Lecture 12:

Optimization Techniques

Markus Hohle

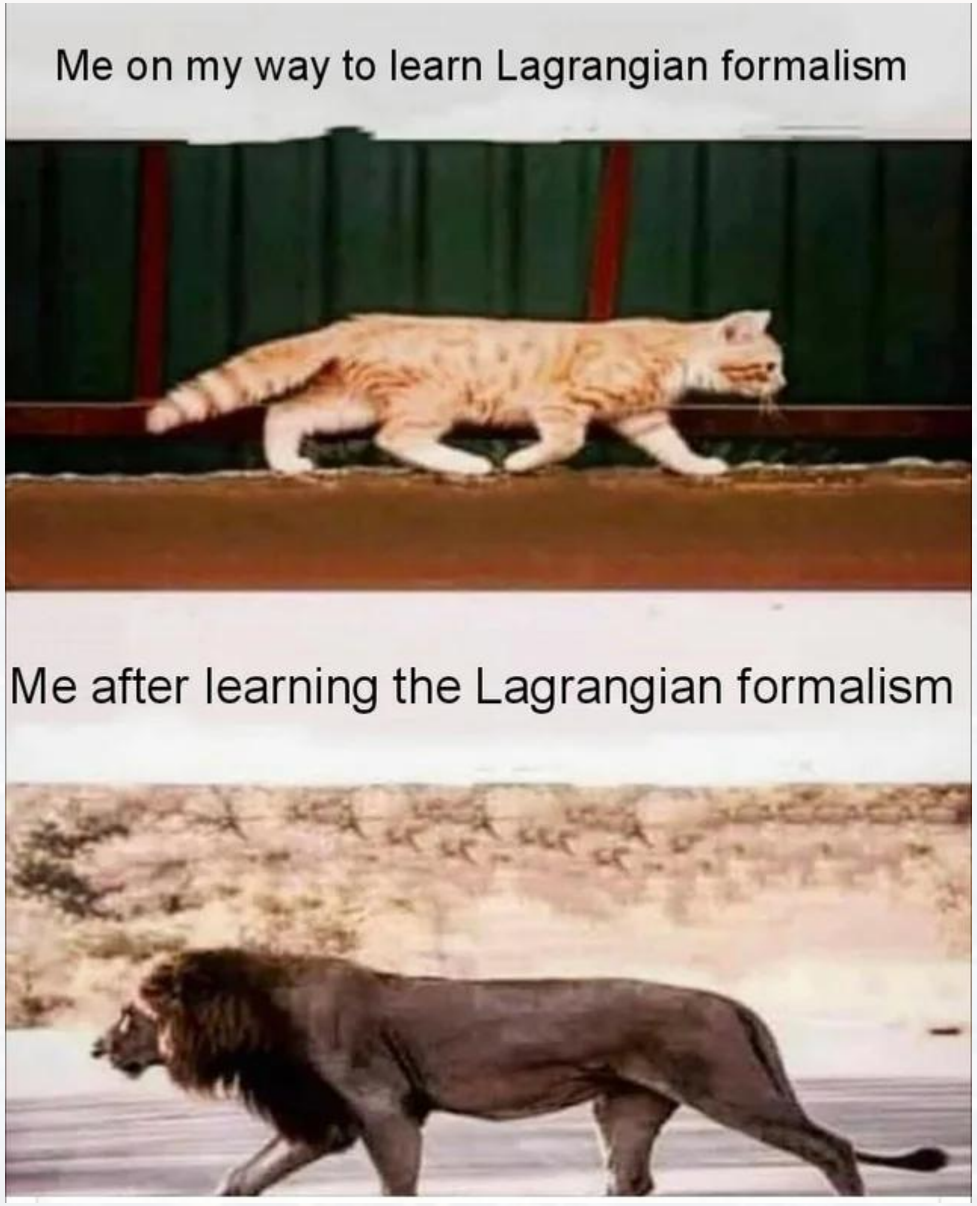
University California, Berkeley

Numerical Methods for Computational Science

Berkeley MSSE
MOLECULAR SCIENCE & SOFTWARE ENGINEERING

Course Map

Week 1:	Introduction to Scientific Computing and Python Libraries
Week 2:	Linear Algebra Fundamentals
Week 3:	Vector Calculus
Week 4:	Numerical Differentiation and Integration
Week 5:	Solving Nonlinear Equations
Week 6:	Probability Theory Basics
Week 7:	Random Variables and Distributions
Week 8:	Statistics for Data Science
Week 9:	Eigenvalues and Eigenvectors
Week 10:	Simulation and Monte Carlo Method
Week 11:	Data Fitting and Regression
Week 12:	**Optimization Techniques**
Week 13:	Machine Learning Fundamentals



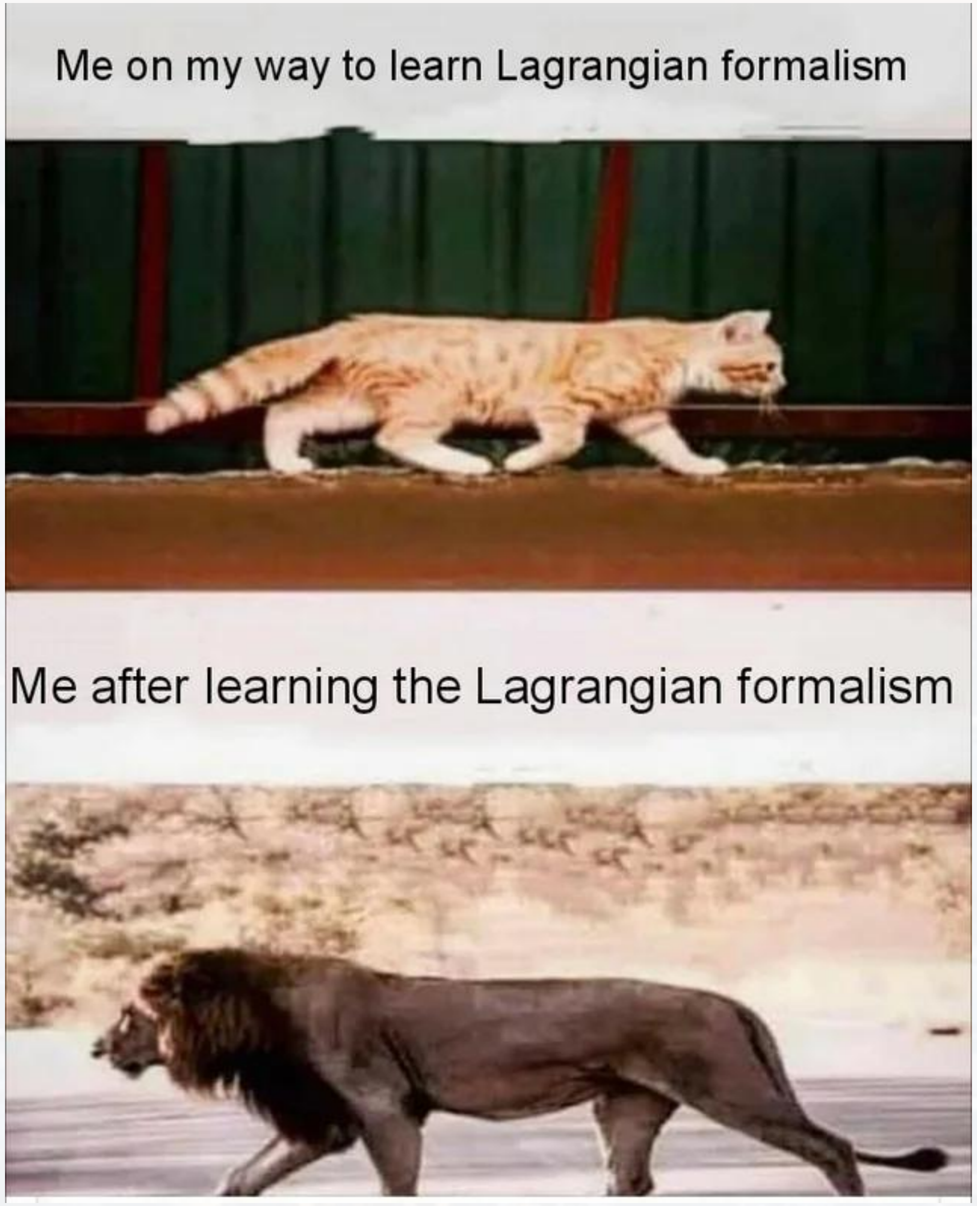
source: SKRyanrr

Outline

Lagrangian Multiplier

Gradient Descent Again

- Vanilla
- Learning Rate Schedule
- Momentum
- L1 and L2
- More Finetuning



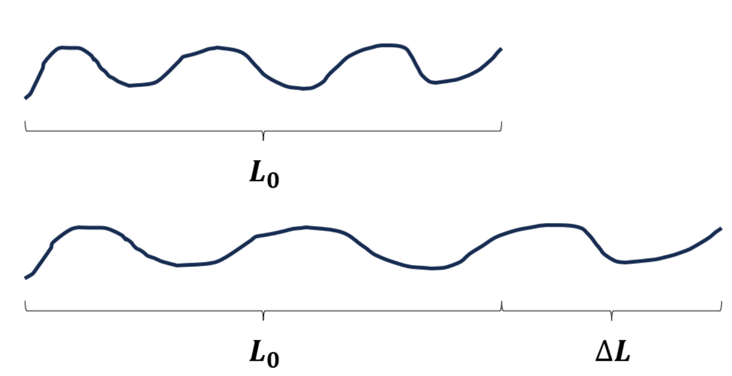
source: SKRyanrr

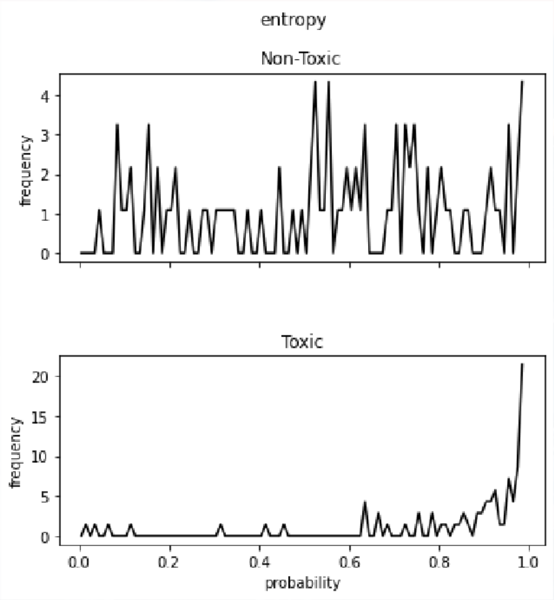
Optimization Techniques:

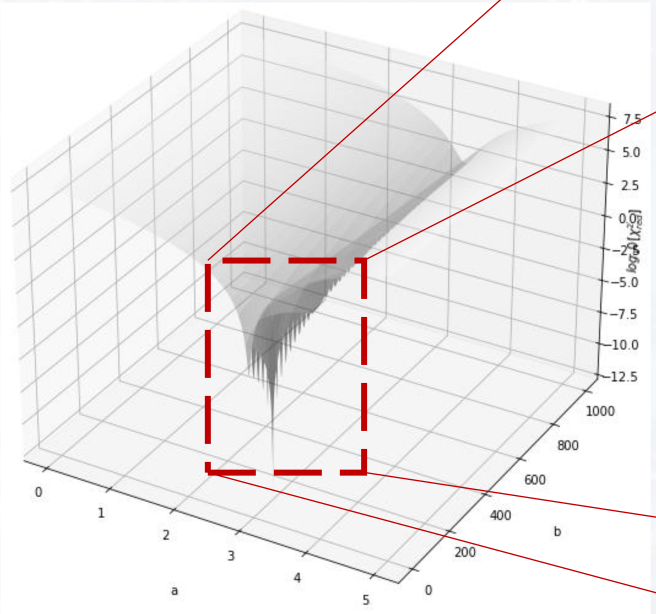
problem: often we need to find **an extreme of a function**:

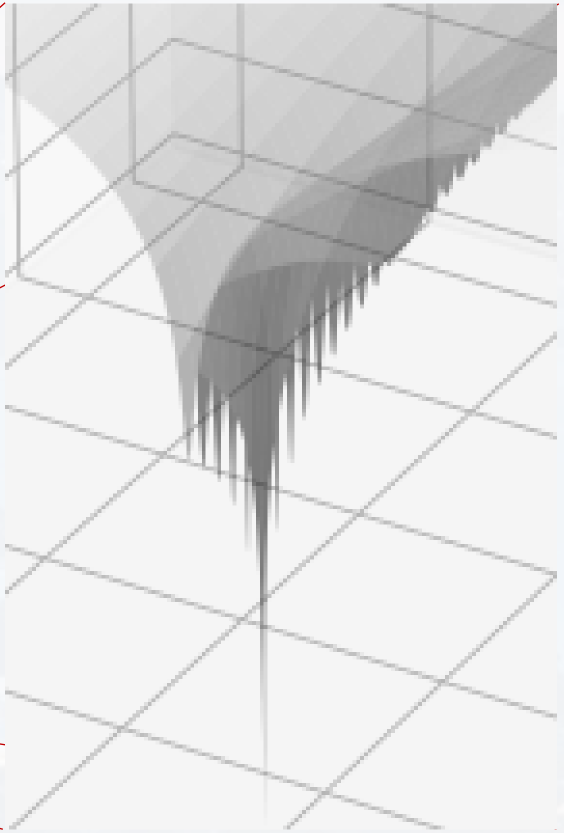
$$P(k|c) = \frac{(c\,\Delta t)^k e^{-c\,\Delta t}}{k!}$$

- equilibrium at **minimum** energy
- distributions for **maximum** entropy
- regression: **minimizing** χ^2_{red} or MSE
- classification: **minimizing** cross-entropy etc







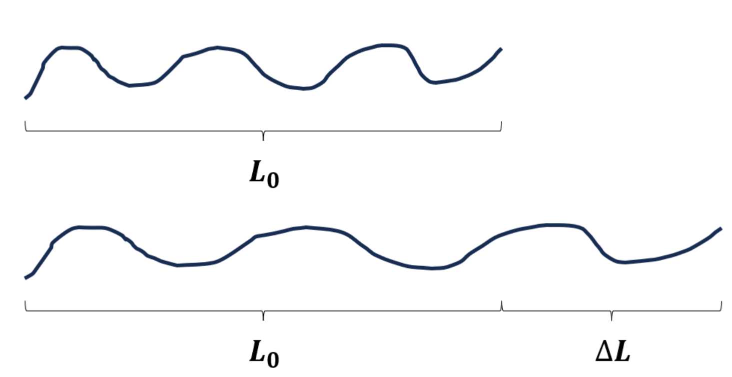


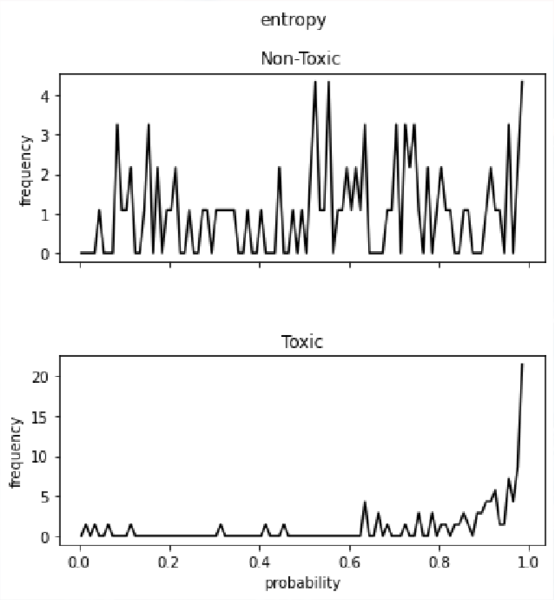
Optimization Techniques:

problem: often we need to find **an extreme of a function**:

- equilibrium at **minimum** energy
- distributions for **maximum** entropy
- regression: **minimizing** χ^2_{red} or MSE
- classification: **minimizing** cross-entropy etc

$$P(k|c) = \frac{(c\,\Delta t)^k e^{-c\,\Delta t}}{k!}$$

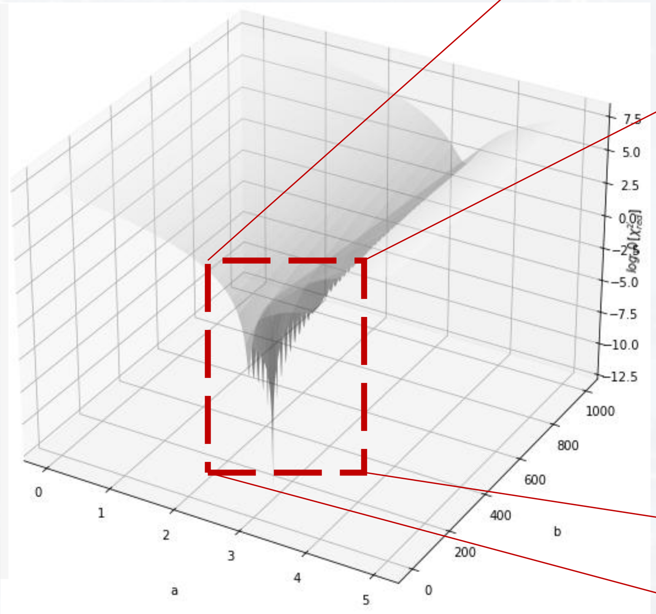


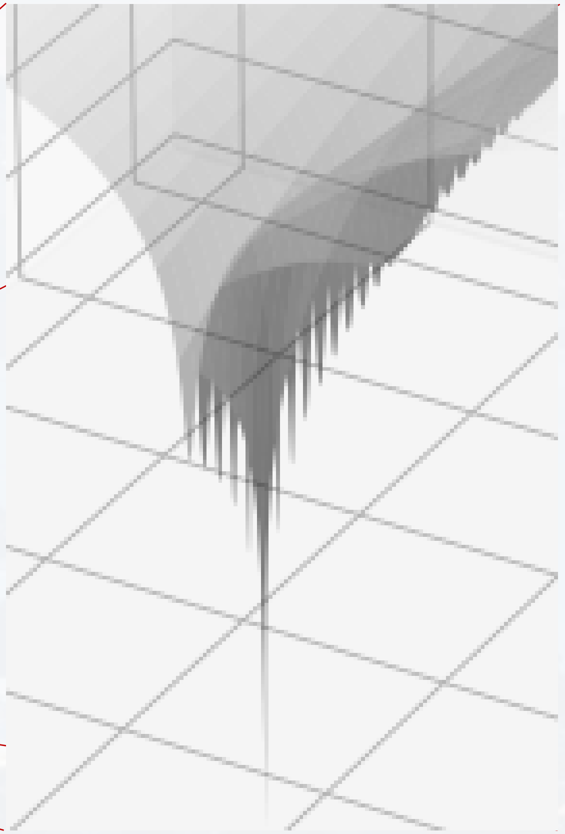


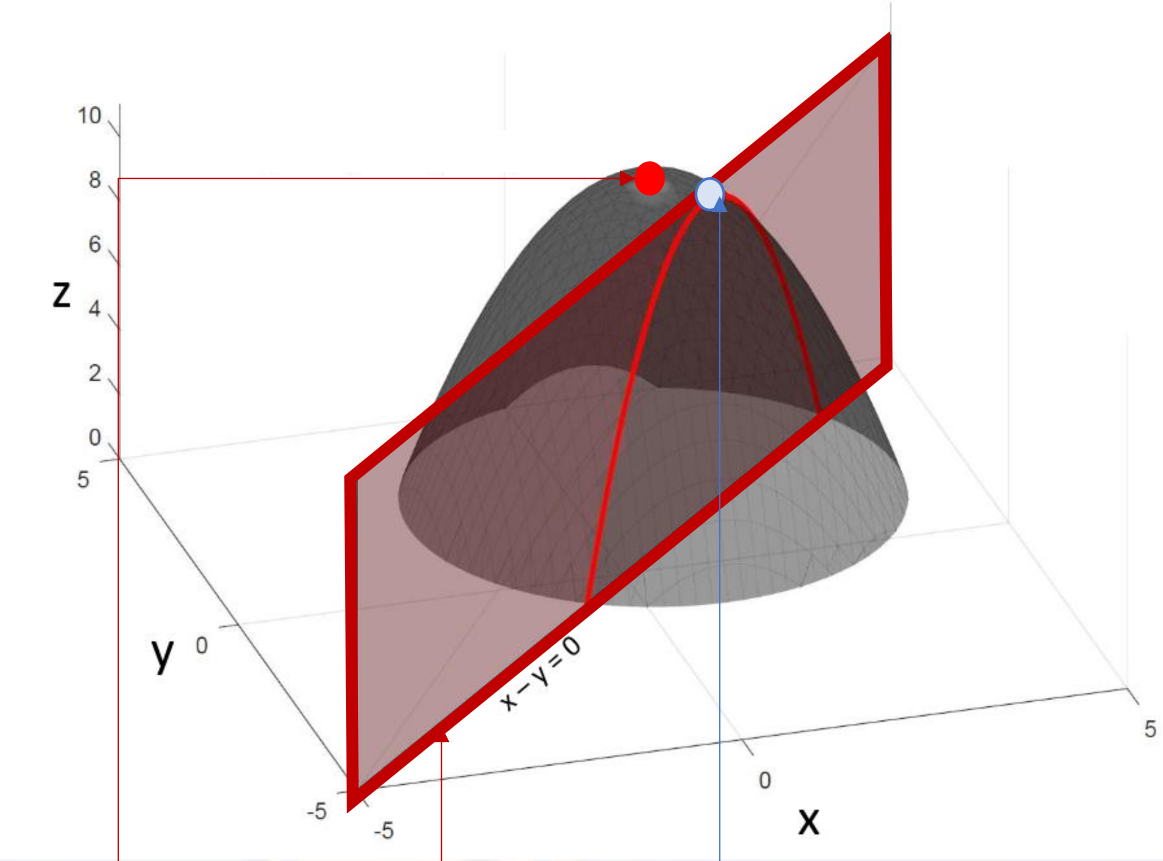
but often, systems are **subject to a set of constrains**

- **mass can only be positive**
- **energy is constant**
- **probabilities sum up to 1**
- **...**

goal: find **extreme within these constrains**!







maximum of the function

$$f(x,y) = z = -(x-2)^2 - (y-1)^2 + 10$$

$$\frac{\partial f(x,y)}{\partial x} = 0 \quad \textbf{and} \quad \frac{\partial f(x,y)}{\partial y} = 0$$

$$-2(x-2) = 0 \qquad -2(y-1) = 0$$

$$x = 2 \qquad y = 1$$

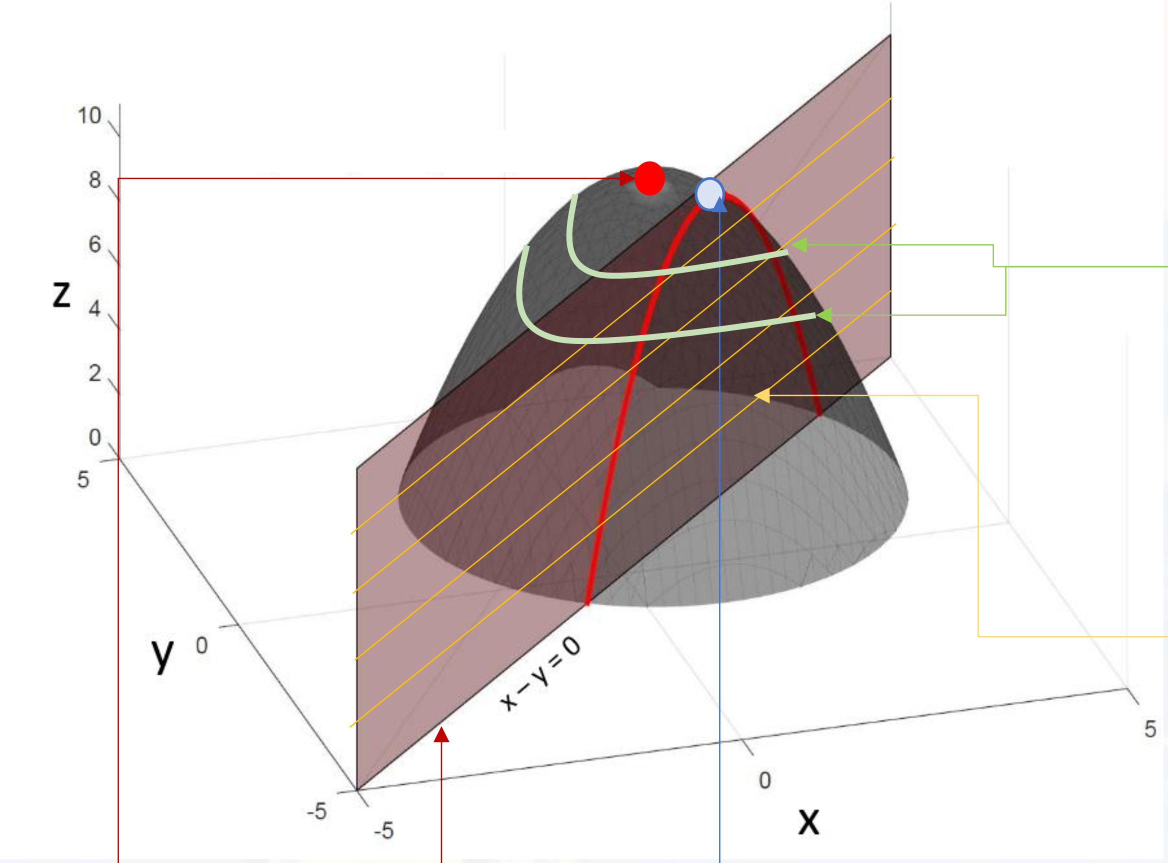
$$f(2,1) = 10$$

$z = 10$ at $x = 2$, $y = 1$

constrain $g(x,y) = x - y = 0$

maximum of the function, subject to $g(x,y)$

Optimization Techniques:



maximum of the function

$$f(x,y) = z = -(x-2)^2 - (y-1)^2 + 10$$

level lines $f(x,y) = const$

$$df(x,y) = \frac{\partial f(x,y)}{\partial x}dx + \frac{\partial f(x,y)}{\partial y}dy = 0$$

$$= grad f \; d\vec{r} = 0$$

level lines $g(x,y) = const$

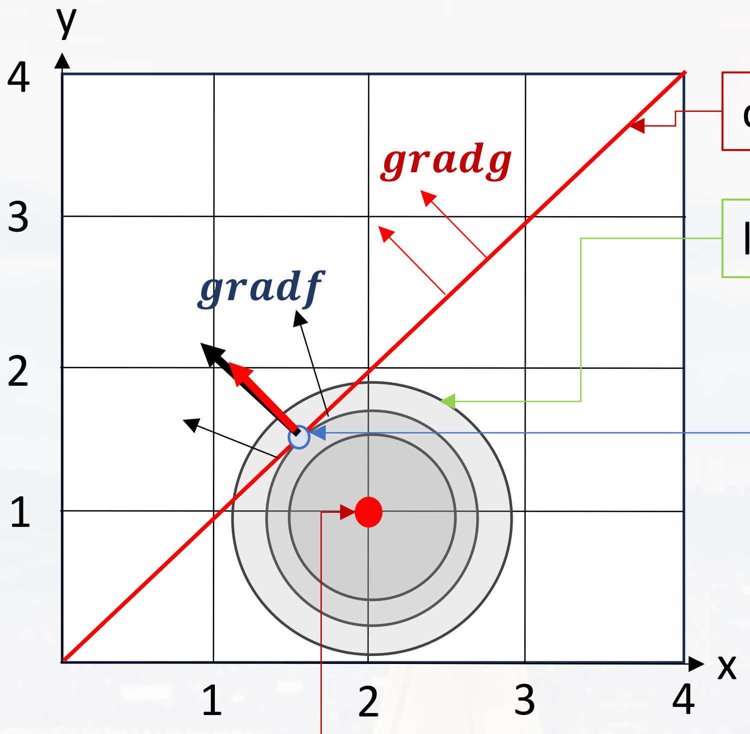
$$dg(x,y) = \frac{\partial g(x,y)}{\partial x}dx + \frac{\partial g(x,y)}{\partial y}dy = 0$$

$$= grad g \; d\vec{r} = 0$$

$z = 10$ at
$x = 2$
$y = 1$

constrain $g(x,y) = x - y = 0$

maximum of the function, subject to $g(x,y)$



constrain $g(x,y) = x - y = 0$

level lines $f(x,y) = const$

$z = 10$ at
$x = 2$
$y = 1$

maximum of the function, subject to $g(x,y)$

the maximum of $f(x,y)$ subject to $g(x,y)$ located where:

$$df(x,y) = dg(x,y)$$

$$gradf\ d\vec{r} = gradg\ d\vec{r}$$

$$gradf = gradg$$

Both gradients need to point in the same direction
(hence, can be multiplied with a constant, say λ)!

$$\boxed{gradf = \lambda\ gradg}$$

λ **Lagrangian Multiplier**

the maximum of $f(x,y)$ subject to $g(x,y)$

$$grad f = \lambda \, grad g$$

$$grad f - \lambda \, grad g = 0$$

$$\int grad f \, d\vec{r} - \lambda \int grad g \, d\vec{r} = const$$

$$f(x,y) - \lambda g(x,y) = const$$

the Lagrangian
$\boldsymbol{L(x,y,\lambda)}$

more general:

$$\boxed{L(x_1, x_2, \dots, x_i, x_N, \lambda_1, \lambda_2, \dots, \lambda_k, \lambda_K) = f(x_1, x_2, \dots, x_i, x_N) - \sum_{k=1}^{K} \lambda_k \, g_k(x_1, x_2, \dots, x_i, x_N)}$$

N dimensions and K $\leq N$ constrains

Optimization Techniques:

the maximum of $f(x,y)$ subject to $g(x,y)$

$$grad f = \lambda\, grad g$$

$$grad f - \lambda\, grad g = 0$$

$$\int grad f\, d\vec{r} - \lambda \int grad g\, d\vec{r} = const$$

$$f(x,y) - \lambda g(x,y) = const$$

the Lagrangian
$\boldsymbol{L(x, y, \lambda)}$

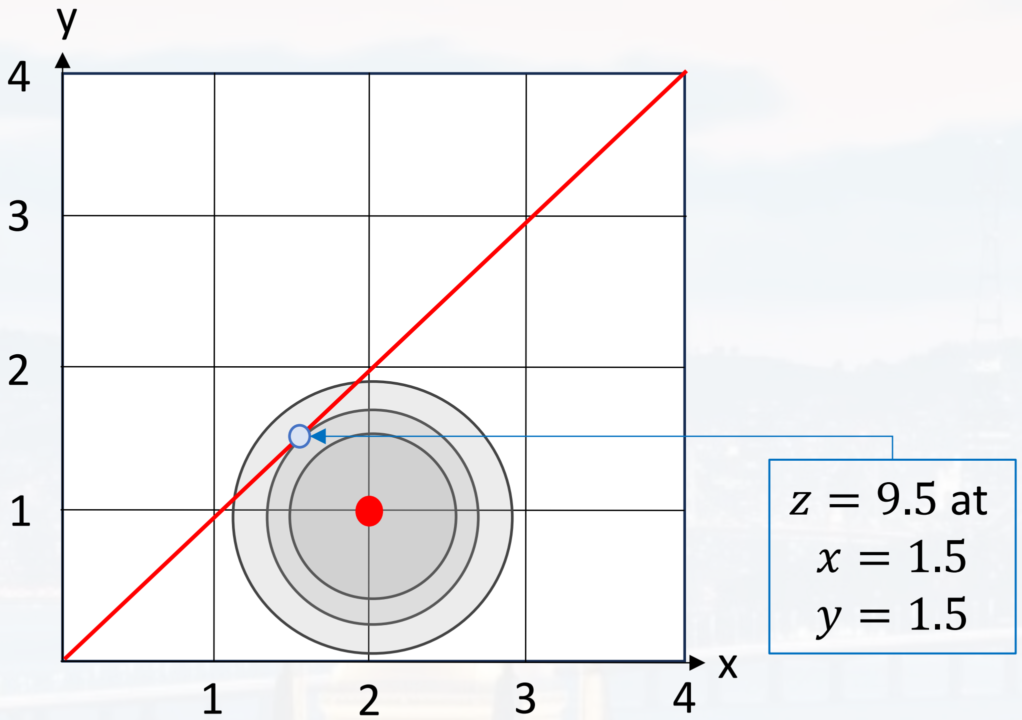
physics:	describes dynamics of a system based on conserved quantities (energy, momentum etc) → L = const
optimization:	more robust results (see later)
machine learning:	loss function

more general:

$$\boxed{L(x_1, x_2, \dots, x_i, x_N, \lambda_1, \lambda_2, \dots, \lambda_k, \lambda_K) = f(x_1, x_2, \dots, x_i, x_N) - \sum_{k=1}^{K} \lambda_k\, g_k(x_1, x_2, \dots, x_i, x_N)}$$

