Training Neural Networks

Trainability

Lecture 16

Pierfrancesco Beneventano
pierb@mit.edu

Massachusetts Institute of Technology

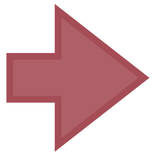
Part 0

What is the goal?

Why and what do we do?

Recap of the Task

- We have data coming in real life: $X \sim P$
- We need to find *a neural network* $f_\theta(X)$ that has the right output $\varphi(X)$.



The question:

Can we find it?

Recap of the Task

- We have data coming in real life: $X \sim P$
- We need to find the neural network $f_\theta(X)$ that has the right output $\varphi(X)$.

The question:

Can we find it?

Ingredient 3:

- We need a measure of the error.

Measure of the Error

Ingredient 3:

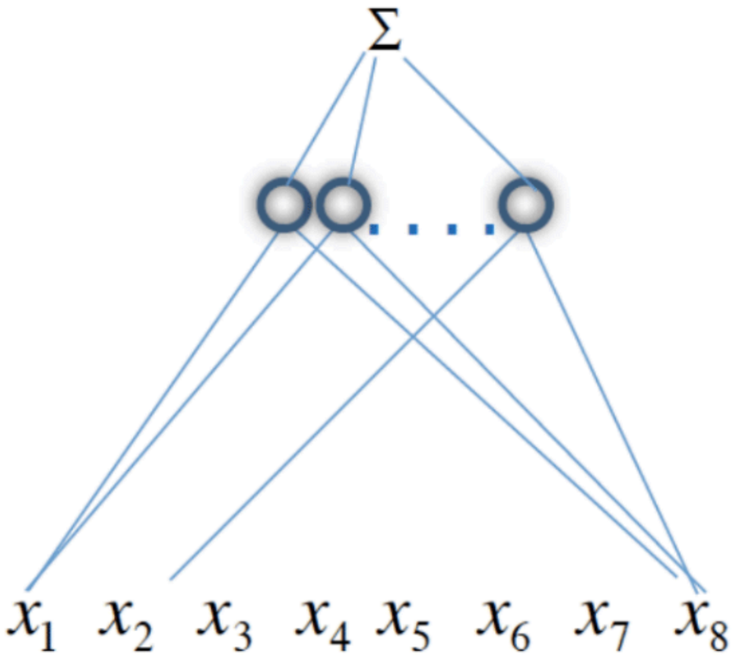
- We need a measure of the error.

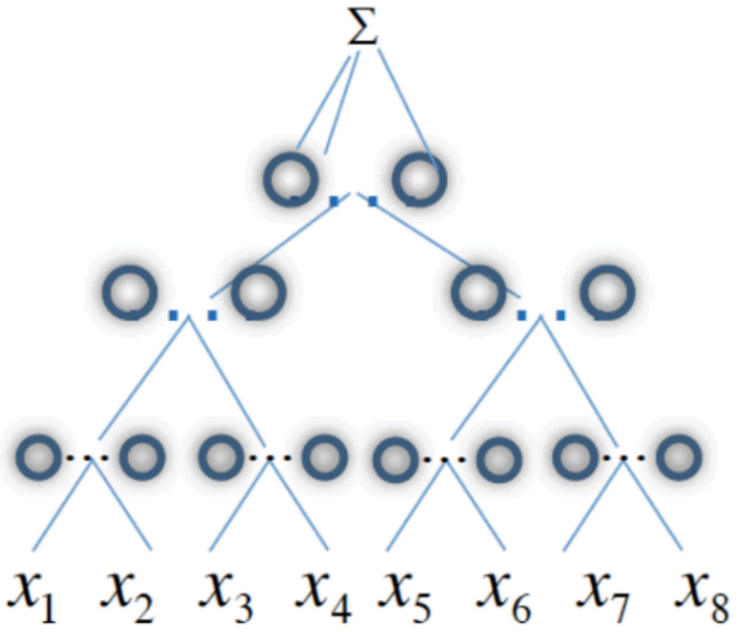
Examples:

- $\sup_X \|f_\theta(X) - \varphi(X)\| = 0$
- $\mathbb{E}_{X\sim P}\|f_\theta(X) - \varphi(X)\|^2 = 0$
- Accuracy 100%.
- The drone self-pilots correctly or not.

Deep and shallow networks: approximation universality

Theorem *Assume as target continuous functions on a compact domain. Shallow, one-hidden layer networks with a a nonlinear $\sigma(x)$ (which is not a polynomial) can approximate such targets arbitrarily well. Deep networks with a nonlinear $\sigma(x)$ (including polynomials) can also approximate such target functions arbitrarily well.*



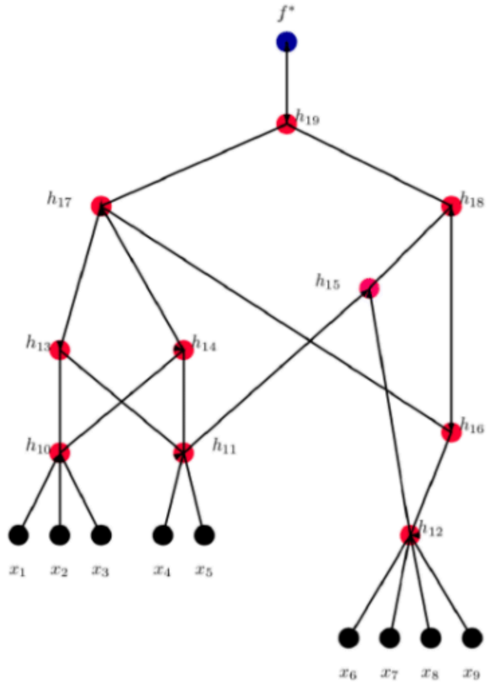


$$\phi(x) = \sum_{i=1}^{r} c_i \left| < w_i , x > + b_i \right|_+$$

Cybenko, Girosi, ….

Definition: Compositional Sparsity

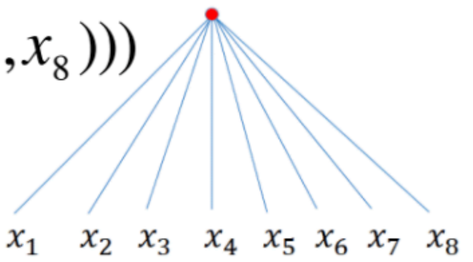
A function $f : \mathcal{X}^d \rightarrow \mathcal{X}$ is *compositionally sparse* if it can be represented as the composition of at most $O(poly(d))$ *constituent functions* each of which is *sparse*, i.e., depends on at most a constant (small) number of variables (so that it does not suffer the curse of dimensionality).

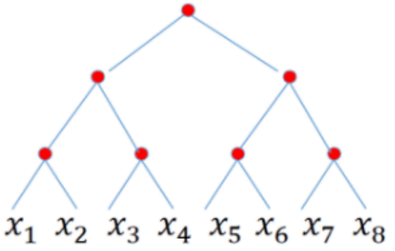


A compositionally sparse function can be represented in terms of a DAG (Directed Acyclic Graph), in which the leaves represent the inputs, the root denotes the output function, and the internal nodes represent the constituent functions.

Deep NNs can avoid the curse for compositional functions

$$f(x_1,x_2,...,x_8) = g_3(g_{21}(g_{11}(x_1,x_2),g_{12}(x_3,x_4))g_{22}(g_{11}(x_5,x_6),g_{12}(x_7,x_8)))$$





Theorem, Mhaskar, Poggio, Liao 2016 (informal statement)

Suppose that a target function of d variables is hierarchically locally compositional, that is has *compositional sparsity*. Both shallow and deep network can approximate f equally well. For a target function with the binary tree graph, the number of parameters of the shallow network depend exponentially on the dimension - that is $O(\epsilon^{-d})$ - whereas the number of parameters of the deep network (with binary tree architecture) is $O(d\epsilon^{-2})$.

H. N. Mhaskar and Poggio, T., "Deep vs. shallow networks: An approximation theory perspective", Analysis and Applications, vol. 14, no. 06, pp. 829 - 848, 2016.

Approximation Recap

- There exists a very wide Shallow network that fits our dataset.
  (Cybenko).

Unfortunately, exponential number of neurons in the dimension!

- What is the role of depth?
  We can approximate sparse compositions without the curse of dimensionality!
  (Poggio & Mhaskar).

They exist.
Still the question:

Can we find them?

Set the Plan

- We have data coming in real life: $X \sim P$
- We need to find the neural network $f_\theta(X)$ that has the right output $\varphi(X)$.
- A measure of the Error $distance(\phi(\,.\,), f_\theta(\,.\,))$.

They exist.
Still the question:

Can we find them?

~~**Idea 1:** Set by hand the parameters such that it works.~~

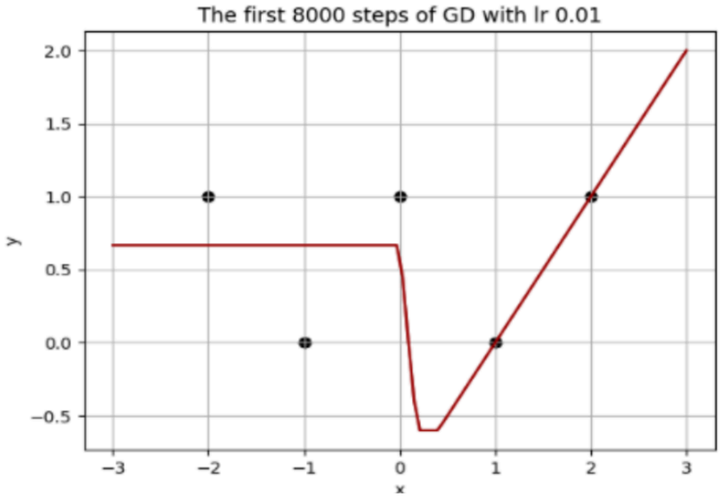
The Error on a Dataset

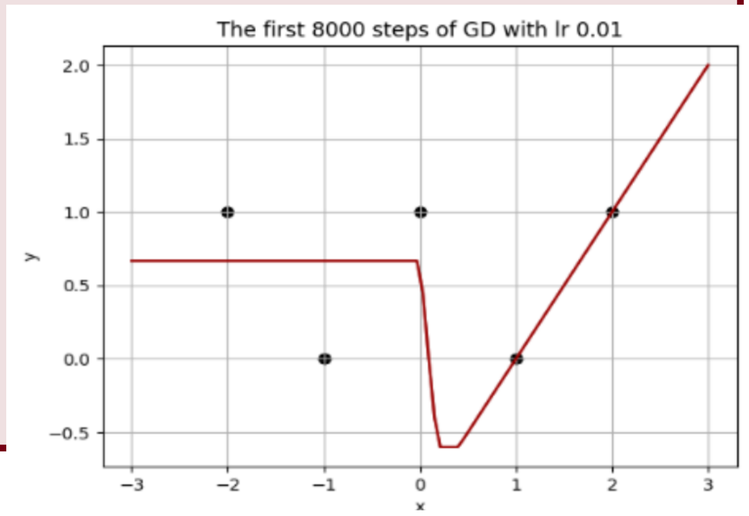
Idea 2:

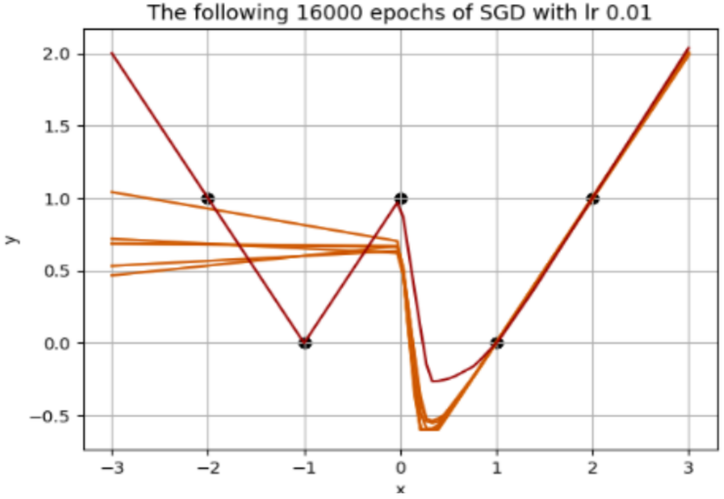
1. Sample some dataset $\mathscr{D} = \{(X_i, \varphi(X_i))\}_{i=1}^{n}$
2. See if we can adapt $f_\theta(\,.\,)$ to $\varphi(\,.\,)$ on $\mathscr{D}$.

Examples:

- $\sup_{X \in \mathscr{D}} \|f_\theta(X) - \varphi(X)\| \approx 0$
- $\sum_{i=1}^{n} \|f_\theta(X_i) - \varphi(X_i)\|^2 \approx 0$
- Accuracy $\approx 100\,\%$







We call this *Training*

Our new *Goal*:

1. Sample some dataset $\mathscr{D} = \{(X_i, \varphi(X_i))\}_{i=1}^{n}$
2. See if we can adapt $f_\theta(\,.\,)$ to $\varphi(\,.\,)$ on $\mathscr{D}$.

How can we do it?

- Solving a non-linear system:

$$\begin{cases} f_\theta(X_1) = \varphi(X_1) \\ f_\theta(X_2) = \varphi(X_2) \\ \quad\vdots \\ f_\theta(X_n) = \varphi(X_n) \end{cases}$$

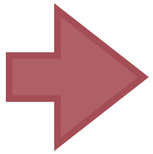
The good algorithms to solve (non-linear) systems are minimizations.

- How do you find it computationally?

Message 1

We have:

- A dataset.
- Some compute.



Training means using those to get a model

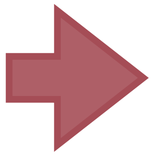
Training means minimizing an error

Learnability *vs* Optimization

We have:

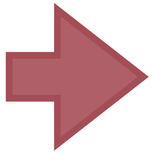
- A dataset.
- Some compute

  (Memory + number of computations constraint).



Do we have enough samples?

Learnability



In case, does the training routine find it within the compute?

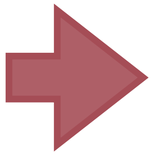
Optimization

Naive Example

- With $\varphi(X) = poly(k, x)$.
- E.g., $\varphi(x[1], x[2], \ldots x[d]) = x[1]^k + x[2]^2 x[3] x[7]^{k-3} + 20x[d]^7$

- Solving a non-linear system:

$$\begin{cases} f_\theta(X_1) = \varphi(X_1) \\ f_\theta(X_2) = \varphi(X_2) \\ \vdots \\ f_\theta(X_n) = \varphi(X_n) \end{cases}$$



By hand, we need d^k samples to set the d^k coefficients, with $> d^k$ computations!

LEARNABILITY

What does it depend on?

- The number of samples needed depend on:
  - The difficulty of $\varphi(\,.\,)$
  - dimensionality of X.

- Vision has 10^3-dim inputs and 10^4 to 10^6 samples

Do we have enough samples?

