Lecture 06:

Optimization

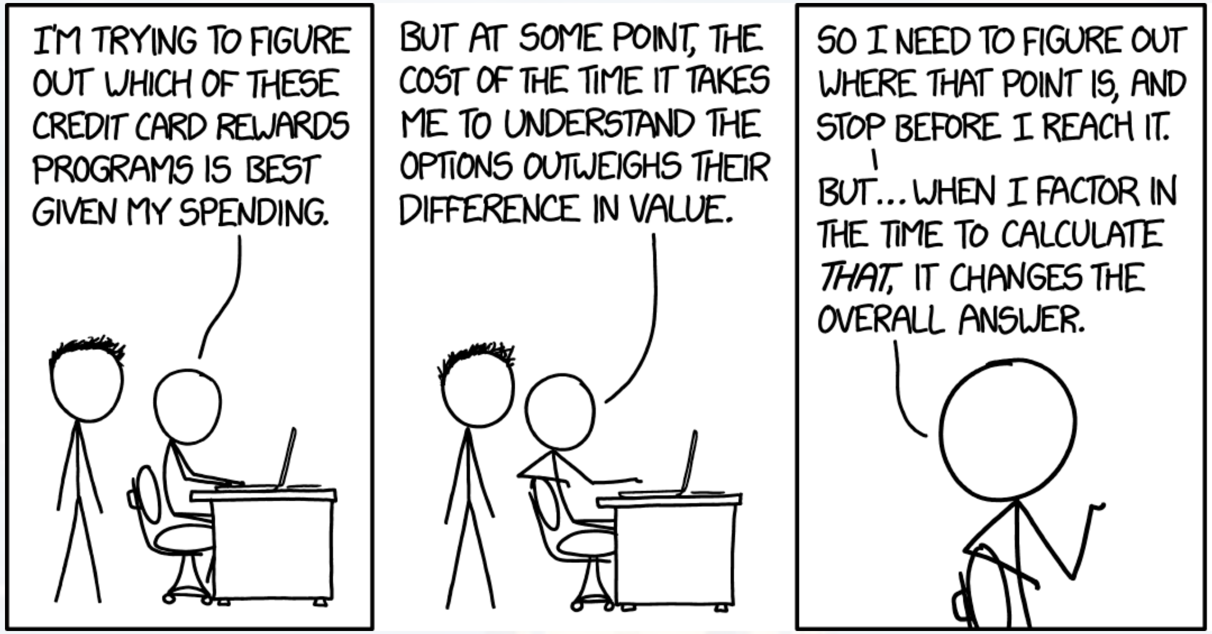
Markus Hohle

University California, Berkeley

Machine Learning Algorithms

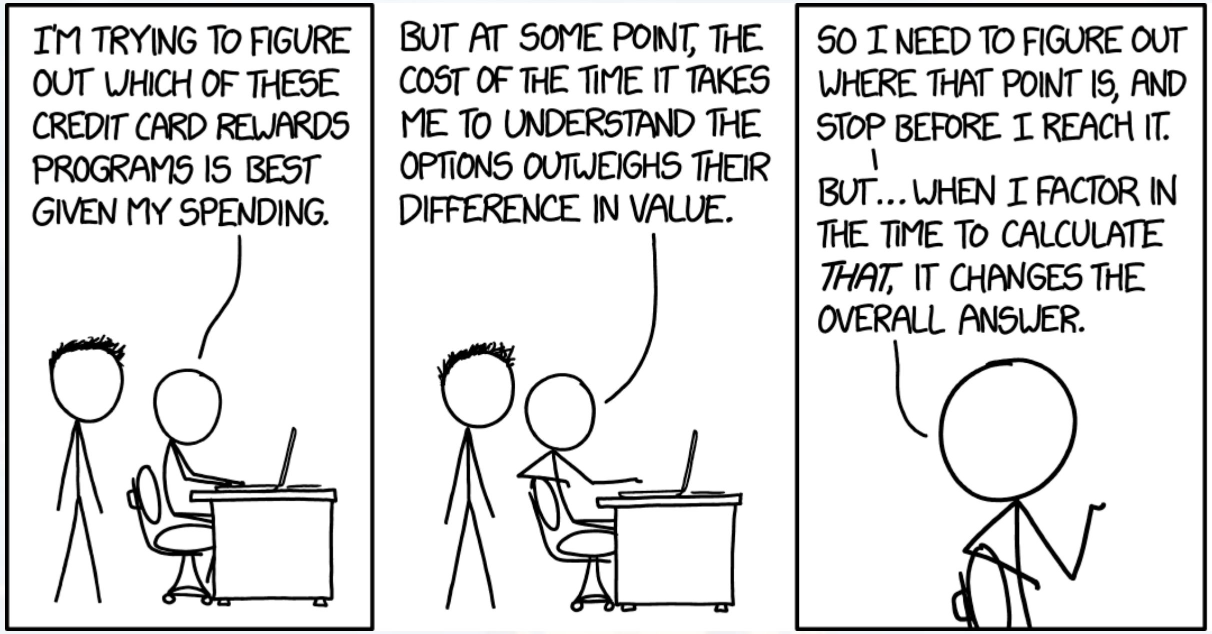
MSSE 277B, 3 Units

Fall 2024



Outline

- **The Problem**
- **Gradient Descent**
  - Vanilla
  - Learning Rate Schedule
  - Momentum
  - L1 and L2
  - More Finetuning



Outline

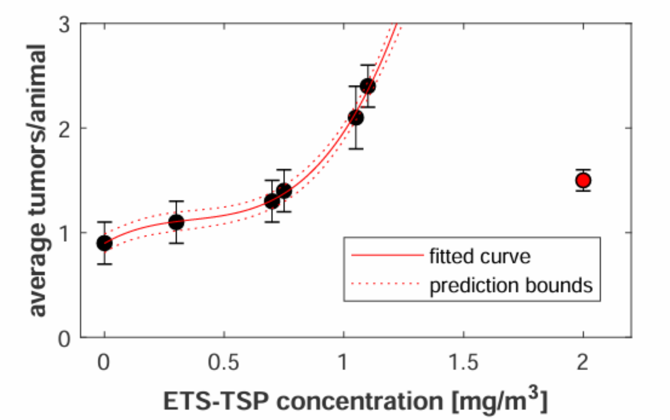
- The Problem

- Gradient Descent

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

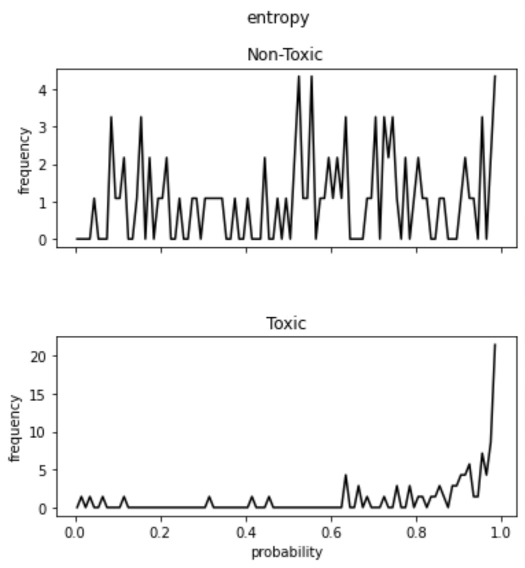
Any algorithm needs a “goal” aka **objective function** that has to be ***optimized*** (finding an **extreme**)

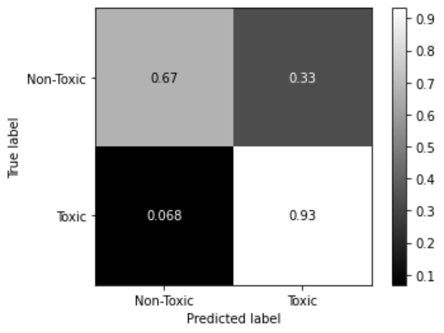
regression, e. g. curve fitting



minimize: $$\chi^2_{red} = \frac{1}{N - p - 1} \sum_{i=1}^{N} \frac{(\bar{y}(model)_i - y_i)^2}{\sigma_i^2}$$

classification



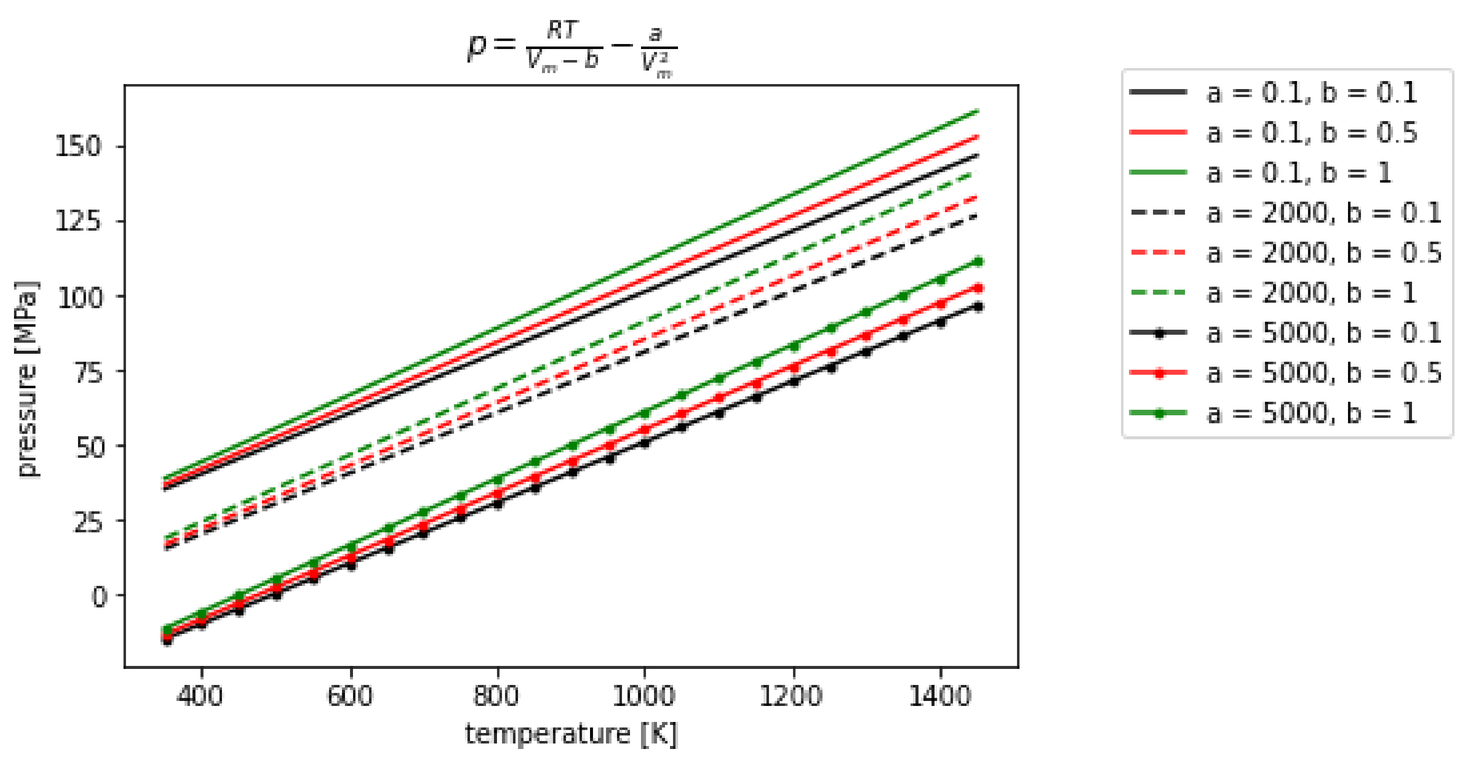


maximize: accuracy

minimize: cross entropy $$S = -\sum_i p(true)_i \cdot \ln p(model)_i$$

Optimization:

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



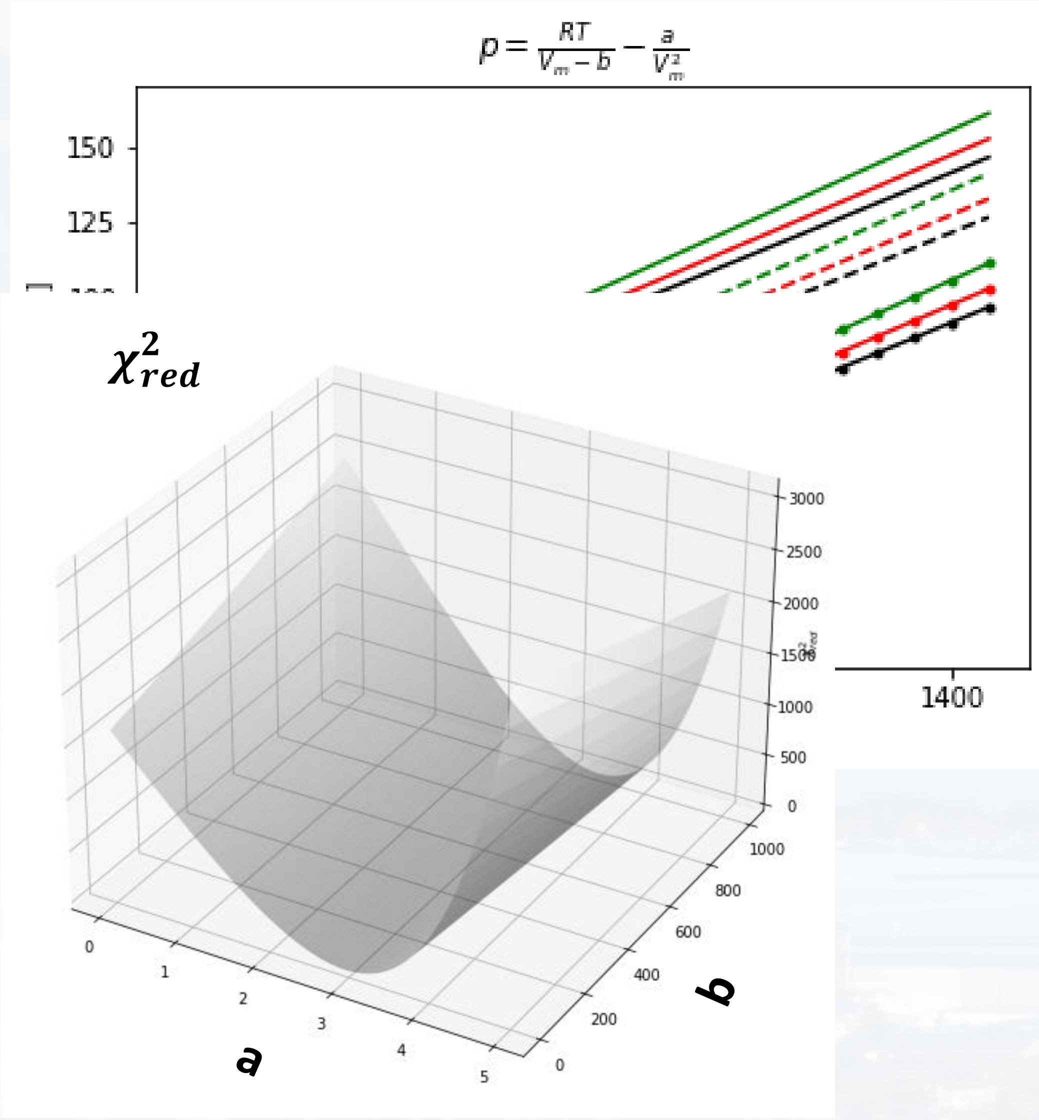
finding ***a*** and ***b*** of
a van-der-Waals gas

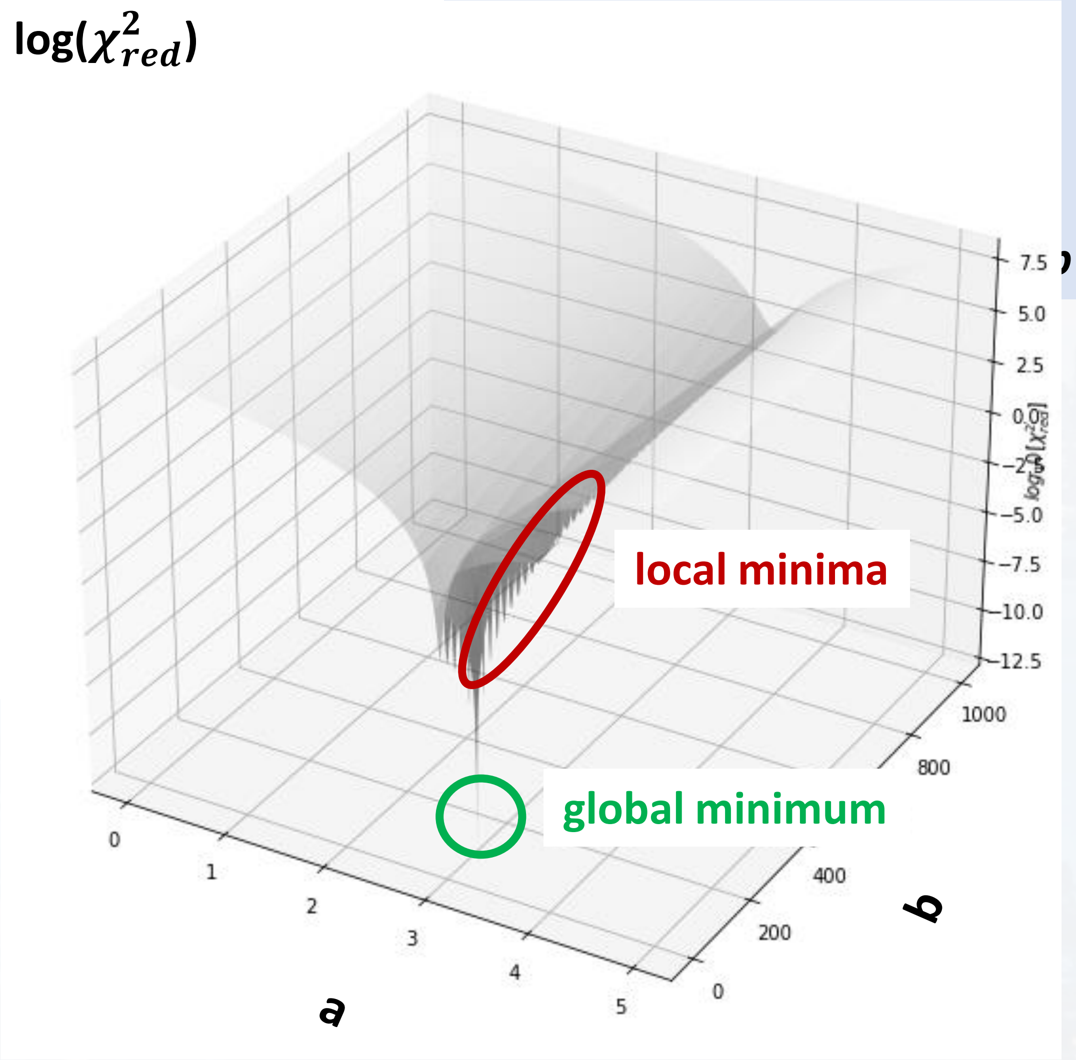
if critical points are not accessible
→ fitting curve, finding ***a*** and ***b***

Optimization: The Problem

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

finding ***a*** and ***b*** of





Optimization: The Problem

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**

cross entropy $$S = -\sum_i p(true)_i \cdot \ln p(model)_i$$ constrain: $$\sum_i p_i = 1$$

→ Lagrangian Multipliers and variational calculus

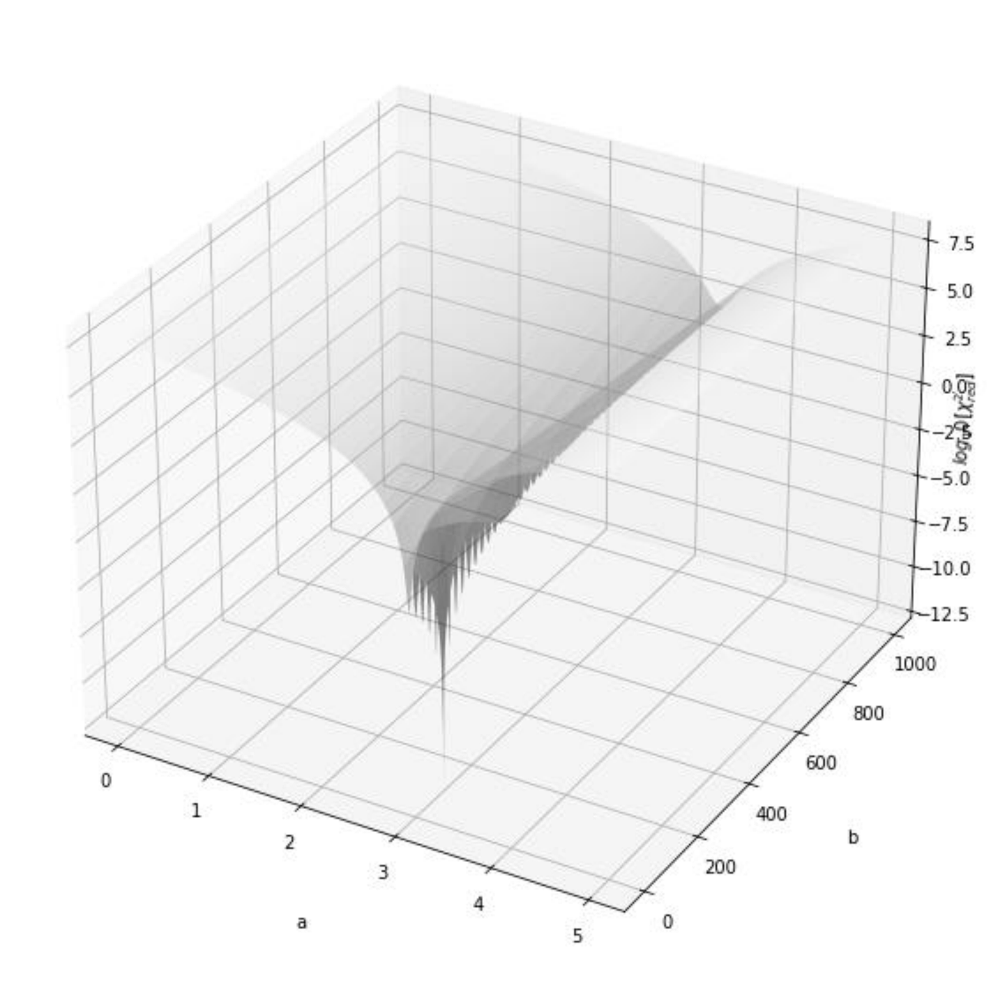
→ mathematically: *Free Energy like term = Energy like term – Entropy term*