N dimensions and K $\leq N$ constrains

the maximum of $f(x,y)$ subject to $g(x,y)$



maximum of the function

$$f(x,y) = z = -(x-2)^2 - (y-1)^2 + 10$$

constrain $g(x,y) = x - y = 0$

constrain $x = y$

$x = 1.5$
$y = 1.5$

$f(1.5, 1.5) = 9.5$

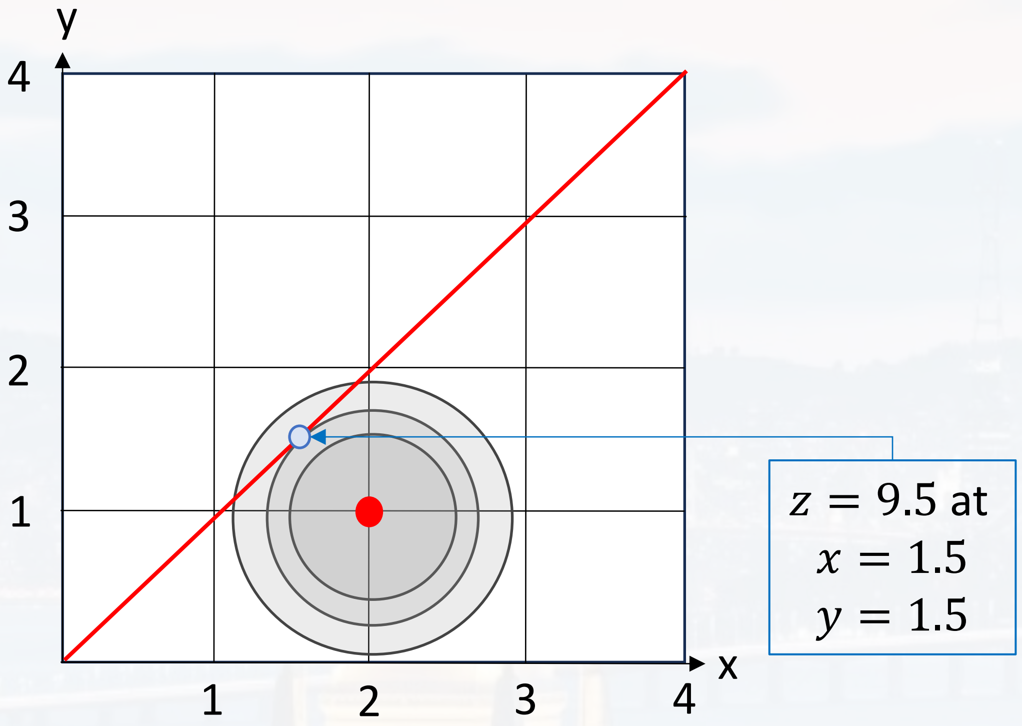
$$grad f = \lambda \, grad g$$

$$\frac{\partial f(x,y)}{\partial x} = \lambda \frac{\partial g(x,y)}{\partial x} \qquad -2(x-2) = \lambda$$

$$\frac{\partial f(x,y)}{\partial y} = \lambda \frac{\partial g(x,y)}{\partial y} \qquad -2(y-1) = -\lambda$$

$$y = -x + 3$$

the maximum of $f(x,y)$ subject to $g(x,y)$



maximum of the function

$$f(x,y) = z = -(x-2)^2 - (y-1)^2 + 10$$

constrain $g(x,y) = x - y = 0$

constrain $x = y$ $\quad x = 1.5$, $y = 1.5$

$$f(1.5, 1.5) = 9.5$$

We need to solve N + K equations!

$$L(x_1, x_2, \dots, x_i, x_N, \lambda_1, \lambda_2, \dots, \lambda_k, \lambda_K) = f(x_1, x_2, \dots, x_i, x_N) - \sum_{k=1}^{K} \lambda_k \, g_k(x_1, x_2, \dots, x_i, x_N)$$

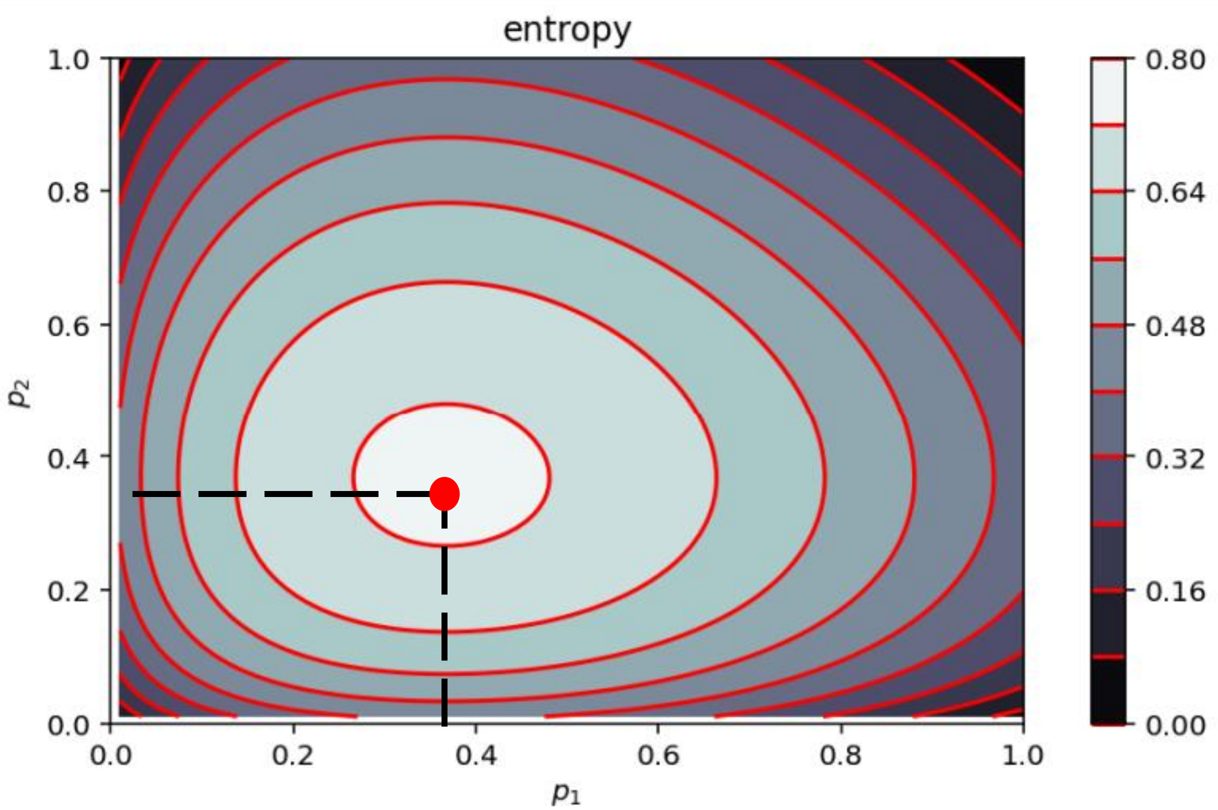
N dimensions and K $\leq N$ constrains

maximum entropy of flipping a coin:

$$f(p_1, p_2) = -p_1 \ln p_1 - p_2 \ln p_2$$

subject to

$$g(p_1, p_2) = p_1 + p_2 = 1$$



absolute maximum:

$$\frac{\partial f(p_1, p_2)}{\partial p_1} = 0 \qquad \frac{\partial f(p_1, p_2)}{\partial p_2} = 0$$

$$-\ln p_1 - 1 = 0 \qquad -\ln p_2 - 1 = 0$$

$$p_1 = p_2 = \frac{1}{e}$$

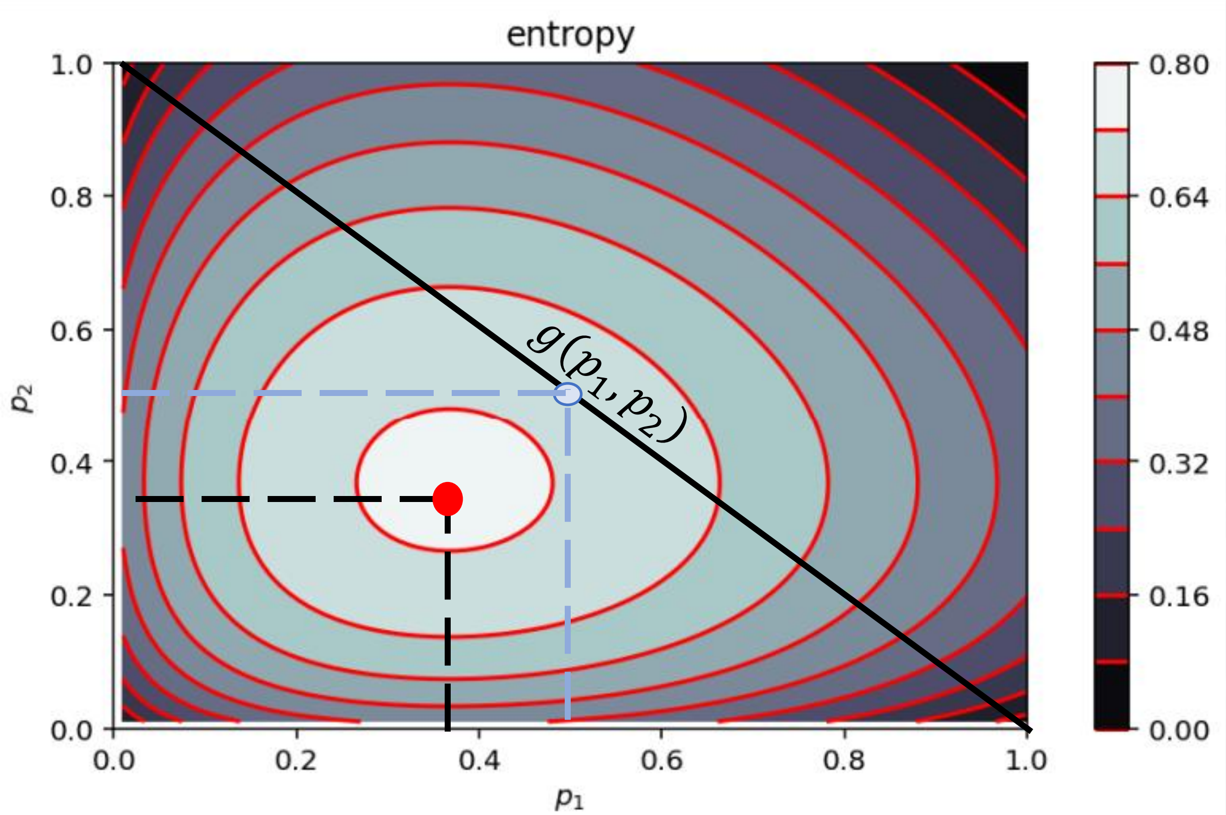
$$f\left(\frac{1}{e}, \frac{1}{e}\right) = \frac{2}{e} \approx 0.74$$

maximum entropy of flipping a coin:

$$f(p_1, p_2) = -p_1 \ln p_1 - p_2 \ln p_2$$

subject to

$$g(p_1, p_2) = p_1 + p_2 = 1$$



maximum subject to $g(p_1, p_2)$:

$$\frac{\partial f(p_1, p_2)}{\partial p_1} = \lambda \frac{\partial g(p_1, p_2)}{\partial p_1}$$

$$\frac{\partial f(p_1, p_2)}{\partial p_2} = \lambda \frac{\partial g(p_1, p_2)}{\partial p_2}$$

$$-\ln p_1 - 1 = \lambda \qquad -\ln p_2 - 1 = \lambda$$

$$p_1 = p_2$$

constrain: $p_1 + p_2 = 1$

$$p_1 = p_2 = \frac{1}{2}$$

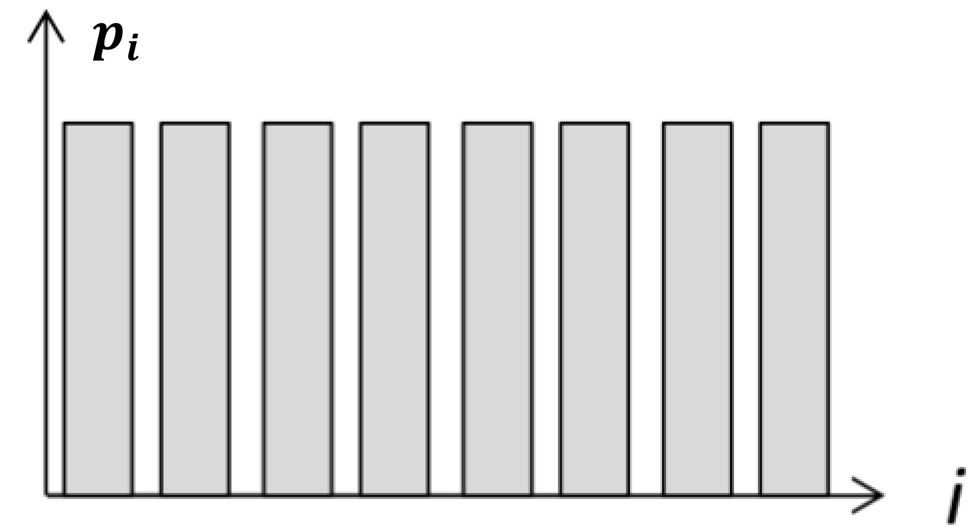
$$f\left(\frac{1}{2}, \frac{1}{2}\right) = ln2 \approx 0.69$$

maximum entropy for I states:

$$f(p_1,..,p_i,...p_I) = -\sum_{i=1}^{I} p_i \, ln p_i$$

subject to

$$g(p_1,..,p_i,...p_I) = \sum_{i=1}^{I} p_i \; = 1$$



maximum subject to $g(p_1,..,p_i,...p_I)$:

$$\frac{\partial f(p_1,..,p_i,...p_I)}{\partial p_i} = \lambda \frac{\partial g(p_1,..,p_i,...p_I)}{\partial p_i}$$

$$-\ln p_i - 1 = \lambda$$

$$p_i = \; e^{-(1+\lambda)}$$

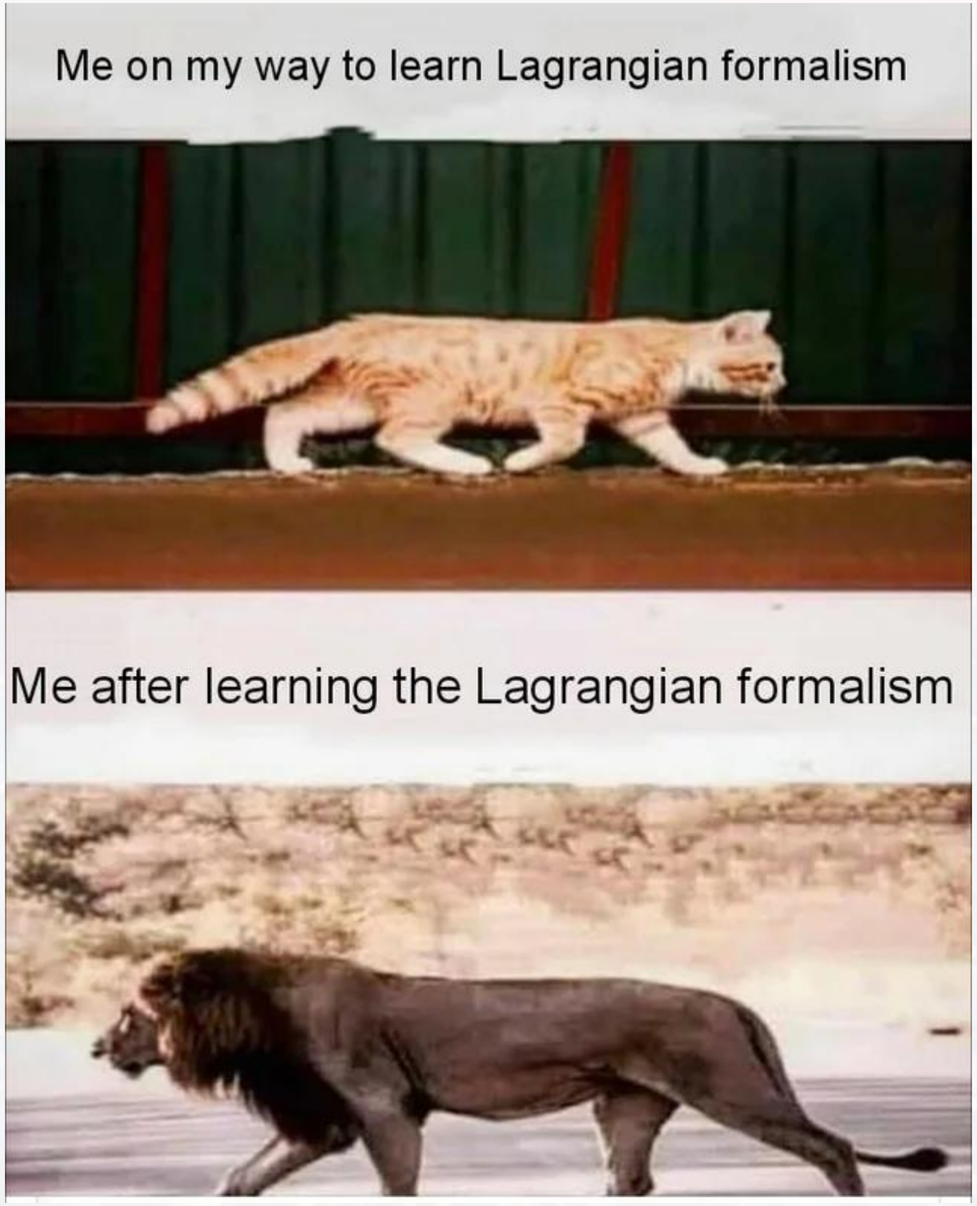
constrain: $$\sum_{i=1}^{I} e^{-(1+\lambda)} = 1$$

$$I e^{-(1+\lambda)} = 1$$

$$e^{-(1+\lambda)} = \frac{1}{I}$$

probabilities are constant!
→ flat distribution!

$$\boxed{p_i = \frac{1}{I}}$$



source: SKRyanrr

<u>Outline</u>

Lagrangian Multiplier

Gradient Descent Again

- Vanilla
- Learning Rate Schedule
- Momentum
- L1 and L2
- More Finetuning

recall the gradient (module 3)

$$\frac{dT}{dt} = \frac{\partial T}{\partial x}\frac{dx}{dt} + \frac{\partial T}{\partial y}\frac{dy}{dt} + \frac{\partial T}{\partial z}\frac{dz}{dt} + \frac{\partial T}{\partial t}$$