Learnability

At most $n \sim d^2$ is viable

LEARNABILITY

The questions are:

- What are the most information I can have in n datapoints?

- How can I design a training pipeline that uses all of that efficiently?

Do we have enough samples?

Learnability

What is Optimization

- Let $\theta = (\theta_1, \theta_2, \ldots, \theta_d) \in \mathbb{R}^d$
- Let $L\colon \mathbb{R}^d \to \mathbb{R}$
- I want $\theta \in \mathbb{R}^d$ that minimizes $L(\theta)$:

$$\min_{\theta=(\theta_1,\theta_2,\ldots)} L(\theta)$$

We want the *best* algorithm (path downhill) for that!

COMPUTE&MEMORY CONSTRAINTS

- We have n samples.
- We have N parameters.

In case, does the training routine find it within the compute?

Optimization

Previously: *Reporting to BIS if* $Compute > 10^{26} \approx \mathcal{O}(n \times N)$ *for GPT-4.*

On **October 30, 2023**, President Biden signed **Executive Order (EO) 14110**. It **required** developers training *potential dual-use foundation models* above certain thresholds to **report information to the U.S. Department of Commerce**. Until Commerce set permanent criteria, the EO set an initial trigger at **>10^26 computational operations (e.g., FLOPs)** (and a lower 10^23 threshold for models trained mainly on biological sequence data). It also required reporting on things like **red-team test results** and **who controls the model weights and how they're protected**. That is *reporting*, not a blanket right for the President to "inspect all models." The White House

Commerce then proposed how to **implement** those reporting obligations: on **September 11, 2024**, the Bureau of Industry and Security (BIS) published a proposed rule that would have required **quarterly notifications** for any training run **using >10^26 operations** and allowed BIS to ask follow-up questions (e.g., about safety testing and how weights are safeguarded). Again, this is an information-reporting regime, **not** government access to the models or weights themselves. Federal Register

Important context today: on **January 23, 2025**, President Trump issued **EO 14179 ("Removing Barriers to American Leadership in AI")**, which **revoked EO 14110** and told agencies to suspend or rescind actions taken under it. In other words, the 2023 reporting framework is **no longer in force**. Federal Register

Compute&Memory Constraints

- We have n samples.
- We have N parameters.

In case, does the training routine find it within the compute?

Optimization

From the naive $n \times N^k$ compute/ memory **per step** of a $k-th$ order optimization algorithm

Hopefully we can optimize in:
$\mathcal{O}(n \times N)$ **total compute**
and $\mathcal{O}(N)$ **total memory.**

COMPUTE&MEMORY CONSTRAINTS

- The dimensionality N of the problem $\left(\theta \in \mathbb{R}^N\right)$ is very high, e.g. 1.7 trillion for GPT-4.

$\Longrightarrow$

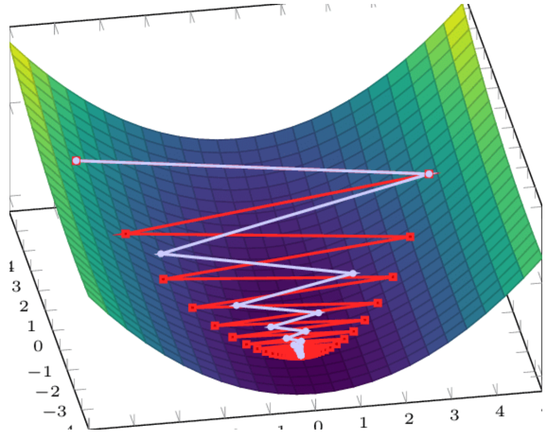
Gradient-based

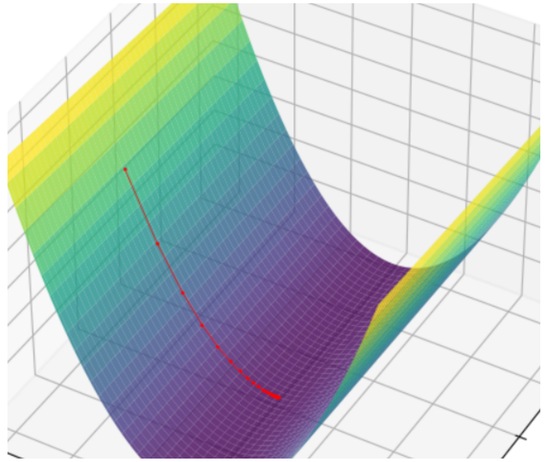
$$\theta_{k+1} = \theta_k - \eta \nabla L(\theta_k)$$

Gradient Descent

- We can look where the gradient points and move back of a small step $\eta > 0$, and reiterate

$$\theta_{i+1} = \theta_i - \eta \cdot \nabla_\vartheta L(\vartheta = \theta_i)$$

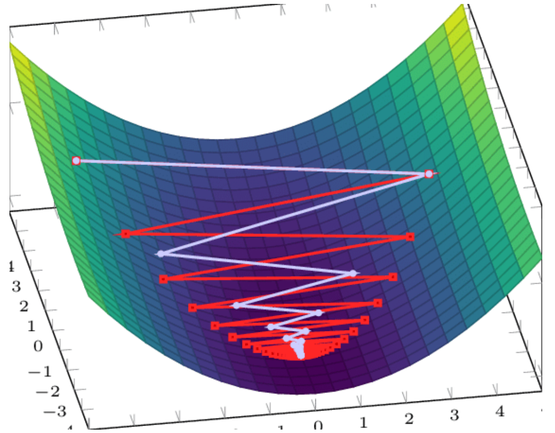


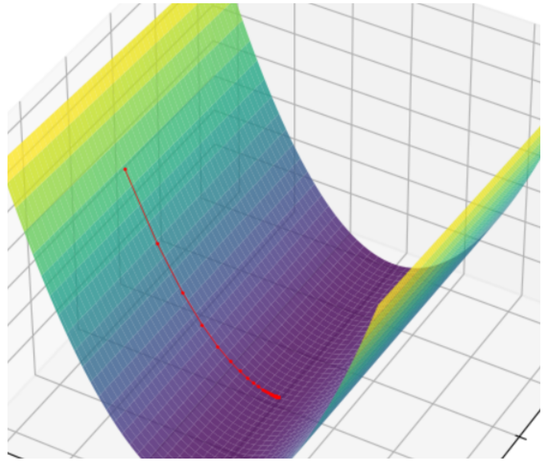


Cauchy (1847), Hadamard (1907), Curry (1944).

Adaptive + Acceleration

- We need (1) Stability, (2) Faster convergence:
  - Expected moving average of the previous gradients.
  - Adaptive stepsize (resizing given the per parameter step-STD).





Polyak (1964), Nesterov (1983),
Duchi (2011), Kingma and Ba (2015).

COMPUTE&MEMORY CONSTRAINTS

- The dimensionality N of the problem $\left(\theta \in \mathbb{R}^N\right)$ is very high, e
- We have many n loss is an

$$\frac{1}{n}\sum_{x \in D} L(\theta, x)$$

We work with:
(for both *learnability* and *compute*)

Mini-batch Gradient Methods

Gradient-based

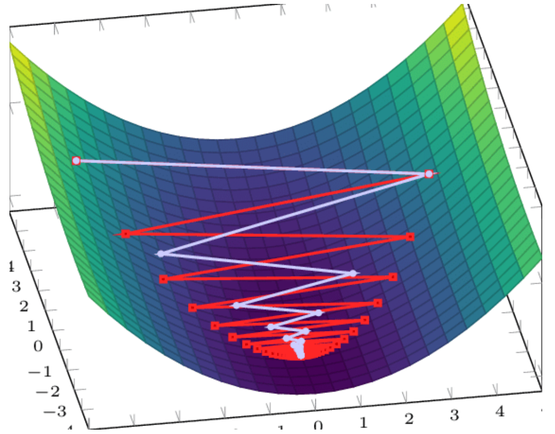
$$\theta_{k+1} = \theta_k - \eta \nabla L(\theta_k)$$

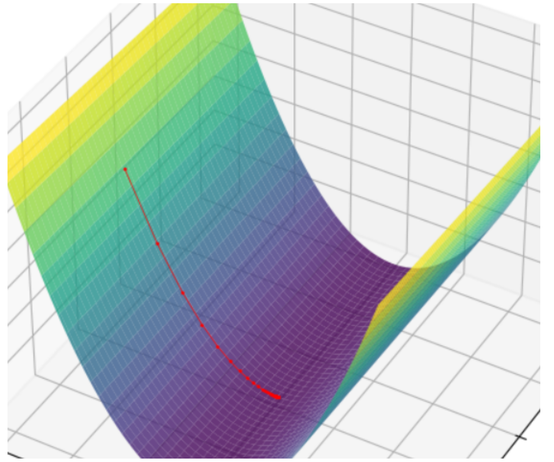
Mini-batch methods

Mini-Batch Gradient Descent

- We have the dataset $\mathscr{D}$, we sample a batch B_{i+1}

$$\theta_{i+1} = \theta_i - \eta \frac{1}{|B_{i+1}|} \cdot \sum_{X \in B_{i+1}} \nabla_\vartheta L(\vartheta = \theta_i, X)$$



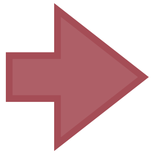


Robbins & Monroe (1951), Rosenblatt (1958) independently, Rumelhart et al. (1986).

Message 1

We have:

- A dataset.
- Some compute.

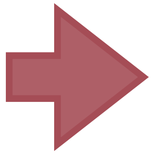


Training means using those to get a model

Message 1b

We have:

- A dataset.
- Some compute.



Enough samples?

Learnability

Enough Compute?

Optimization

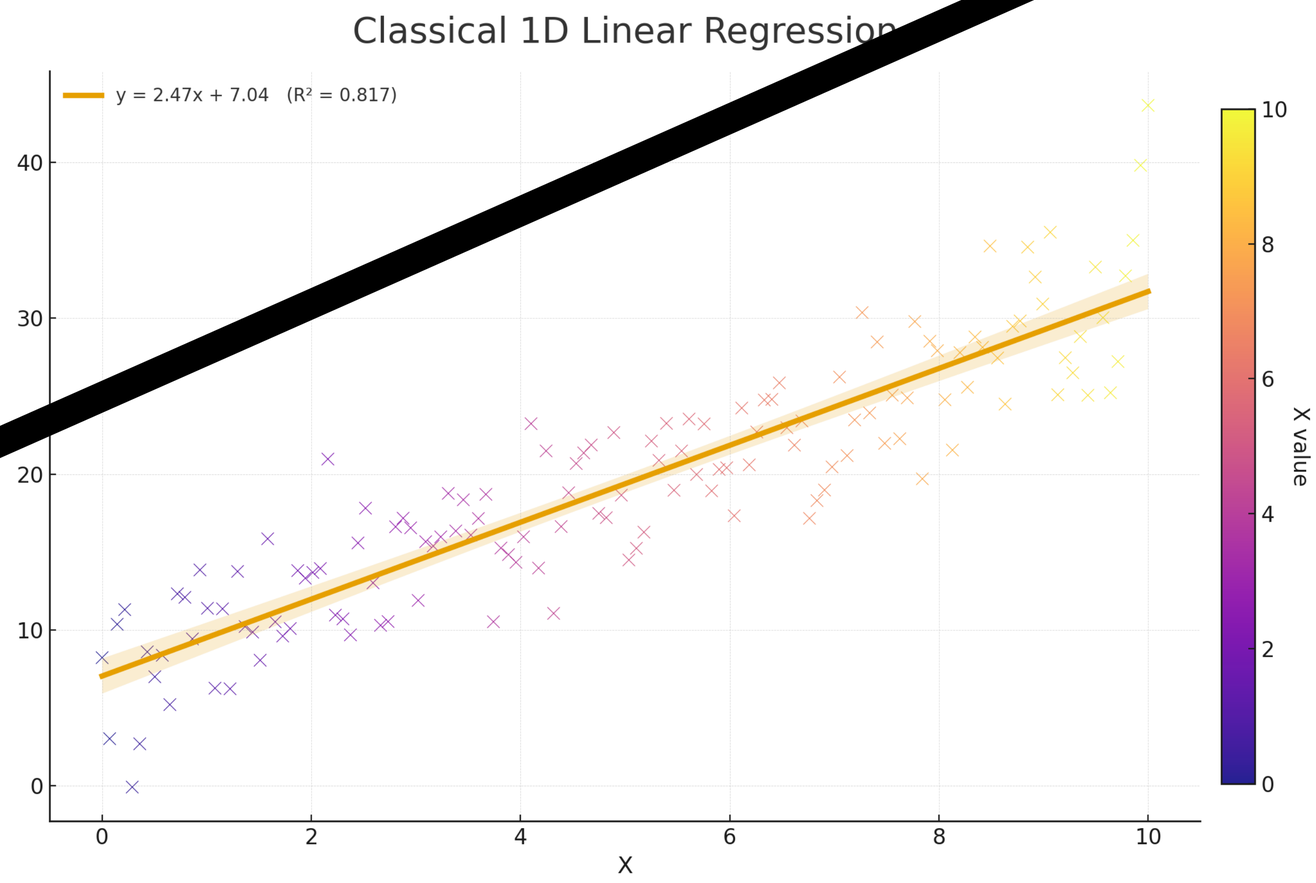
Part 1

The regimes

Regime 1

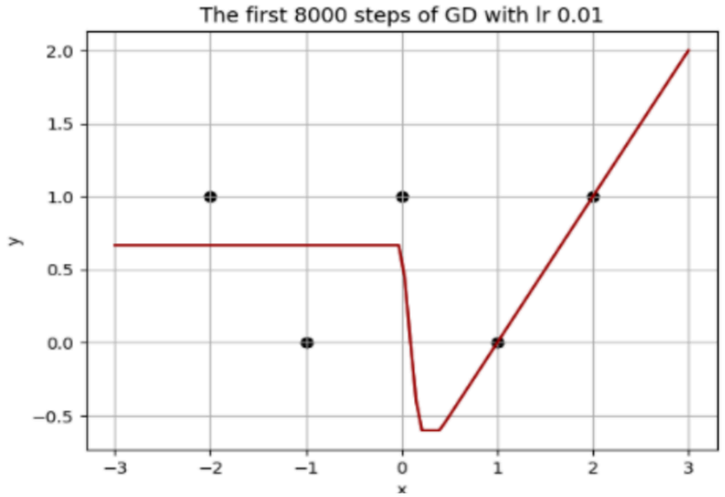
- Enough noisy data
- Enough compute

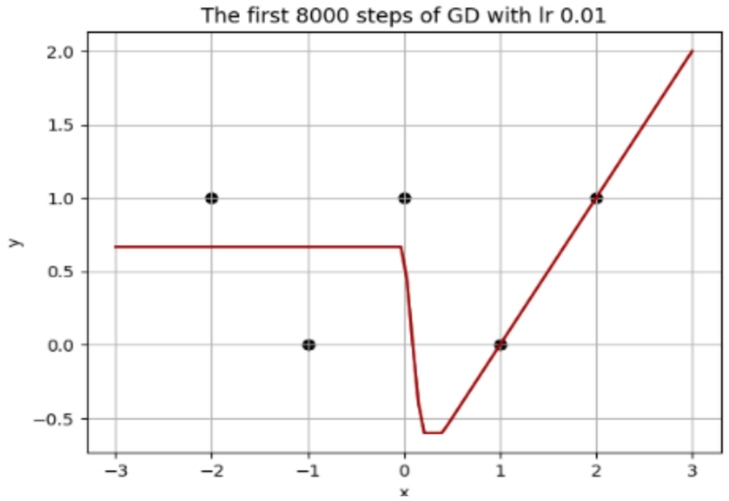
Lorenzo dealt with it.

