上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Lecture 10: Optimization

Tao Huang

John Hopcroft Center, School of Computer Science, Shanghai Jiao Tong University

https://taohuang.info/cs3317

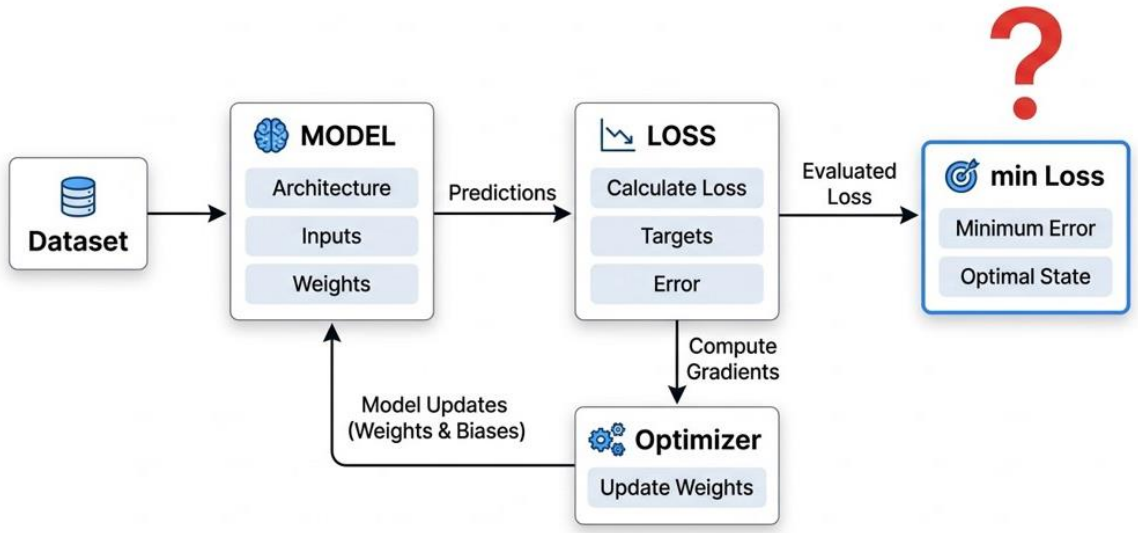
https://oc.sjtu.edu.cn/courses/89538

Part of lecture credits: CS221 - Stanford University

Where We Are

Recap

- Model: $f_w(x)$
- Loss: $\mathcal{L}(x, y, w)$
- Goal: $\min_w \mathcal{L}(x, y, w)$



Question:

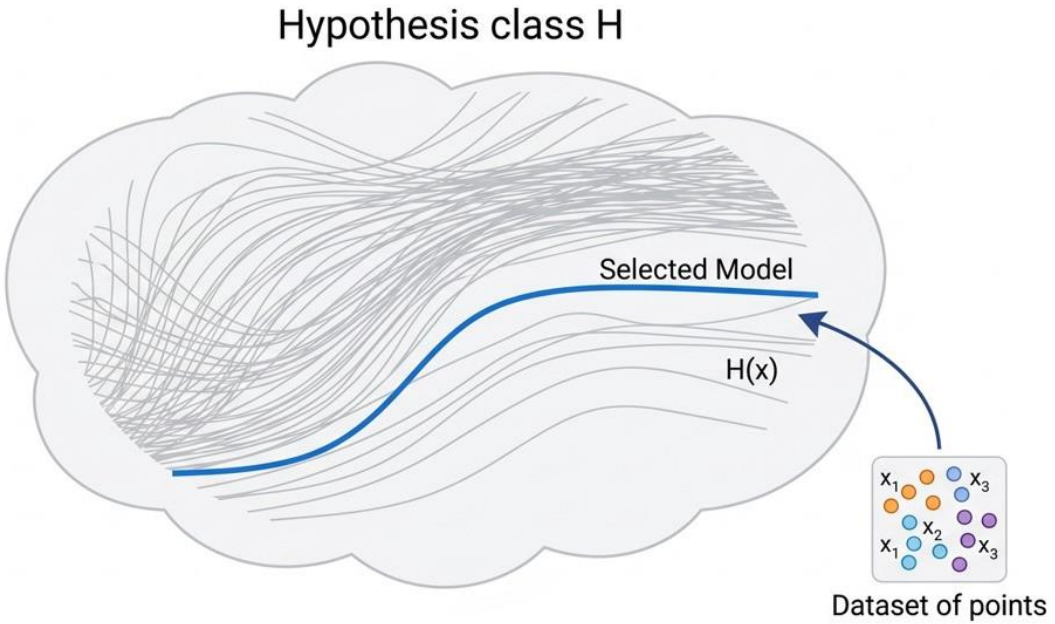
How do we actually find w?