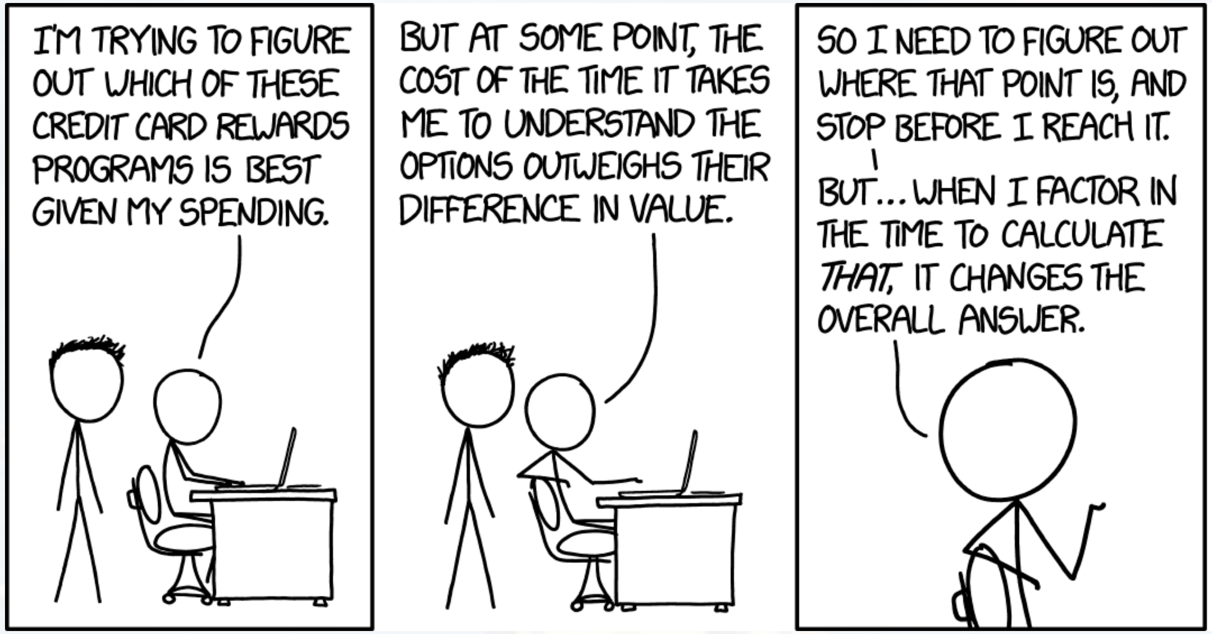
examples:
- KL divergence
- Lasso method (linear regression)
- actual energy → Boltzmann distribution

etc

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

These functions are very complicated, not analytical at all





Outline

- The Problem

- **Gradient Descent**

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

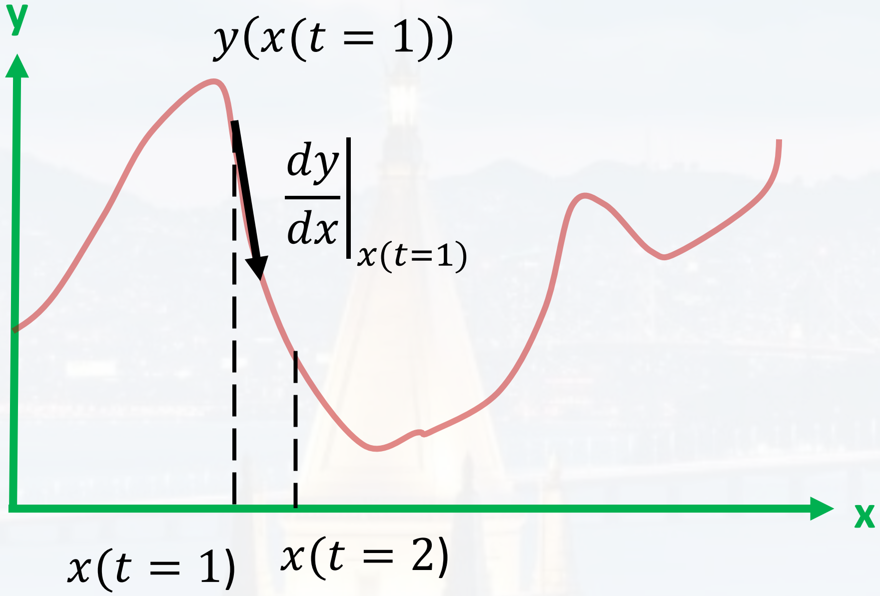
Optimization: Gradient Descent

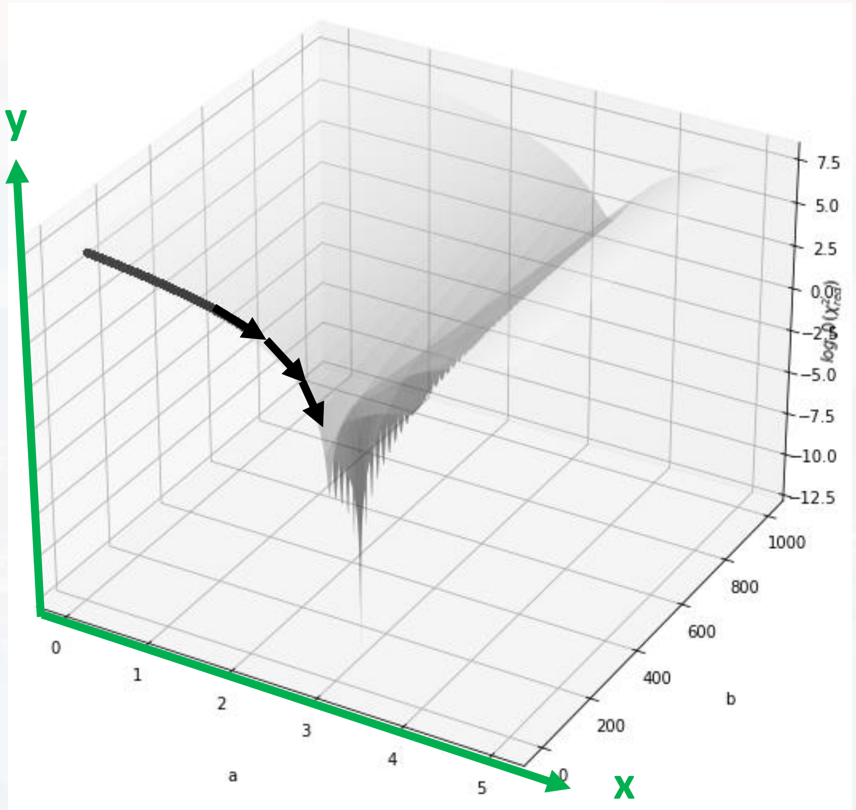
main application: **ANN!**

Optimization: Gradient Descent

Vanilla

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



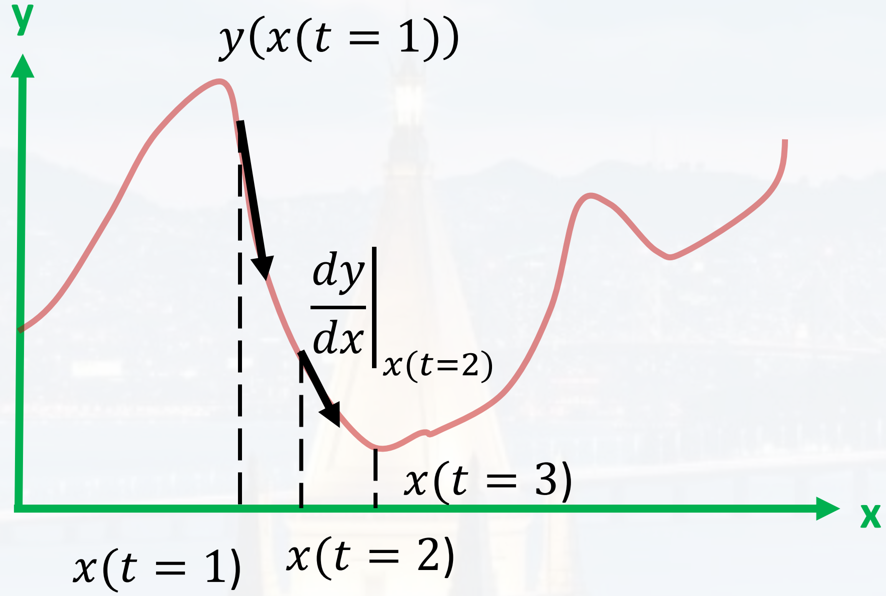


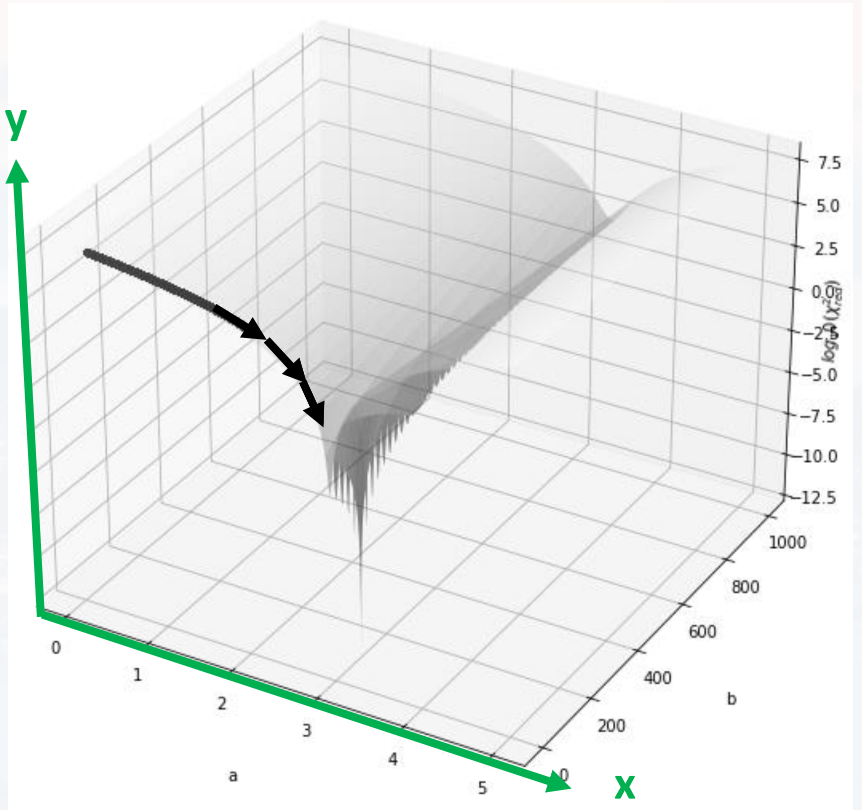
$$x(t=2) = x(t=1) - \varepsilon \left.\frac{dy}{dx}\right|_{x(t=1)}$$

ε > 0

Vanilla

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



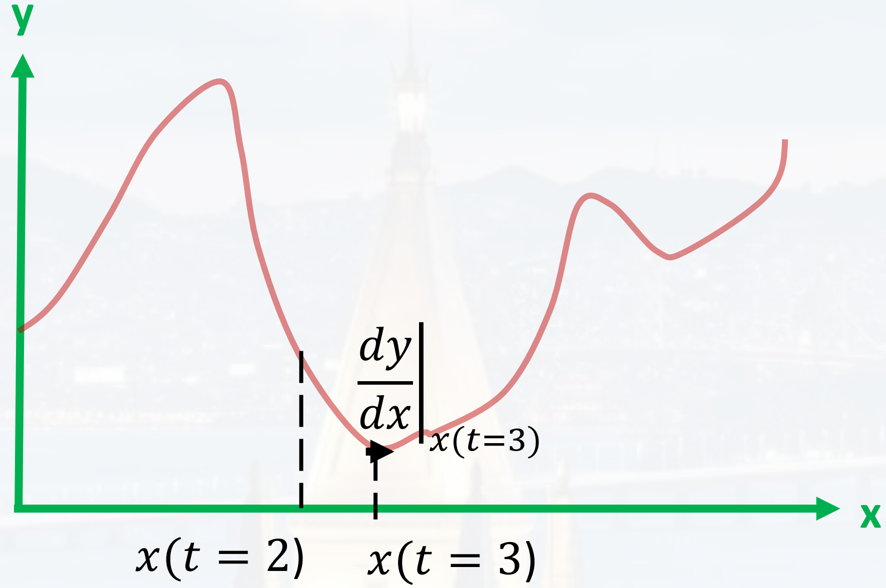


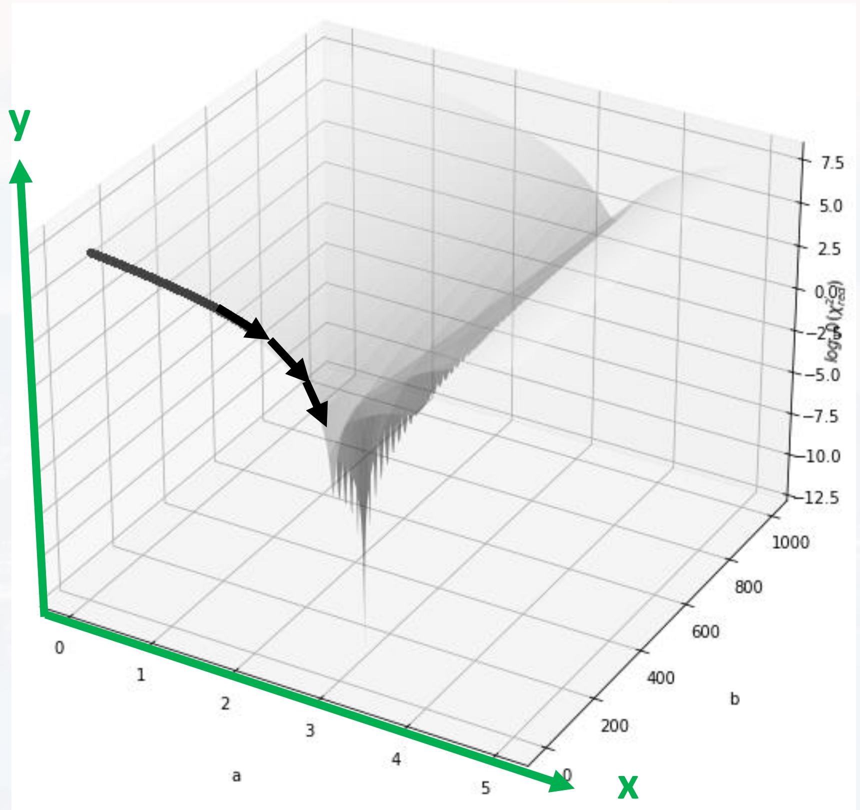
$$x(t=3) = x(t=2) - \varepsilon \left.\frac{dy}{dx}\right|_{x(t=2)}$$