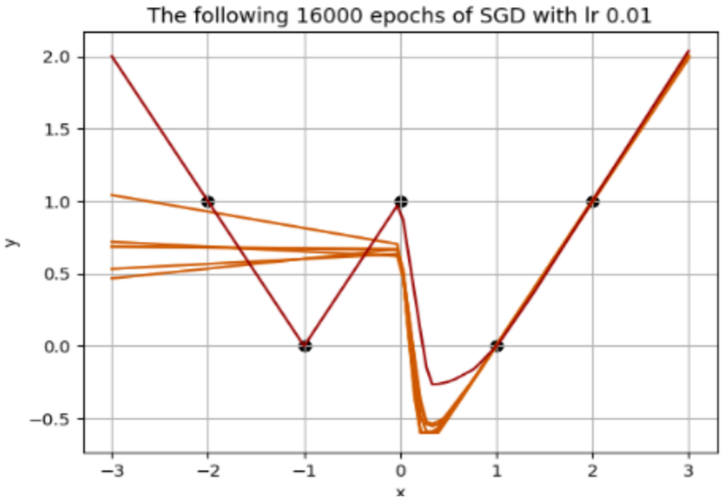


REGIME 2

- ***Not enough exact*** data!
- Enough compute

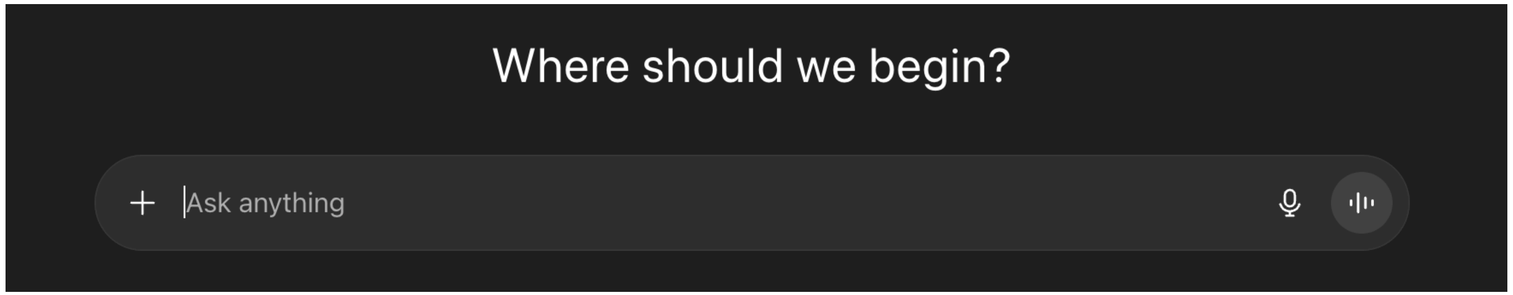






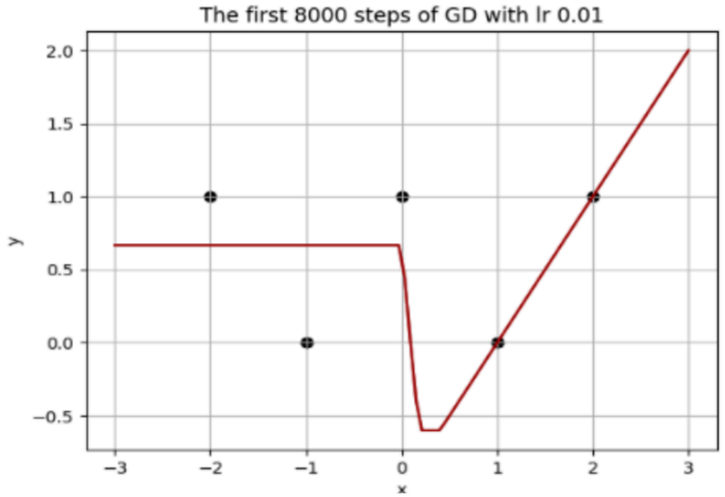
Regime 3

- Enough exact data (all of them!)
- ***Not enough*** compute!



REGIME 2

- ***Not enough exact*** data!
- Enough compute



We want a model that:

- Can fit them all.
- It's the simplest such model.

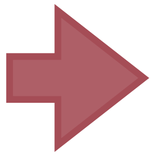
A training pipeline which goes to the best global minima.

Regime 3

- Enough exact data (all of them!)
- ***Not enough*** compute!

We want a model that:

- (Just) can fit them all.

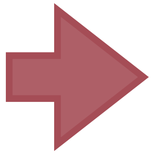


A training pipeline that allows it within my compute!

Message 1

We have:

- A dataset.
- Some compute.



Training means using those to get a model

Message 2

We have 2 regimes *(bottlenecks)*:

- Not enough data.
- Not enough compute.

Part 2

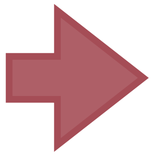
Small Digression on Learnability

Learnability *vs* Optimization

We have:

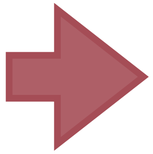
- A dataset.
- Some compute

  (Memory + number of computations constraint).



Do we have enough samples?

Learnability



In case, does the training routine find it within the compute?

Optimization

LEARNABILITY

The questions are:

- What are the most information I can have in n datapoints?

- How can I design a training pipeline that uses all of that efficiently?

Do we have enough samples?

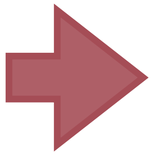
Learnability

Naive Example

- With $\varphi(X) = poly(k, x)$.
- E.g., $\varphi(x[1], x[2], \ldots x[d]) = x[1]^k + x[2]^2 x[3] x[7]^{k-3} + 20x[d]^7$

- Solving a non-linear system:

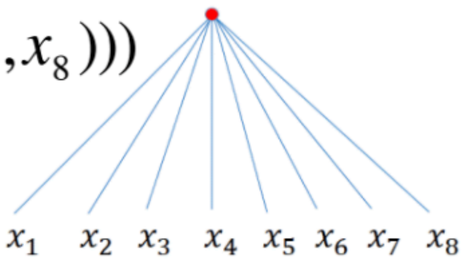
$$\begin{cases} f_\theta(X_1) = \varphi(X_1) \\ f_\theta(X_2) = \varphi(X_2) \\ \vdots \\ f_\theta(X_n) = \varphi(X_n) \end{cases}$$

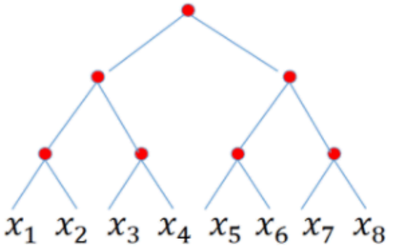


By hand, we need d^k samples to set the d^k coefficients, with $> d^k$ computations!

Deep NNs can avoid the curse for compositional functions

$$f(x_1,x_2,...,x_8) = g_3(g_{21}(g_{11}(x_1,x_2),g_{12}(x_3,x_4))g_{22}(g_{11}(x_5,x_6),g_{12}(x_7,x_8)))$$





Theorem, Mhaskar, Poggio, Liao 2016 (informal statement)

Suppose that a target function of d variables is hierarchically locally compositional, that is has *compositional sparsity*. Both shallow and deep network can approximate f equally well. For a target function with the binary tree graph, the number of parameters of the shallow network depend exponentially on the dimension - that is $O(\epsilon^{-d})$ - whereas the number of parameters of the deep network (with binary tree architecture) is $O(d\epsilon^{-2})$.

H. N. Mhaskar and Poggio, T., "Deep vs. shallow networks: An approximation theory perspective", Analysis and Applications, vol. 14, no. 06, pp. 829 - 848, 2016.

Single Index Model

Do we have enough samples?
To learn the constituent sparse functions?

Learnability

Let $X \sim N(0, I_d)$ be a Gaussian $d - dim$ random v, $\varphi(X) = H_k(X)$.

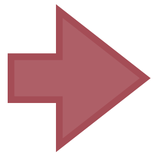
Let $X \sim B(1/2)^d$ be uniform $\{-1,1\}^d$ random v, $\varphi(X) = \prod_{i=1}^{k} X[i]$ *(k−juntas/parity)*.

How many samples do we need to recover them with a NN?

LEARNABILITY

- The number of samples needed depend on:
  - The difficulty of $\varphi(\,.\,)$
  - dimensionality of X.

- Vision has 10^3-dim inputs and 10^4 to 10^6 samples



Do we have enough samples?

Learnability

At most $n \sim d^2$ is viable

Single Index Model

Let $X \sim N(0, I_d)$ or $X \sim B(1/2)^d$ be a Gaussian $d - dim$ random v, $\varphi(X) = H_k(X)$.

Let $X \sim B(1/2)^d$ be uniform $\{-1,1\}^d$ random v, $\varphi(X) = \prod_{i=1}^{k} X[i]$ *(k−juntas/parity)*.

How many samples do we need to recover them with a NN?

- **Information theoretic** (noiseless): d.
- **Online SGD** (with noise): d^i where $i = \min\{k \,|\, \langle \varphi, H_k(X) \rangle \neq 0\}$.
- **Multi-pass SGD** (with noise): d^l **generative exponent** *(Damian et al.)*.

Single Index Model

Let $X \sim B(1/2)^d$ be uniform $\{-1,1\}^d$ random v, $\varphi(X) = \prod_{i=1}^{k} X[i]$ *(k−juntas/parity)*.

- **Information theoretic** (noiseless): d.
- **Online SGD** (with noise): d^i where $i = \min\{k \,|\, \langle \varphi, H_k(X) \rangle \neq 0\}$.

Cornacchia, Mikulincer, Mossel (2024): **biased distribution**

If $X \sim B(p)$ with $p \sim Unif(1/2 - \eta, 1/2 + \eta)$:

With high probability over p_i, SGD on a 2-layer network of size $O(2^k d\eta^{-k})$ can learn any k-sparse function with $\tilde{O}(d\eta^{-2k})$ samples and $\tilde{O}(\eta^{-2k})$ steps.

LEARNABILITY

- The number of samples needed depend on:
  - The difficulty of
  - dimensionality (

- Vision has 10^3-dim inputs and 10^4 to 10^6 samples

Do we have enough samples?

Learnability

Not solved, but:

There is hope!

At most $n \sim d^2$ is viable

Part 3

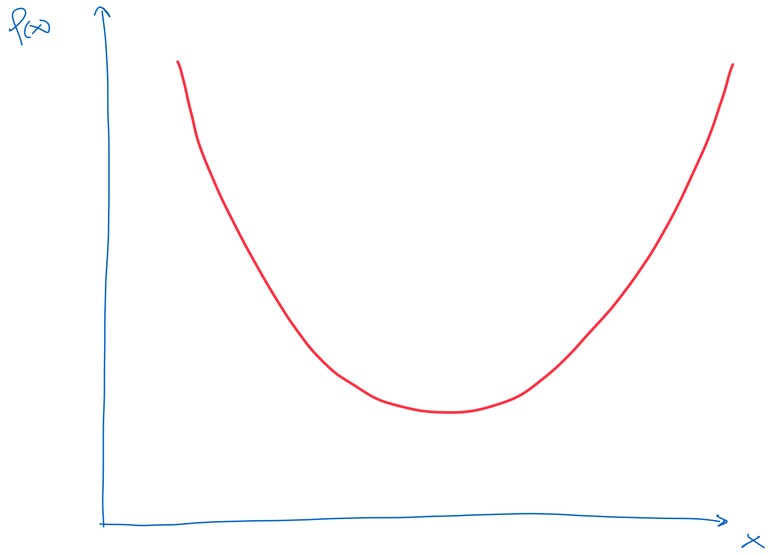
Classical optimization theory
&
Which algorithms

Ingredient 1: Convexity

- A convex function is a function that is curved up-ward at every point.

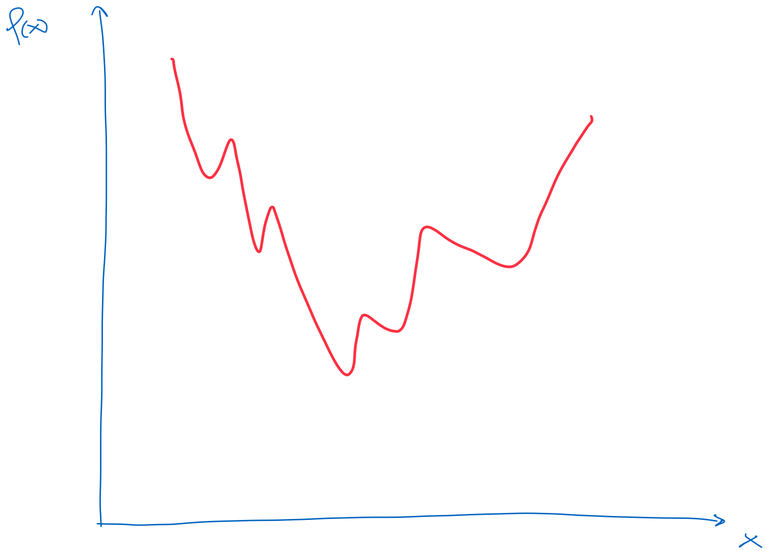
- A convex function is a function in which if you go down you go to a minimum.

