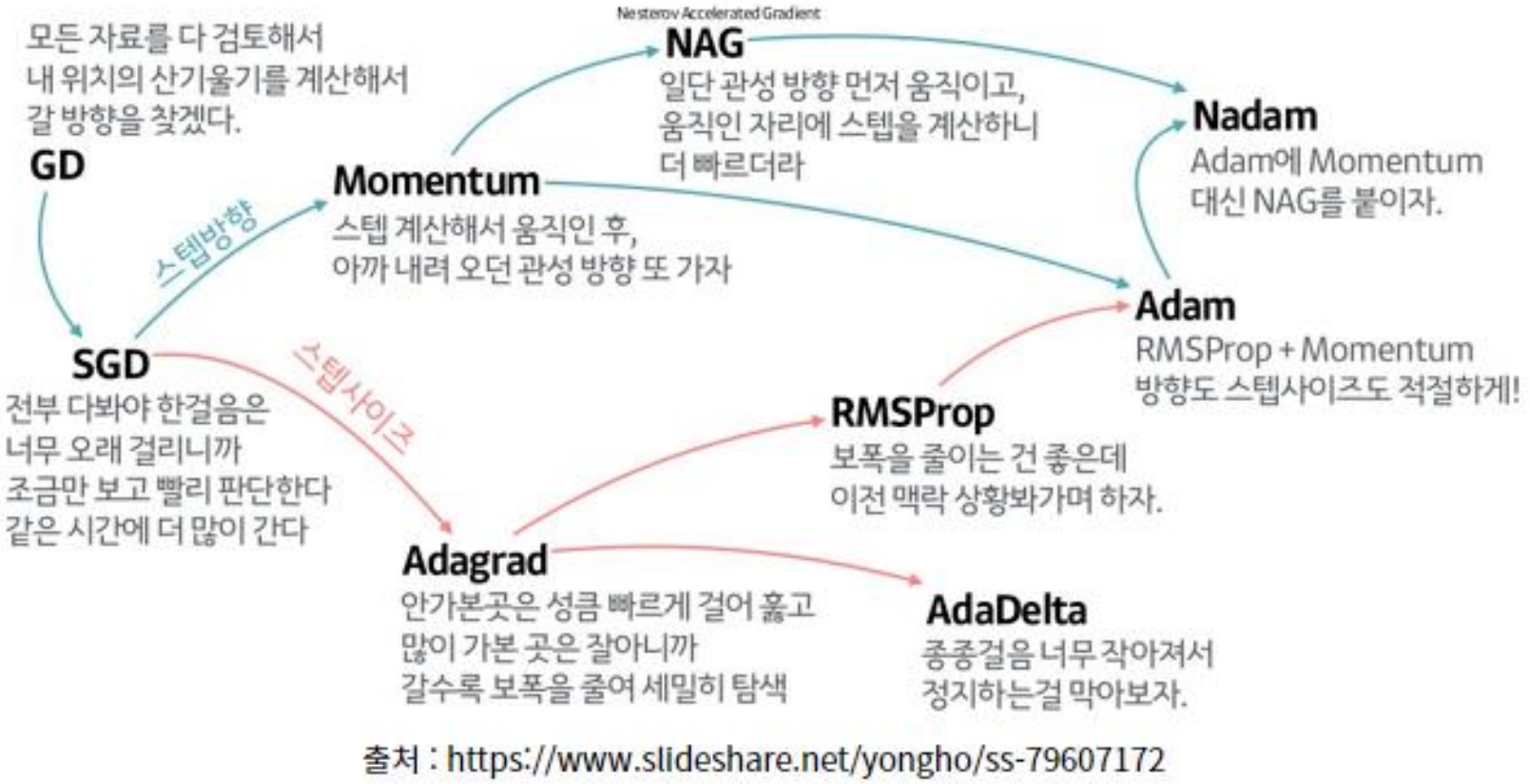
# Optimizer: RMSProp

- To resolve Adagrad's radically diminishing learning rates
  - restricts the window of accumulated past gradients
- Instead of inefficiently storing $w$ previous squared gradients, the sum of gradients is recursively defined as a decaying average
  - $E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$ (exponentially moving average)
  - $x_{t+1,i} = x_{t,i} - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} g_{t,i}$
- Hinton suggests $\gamma$ to be set to 0.9, while a good default value for $\alpha = 0.001$