ε > 0

Vanilla

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



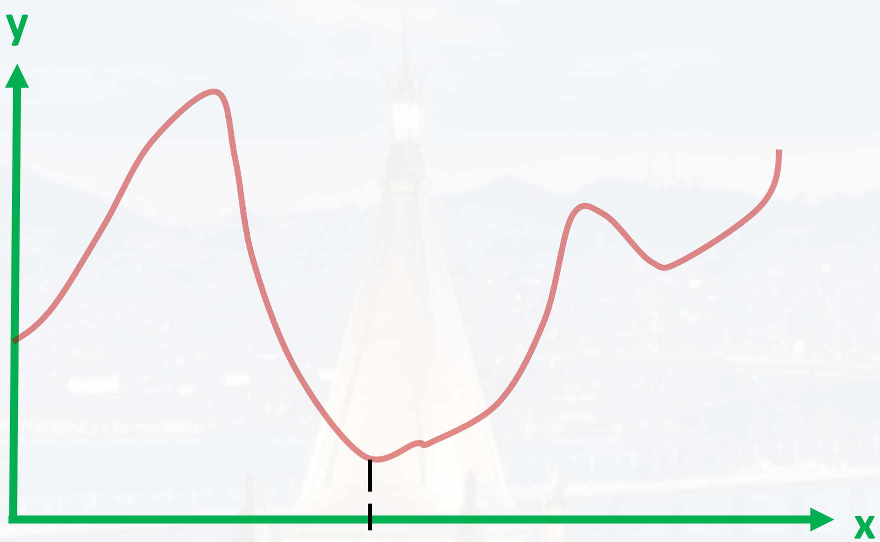


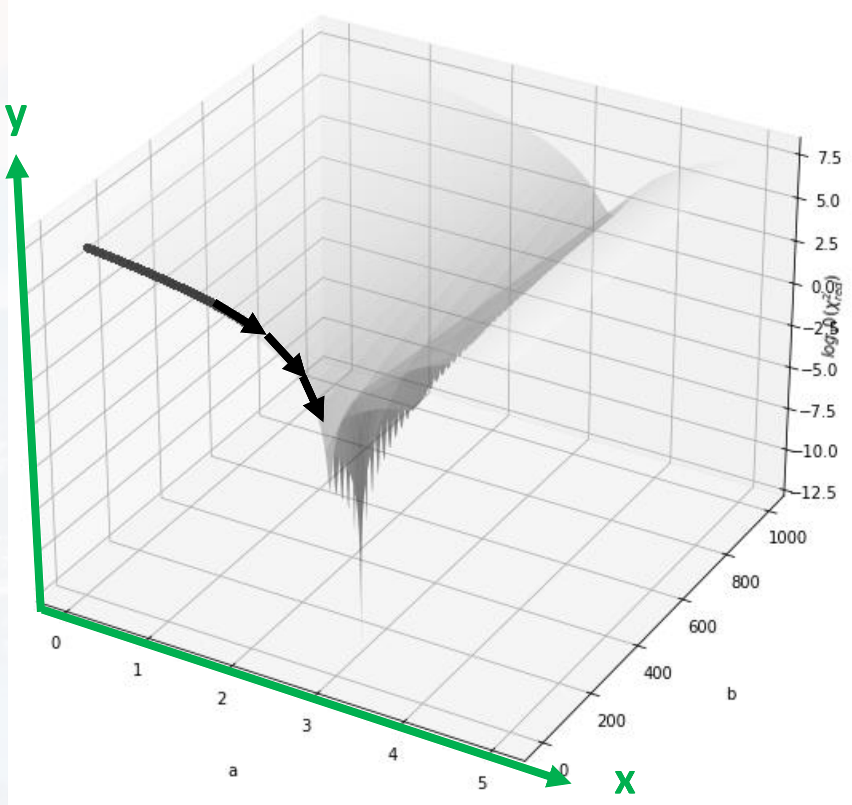
$$x(t=4) = x(t=3) - \varepsilon \left.\frac{dy}{dx}\right|_{x(t=3)}$$

$\varepsilon > 0$

Vanilla

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



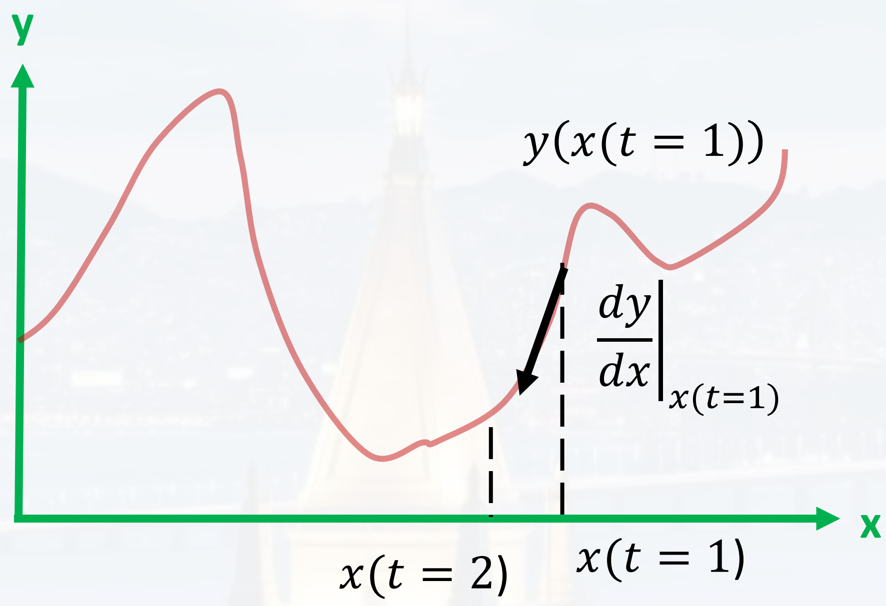


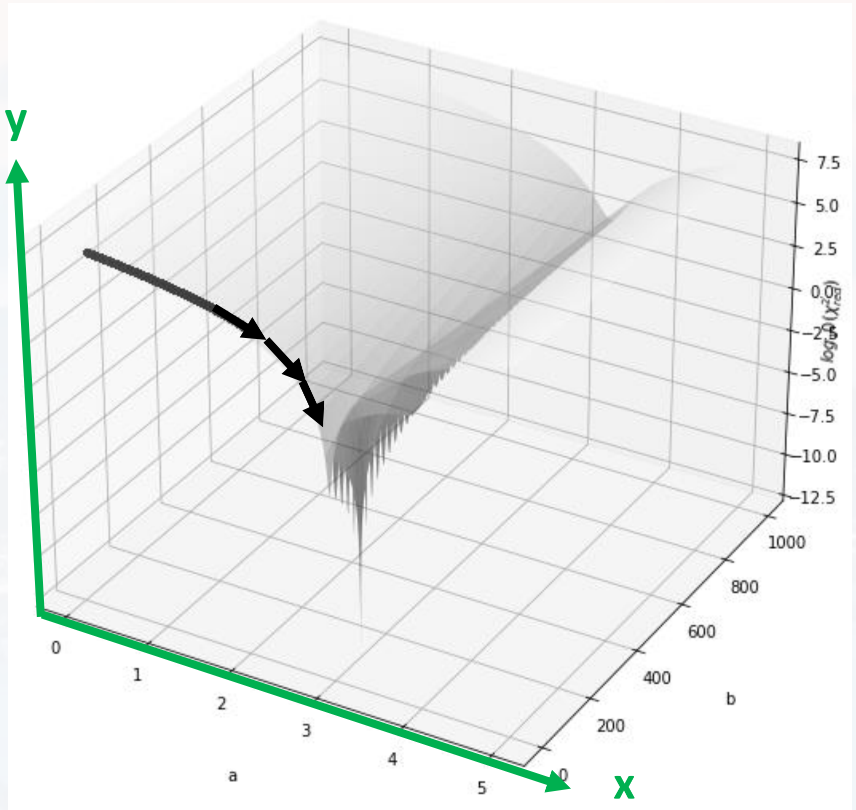
$$x(t=4) = x(t=3)\ -\ \varepsilon\left.\frac{dy}{dx}\right|_{x(t=3)}$$

$\varepsilon > 0$

Vanilla

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





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

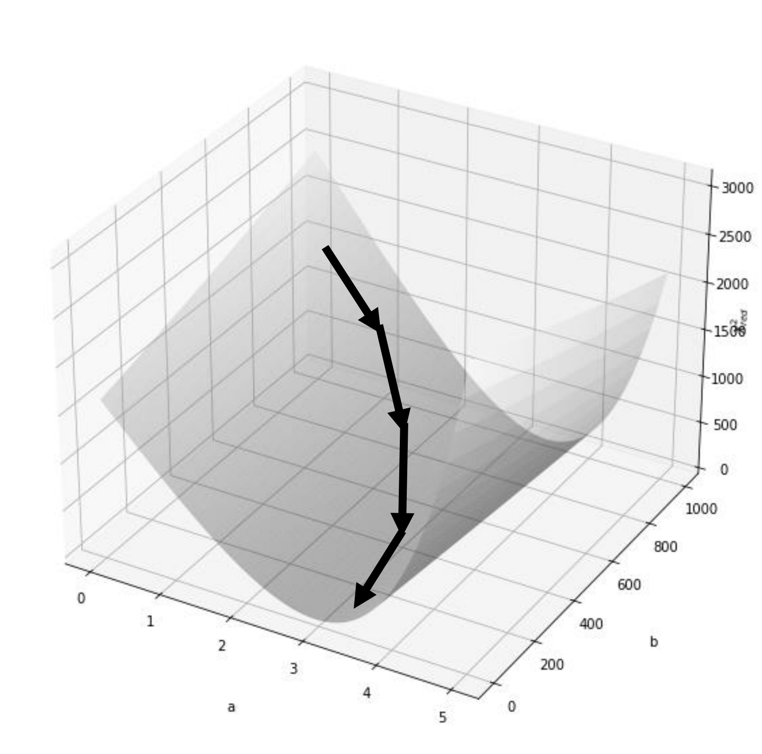
ε > 0

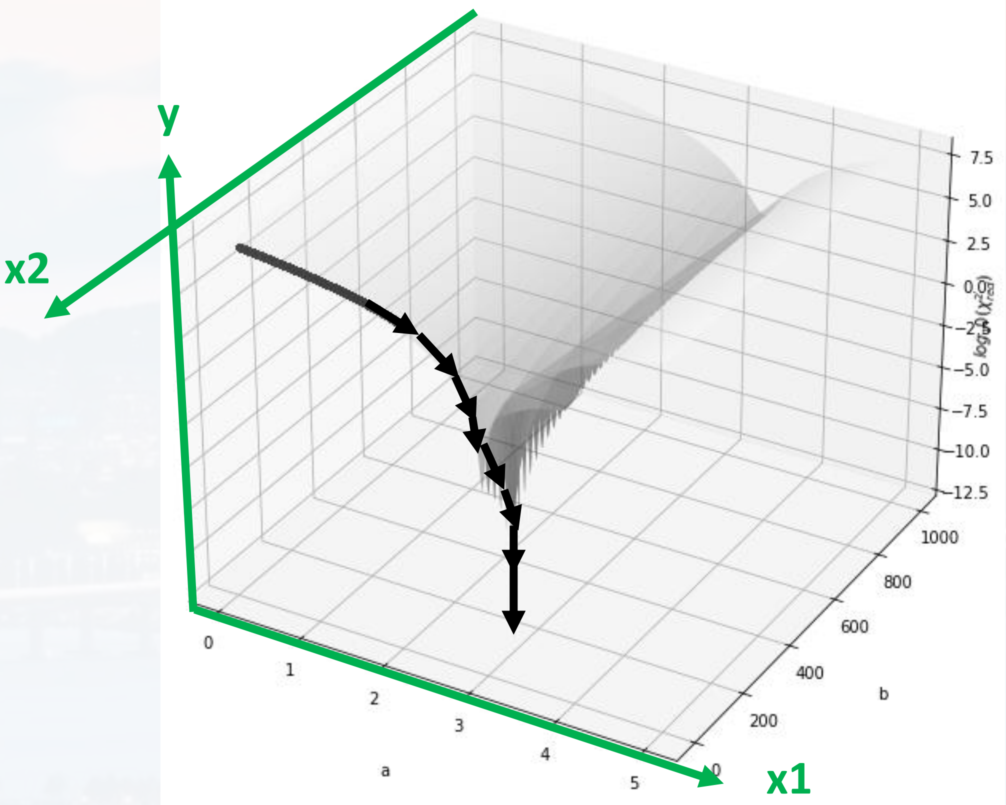
Optimization:

Vanilla

$$\left.\frac{dy}{dx_1}\right|_{x_1(0)} \approx \frac{y(x_1(0)+\Delta x_1)\ - y(x_1(0\ )-\Delta x_1)}{2\Delta x_1}$$

$$\left.\frac{dy}{dx_2}\right|_{x_2(0)} \approx \frac{y(x_2(0)+\Delta x_2)\ - y(x_2(0\ )-\Delta x_2)}{2\Delta x_2}$$





Vanilla

$$\left.\frac{dy}{dx_1}\right|_{x_1(0)} \approx \frac{y(x_1(0)+\Delta x_1) \; - y(x_1(0\,) - \Delta x_1)}{2\Delta x_1}$$

$$\left.\frac{dy}{dx_2}\right|_{x_2(0)} \approx \frac{y(x_2(0)+\Delta x_2) \; - y(x_2(0\,) - \Delta x_2)}{2\Delta x_2}$$

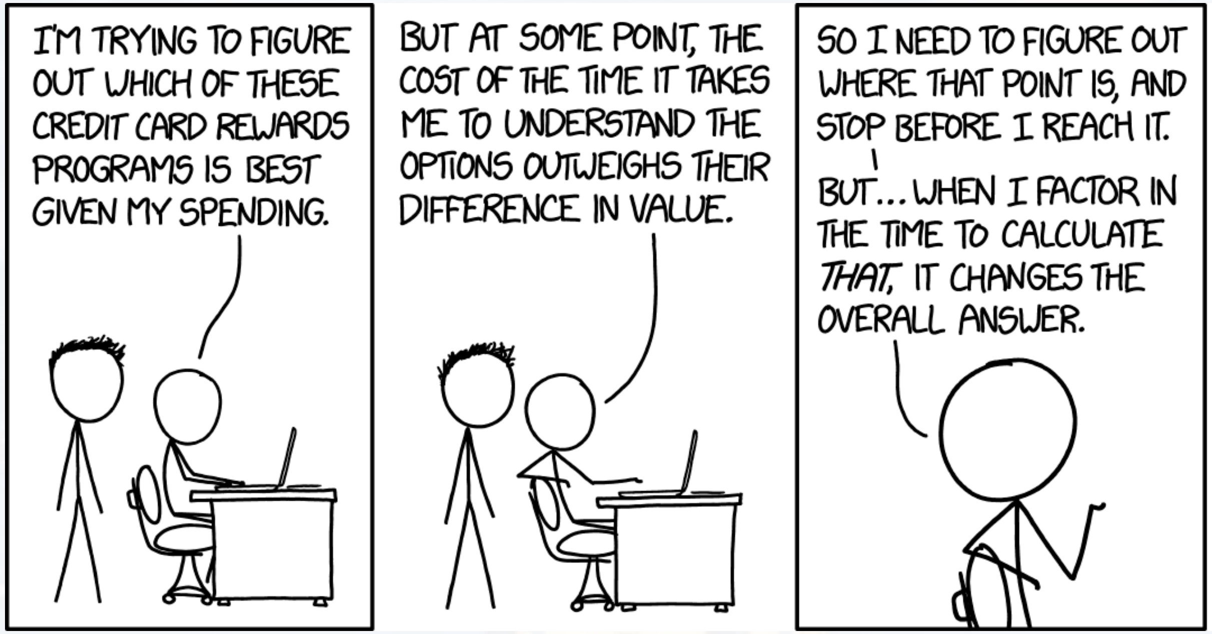
$$\vdots$$

$$\left.\frac{dy}{dx_i}\right|_{x_i(0)} \approx \frac{y(x_i(0)+\Delta x_i) \; - y(x_i(0\,) - \Delta x_i)}{2\Delta x_i}$$

$$\vdots$$

$$\left.\frac{dy}{dx_N}\right|_{x_N(0)} \approx \frac{y(x_N(0)+\Delta x_N) \; - y(x_N(0\,) - \Delta x_N)}{2\Delta x_N}$$

$$\left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \begin{pmatrix} \left.\frac{dy}{dx_1}\right|_{x_1(0)} \\ \dots \\ \left.\frac{dy}{dx_i}\right|_{x_i(0)} \\ \dots \\ \left.\frac{dy}{dx_N}\right|_{x_N(0)} \end{pmatrix} = grad(y)_x$$



Outline

- The Problem
- **Gradient Descent**
  - Vanilla
  - Learning Rate Schedule
  - Momentum
  - L1 and L2
  - More Finetuning

Learning Rate Schedule

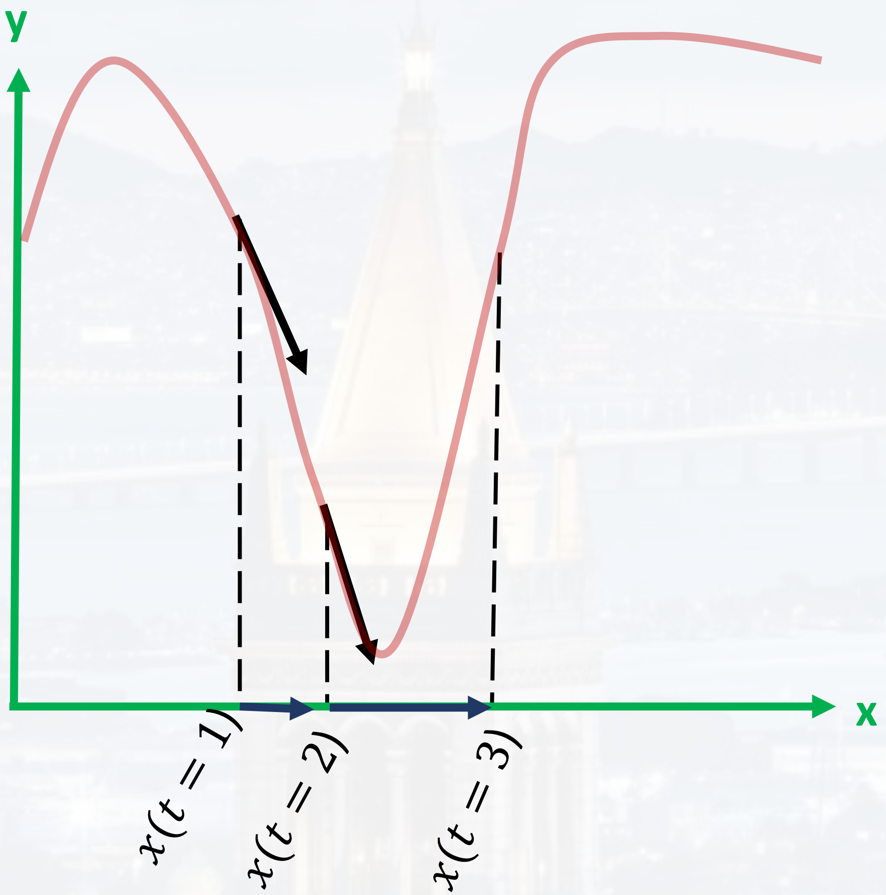
$$\left.\frac{dy}{dx}\right|_{x_0} \approx \frac{y(x_0+\Delta x) - y(x_0-\Delta x)}{2\Delta x}$$

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

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

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

defines how large the leap Δx is



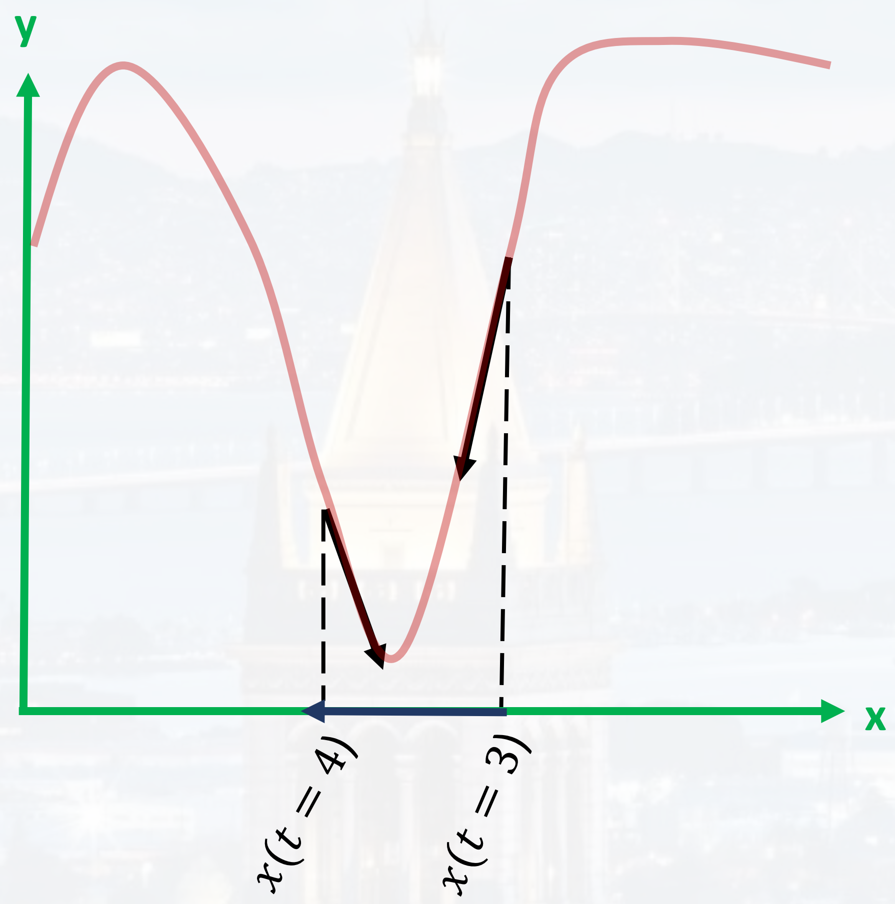
Optimization:

Gradient Descent

Learning Rate Schedule

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

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



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

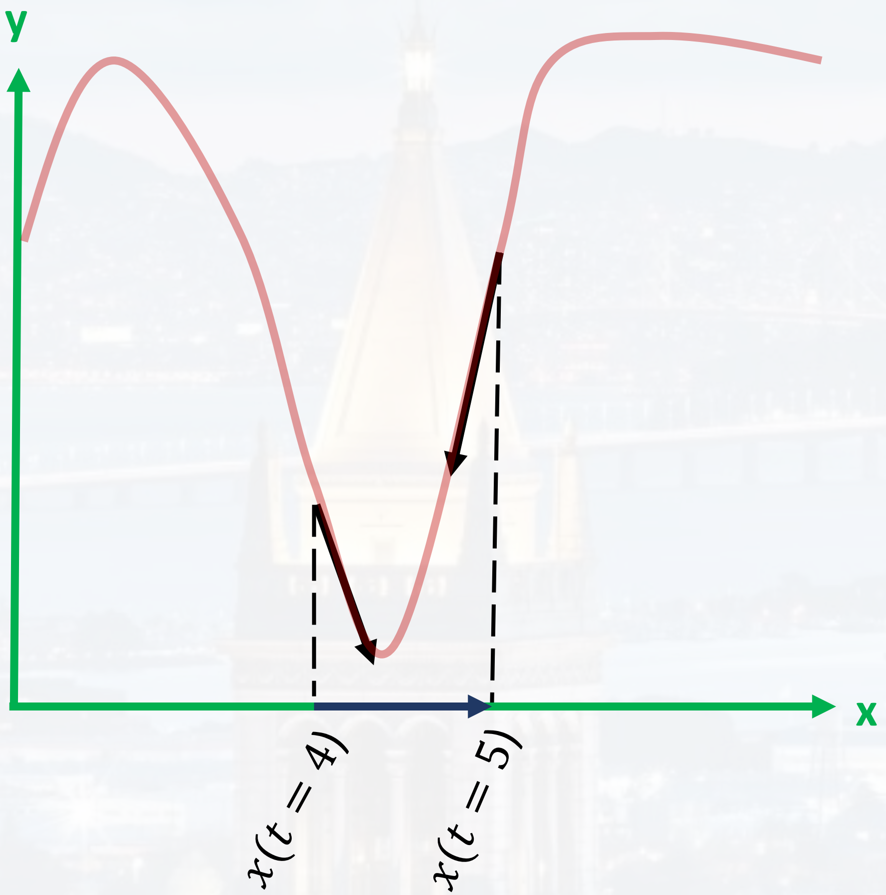
$$\Delta x = -\ \boldsymbol{\varepsilon}\left.\frac{dy}{dx}\right|_{x(t)}$$

defines how large the leap Δx is

Learning Rate Schedule

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

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



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

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

defines how large the leap Δx is

… and so on…

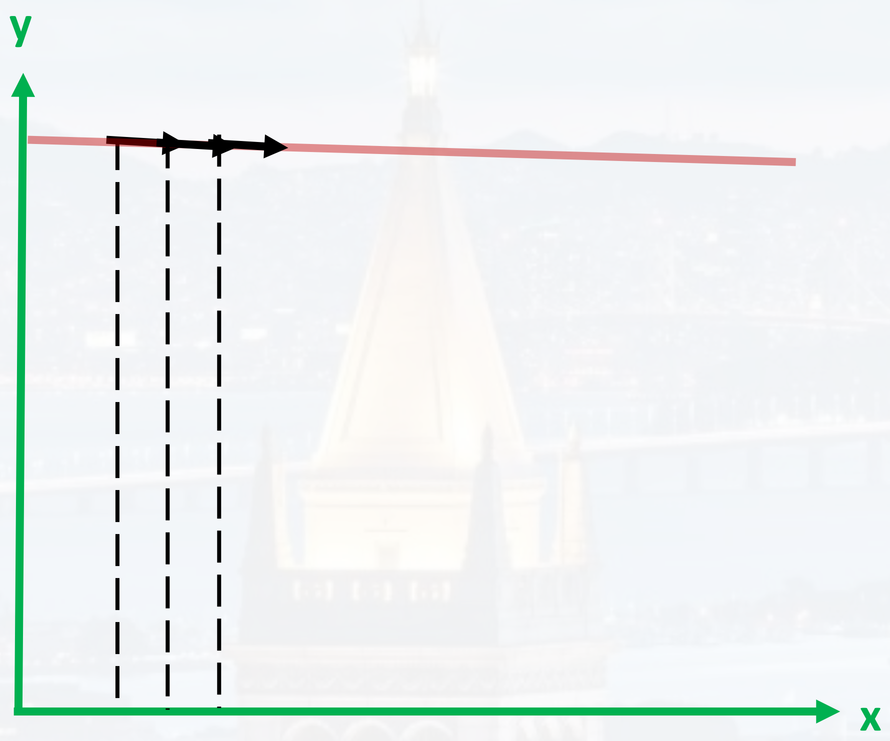
→ smaller $\boldsymbol{\varepsilon}$?

Optimization: Gradient Descent

Learning Rate Schedule

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

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



$\boldsymbol{\varepsilon} > 0$ 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: Gradient Descent

Learning Rate Schedule

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

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

$\boldsymbol{\varepsilon} > 0$ 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

Optimization: Gradient Descent

Learning Rate Schedule

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

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

$\boldsymbol{\varepsilon} > 0$ 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 Schedule

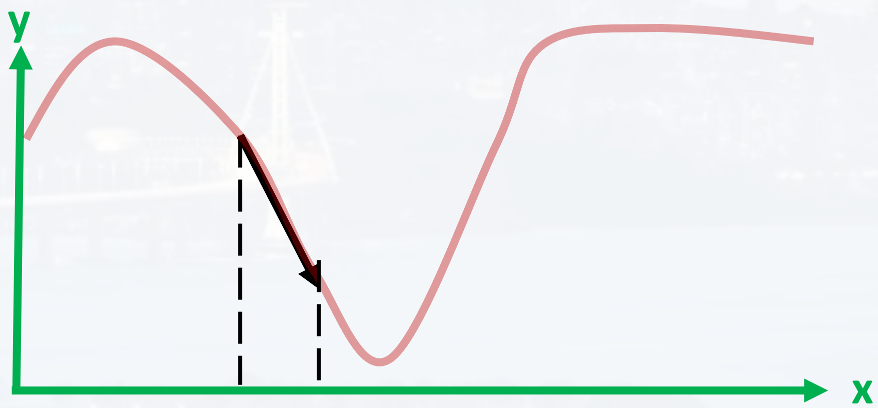
learning rate as function of t:

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

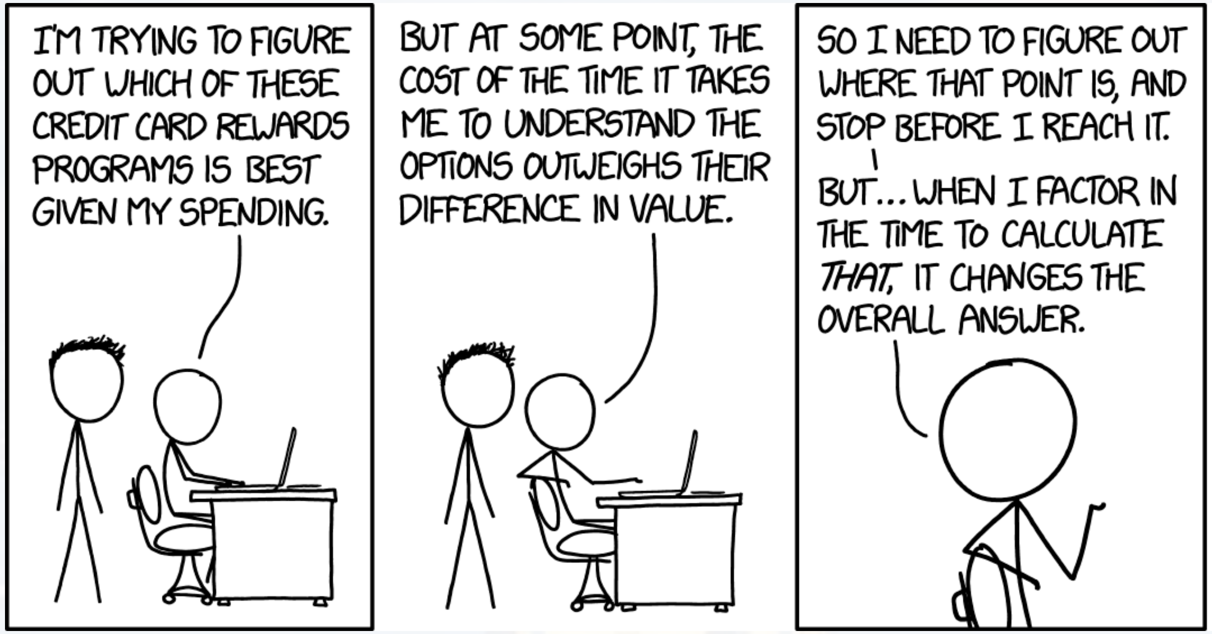
$$\Delta x = -\,\boldsymbol{\varepsilon} \left.\frac{dy}{dx}\right|_{x(t)}$$

defines how large the leap Δx is

can also be a stepwise function (learning rate schedule)



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

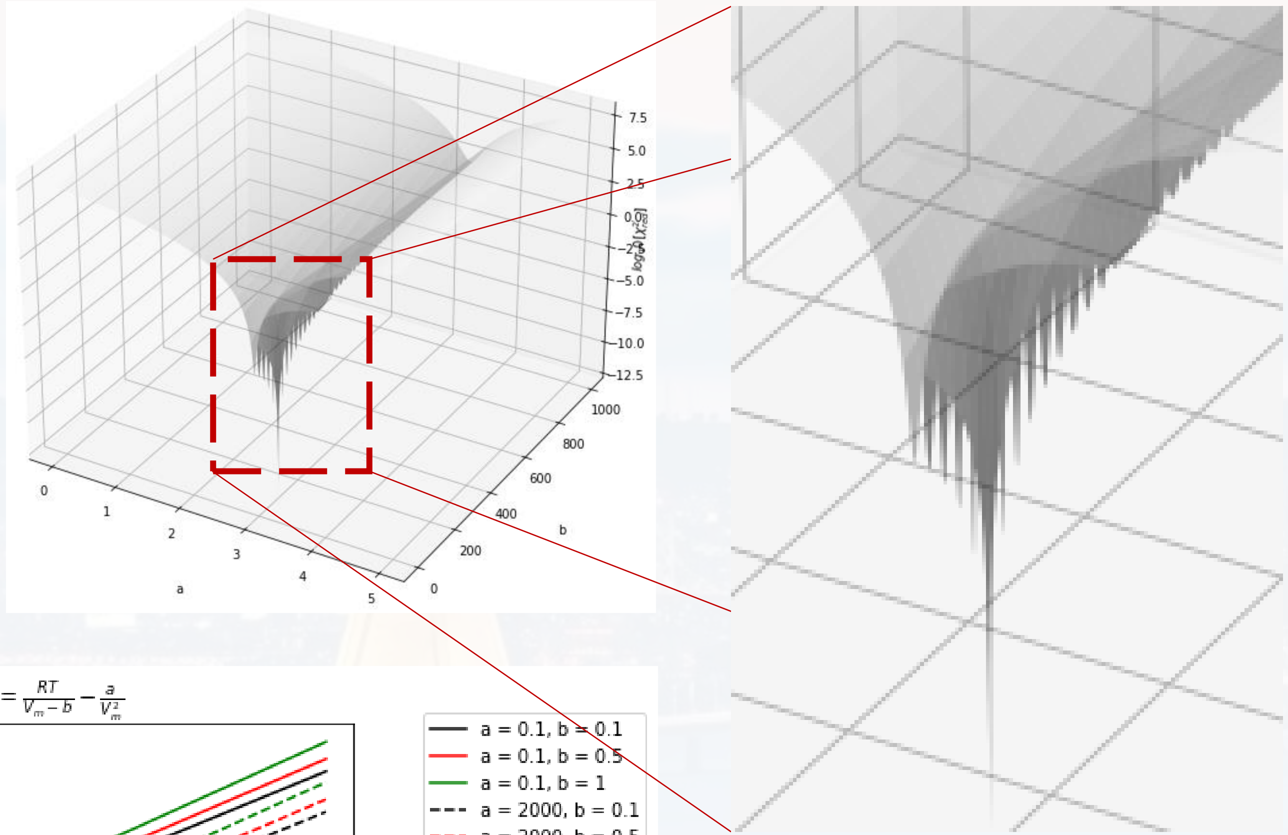


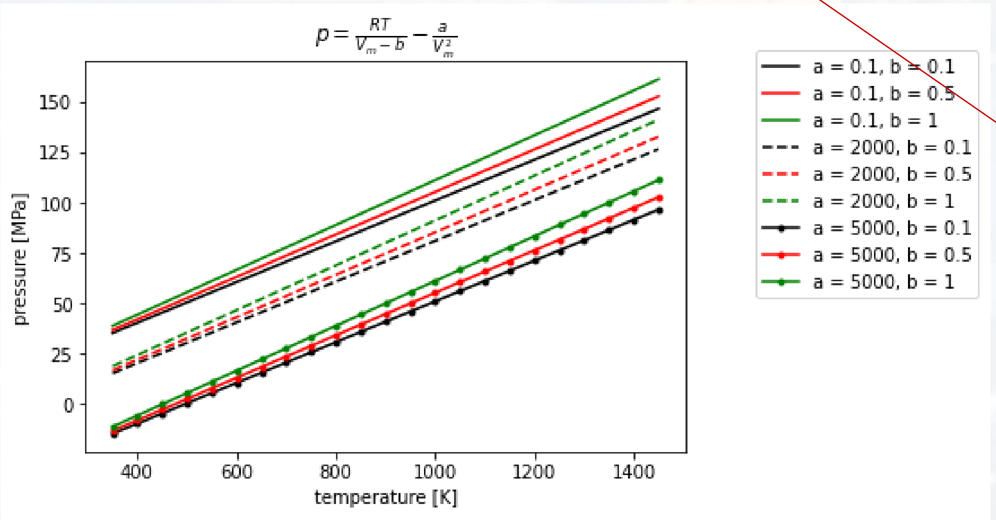
Outline

- The Problem
- **Gradient Descent**
  - Vanilla
  - Learning Rate Schedule
  - Momentum
  - L1 and L2
  - More Finetuning