Convexity is about:
the local steps
work out globally

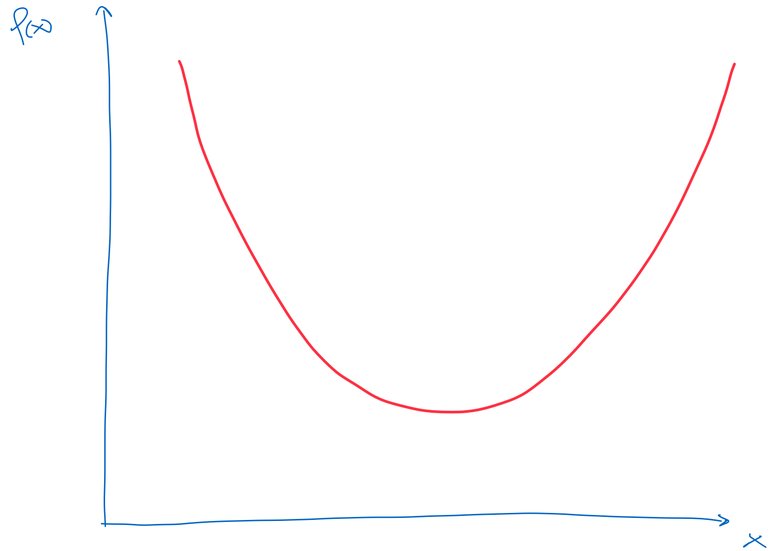


Convex

Convexity

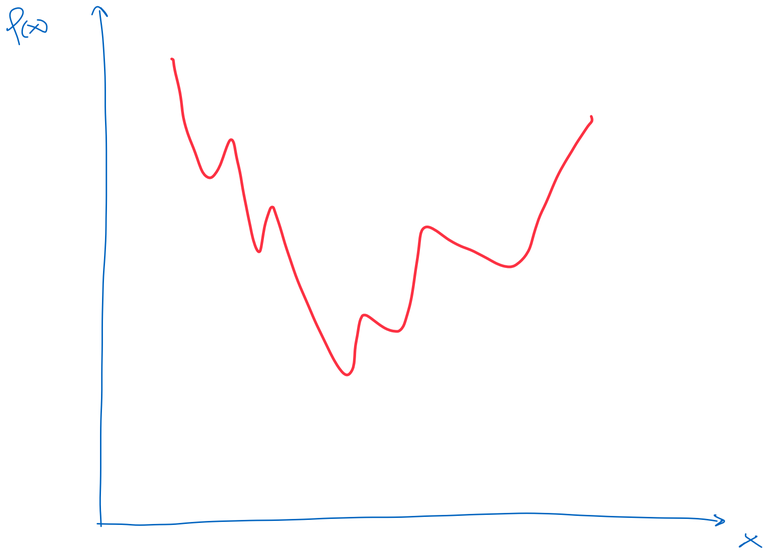


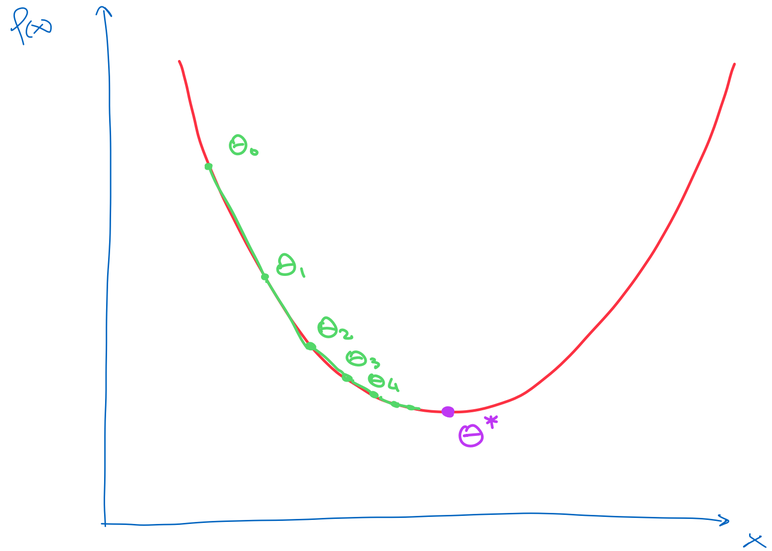
Non-Convex



Convex

Convexity

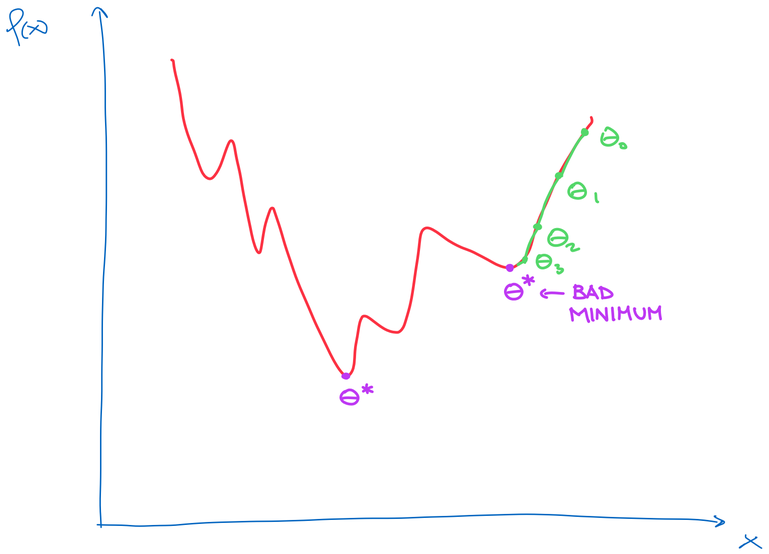


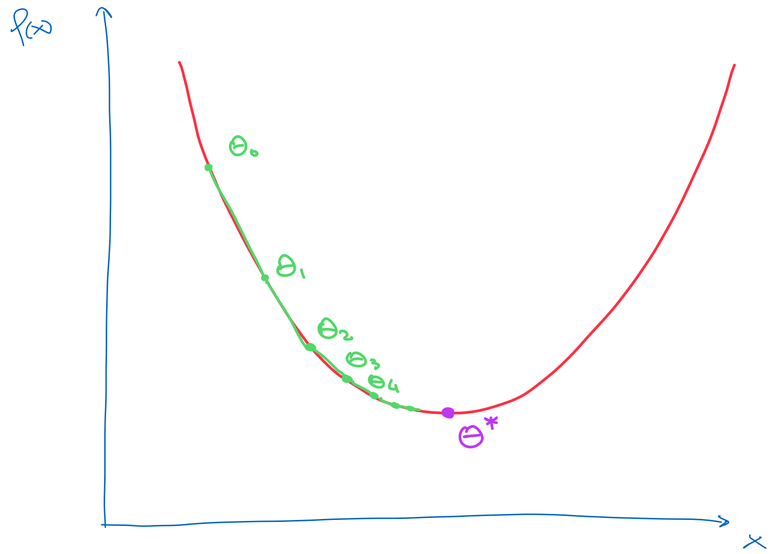


Non-Convex

Convex

Convexity





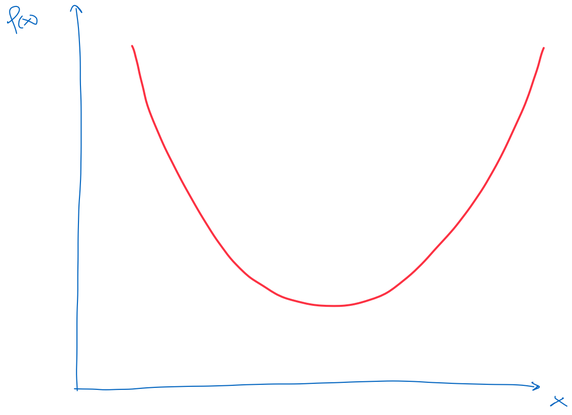
Non-Convex

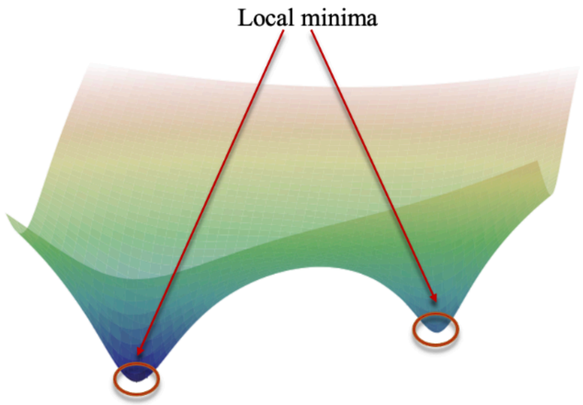
Convex

Lack of Convexity

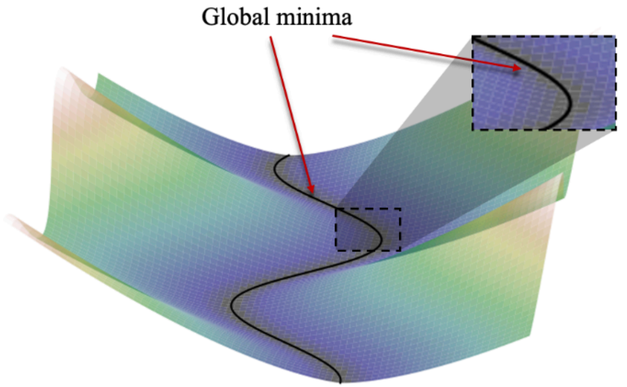
Is Neural Networks' training loss convex?

Let's play with an example





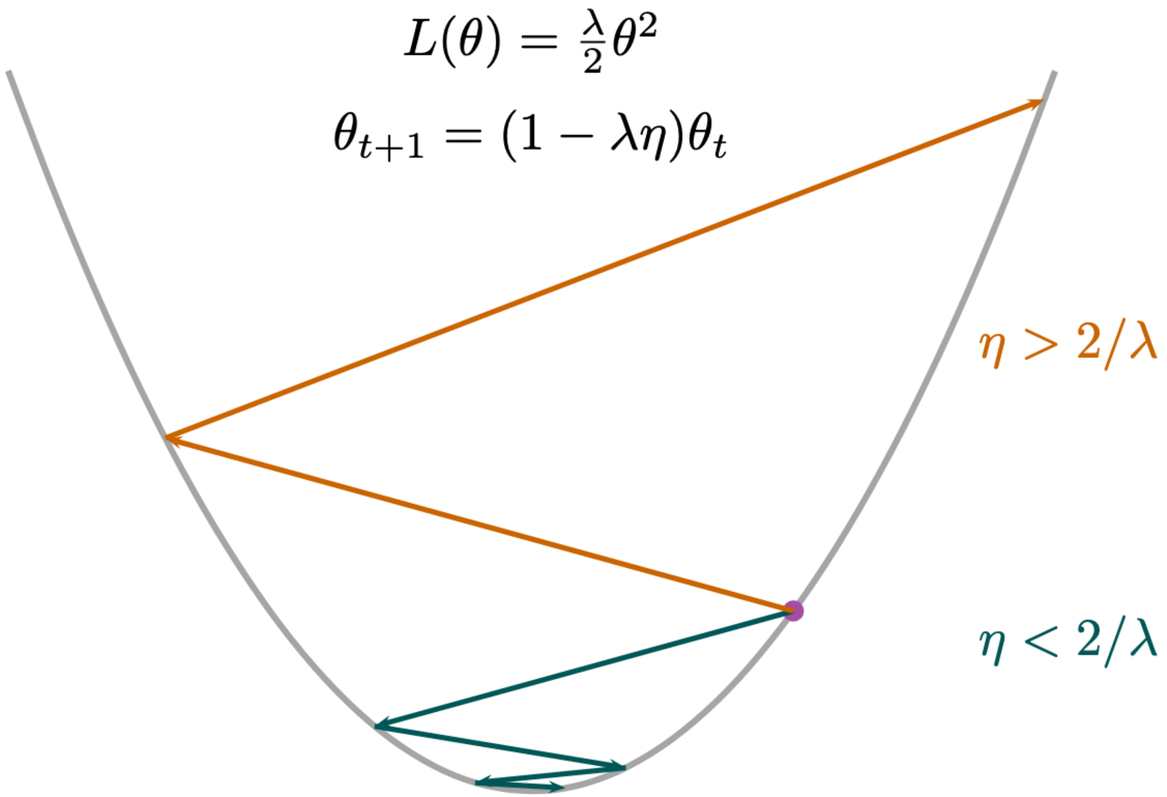
(a) Loss landscape of under-parameterized models



(b) Loss landscape of over-parameterized models

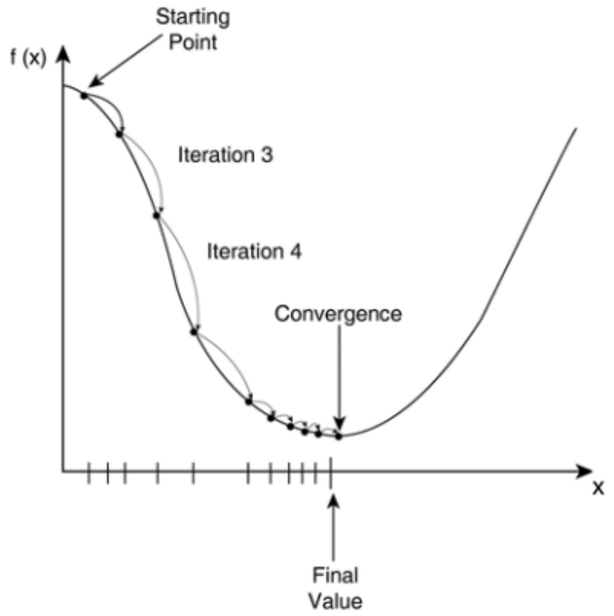
Liu, Zhu, Belkin, (2020), Arxiv link.

INGREDIENT 2: STABILITY



Gradient descent

From Class 4



Let $F : \mathbb{R}^d \to \mathbb{R}$ differentiable, convex and such that

$$\|\nabla F(w) - \nabla F(w')\| \leq B\|w - w'\|$$

(e.g. $\sup_w \|\underbrace{\nabla^2 F(w)}_{\text{Hessian}}\| \leq B$)

γ STEP SIZE

$\gamma = 1/B$

Then

$$w_0 = 0, \quad w_{t+1} = w_t - \boxed{\frac{1}{B}}\nabla F(w_t),$$

converges to a minimizer w^* of F.

θ is now w

B will be $\lambda_{\max}$

$\eta = 1/B$ here!

Message 3

Classically we need some assumptions for

(e.g., Boyd and Vanderberge (2004)):

- You go down $\Longrightarrow$ stable steps
- If you go down you converge well $\Longrightarrow$ convexity/PL

Part 4

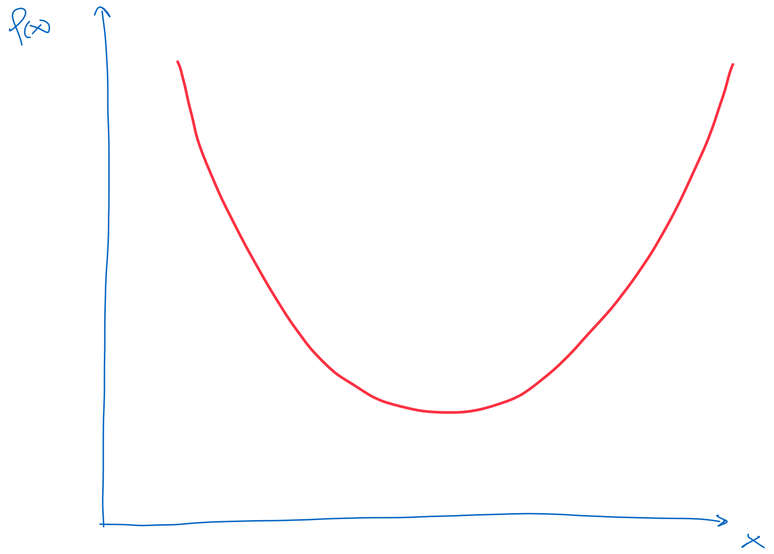
How are the landscape?
&
How do optimizers travel it?