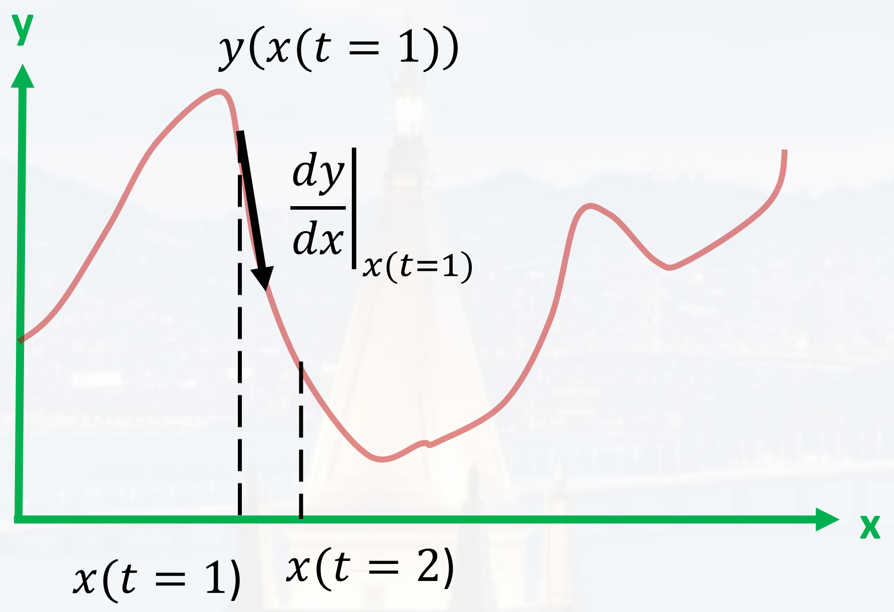
$$\left(\frac{\partial T}{\partial x} ; \frac{\partial T}{\partial y} ; \frac{\partial T}{\partial z}\right) := \text{gradient of } T$$

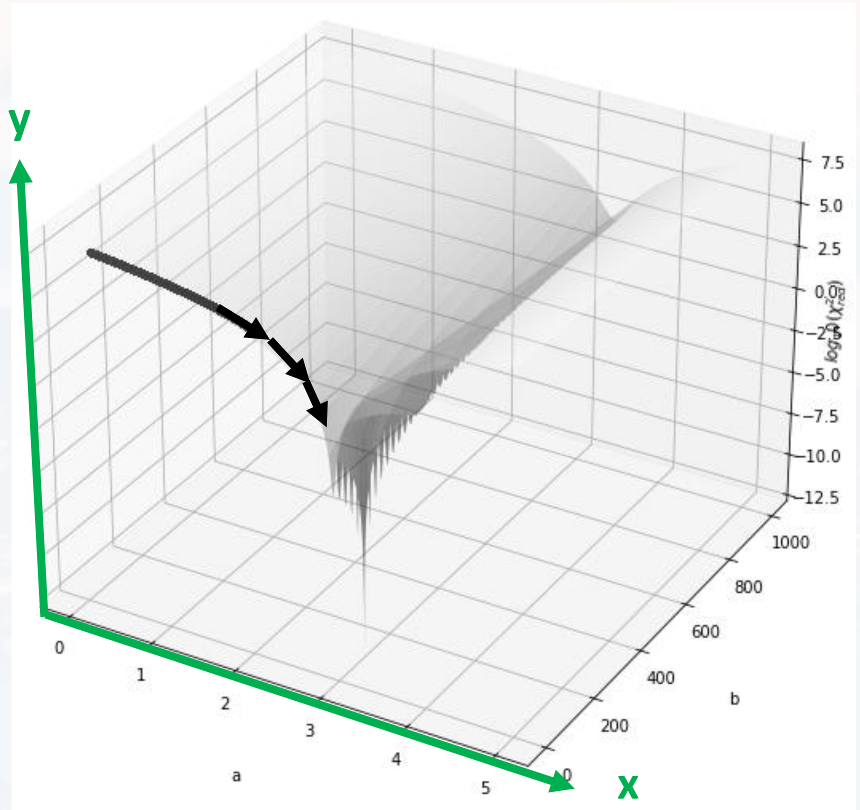
$$\text{grad } T = \begin{pmatrix} \frac{\partial T}{\partial x} \\ \frac{\partial T}{\partial y} \\ \frac{\partial T}{\partial z} \end{pmatrix} \equiv \vec{\nabla} T$$

Optimization Techniques:

Vanilla

$$\left.\frac{dy}{dx}\right|_{x_0} \approx \frac{y(x_0+\Delta x) - y(x_0-\Delta x)}{2\Delta x}$$





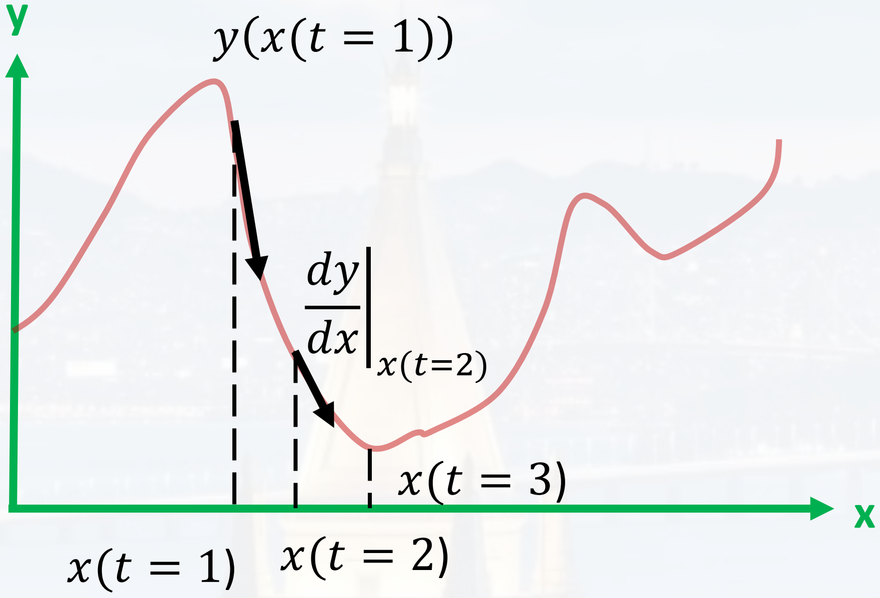
$$x(t=2) = x(t=1) - \varepsilon \left.\frac{dy}{dx}\right|_{x(t=1)}$$

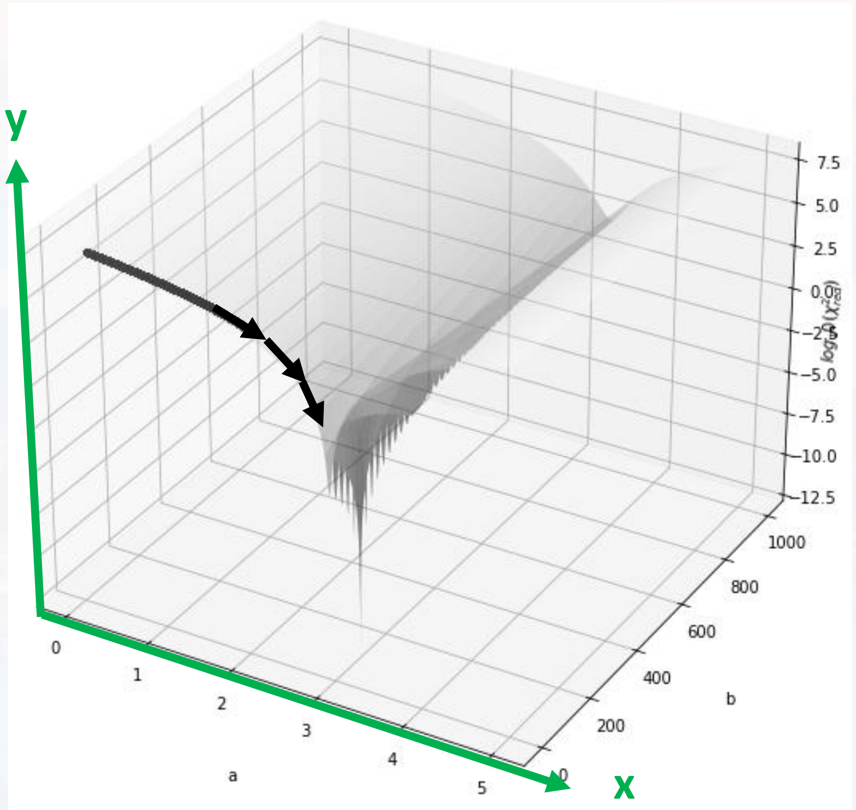
$\varepsilon > 0$

Optimization Techniques:

Vanilla

$$\left.\frac{dy}{dx}\right|_{x_0} \approx \frac{y(x_0 + \Delta x) - y(x_0 - \Delta x)}{2\Delta x}$$

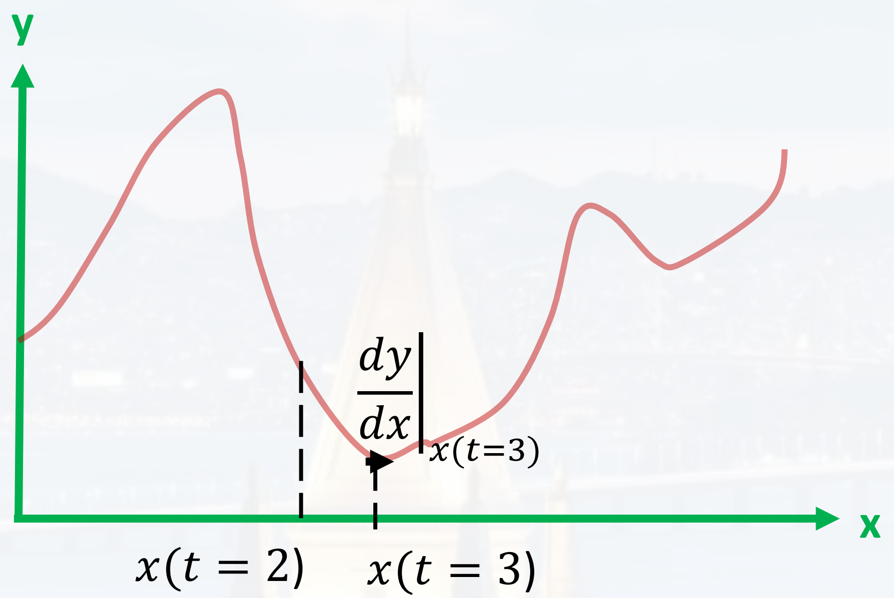


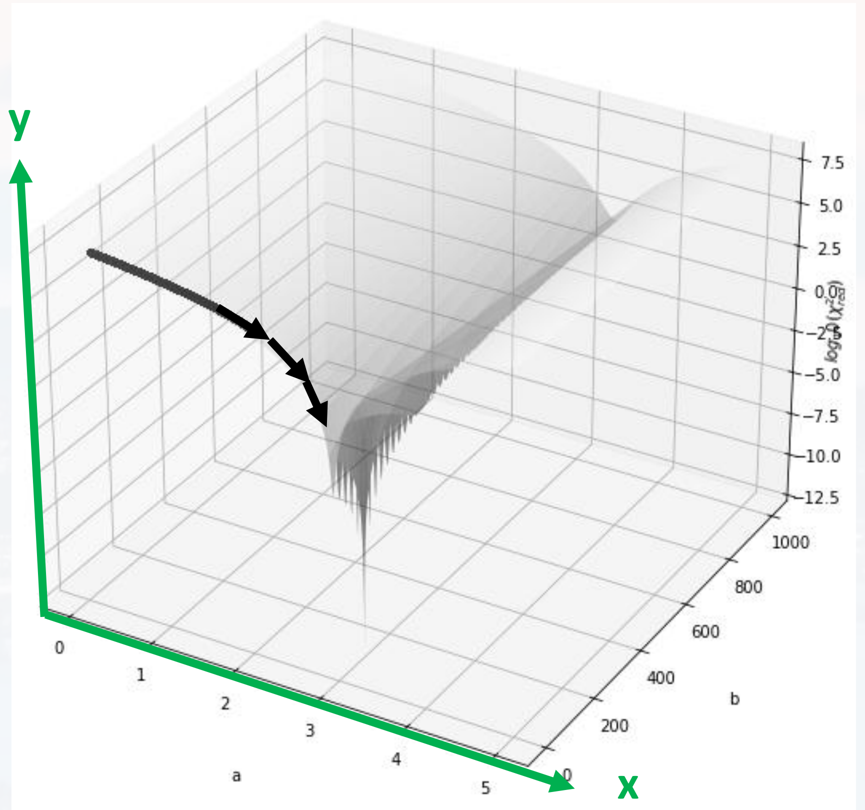


$$x(t = 3) = x(t = 2) - \varepsilon \left.\frac{dy}{dx}\right|_{x(t=2)}$$

$\varepsilon > 0$

$$\left.\frac{dy}{dx}\right|_{x_0} \approx \frac{y(x_0+\Delta x) - y(x_0-\Delta x)}{2\Delta x}$$



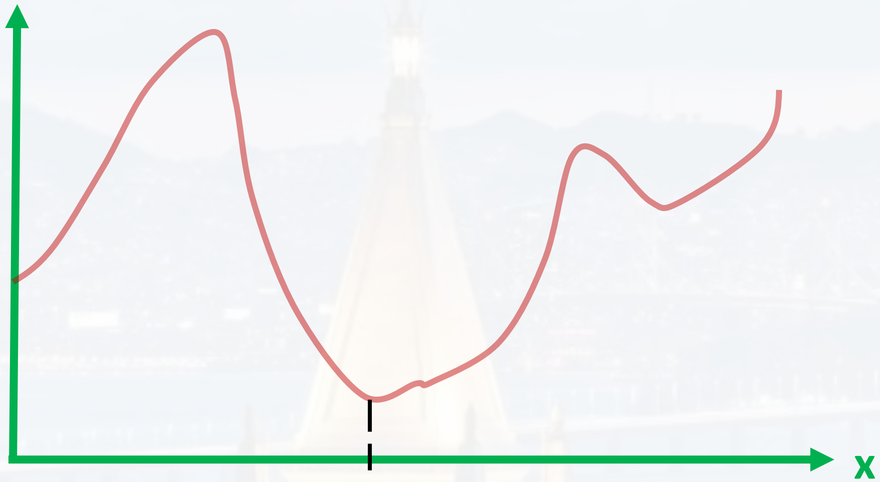


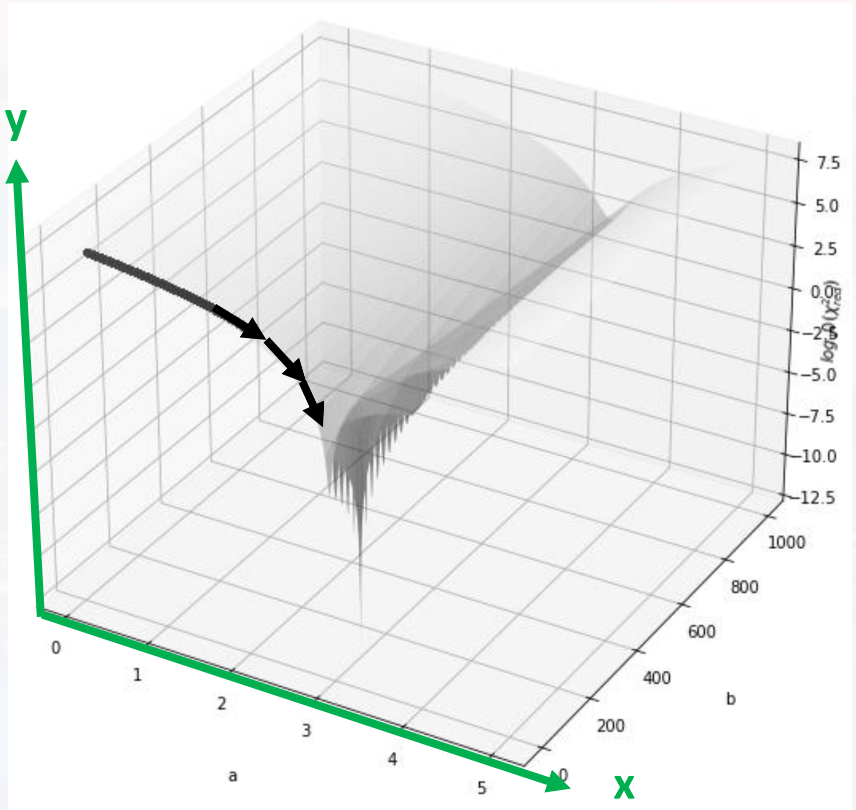
$$x(t=4) = x(t=3) - \varepsilon \left.\frac{dy}{dx}\right|_{x(t=3)}$$

$\varepsilon > 0$

Optimization Techniques:

$$\left.\frac{dy}{dx}\right|_{x_0} \approx \frac{y(x_0+\Delta x) - y(x_0-\Delta x)}{2\Delta x}$$