Optimization: Gradient Descent

Momentum

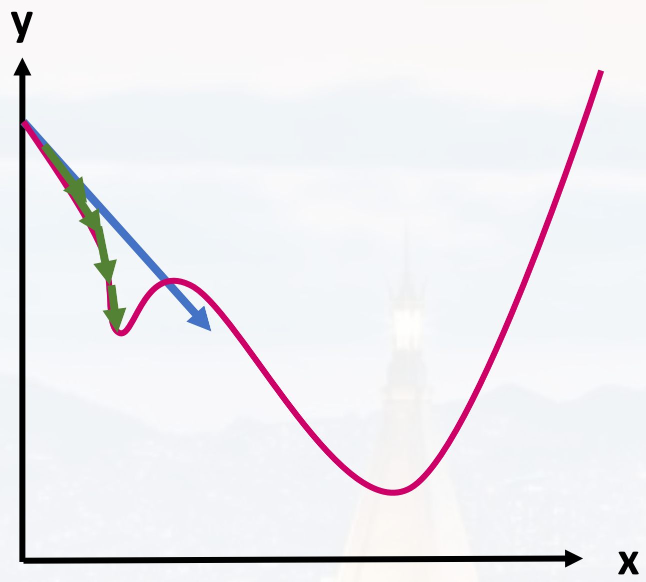




even with AdaGrad and learning rate schedule
→ would get stuck in local minimum

need to roll over → **momentum**

Momentum



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

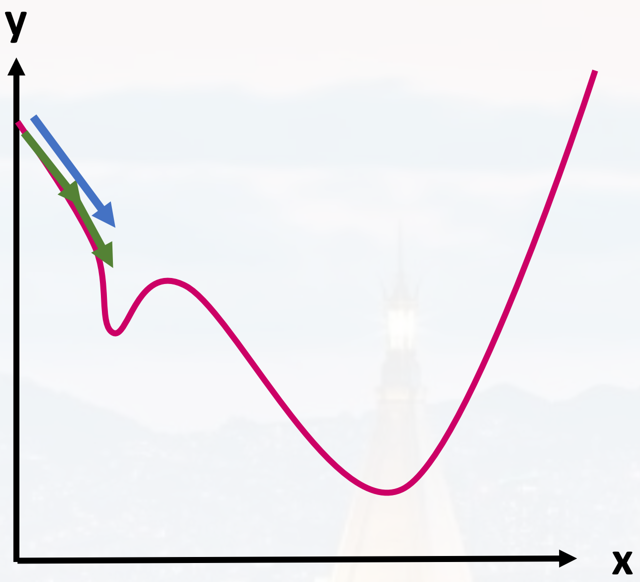
$$\langle grad(y)_{x(t)} \rangle = \frac{1}{N}[grad(y)_{x(t-1)} + grad(y)_{x(t-2)} + \ldots + grad(y)_{x(t-N)}]$$

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



Momentum

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

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

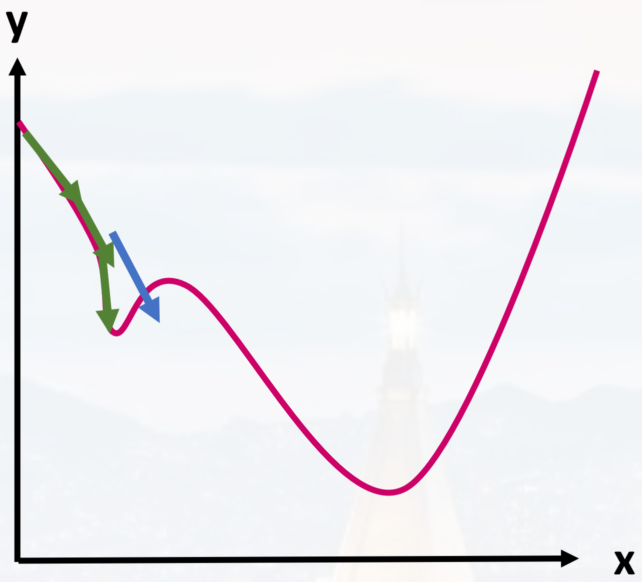
$$\langle grad(y)_{x(0)} \rangle = grad(y)_{x(0)} \qquad \mu_0 = (0,1)$$

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

Momentum

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

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



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

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

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

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

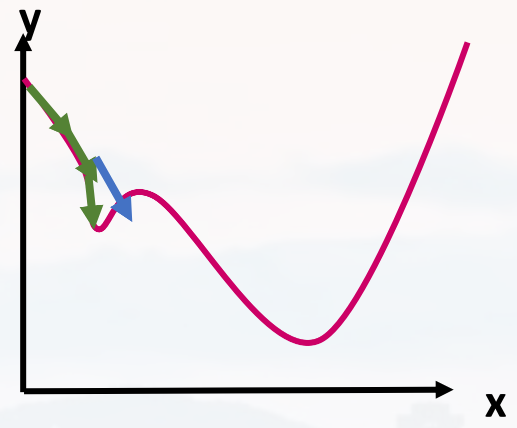
$$\langle \boldsymbol{grad(y)_{x(3)}} \rangle = grad(y)_{x(3)} + \boxed{\mu_0} \left[grad(y)_{x(2)} + \boxed{\mu_0} [grad(y)_{x(1)} + \boxed{\mu_0} \cdot grad(y)_{x(0)}] \right]$$

… and so on…

Momentum

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

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

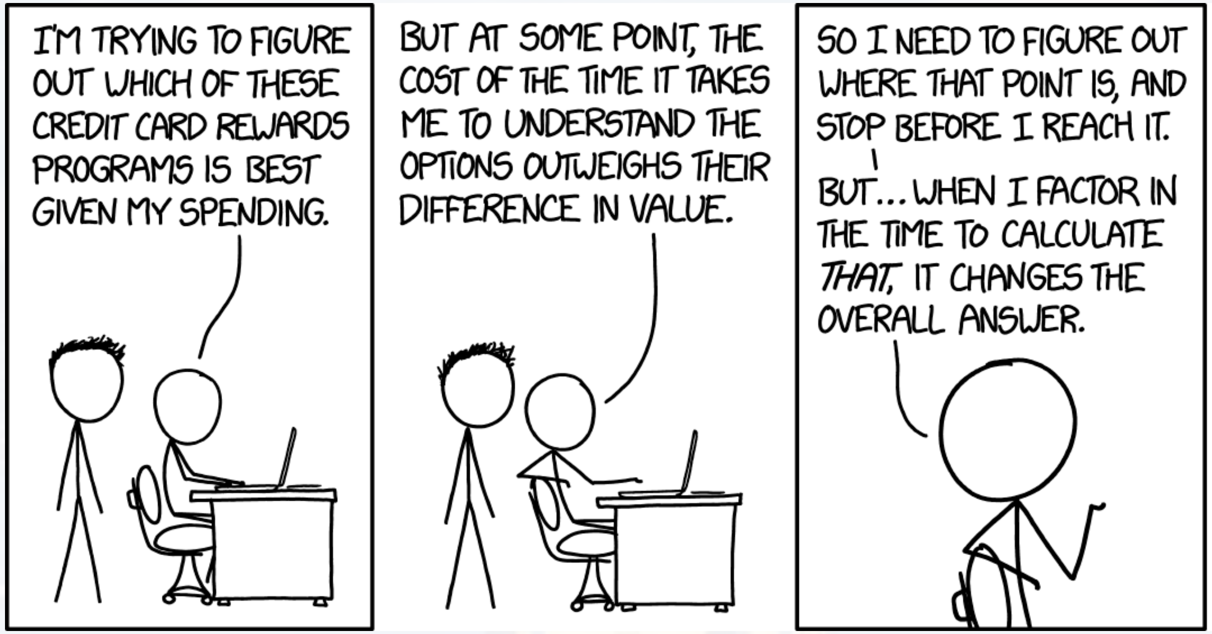


$$\langle \boldsymbol{grad(y)_{x(3)}} \rangle = grad(y)_{x(3)} \; +$$

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

```python
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
```



Outline

- The Problem
- **Gradient Descent**
  - Vanilla
  - Learning Rate Schedule
  - Momentum
  - L1 and L2
  - More Finetuning

L1 and L2

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 $\boldsymbol{\beta}$**

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

λ *Lagrangian Multiplier*

called **L1 regularization**, or LASSO

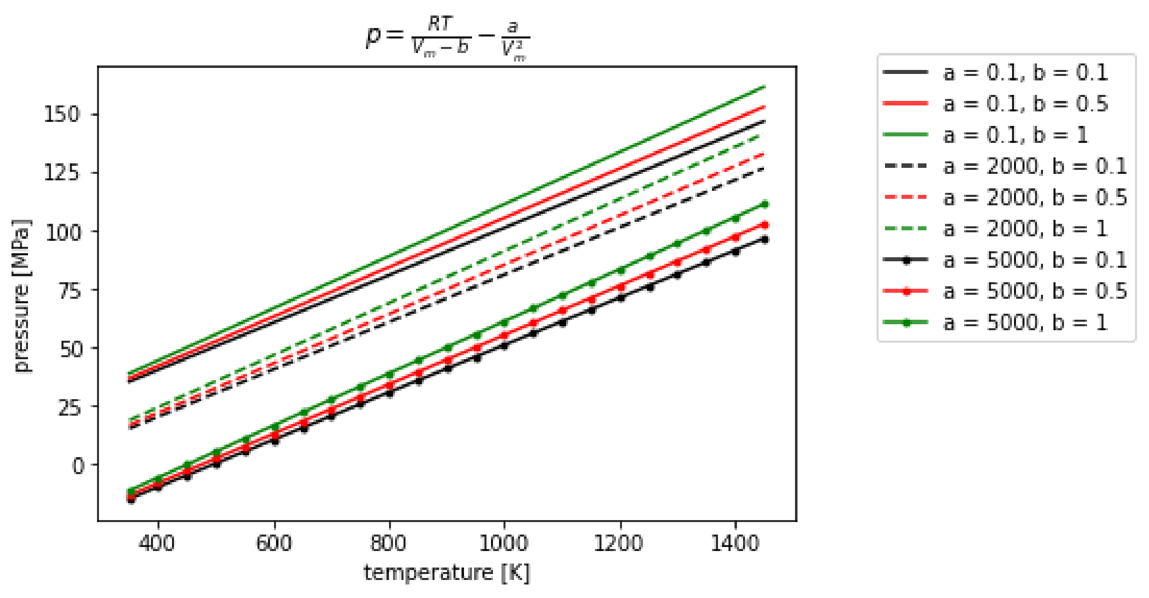
L1 and L2

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!