Ingredient 1: Convexity

- A convex function is a function that is curved up-ward at every point.

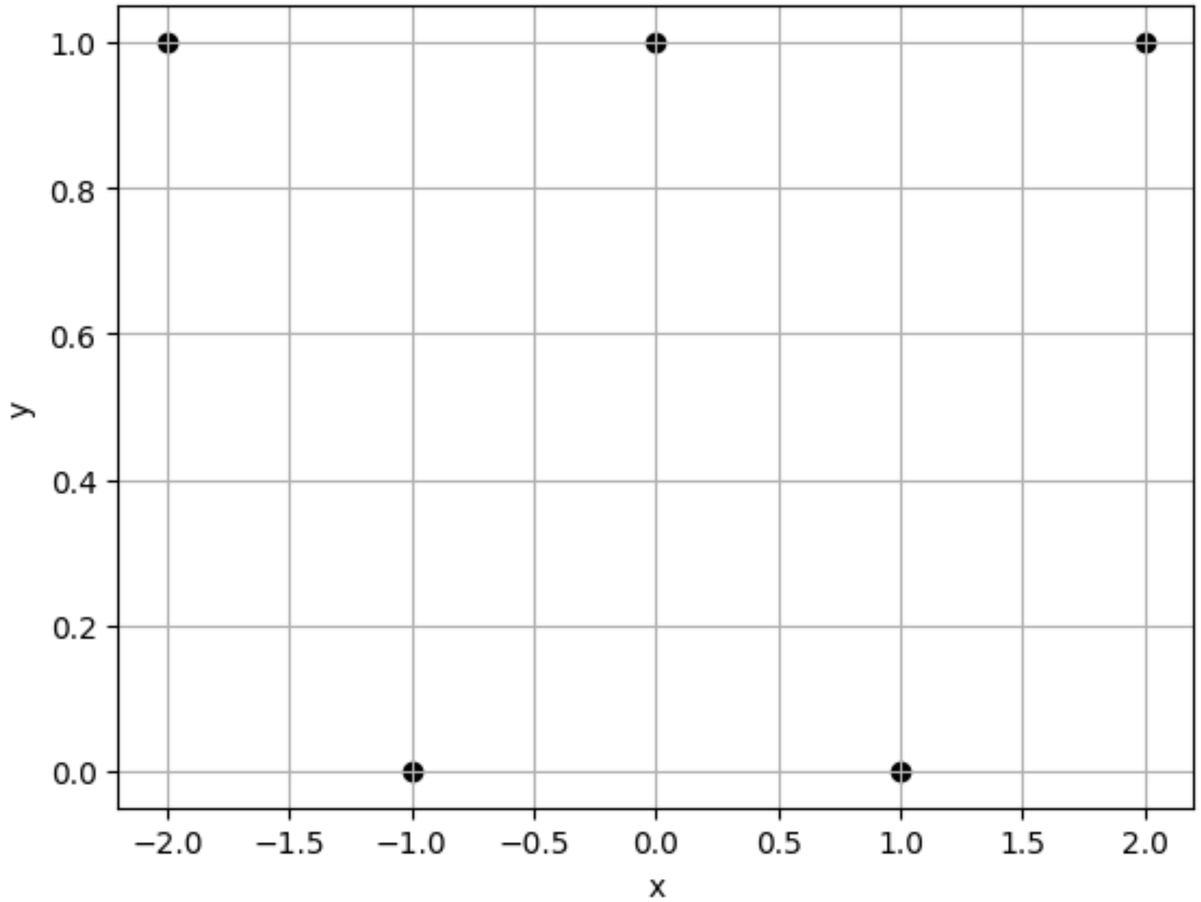
- A convex function is a function in which if you go down you go to a minimum.

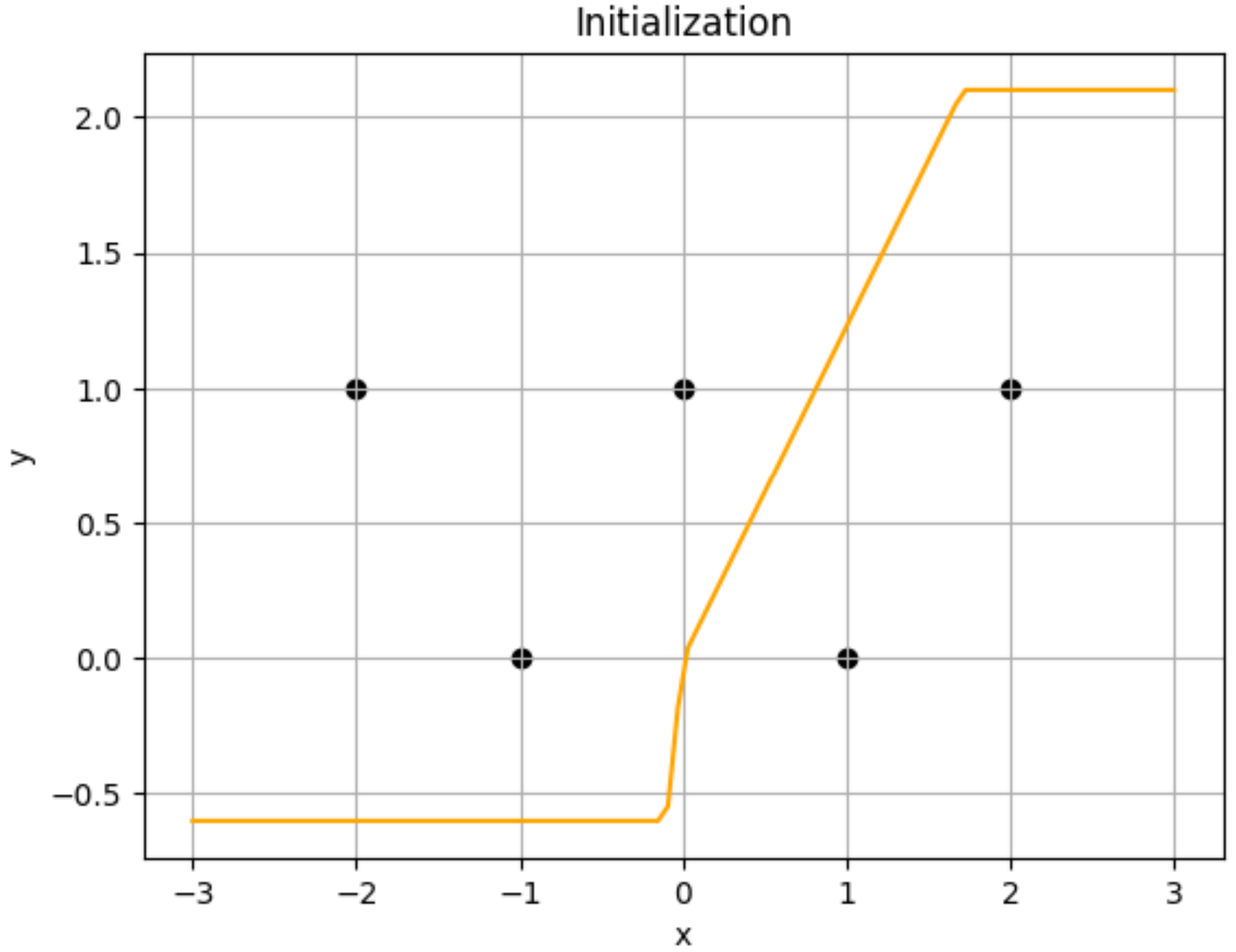
Convexity is about:
the local steps
work out globally