First Thought

- **Idea:**
  Try all possible w from hypothesis class

- **Problem**
  w is continuous
  => Infinite possibilities

Impossible.



Second Thought

Linear Regression: Find the closed-form solution of w that minimizes $\mathcal{L}(w)$ using least squared.

$$\begin{aligned} L(w) &= \|Xw - y\|_2^2 \\ &= (Xw - y)^T (Xw - y) \\ &= w^T X^T X w - 2y^T X w + y^T y \end{aligned}$$

Compute the gradient w.r.t. w:

$$\nabla_w L(w) = 2X^T X w - 2X^T y$$

At optimum:

$$\nabla_w L(w) = 0$$

So:

$$2X^T X w - 2X^T y = 0$$

$$X^T X w = X^T y$$

If $X^T X$ is invertible:

$$\boxed{w^* = (X^T X)^{-1} X^T y}$$

Complexity: $O(nd^2 + d^3)$ n: number of samples d: number of features

Too expensive!

[1] Detailed derivations of solving LR with least squared: https://see.stanford.edu/materials/aimlcs229/cs229-notes1.pdf

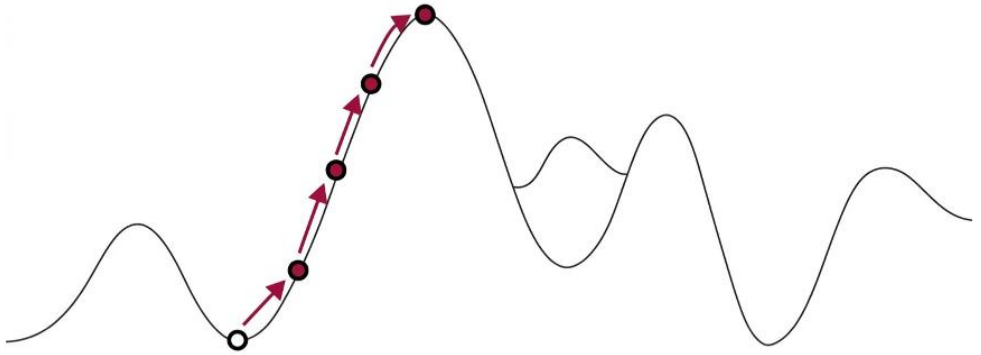
So What Do We Need?

Requirements

- A general method
- Works for any differentiable loss
- Scalable to large data

Key Question

How do we move toward better solutions?