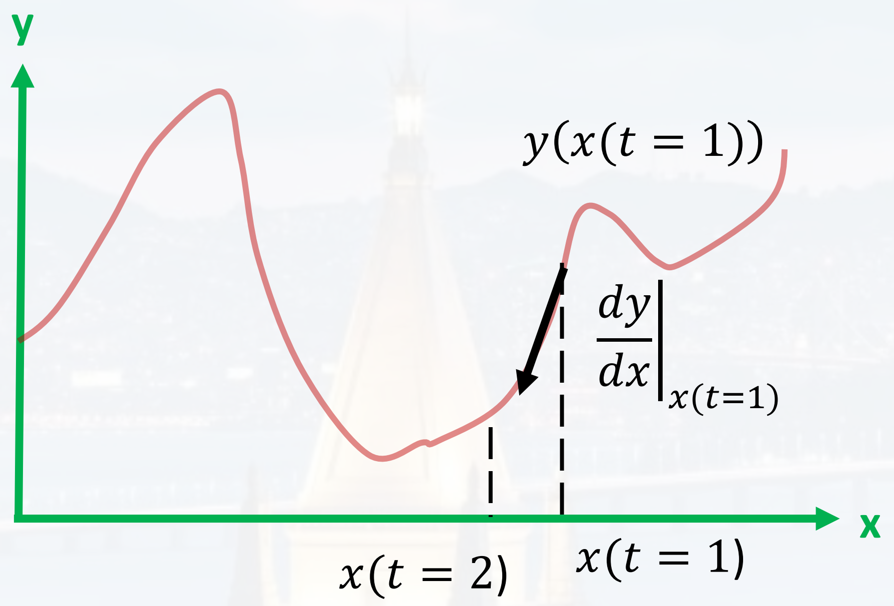


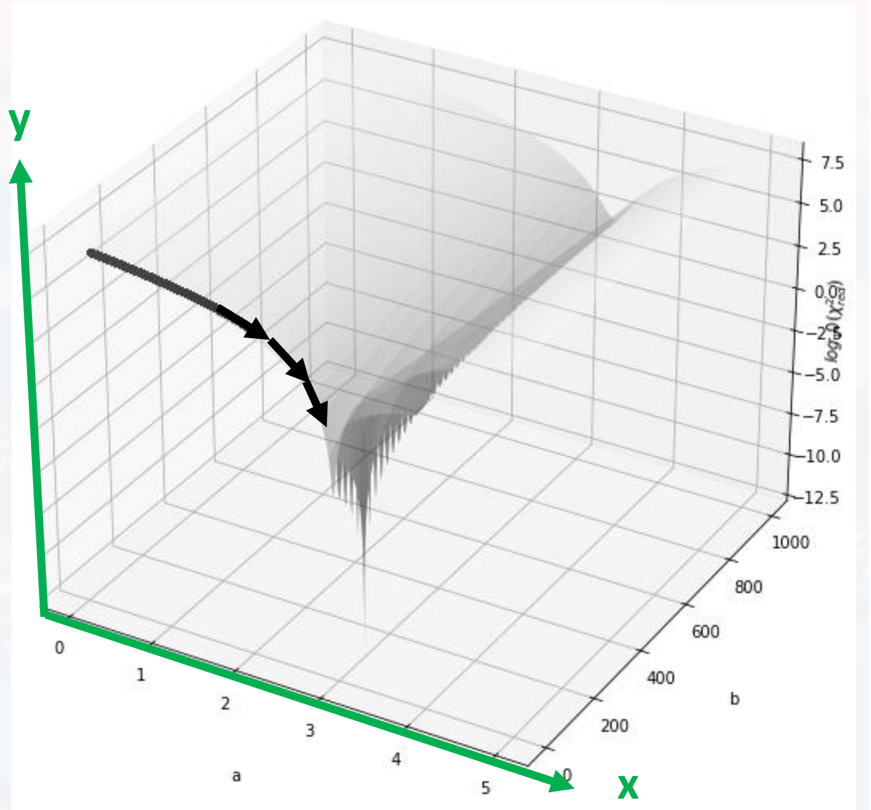
$$x(t=4) = x(t=3) - \varepsilon \left.\frac{dy}{dx}\right|_{x(t=3)}$$

ε > 0

Optimization Techniques:

$$\left.\frac{dy}{dx}\right|_{x_0} \approx \frac{y(x_0 + \Delta x) - y(x_0 - \Delta x)}{2\Delta x}$$





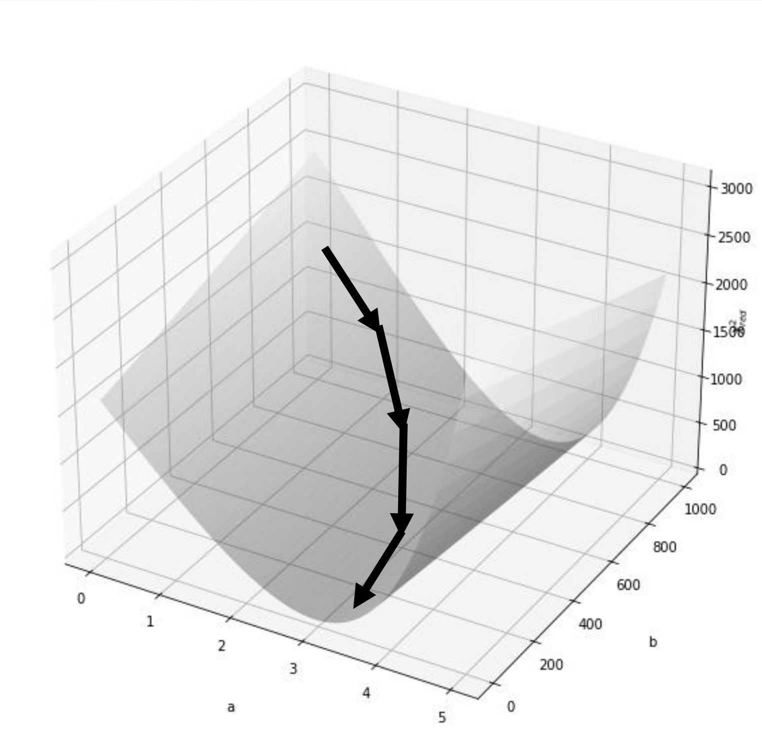
$$x(t=2) = x(t=1) - \varepsilon \left.\frac{dy}{dx}\right|_{x(t=1)}$$

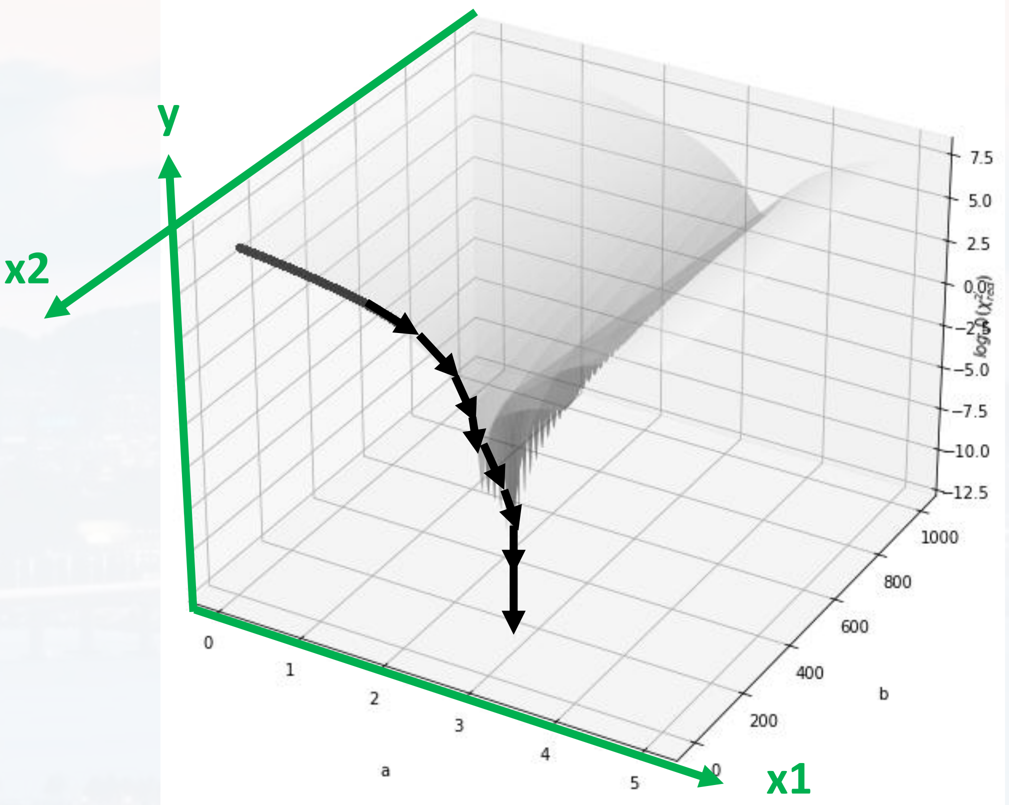
ε > 0

Optimization Techniques:

$$\left.\frac{\partial y}{\partial x_1}\right|_{x_1^*;\,x_2^*} \approx \frac{y(x_1^* + \Delta x_1,\ x_2^*) - y(x_1^* - \Delta x_1,\ x_2^*)}{2\Delta x_1}$$

$$\left.\frac{\partial y}{\partial x_2}\right|_{x_1^*;\,x_2^*} \approx \frac{y(x_1^*,\ x_2^* + \Delta x_2)\ - y(x_1^*,\ x_2^* - \Delta x_2)}{2\Delta x_2}$$





Optimization Techniques:

$$\left.\frac{\partial y}{\partial x_1}\right|_{x_1^*;\, x_2^*;\ldots;\, x_N^*} \approx \frac{y(x_1^* + \Delta x_1,\ x_2^*, \ldots, x_N^*) - y(x_1^* - \Delta x_1,\ x_2^*, \ldots, x_N^*)}{2\Delta x_1}$$

$$\left.\frac{\partial y}{\partial x_2}\right|_{x_1^*;\, x_2^*;\ldots;\, x_N^*} \approx \frac{y(x_1^*, x_2^* + \Delta x_2, \ldots, x_N^*) - y(x_1^*, x_2^* - \Delta x_2, \ldots, x_N^*)}{2\Delta x_2}$$

$$\vdots$$

$$\left.\frac{\partial y}{\partial x_i}\right|_{x_1^*;\, x_2^*;\ldots;\, x_N^*} \approx \frac{y(\ldots, x_i^* + \Delta x_i, \ldots, x_N^*) - y(\ldots, x_i^* - \Delta x_i, \ldots, x_N^*)}{2\Delta x_i}$$

$$\vdots$$

$$\left.\frac{\partial y}{\partial x_N}\right|_{x_1^*;\, x_2^*;\ldots;\, x_N^*} \approx \frac{y(x_1^*, x_2^*, \ldots, x_N^* + \Delta x_N) - y(x_1^*, x_2^*, \ldots, x_N^* - \Delta x_N)}{2\Delta x_N}$$

$$\begin{pmatrix} \left.\frac{\partial y}{\partial x_1}\right|_{x_1^*;\, x_2^*;\ldots;\, x_N^*} \\ \ldots \\ \left.\frac{\partial y}{\partial x_i}\right|_{x_1^*;\, x_2^*;\ldots;\, x_N^*} \\ \ldots \\ \left.\frac{\partial y}{\partial x_N}\right|_{x_1^*;\, x_2^*;\ldots;\, x_N^*} \end{pmatrix} = grad(y)_x$$

gradient of *y* wrt *x*

Optimization Techniques:

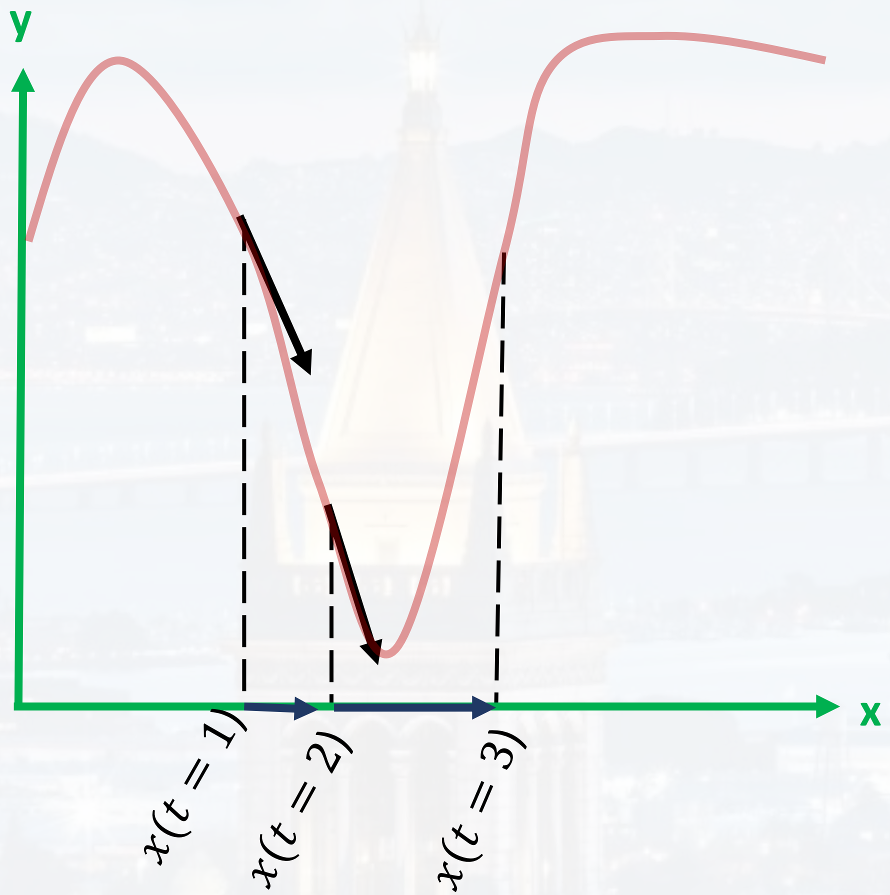
$$\left.\frac{dy}{dx}\right|_{x_0} \approx \frac{y(x_0+\Delta x) - y(x_0-\Delta x)}{2\Delta x}$$

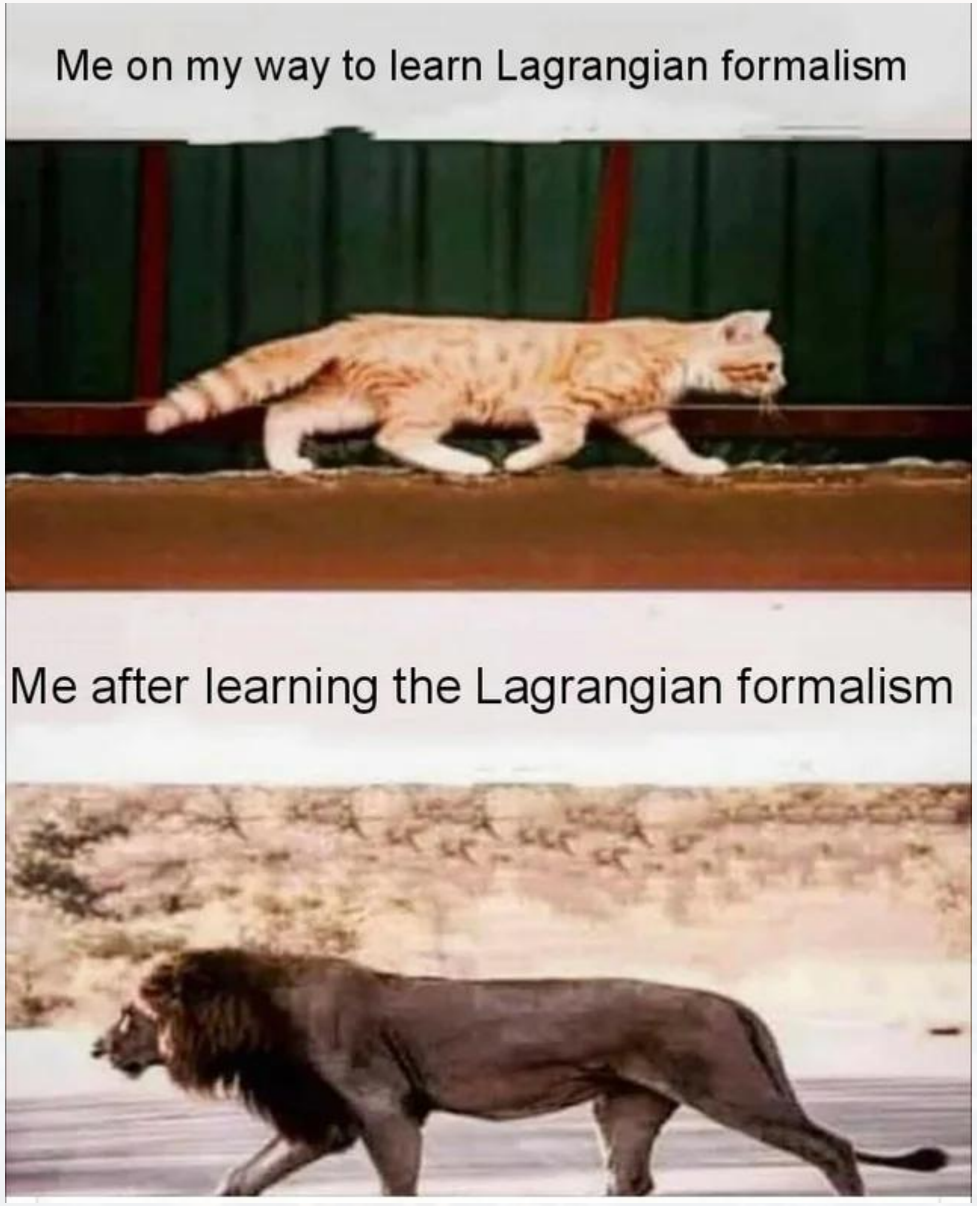
$$x(t+1) = x(t) - \boldsymbol{\varepsilon} \left.\frac{dy}{dx}\right|_{x(t)}$$