Convex

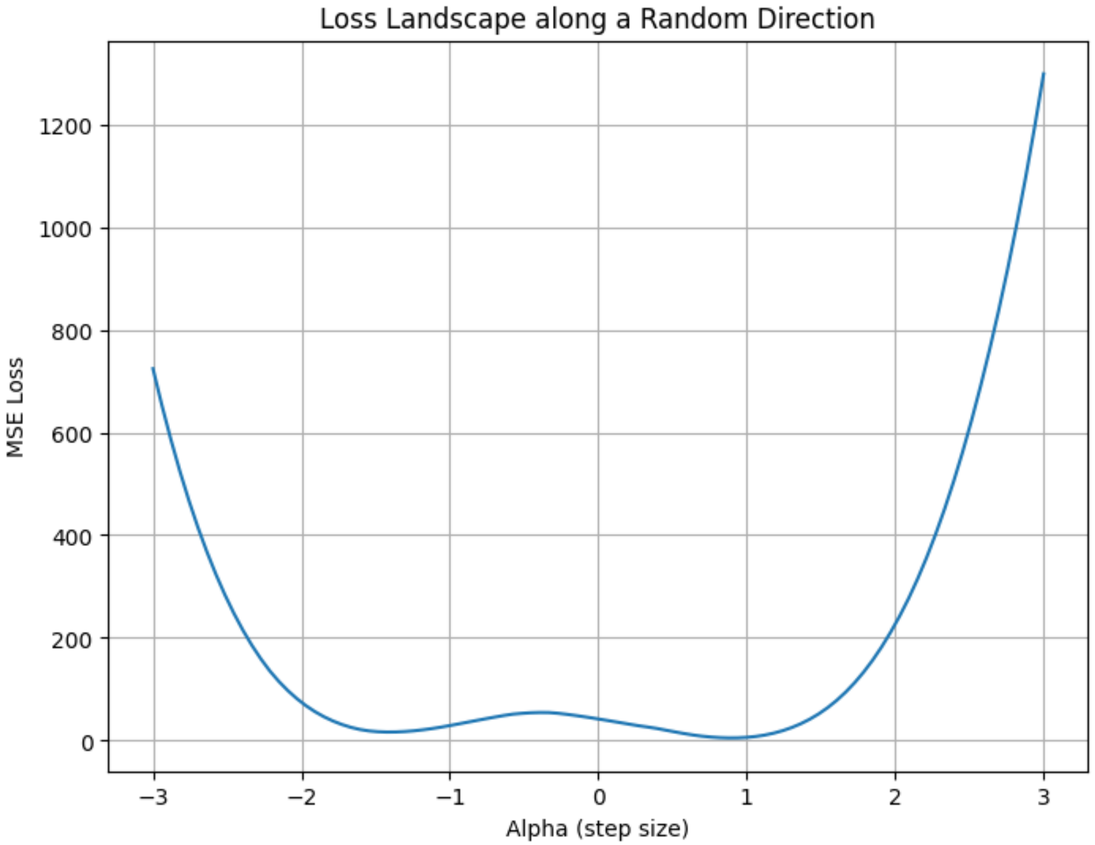
Data on a W

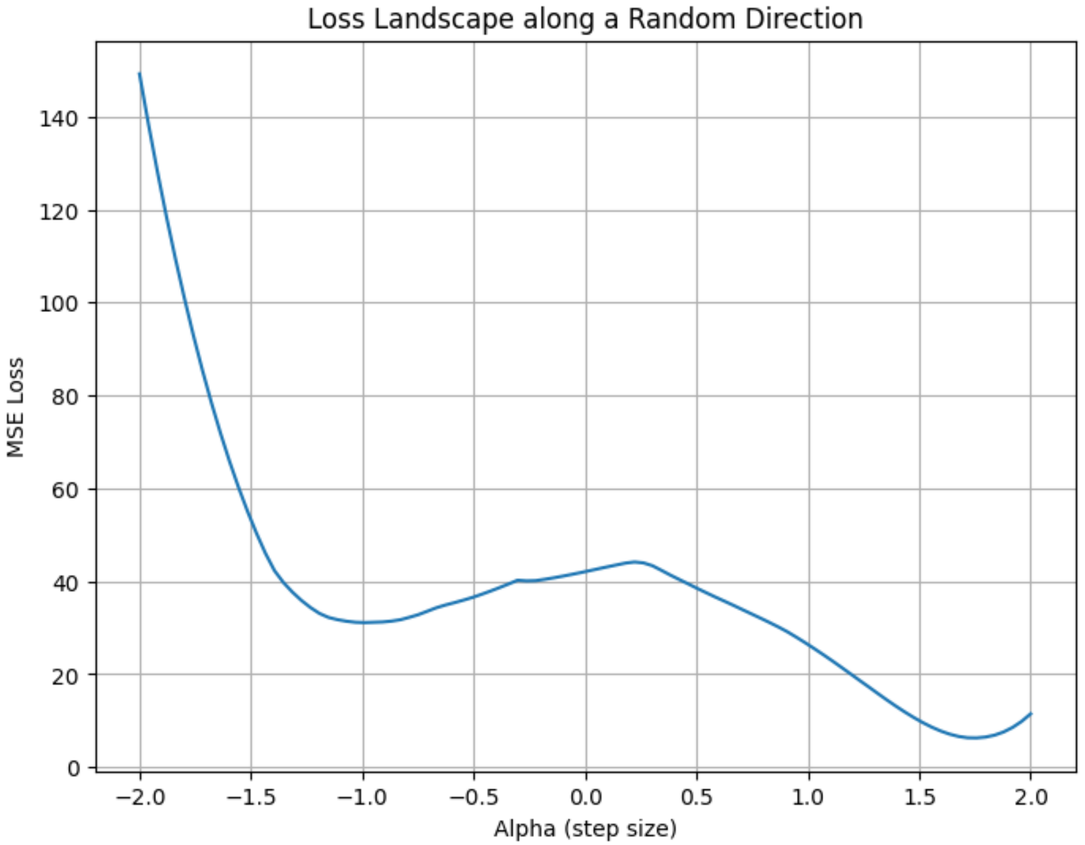




Model $f_{a,b,c}(x) = \sum_{i=1}^{20} a_i \cdot ReLU(b_i x + c_i)$

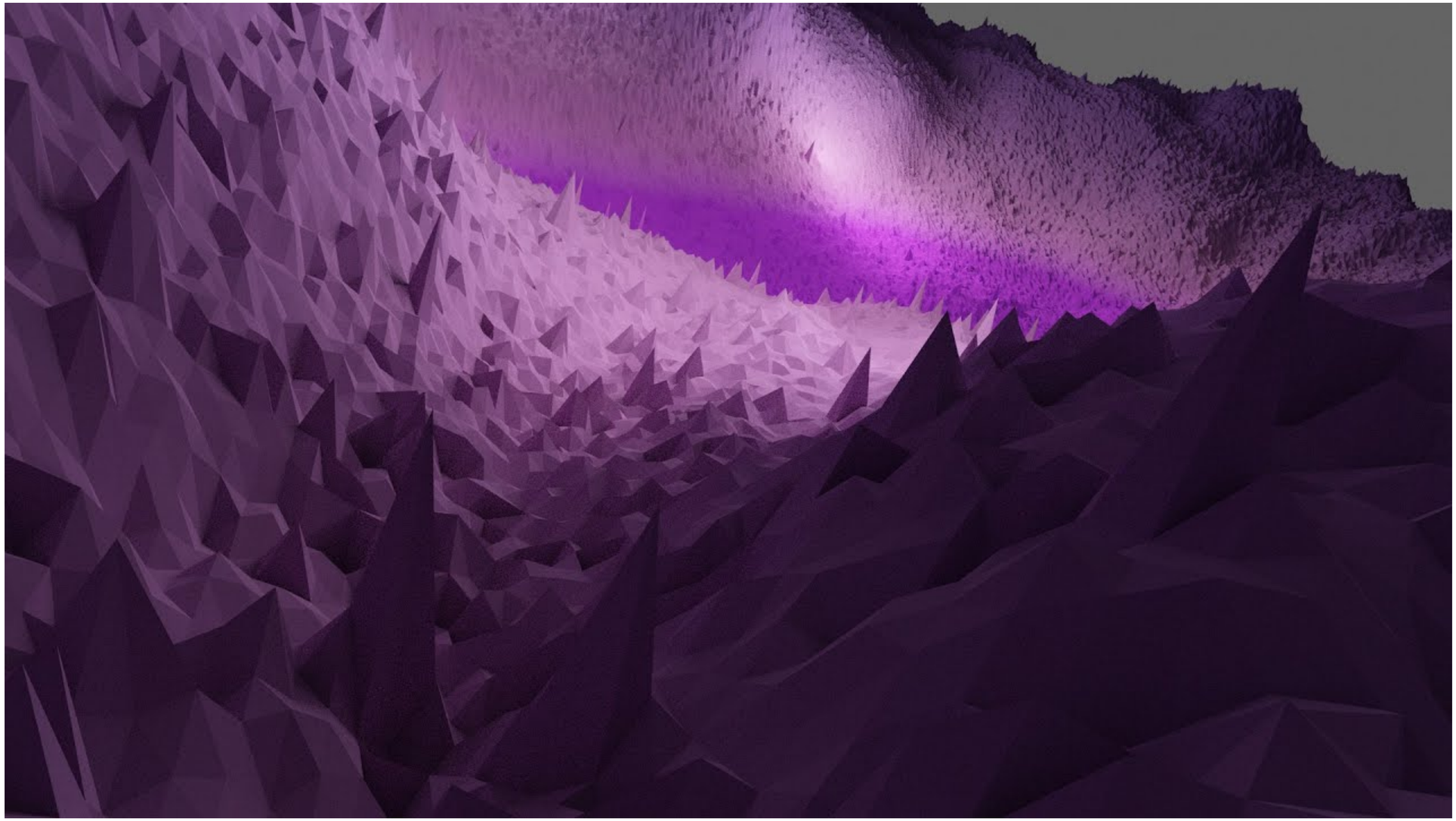
Lack of Convexity



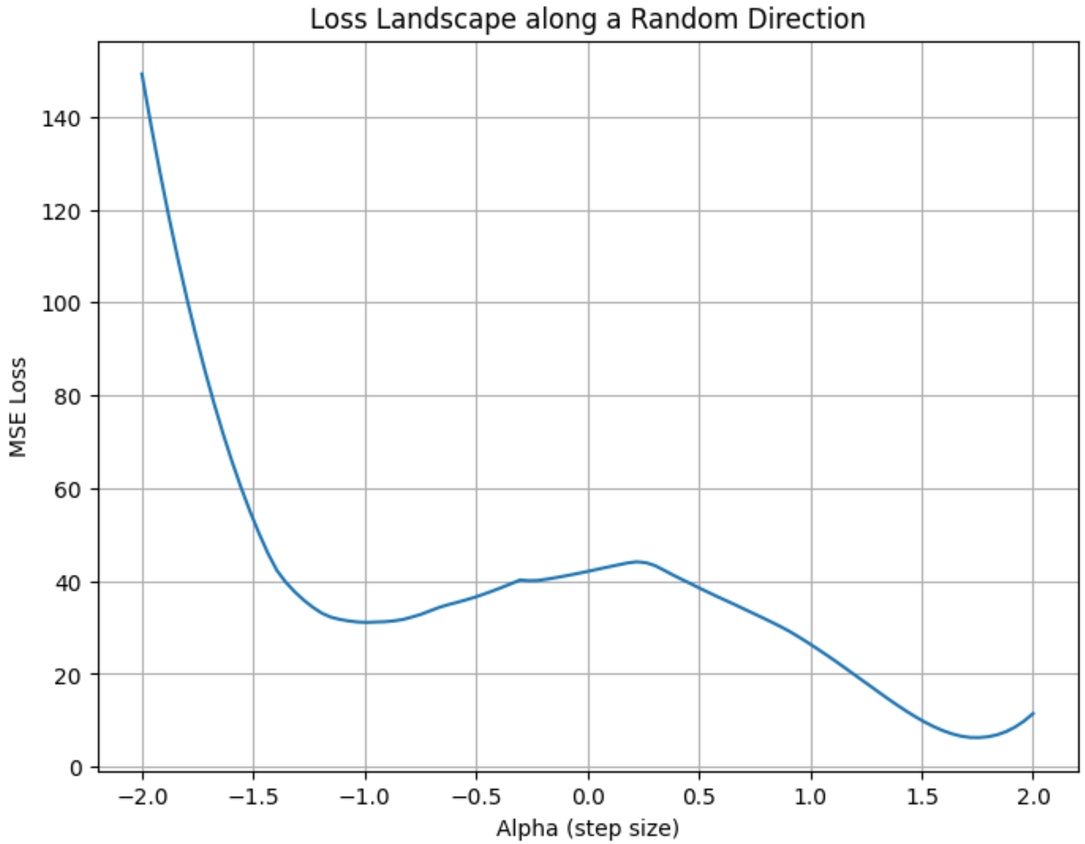


Plot of the loss landscape along a line

Lack of Convexity



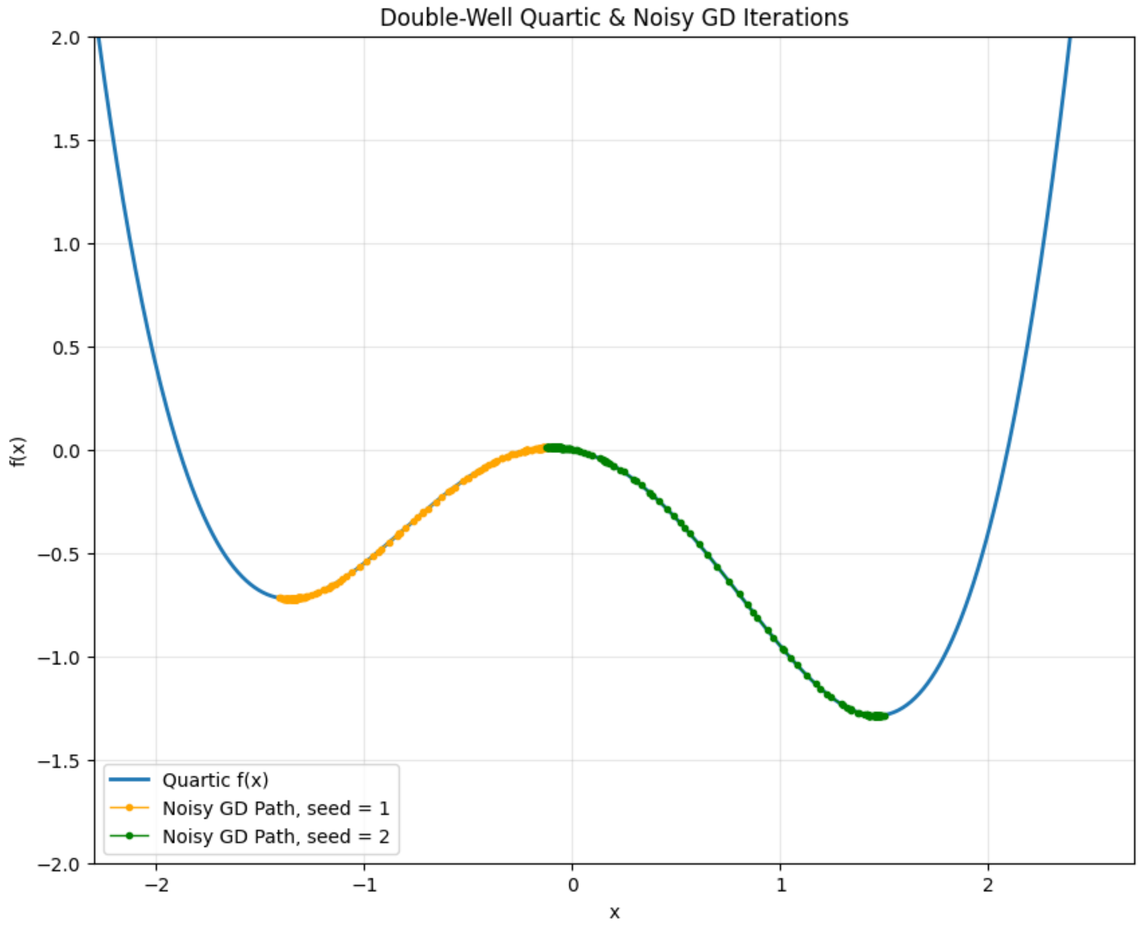
Lack of Convexity



$$f_{a,b,c}(x) = \sum_{i=1}^{20} a_i \cdot ReLU(b_i x + c_i)$$

$(a_i b_i x - a_i c_i - y)^2$ is a quartic

Lack of Convexity



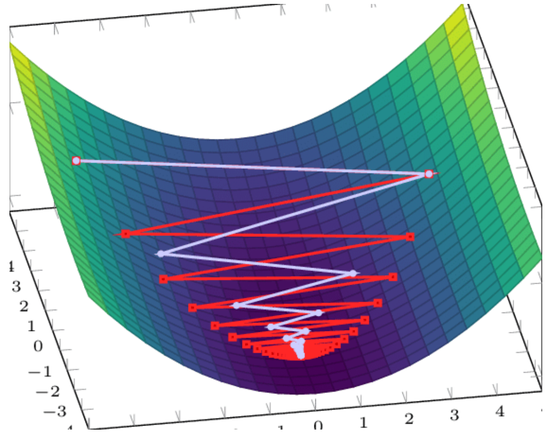
$$f_{a,b,c}(x) = \sum_{i=1}^{20} a_i \cdot ReLU(b_i x + c_i)$$

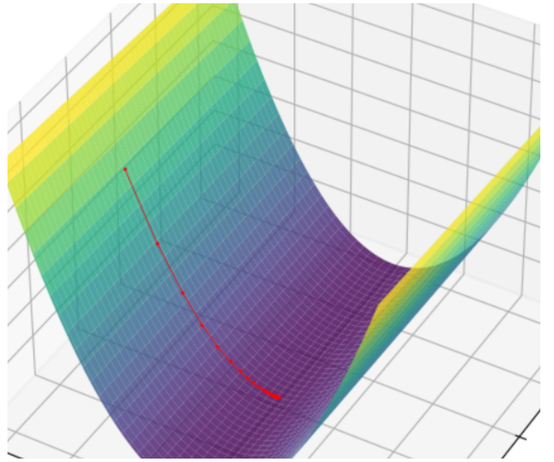
$(a_i b_i x - a_i c_i - y)^2$ is a quartic

$$f(x) = \frac{1}{4}x^4 - x^2 - \frac{1}{5}x$$

For Neural Networks (1950-80s)

- Historically, one relaxes the landscape to convex.
- In neural networks the practice (first such field!):
  - Add neurons.
  - Gradient descent.
  - Prune.

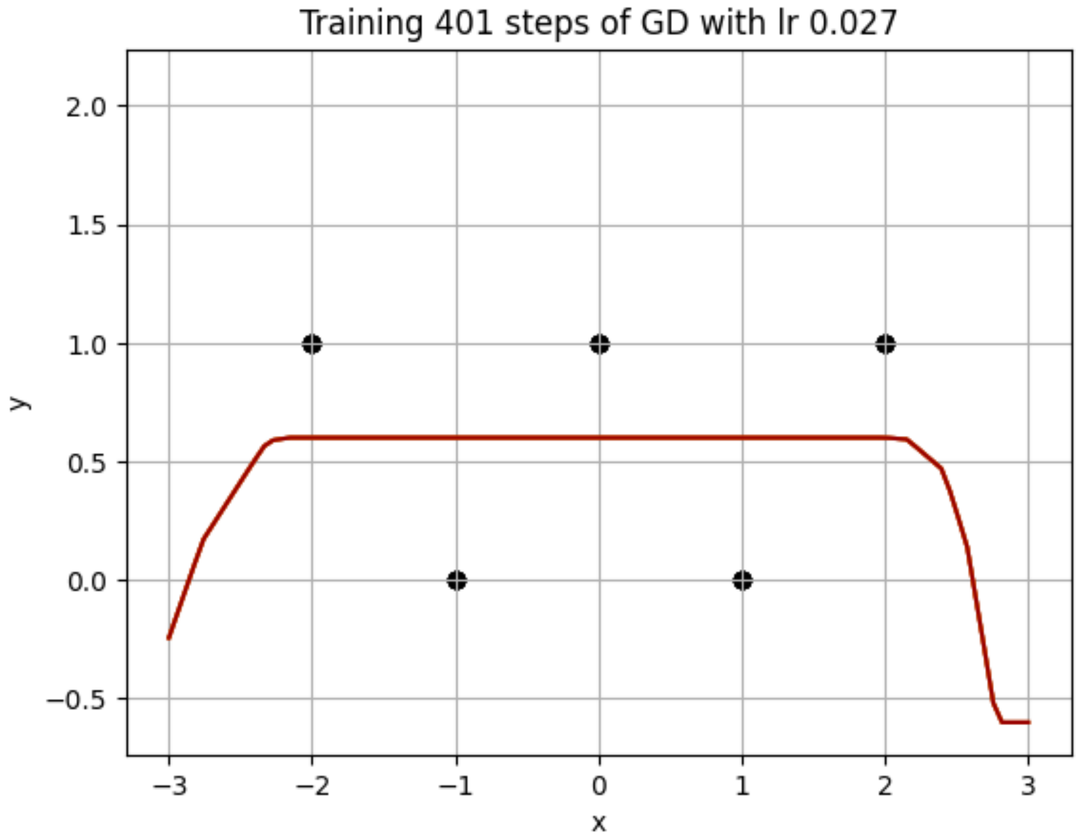


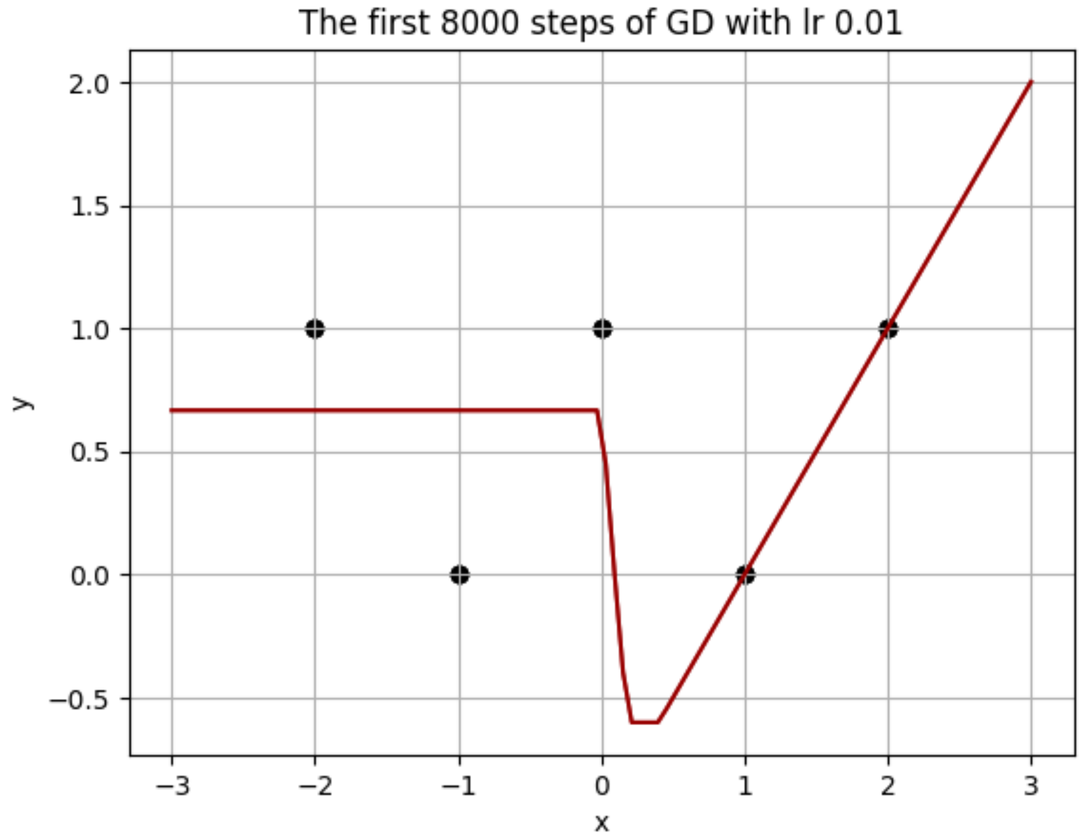


Rumelhart et al. (1986),
Plaut et al. (1986),
Sietsma and Dow (1988).