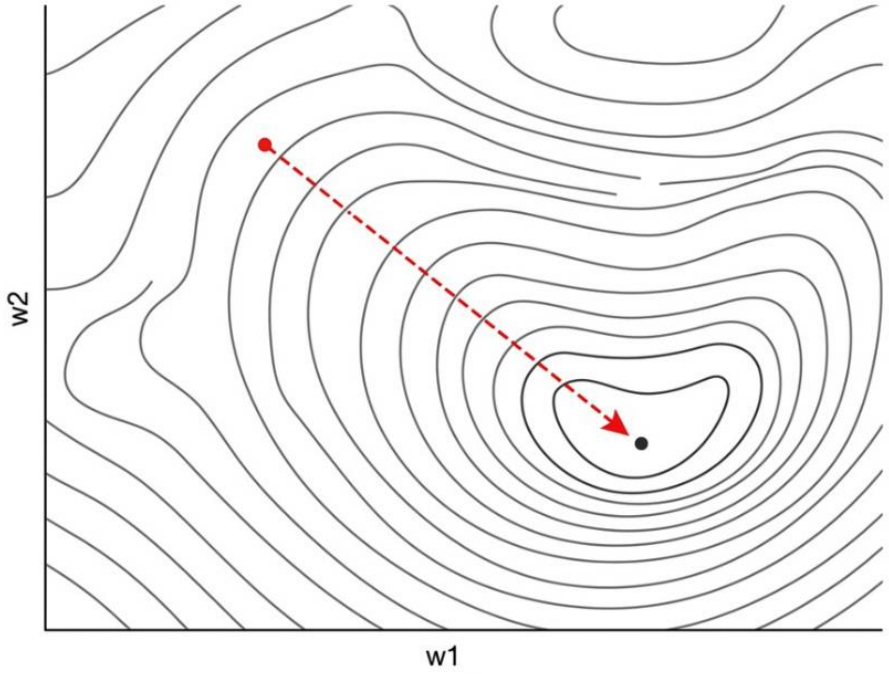
Recap: local search

Optimization as Search in Parameter Space

- Parameters (w) define a point in parameter space.
- Each point has a loss value:

$$w \rightarrow \hat{\mathcal{R}}(w)$$

- **Training means:** search for parameters with low loss
- This is a new kind of search:
  - Not over states in a graph
  - But over a **continuous space**



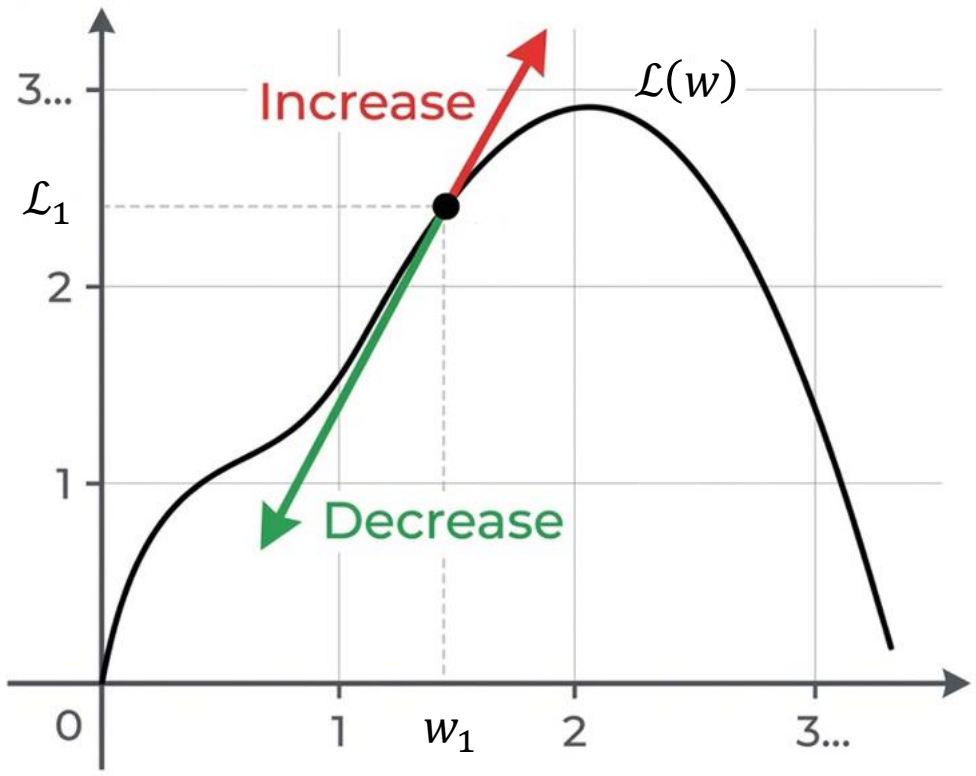
Key Idea

If we know:

- Which direction increases loss
- Which direction decreases loss

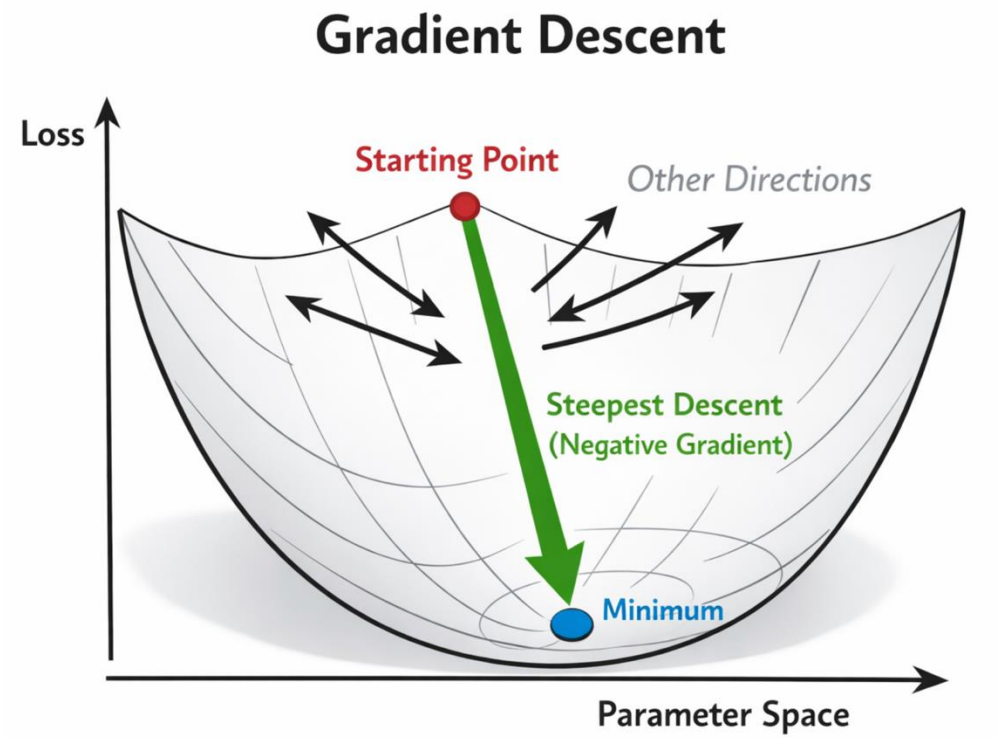
Then:

- We can move in the right direction



The Gradient

- **Definition**
  Gradient: $\nabla \mathcal{L}(w)$
- **Interpretation**
  Direction of steepest increase
- **Therefore**
  Negative gradient = Steepest decrease