# Optimizer: Adam

- Adaptive momentum estimation (Adam)
  - In addition to storing an EMA of past squared gradients, Adam keeps an EMA of past gradients, similar to momentum
  - $E[g^2]_t = \gamma_1 E[g^2]_{t-1} + (1-\gamma_1) g_t^2$
  - $m_t = \gamma_2 m_{t-1} + (1-\gamma_2) g_t$
  - $m_t$ and $E[g^2]_t$ are biased towards zero initially, thus correct it
  - $\hat{m}_t = \frac{m_t}{1-\gamma_2^t}, \hat{E}[g^2]_t = \frac{E[g^2]_t}{1-\gamma_1^t}$
  - $x_{t+1,i} = x_{t,i} - \frac{\alpha}{\sqrt{\hat{E}[g^2]_t + \epsilon}} \hat{m}_t$
  - Authors propose default values of $(0.999\ 0.9)$ for $(\gamma_1, \gamma_2)$

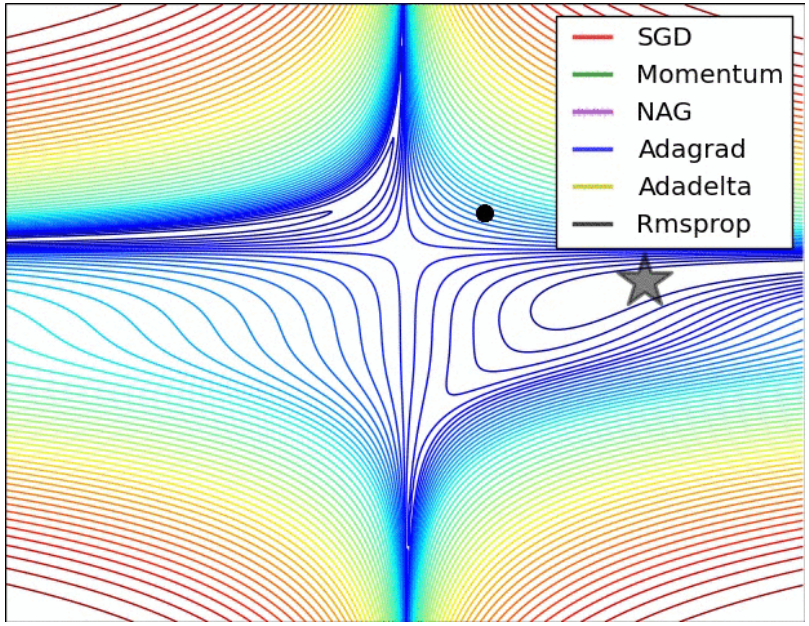
# Optimizers

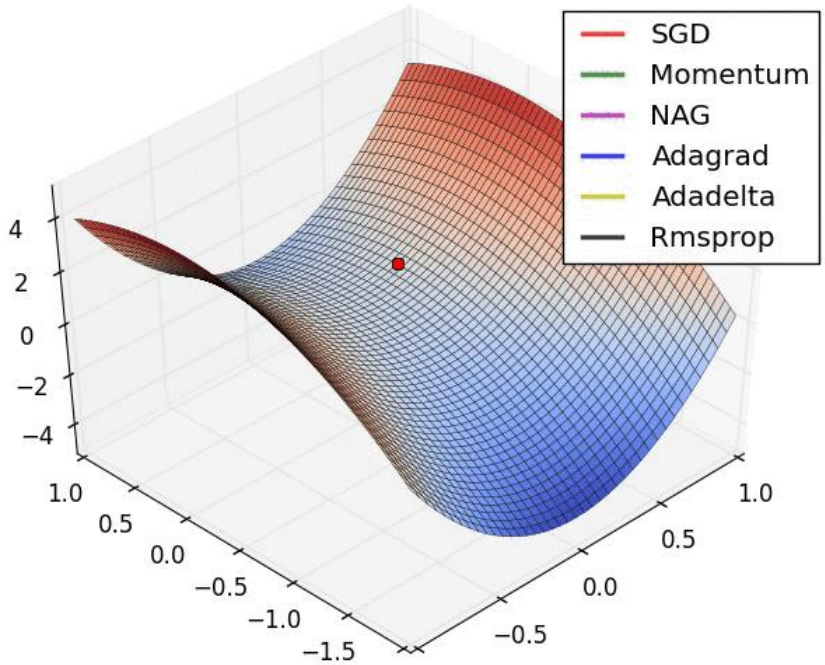



출처 : https://www.slideshare.net/yongho/ss-79607172