Optimization Works (?)

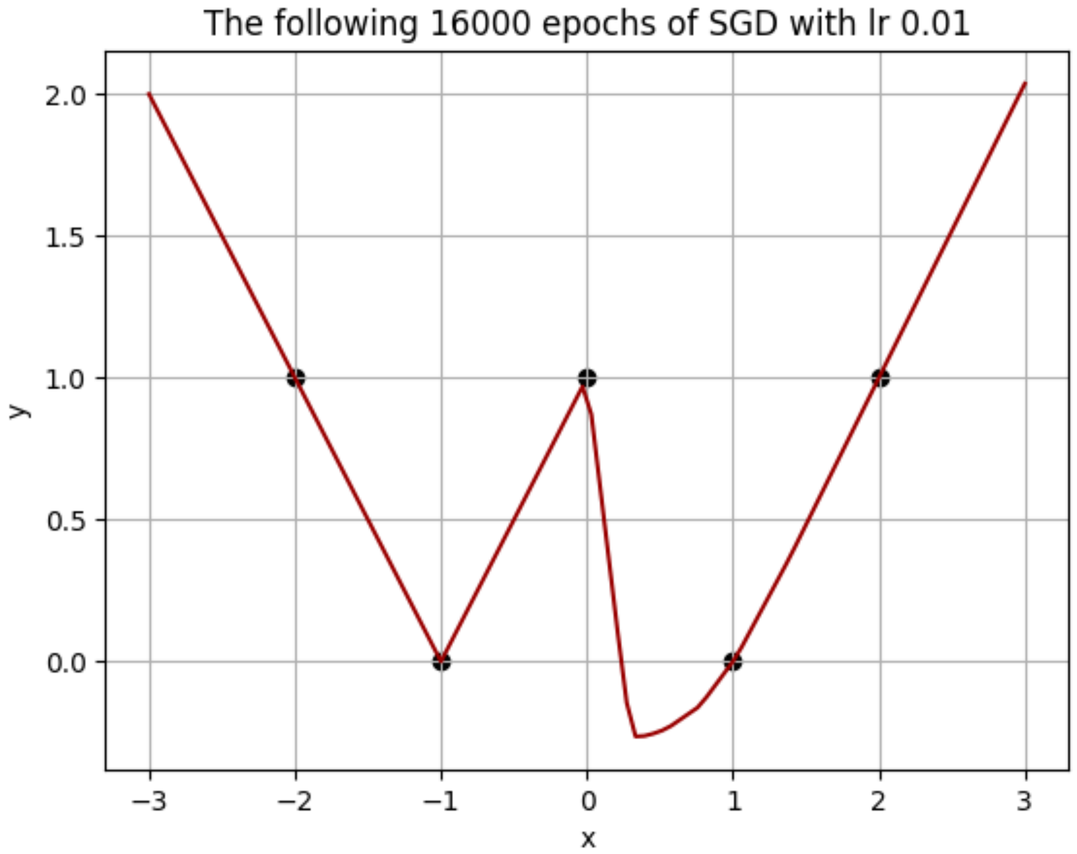
Does GD work?



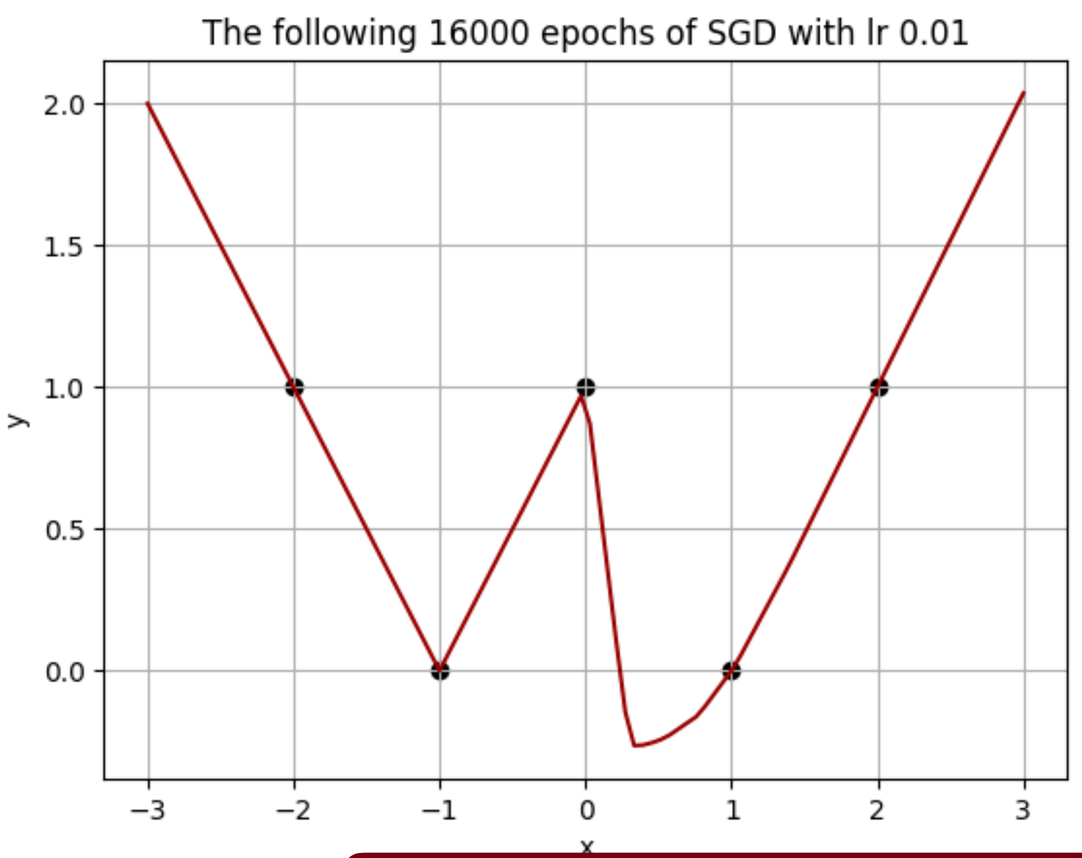


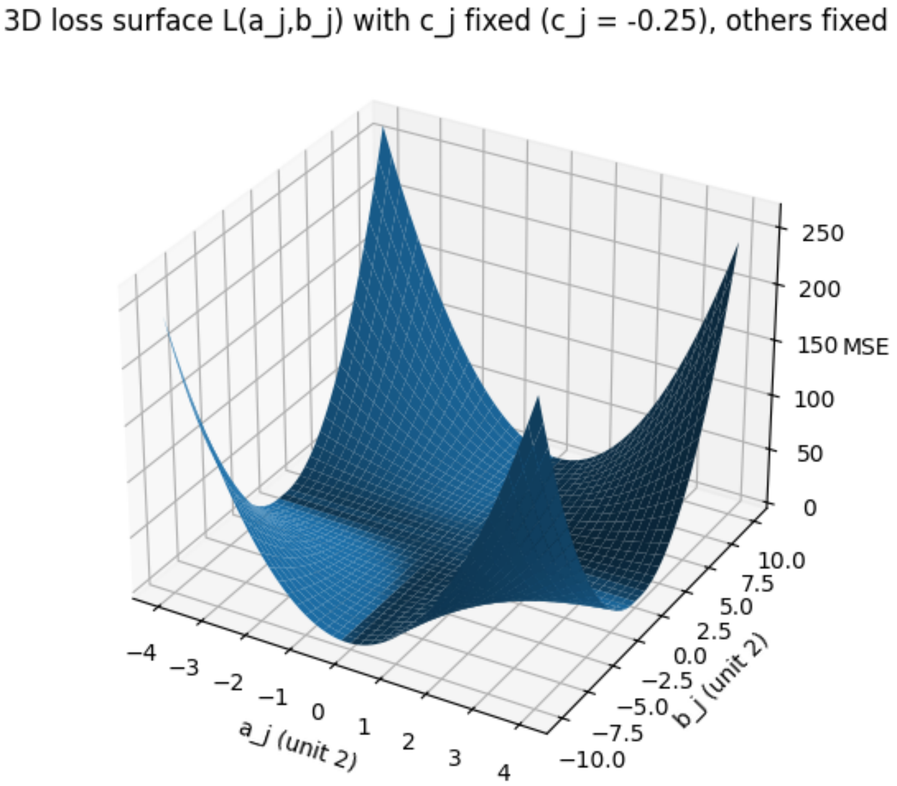
Optimization Works

SGD or Adam do!



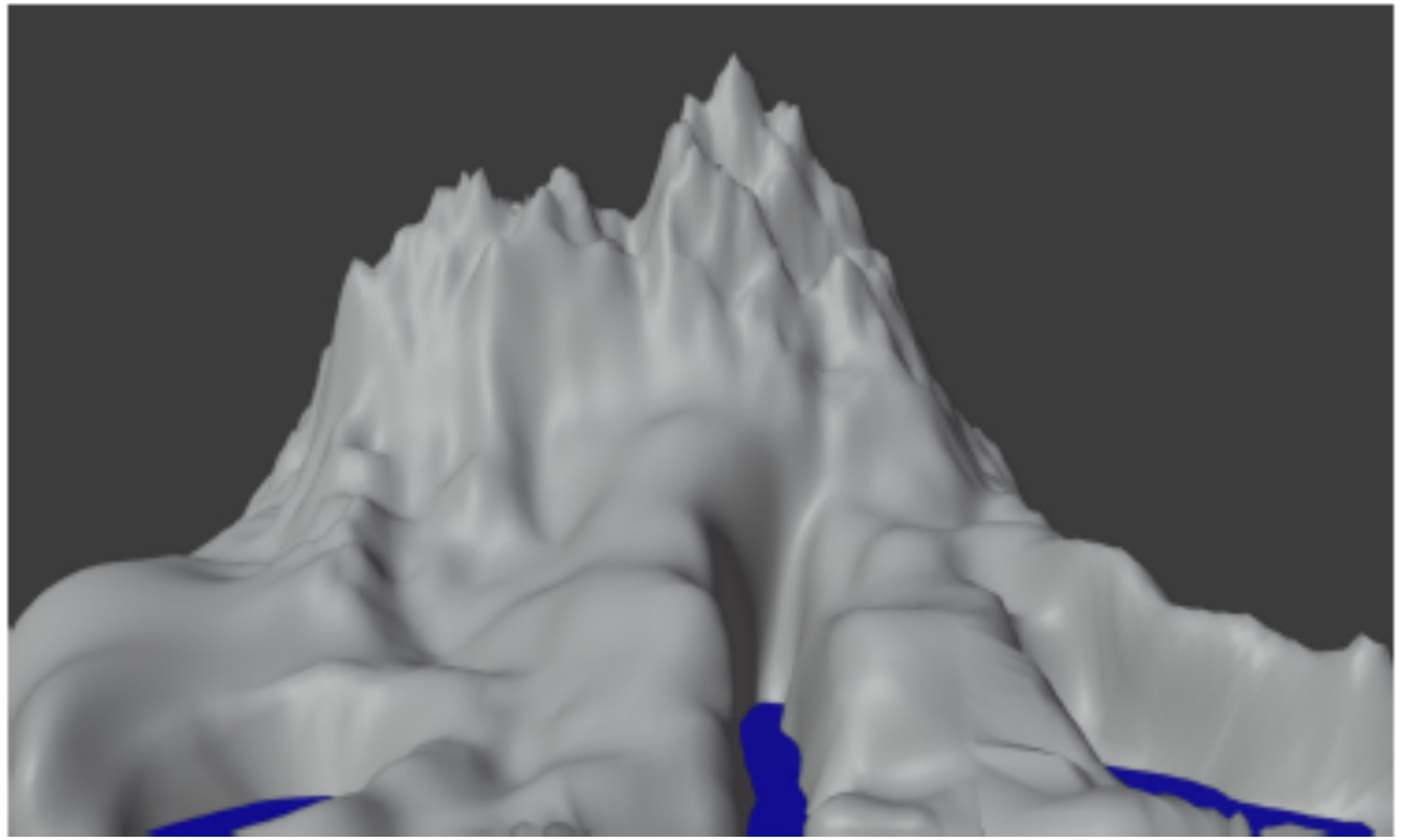
Optimization Works

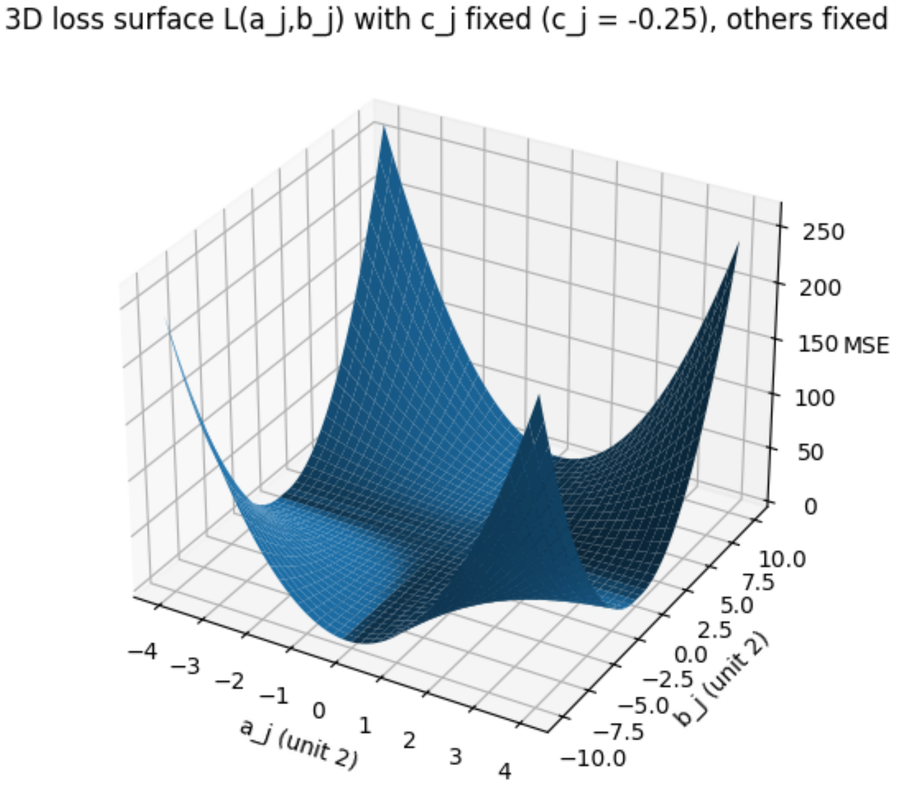




It should not be, but it is still easy!