$\boldsymbol{\varepsilon} > 0$ called *learning rate*

$$\Delta x = -\,\boldsymbol{\varepsilon} \left.\frac{dy}{dx}\right|_{x(t)}$$

defines how large the leap Δx is





source: SKRyanrr

Outline

Lagrangian Multiplier

Gradient Descent Again

- Vanilla
- Learning Rate Schedule
- Momentum
- L1 and L2
- More Finetuning

Optimization Techniques:

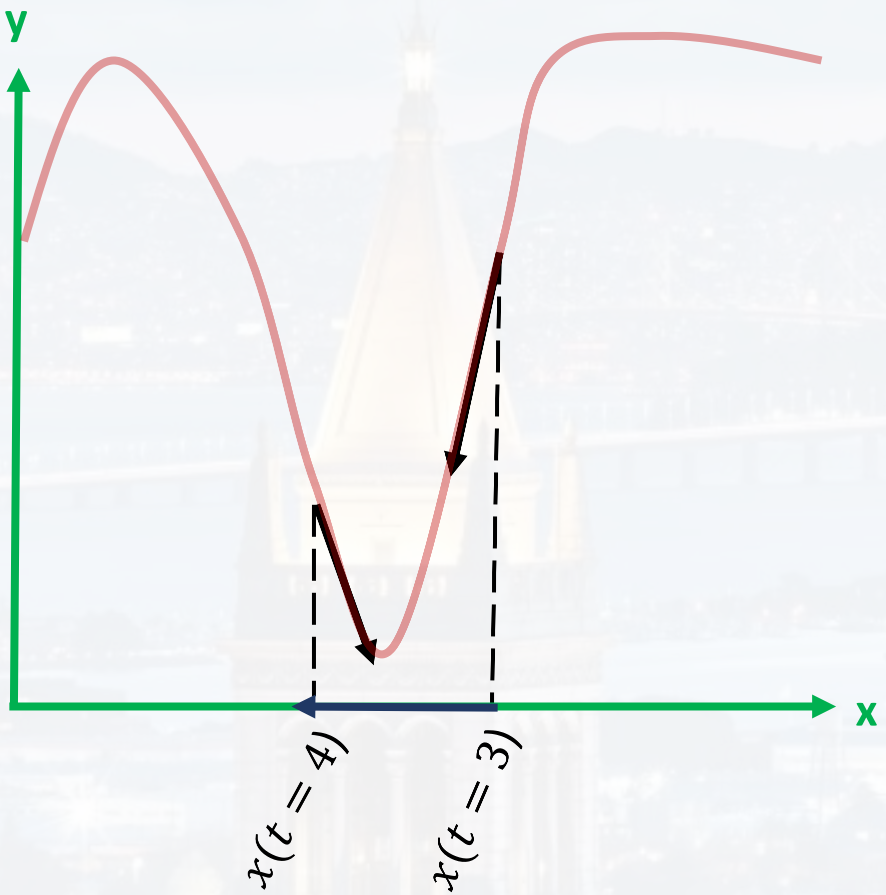
$$\left.\frac{dy}{dx}\right|_{x_0} \approx \frac{y(x_0+\Delta x) - y(x_0-\Delta x)}{2\Delta x}$$

$$x(t+1) = x(t) - \boldsymbol{\varepsilon} \left.\frac{dy}{dx}\right|_{x(t)}$$

$\boldsymbol{\varepsilon} > 0$ called *learning rate*

$$\Delta x = -\boldsymbol{\varepsilon} \left.\frac{dy}{dx}\right|_{x(t)}$$

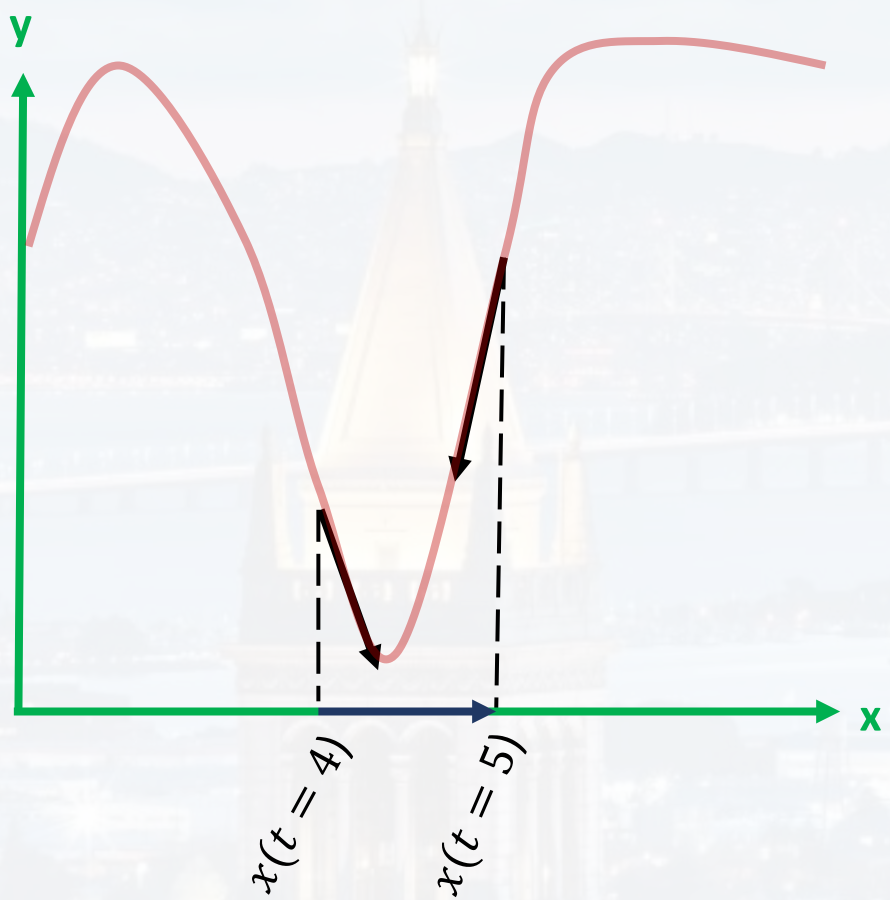
defines how large the leap Δx is



Optimization Techniques:

$$\left.\frac{dy}{dx}\right|_{x_0} \approx \frac{y(x_0+\Delta x) - y(x_0-\Delta x)}{2\Delta x}$$

$$x(t+1) = x(t) - \boldsymbol{\varepsilon}\left.\frac{dy}{dx}\right|_{x(t)}$$



$\boldsymbol{\varepsilon} > 0$ called *learning rate*

$$\Delta x = -\,\boldsymbol{\varepsilon}\left.\frac{dy}{dx}\right|_{x(t)}$$

defines how large the leap Δx is

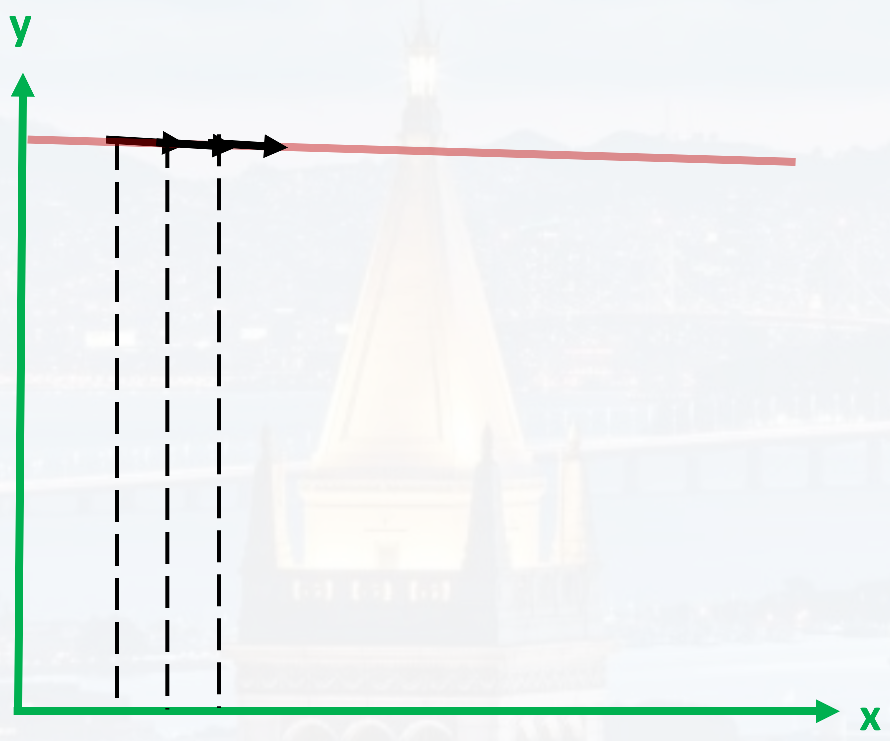
… and so on…

→ smaller $\boldsymbol{\varepsilon}$?

Optimization Techniques:

$$\left.\frac{dy}{dx}\right|_{x_0} \approx \frac{y(x_0+\Delta x) - y(x_0-\Delta x)}{2\Delta x}$$

$$x(t+1) = x(t) - \boldsymbol{\varepsilon} \left.\frac{dy}{dx}\right|_{x(t)}$$



$\boldsymbol{\varepsilon} > 0$ called *learning rate*

$$\Delta x = -\boldsymbol{\varepsilon} \left.\frac{dy}{dx}\right|_{x(t)}$$

defines how large the leap Δx is

… and so on…

→ smaller $\boldsymbol{\varepsilon}$?

Takes too long!

Optimization Techniques:

$$\left.\frac{dy}{dx}\right|_{x_0} \approx \frac{y(x_0+\Delta x) - y(x_0-\Delta x)}{2\Delta x}$$

$$x(t+1) = x(t) - \boldsymbol{\varepsilon} \left.\frac{dy}{dx}\right|_{x(t)}$$

<u>learning rate as function of t:</u>

$\boldsymbol{\varepsilon} > 0$ called *learning rate*

$$\boldsymbol{\varepsilon}(t) = \frac{\boldsymbol{\varepsilon}_0}{1 + \kappa\, t}$$

decay rate κ

$$\Delta x = -\boldsymbol{\varepsilon} \left.\frac{dy}{dx}\right|_{x(t)}$$

defines how large the leap Δx is

Optimization Techniques:

$$\left.\frac{dy}{dx}\right|_{x_0} \approx \frac{y(x_0 + \Delta x) - y(x_0 - \Delta x)}{2\Delta x}$$

$$x(t+1) = x(t) - \boldsymbol{\varepsilon} \left.\frac{dy}{dx}\right|_{x(t)}$$

$\boldsymbol{\varepsilon} > 0$ called *learning rate*

$$\Delta x = -\,\boldsymbol{\varepsilon} \left.\frac{dy}{dx}\right|_{x(t)}$$

defines how large the leap Δx is

<u>learning rate as function of t:</u>

$$\boldsymbol{\varepsilon}(t) = \frac{\boldsymbol{\varepsilon}_0}{1 + \kappa\, t}$$

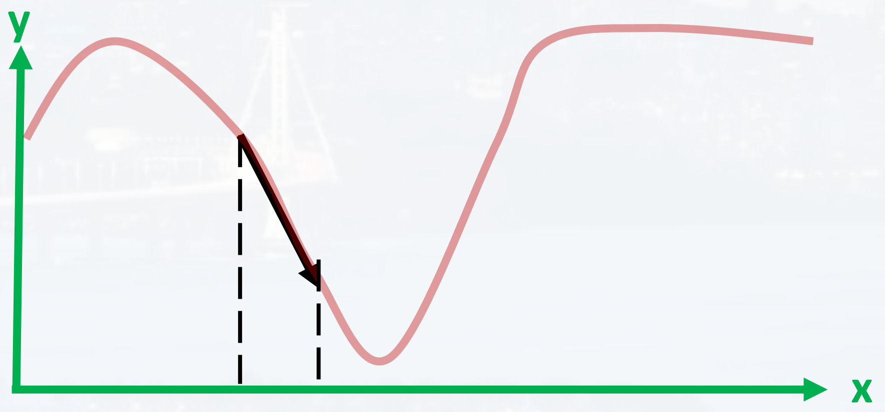
decay rate κ

can also be a stepwise function (learning rate schedule) learning rate as function of t:

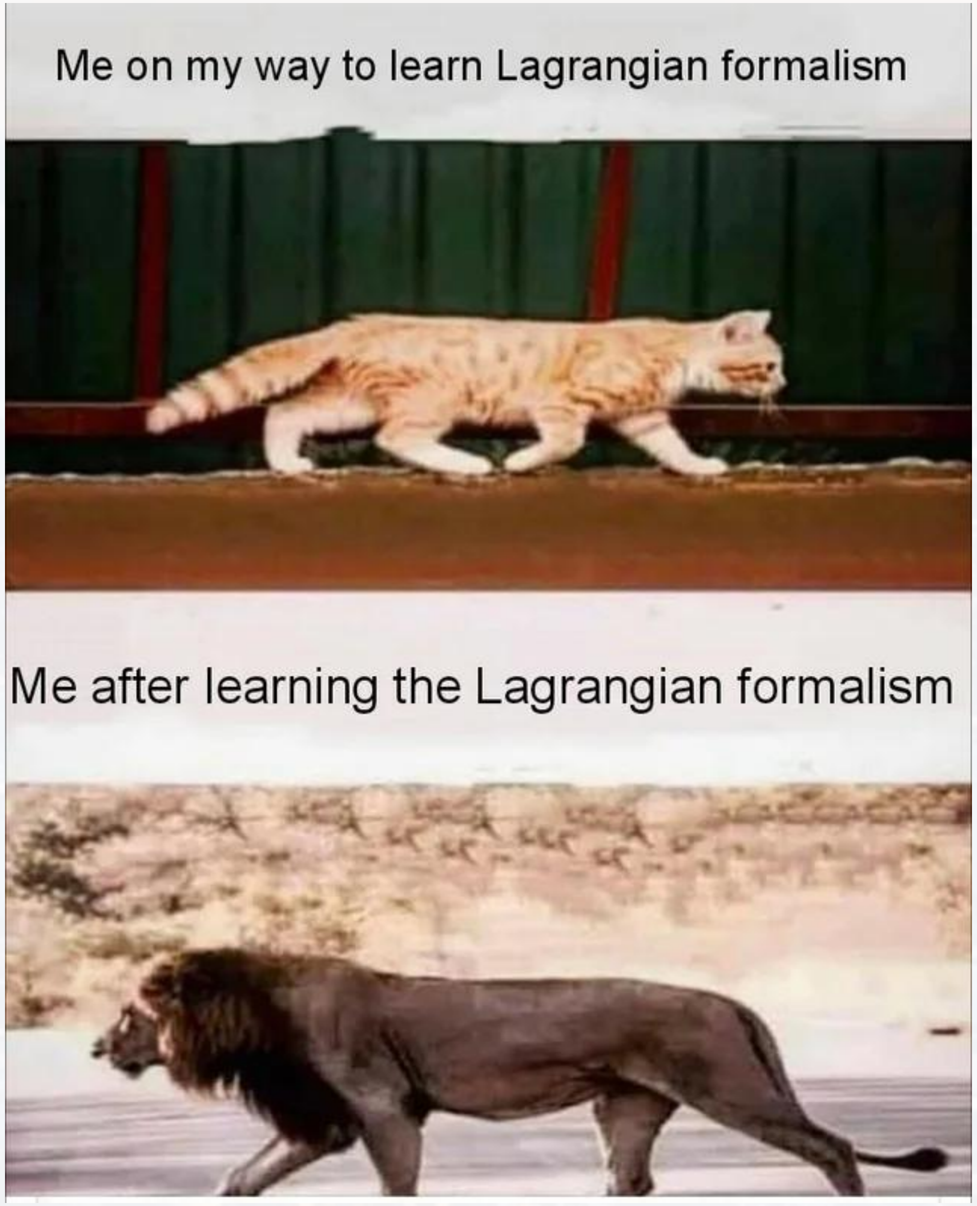
$$\varepsilon(t) = \frac{\varepsilon_0}{1 + \kappa t}$$ decay rate κ

$$\Delta x = -\varepsilon \frac{dy}{dx}\bigg|_{x(t)}$$ defines how large the leap Δx is

can also be a stepwise function (learning rate schedule)



$$\varepsilon \rightarrow \frac{\varepsilon}{\sqrt{grad(y)_x}}$$ adaptive gradient, aka **AdaGrad**