# Optimizers

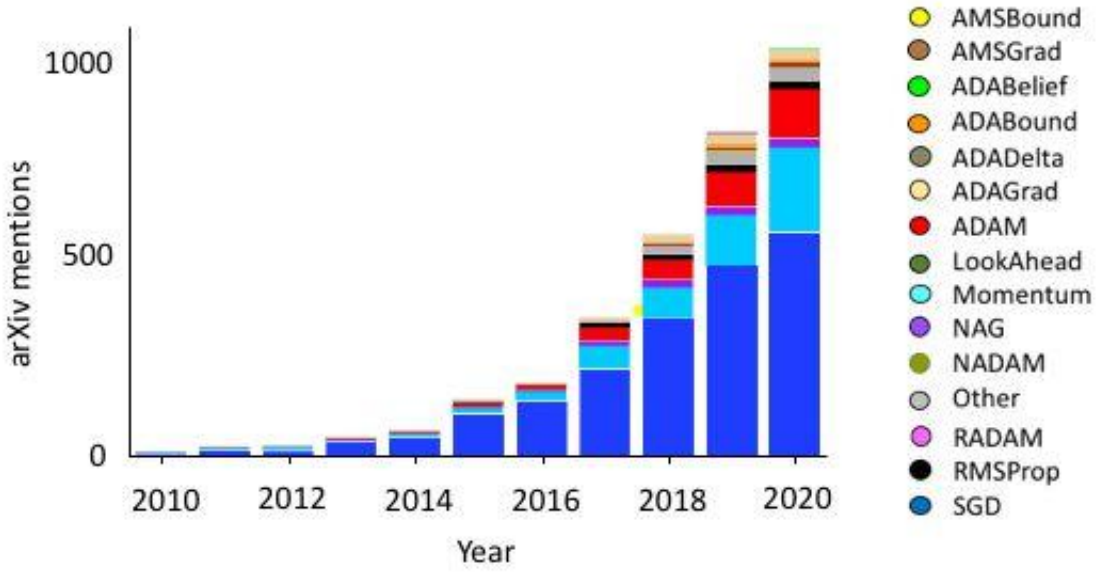
> Convergence performance



# Optimizers

- Which optimizer to use?
  - Adam / AdamW are dominant and default choices
  - In cases where another optimizer did better than Adam, it was usually RMSProp or NAG
  - Adafactor, Lion, Signum are promising for LLMs
  - SGD (+momentum) is good for vision tasks, but requires careful tuning



# Reference

- An overview of gradient descent optimization algorithms
  - https://www.ruder.io/optimizing-gradient-descent/
- Simple performance example
  - https://tiddler.github.io/optimizers/
- Lecture notes
  - Theoretical derivation and mathematics:
    - optimization in machine learning – Simon Weissmann
    - optimization for machine learning – Bernd Gartner and Martin Jaggi