Optimization Works



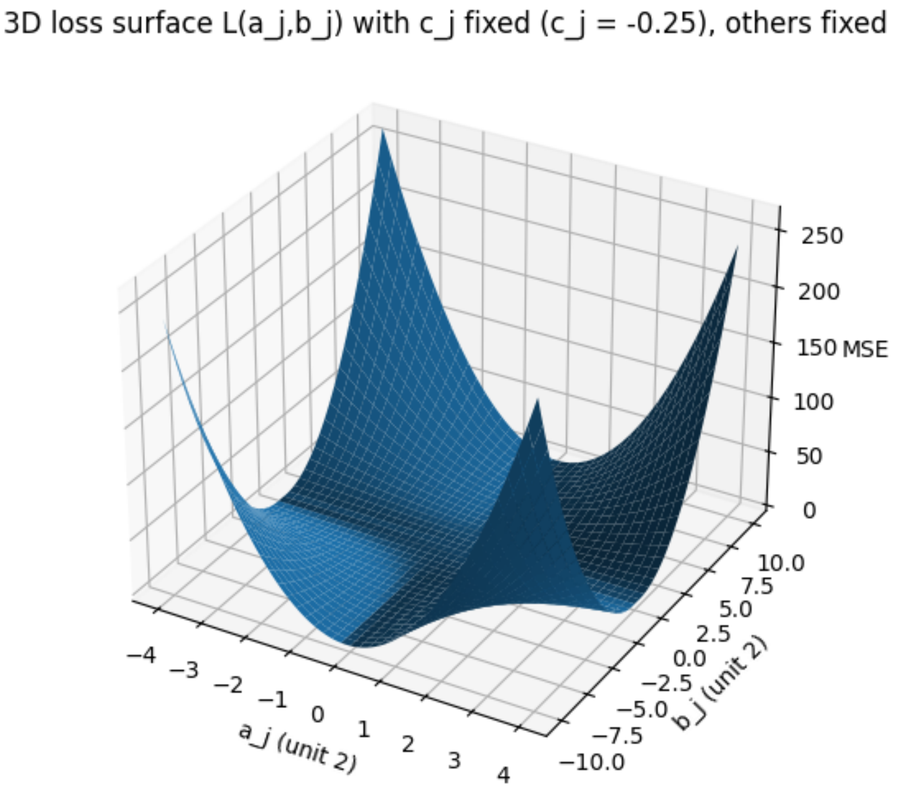


It is non convex but it’s easy to go down!

Optimization Works (?)

It is non convex but if you go down, you are good!

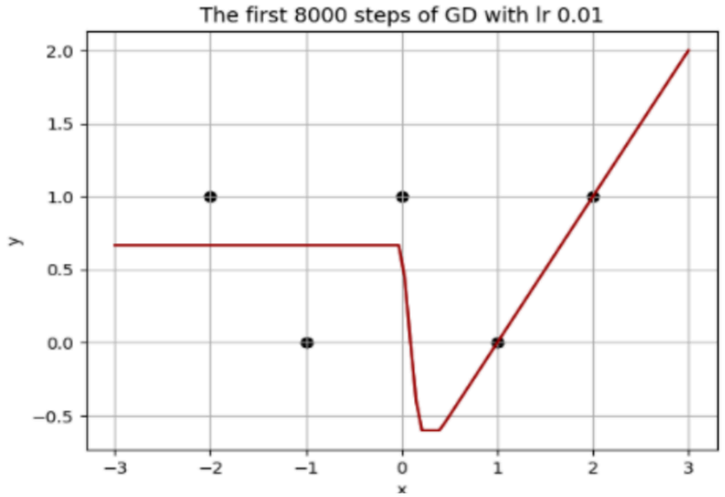
- Is this actually the case?
- Why?
- Can we say something theoretical about this?
- Can we propose a better algorithm?

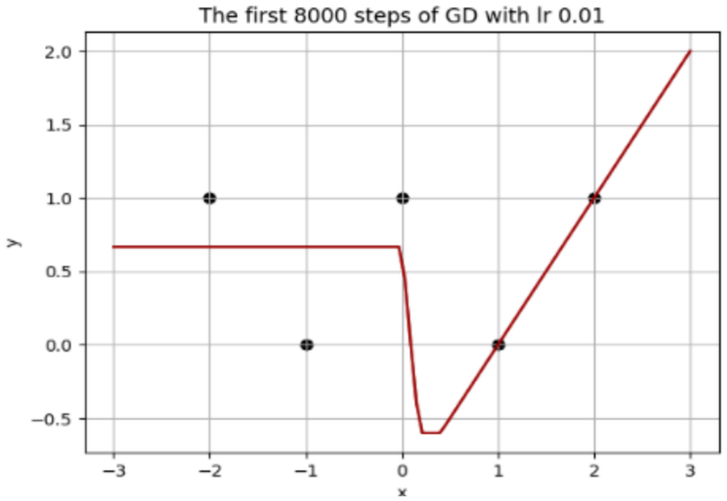


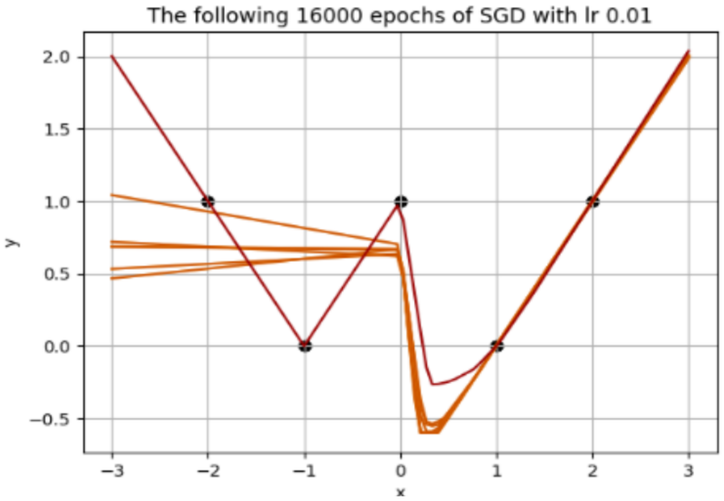
REGIME 2

- ***Not enough exact*** data!
- Enough compute

And big big model







Optimization Works (?)

L-smoothness (Descent Lemma): $L(\vartheta) \le L(\theta) + \langle \nabla L(\theta), \vartheta - \theta \rangle + \frac{L}{2}\|\vartheta - \theta\|^2.$

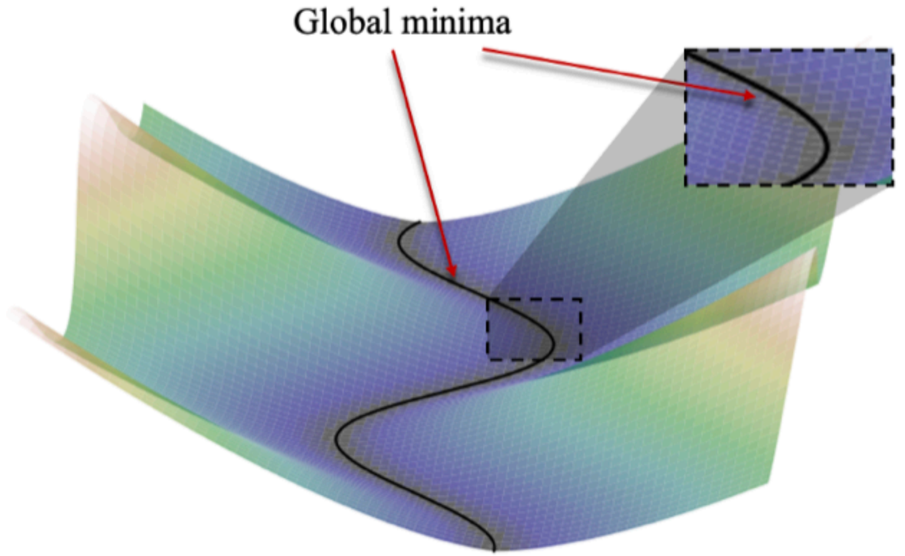
Gradient step: $x^+ = x - \eta \nabla L(\theta), \quad f(x^+) \le L(\theta) - \left(\eta - \frac{L\eta^2}{2}\right) \|\nabla L(\theta)\|^2.$

PL condition: $2\mu\,(L(\theta) - L^*) \le \|\nabla L(\theta)\|^2 \iff L(\theta) - L^* \le \frac{1}{2\mu}\,\|\nabla L(\theta)\|^2.$

Linear rate (choose $\eta = \frac{1}{L}$): $L(\theta_k) - L^* \le \left(1 - \frac{\mu}{L}\right)^k \left(L(\theta_0) - L^*\right).$

No spurious stationary points under PL: $\nabla L(\theta) = 0 \;\Rightarrow\; L(\theta) = L^*.$

Implication: μ-strong convex $\;\Rightarrow\;$ PL(μ).



(b) Loss landscape of over-parameterized models

YES IN REGIME 2

A number of works tried to make sense of this:

- **All the sublevel sets are connected ($width > 8d$)** *(Nguyen, 2019)*
- **Overparameterized networks are essentially PL** *(Allen Zhu et al., 2019 ++)*
- **You may escape saddle points** *(Lee et al., 2016)*

But the problem is open in general

Question

Classically we need some assumptions for

(e.g., Boyd and Vanderberge (2004)):

- If you go down you converge well $\Longrightarrow$ We got them for very overparameterized models only. What in general?

Conclusion

Why and when can we train neural networks?

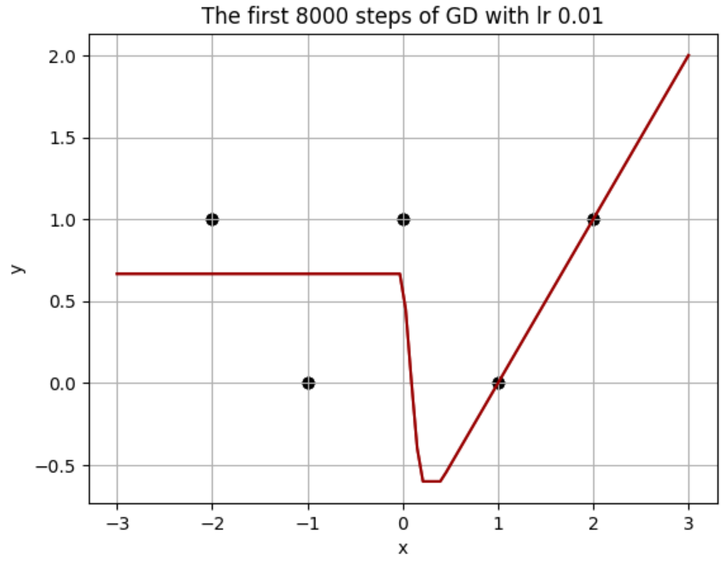
Paradigm Shift

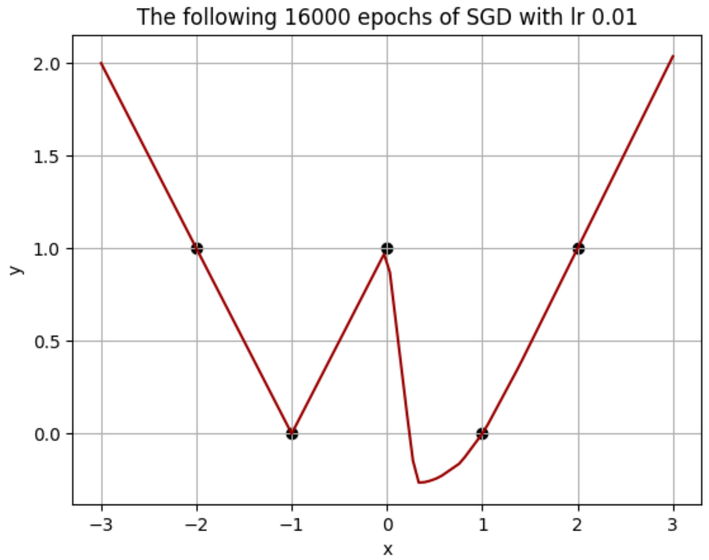
Goal of training neural networks:

Find the one which
Performs best on your task

What we can work with:

- **The data we got**
- **The measure of error $L(\theta)$**
- **(Within) a compute budget**

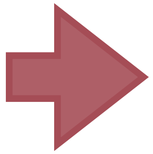




Message 1

We have:

- A dataset.
- Some compute.

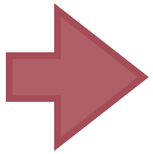


Training means using those to get a model

Message 1b

We have:

- A dataset.
- Some compute.



Enough samples?

Learnability

Enough Compute?

Optimization

Message 2

We have 2 regimes *(bottlenecks)*:

- Not enough data.
- Not enough compute.

Message 3

Classically we need some assumptions for

(e.g., Boyd and Vanderberge (2004)):

- You go down $\Longrightarrow$ stable steps
- If you go down you converge well $\Longrightarrow$ convexity/PL

Message 4

- If you go down you converge well $\Longrightarrow$ We got them for very overparameterized models only. What in general?

- In general, though, optimizers converge down, and it may be that it is about:
  - They do not really find local minima,
    only escapable flat manifolds (saddles are present).
  - They travel flat manifolds (fast).