source: SKRyanrr

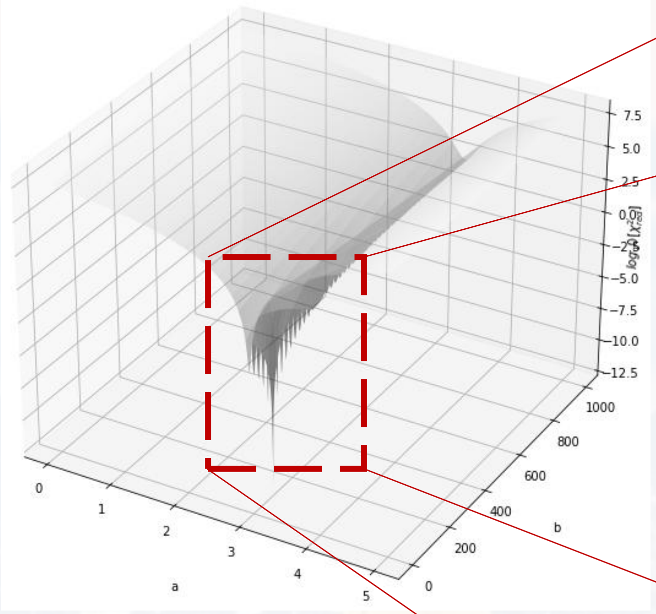
Outline

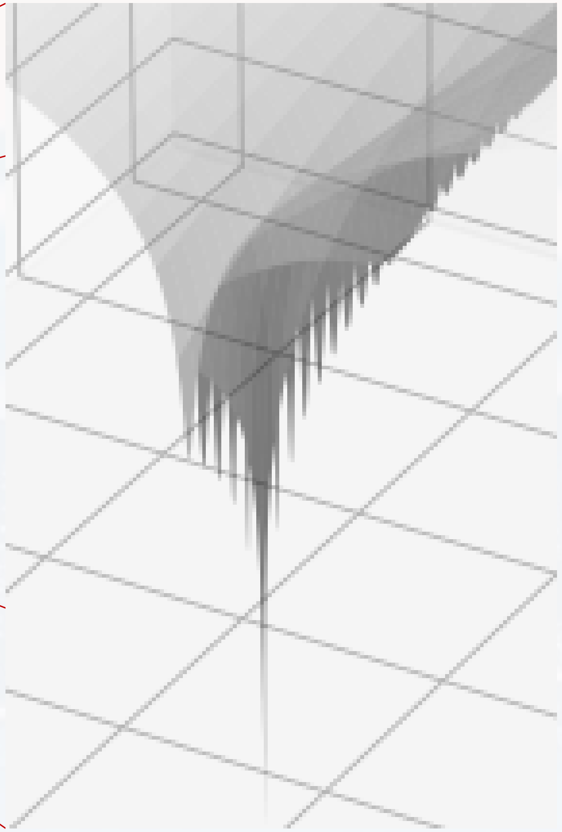
Lagrangian Multiplier

Gradient Descent Again

- Vanilla
- Learning Rate Schedule
- Momentum
- L1 and L2
- More Finetuning

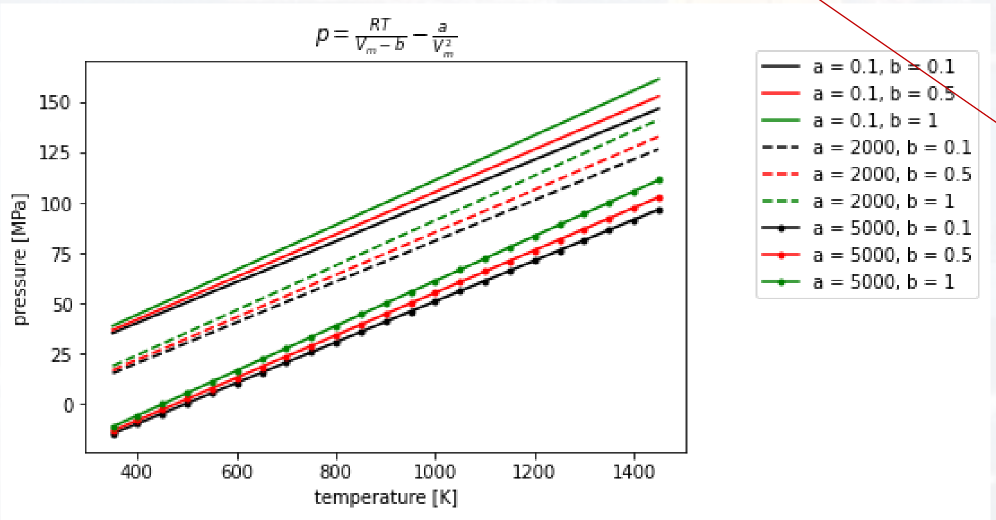
Optimization Techniques:

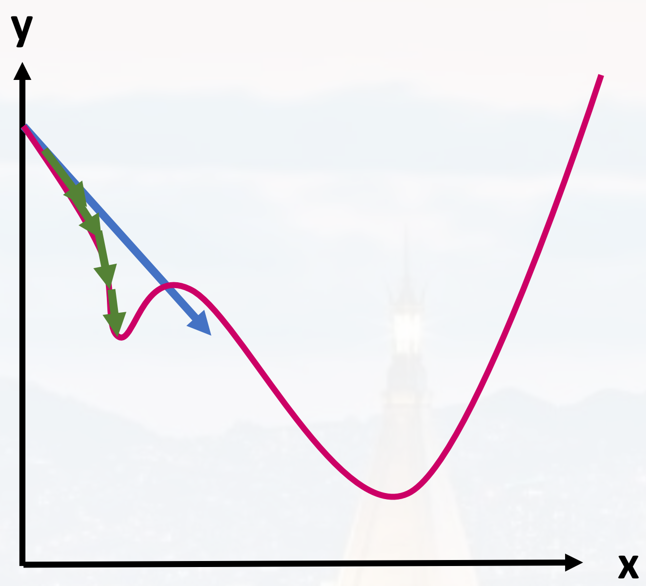




even with AdaGrad and learning rate schedule
→ would get stuck in local minimum

need to roll over → **momentum**





taking the **average** of ***N*** previous gradients

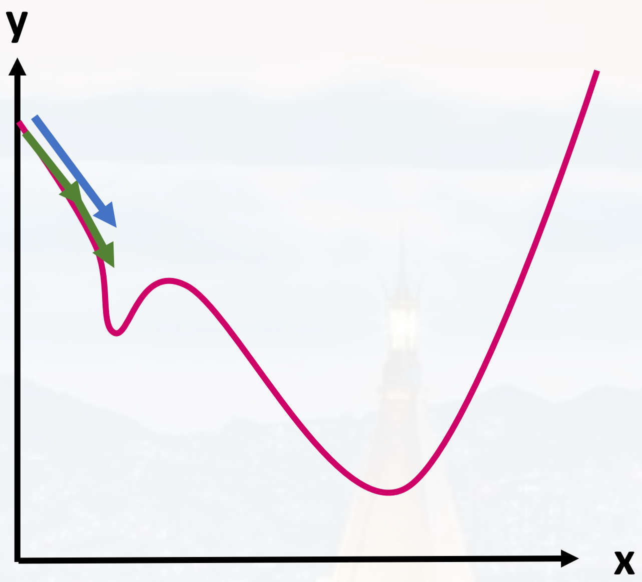
$$\langle grad(y)_{x(t)} \rangle = \frac{1}{N}[grad(y)_{x(t-1)} + grad(y)_{x(t-2)} + \ldots + grad(y)_{x(t-N)}]$$

but we want more recent gradients to contribute more than older gradients

→ **weighted average** with weighting factor $\boldsymbol{\mu_k}$

$$\langle grad(y)_{x(t)} \rangle = \sum_{k=t-N}^{t-1} \mu_k \cdot grad(y)_{x(k)}$$

Finding a clever way to adjust $\boldsymbol{\mu_k}$ during every iteration *t*



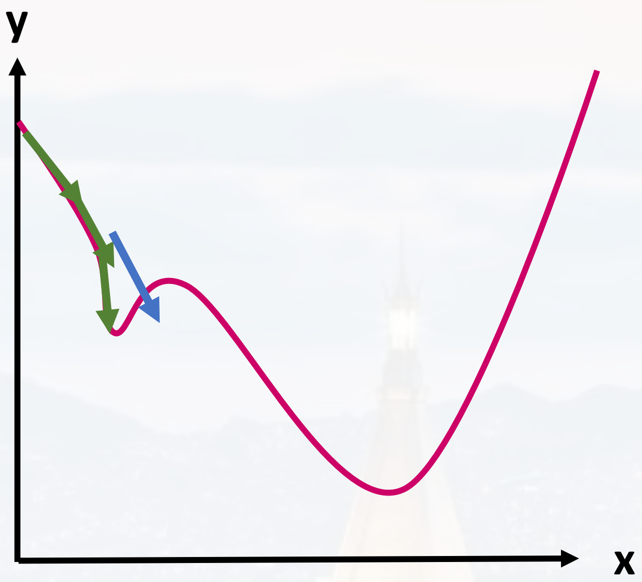
weighted average with weighting factor $\boldsymbol{\mu_k}$

Finding a clever way to adjust $\boldsymbol{\mu_k}$ during every iteration t

$$\langle grad(y)_{x(0)} \rangle = grad(y)_{x(0)} \qquad \mu_0 = (0,1)$$

$$\langle \boldsymbol{grad(y)_{x(1)}} \rangle = grad(y)_{x(1)} + \mu_0 \cdot grad(y)_{x(0)}$$

Optimization Techniques:



weighted average with weighting factor $\boldsymbol{\mu_k}$

Finding a clever way to adjust $\boldsymbol{\mu_k}$ during every iteration t

$$\langle grad(y)_{x(0)} \rangle = grad(y)_{x(0)} \qquad \mu_0 = (0,1)$$

$$\langle \boldsymbol{grad(y)_{x(1)}} \rangle = grad(y)_{x(1)} + \mu_0 \cdot grad(y)_{x(0)}$$

$$\langle \boldsymbol{grad(y)_{x(2)}} \rangle = grad(y)_{x(2)} + \mu_0 \left[grad(y)_{x(1)} + \mu_0 \, grad(y)_{x(0)} \right]$$

$$\boldsymbol{\mu_{k=2}} = \mu_0 \, \mu_0 = \mu_0^2$$

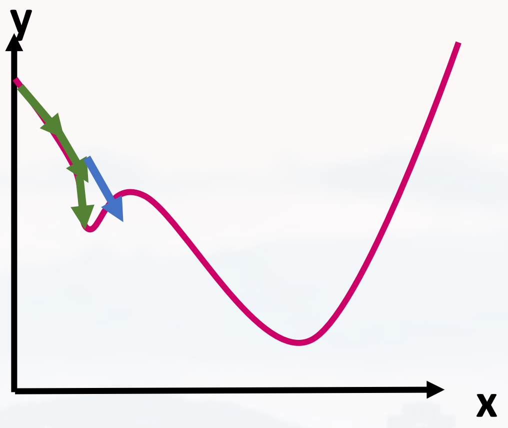
$$\langle \boldsymbol{grad(y)_{x(3)}} \rangle = grad(y)_{x(3)} + \boxed{\mu_0} \left[grad(y)_{x(2)} + \boxed{\mu_0} \left[grad(y)_{x(1)} + \boxed{\mu_0} \cdot grad(y)_{x(0)} \right] \right]$$

… and so on…

Optimization Techniques:

weighted average with weighting factor $\boldsymbol{\mu_k}$

$\mu_0 = (0,1)$ called ''momentum'



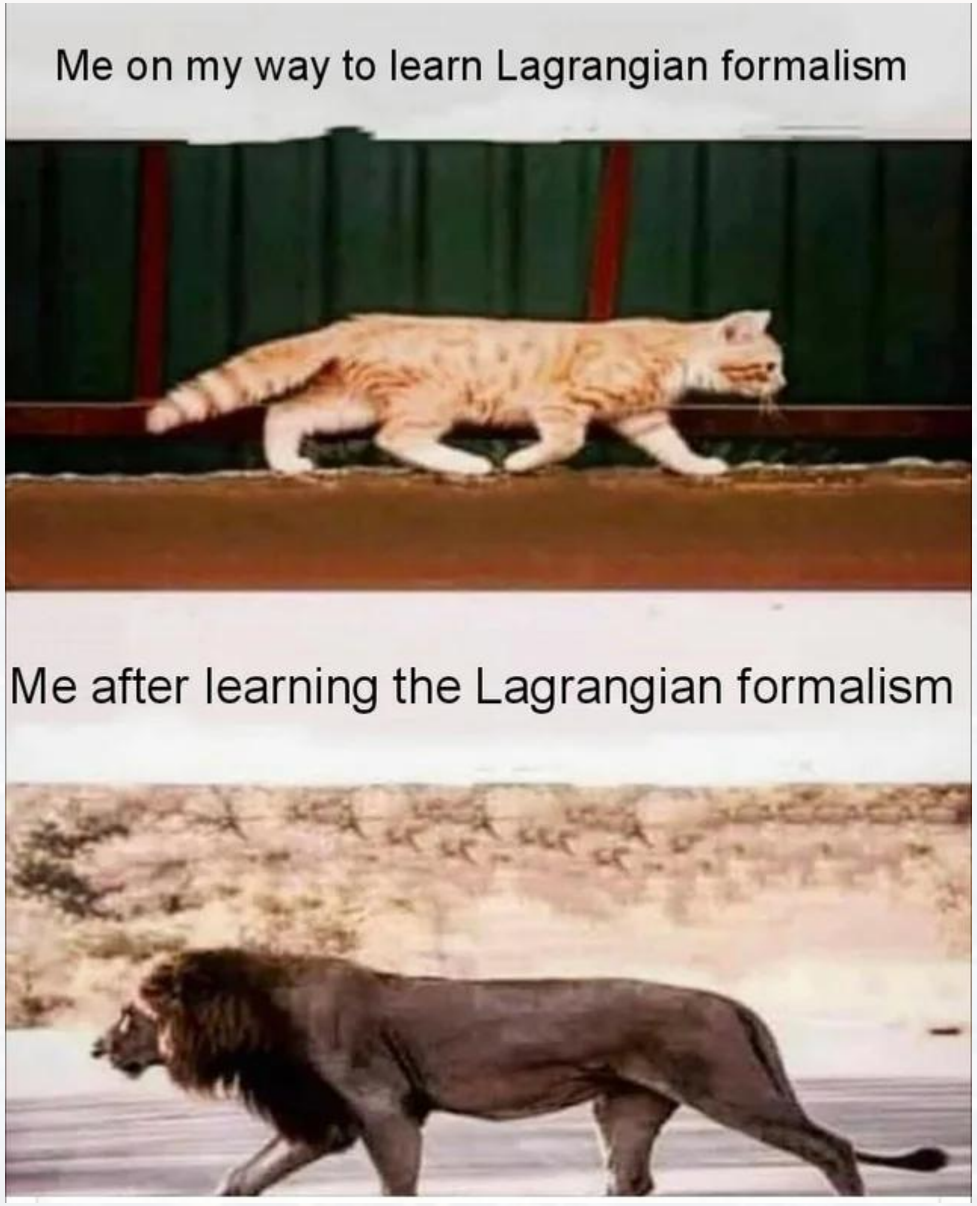
$$\langle \boldsymbol{grad(y)_{x(3)}} \rangle = grad(y)_{x(3)} \; +$$

$$\mu_0 \left[grad(y)_{x(2)} + \; \mu_0 \left[grad(y)_{x(1)} + \mu_0 \cdot grad(y)_{x(0)} \right] \right]$$

... and so on...

```
class Optimizer:

     def __init__(self, learning_rate = 0.1, decay = 0, momentum = 0):
       self.learning_rate         = learning_rate
       self.decay                 = decay
       self.current_learning_rate = learning_rate
       self.iterations            = 0
       self.momentum              = momentum
```



source: SKRyanrr

Outline

Lagrangian Multiplier

Gradient Descent Again

- Vanilla
- Learning Rate Schedule
- Momentum
- L1 and L2
- More Finetuning

Optimization Techniques:

Any algorithm needs a “goal” aka **objective function** that has to be ***optimized*** (finding an **extreme**)

Often, the extreme of the objective function is subject to **constrains**

sometimes we have some **prior knowledge** about the **independent variables**

recall: linear regression

finding best β by

$$\min_{\beta}\left\{\frac{1}{N}\|Y - X\beta\|^2\right\}$$

now:
constrain: **encourages sparsity of $\boldsymbol{\beta}$**

$$\min_{\beta}\left\{\frac{1}{N}\|Y - X\beta\|^2 + \boldsymbol{\lambda}\|\boldsymbol{\beta}\|^{\mathbf{1}}\right\}$$

the Lagrangian
$\boldsymbol{L}(\boldsymbol{X}, \boldsymbol{Y}, \lambda)$

λ *Lagrangian Multiplier*

called **L1 regularization**, or LASSO

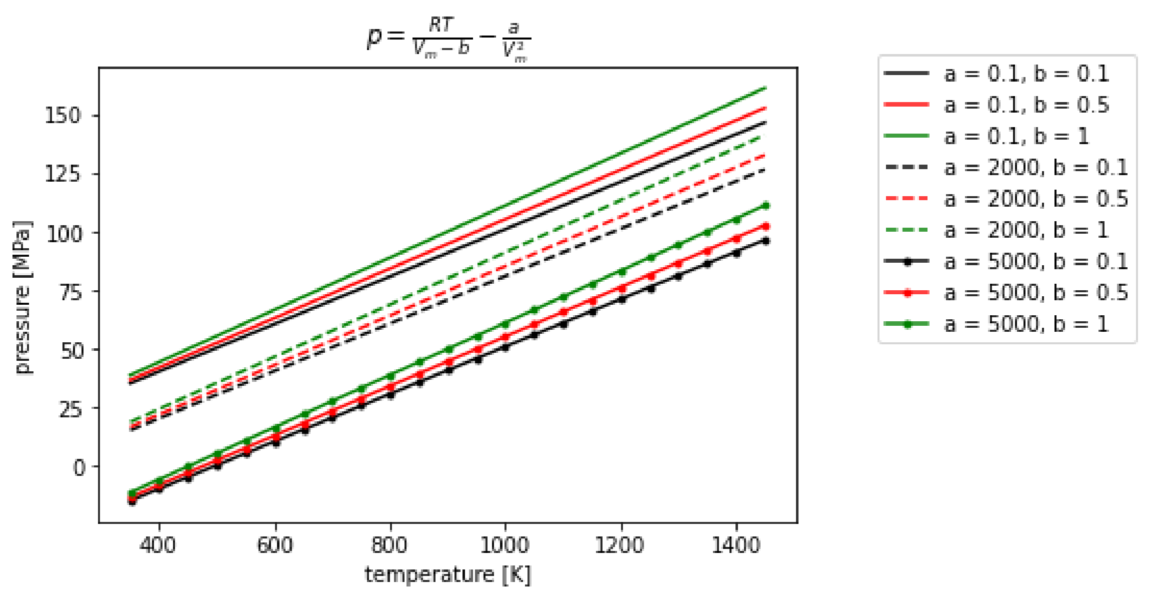
Optimization Techniques:

Any algorithm needs a “goal” aka **objective function** that has to be ***optimized*** (finding an **extreme**)

Often, the extreme of the objective function is subject to **constrains**

sometimes we have some **prior knowledge** about the **independent variables**

L1 regularization



We often have even hard constrains based on the laws of physics!