L1 and L2

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:

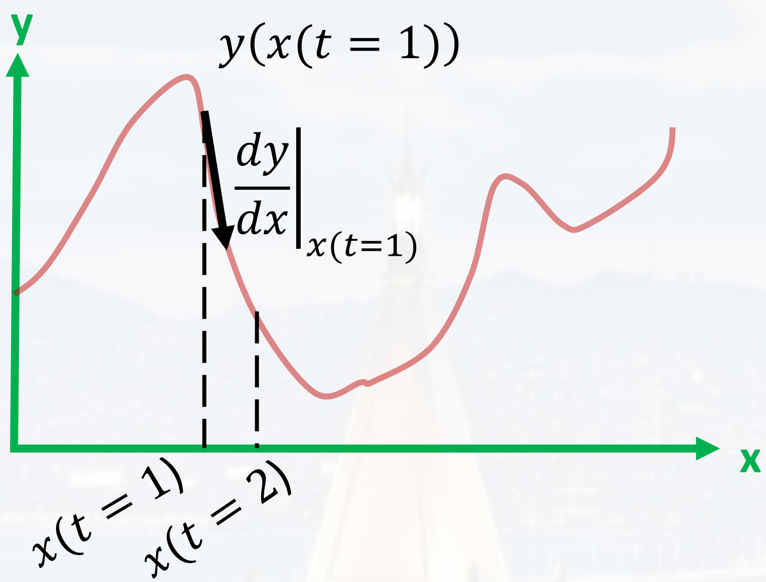
$$\hat{\beta} = (X^T X + \boldsymbol{\lambda I})^{-1} X^T Y \longrightarrow \min_{\beta} \left\{ \frac{1}{N} \|Y - X\beta\|^2 + \boldsymbol{\lambda} \|\boldsymbol{\beta}\|^2 \right\}$$

λ *Lagrangian Multiplier*

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

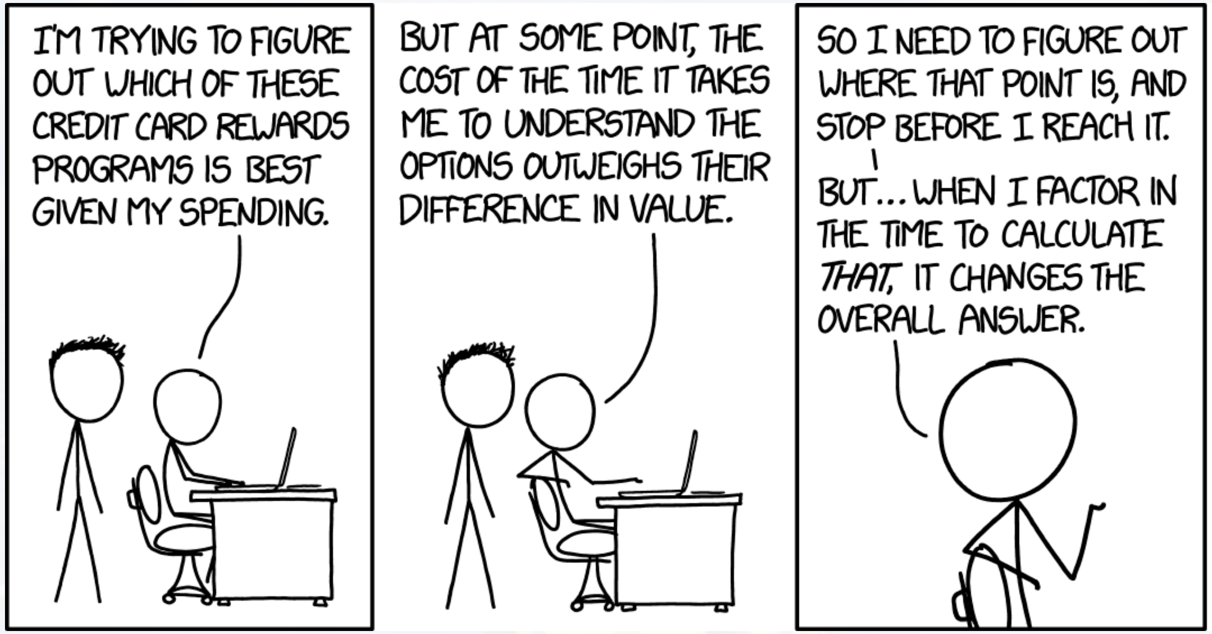
L1 and L2 regularization

L1 and L2



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

- 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



Outline

- The Problem

- Gradient Descent

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

More Fine Tuning

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

↓

Learning Rate Schedule, L1, L2

↓

Momentum

↓

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

↓

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

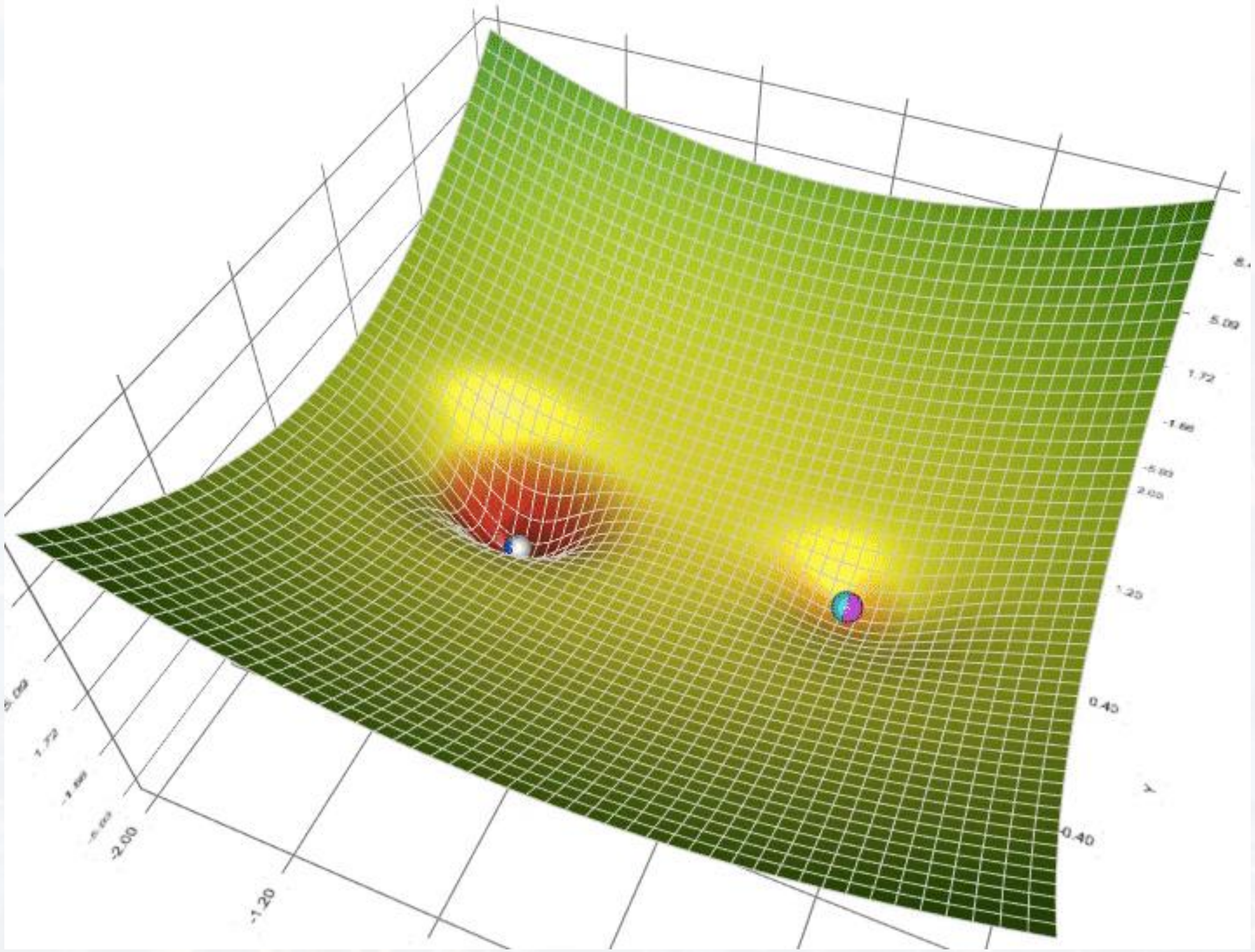
different scaling for all different direction

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

Lili Jiang

TowardsDataScience

More Fine Tuning



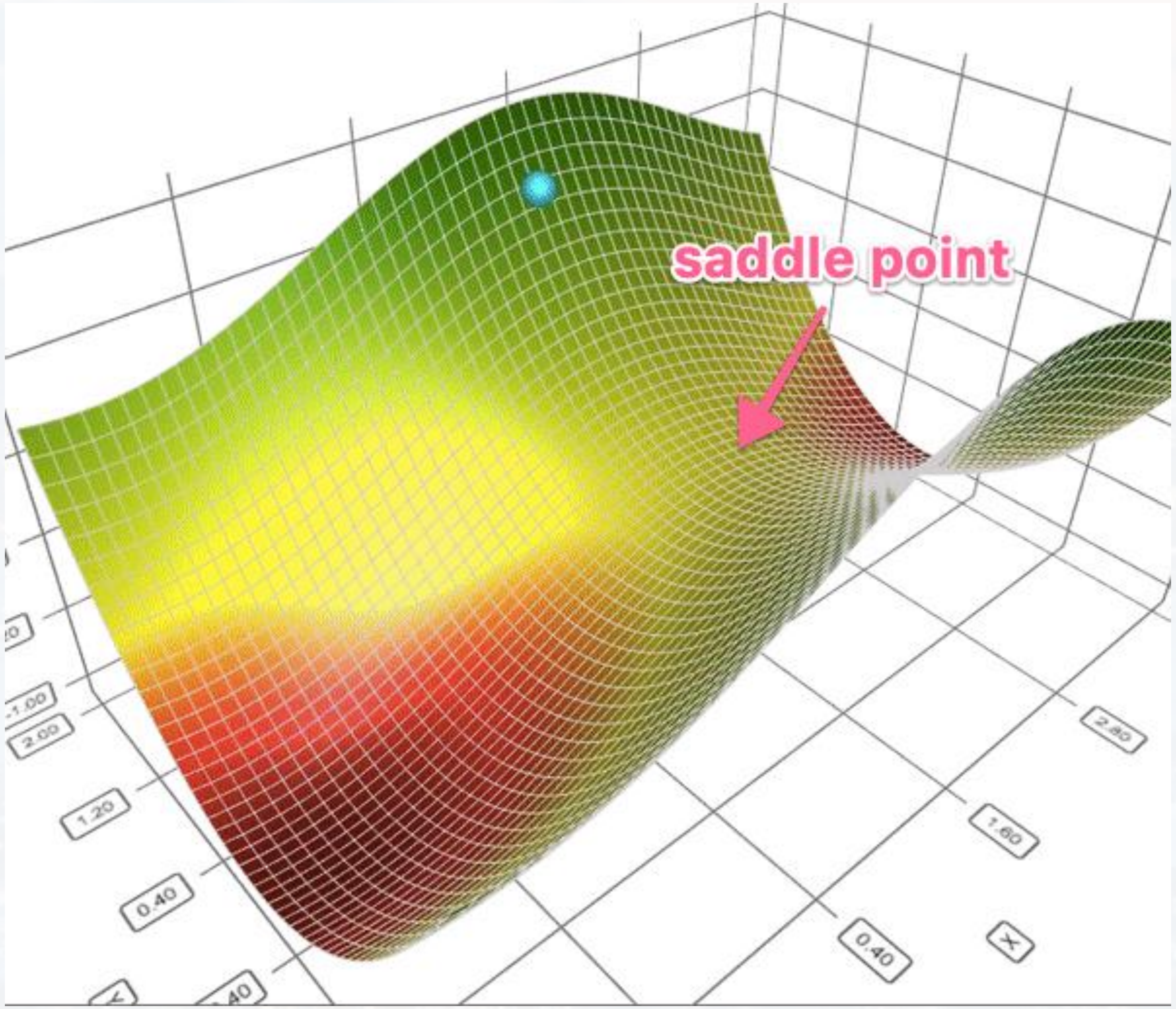
gradient descent (cyan),
momentum (magenta),
AdaGrad (white),
RMSProp (green),
Adam (blue)

Optimization: Gradient Descent

Lili Jiang

[TowardsDataScience](#)

More Fine Tuning



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

Thank you very much for your attention!