Optimization Techniques:

Any algorithm needs a “goal” aka **objective function** that has to be ***optimized*** (finding an **extreme**)

Often, the extreme of the objective function is subject to **constrains**

sometimes we have some **prior knowledge** about the **independent variables**

recall: linear regression

finding best β by

$$\min_{\beta} \left\{ \frac{1}{N} \|Y - X\beta\|^2 \right\}$$

now:

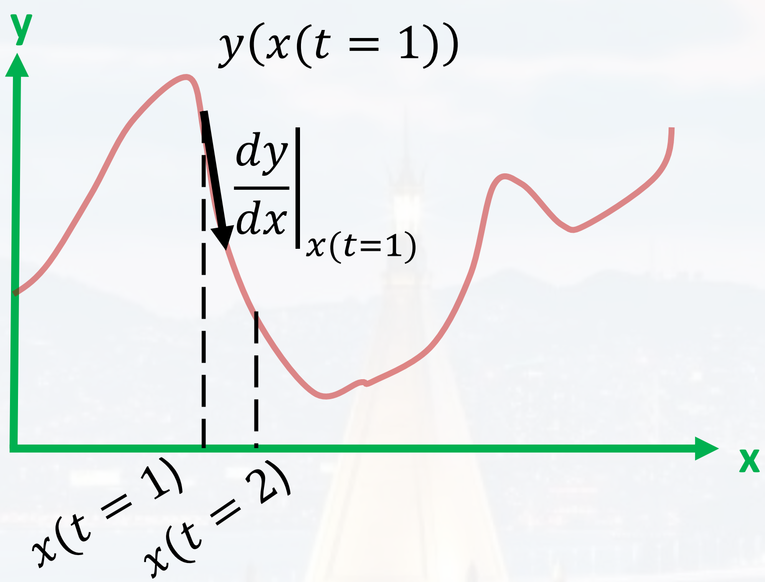
$$\hat{\beta} = (X^T X + \boldsymbol{\lambda I})^{-1} X^T Y \longrightarrow \min_{\beta} \left\{ \frac{1}{N} \|Y - X\beta\|^2 + \boldsymbol{\lambda} \|\boldsymbol{\beta}\|^2 \right\}$$

the Lagrangian
$\boldsymbol{L}(\boldsymbol{X}, \boldsymbol{Y}, \lambda)$

λ *Lagrangian Multiplier*

called **L2 regularization**, or RIDGE penalizes large β

L1 and L2 regularization

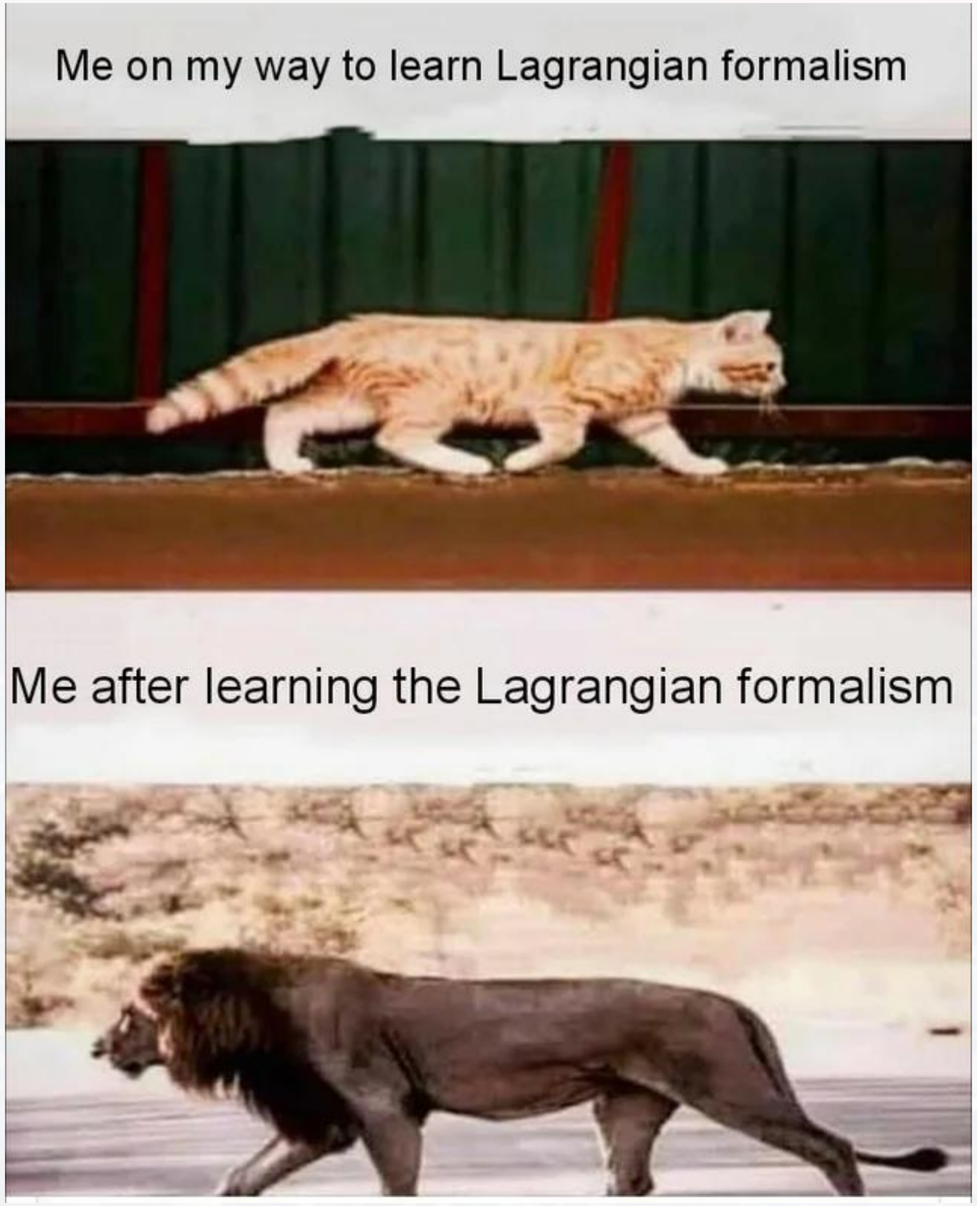


$$x(t=2) = x(t=1) - \varepsilon \left.\frac{d[y + \lambda_1 \|x\|^1 + \lambda_2 \|x\|^2]}{dx}\right|_{x(t=1)}$$

$$x(t=2) = x(t=1) - \varepsilon \left.\frac{dy}{dx}\right|_{x(t=1)}$$

$$- \varepsilon \left.\frac{\lambda_1 d\|x\|^1}{dx}\right|_{x(t=1)} \quad - \varepsilon \left.\frac{\lambda_2 d\|x\|^2}{dx}\right|_{x(t=1)}$$

- gradient descent does not stop if values for x are too large and prefers sparsity
- note: the derivative of $\|x\|^1$ returns the sign (i. e. direction)
- usually $\boldsymbol{\lambda} \ll \|x\|^n$
- will be important for ANNs later



source: SKRyanrr

Outline

Lagrangian Multiplier

Gradient Descent Again

- Vanilla
- Learning Rate Schedule
- Momentum
- L1 and L2
- More Finetuning

Optimization Techniques:

Vanilla Gradient Descent → **S**tochastic **G**radient **D**escent

↓

Learning Rate Schedule, L1, L2

different scaling for all different direction

↓

Momentum

↓

$\varepsilon \rightarrow \frac{\varepsilon}{\sqrt{grad(y)_x}}$ adaptive gradient, aka **AdaGrad**

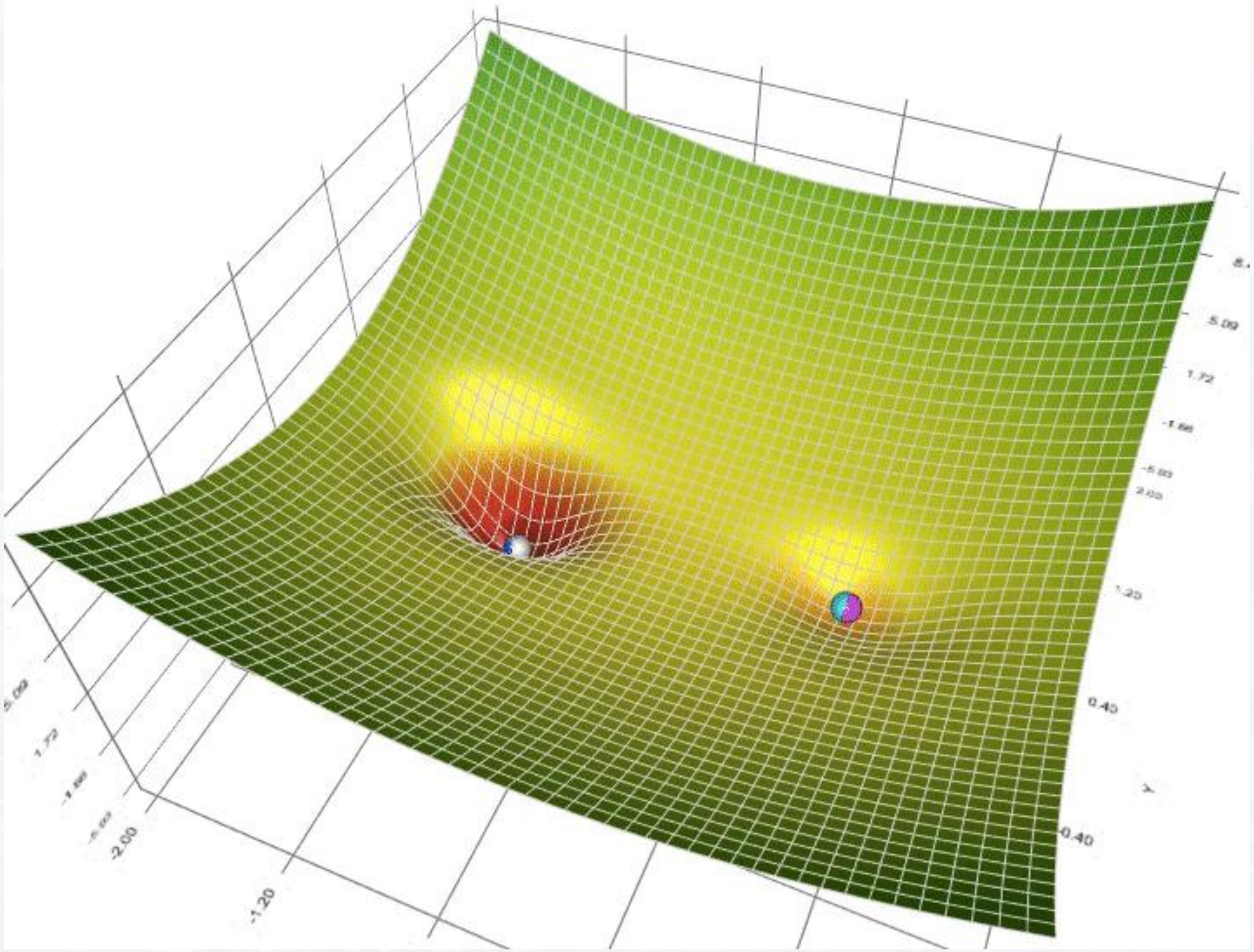
↓

Multiplying a decay factor to the sum of gradient squared (similar to momentum), aka **R**oot **M**ean **S**quare **Prop**agation **RMSProp**

all combined:
Adaptive **M**oment Estimation
aka **Adam**

Lili Jiang

[TowardsDataScience]

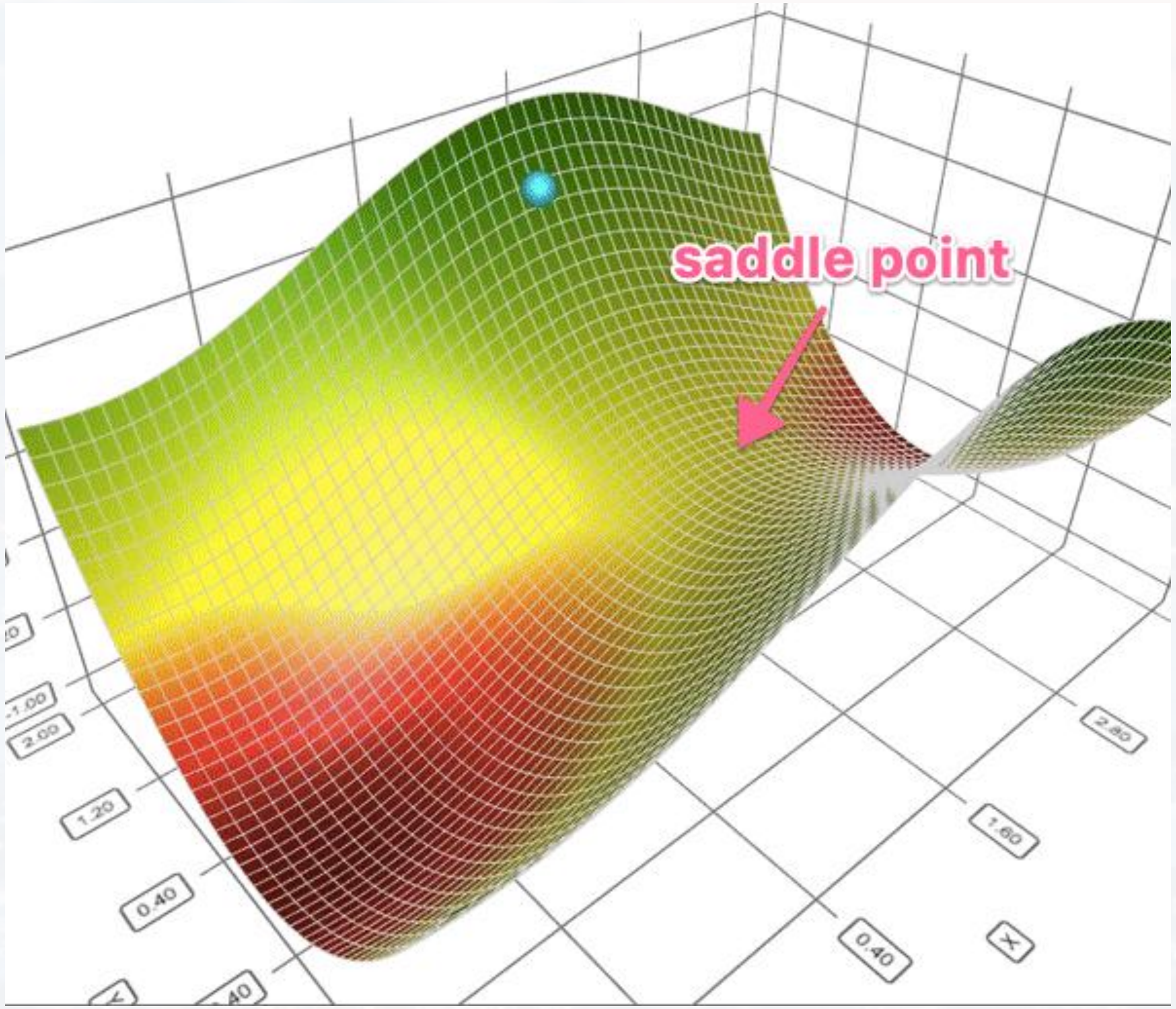


gradient descent (cyan),
momentum (magenta),
AdaGrad (white),
RMSProp (green),
Adam (blue)

Optimization Techniques:

Lili Jiang

[TowardsDataScience]



gradient descent (cyan),
momentum (magenta),
AdaGrad (white),
RMSProp (green),
Adam (blue)

Thank you very much for your attention!