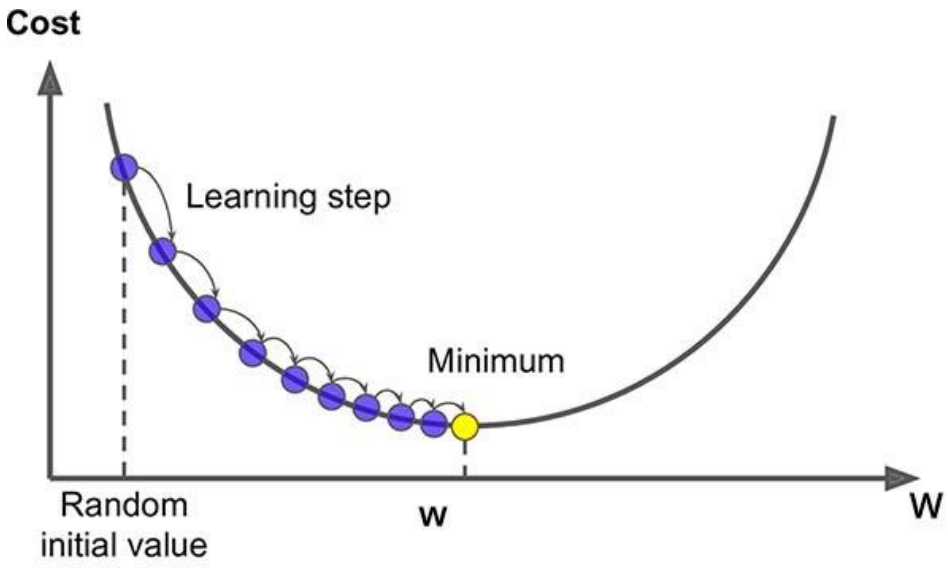
Gradient Descent

- **Update Rule**

$$w \leftarrow w - \eta \nabla \mathcal{L}(w)$$

- **η = learning rate (step size)**

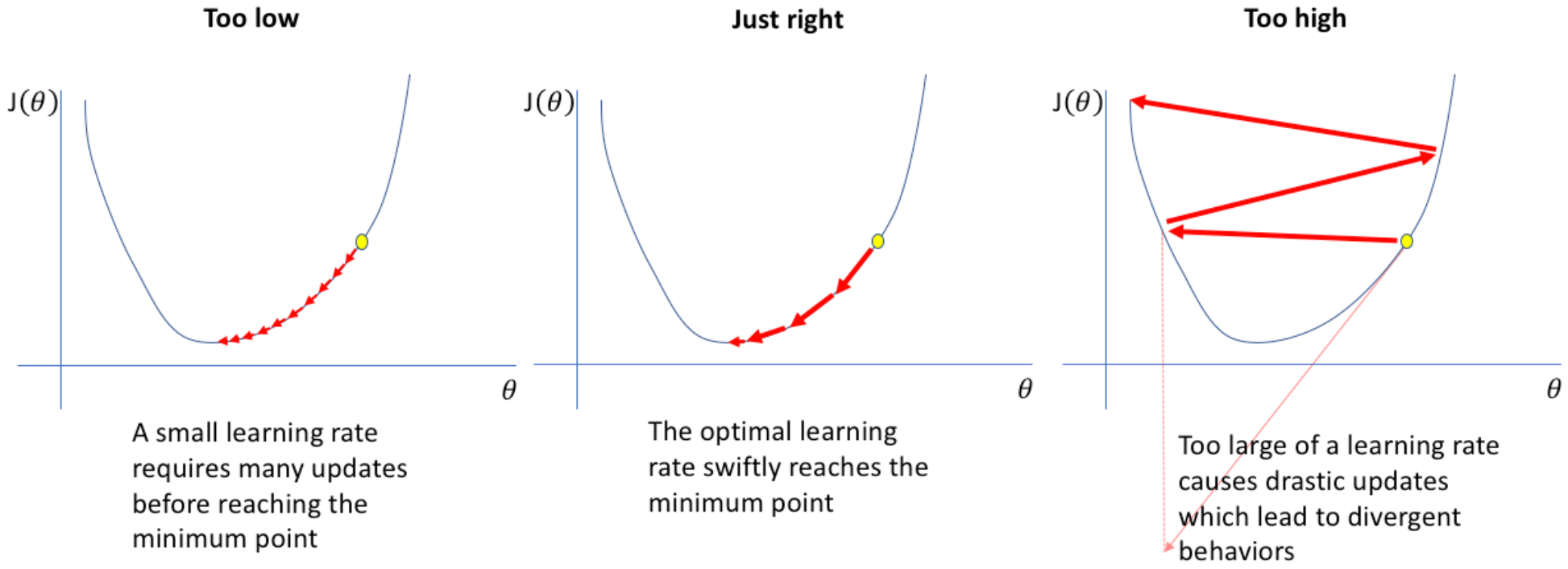
- **Interpretation**
  - Step in downhill direction
  - Repeat until convergence



Key Hyperparameter

Learning rate η

- Too small: Slow convergence
- Too large: Overshoot, divergence



Full Algorithm

- Initialize w randomly
- Repeat
  - Compute gradient
  - Update w
- Until
  - Convergence
  - Reach maximum steps

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

Computing the Gradient of LR

- Objective function:

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} (\mathbf{w} \cdot \phi(x) - y)^2$$

- Gradient (use chain rule):

$$\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} 2(\underbrace{\mathbf{w} \cdot \phi(x) - y}_{\text{prediction} - \text{target}})\phi(x)$$

Gradient Descent Example

training data $\mathcal{D}_{\text{train}}$

x	y
1	1
2	3
4	3

$$\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} 2(\mathbf{w} \cdot \phi(x) - y)\phi(x)$$

Gradient update: $\mathbf{w} \leftarrow \mathbf{w} - 0.1 \nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

t	$\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$	$\mathbf{w}$
		$[0, 0]$
1	$\underbrace{\frac{1}{3}(2([0,0] \cdot [1,1] - 1)[1,1] + 2([0,0] \cdot [1,2] - 3)[1,2] + 2([0,0] \cdot [1,4] - 3)[1,4])}_{=[-4.67, -12.67]}$	$[0.47, 1.27]$
2	$\underbrace{\frac{1}{3}(2([0.47,1.27] \cdot [1,1] - 1)[1,1] + 2([0.47,1.27] \cdot [1,2] - 3)[1,2] + 2([0.47,1.27] \cdot [1,4] - 3)[1,4])}_{=[2.18, 7.24]}$	$[0.25, 0.54]$
...	...	...
200	$\underbrace{\frac{1}{3}(2([1,0.57] \cdot [1,1] - 1)[1,1] + 2([1,0.57] \cdot [1,2] - 3)[1,2] + 2([1,0.57] \cdot [1,4] - 3)[1,4])}_{=[0,0]}$	$[1, 0.57]$

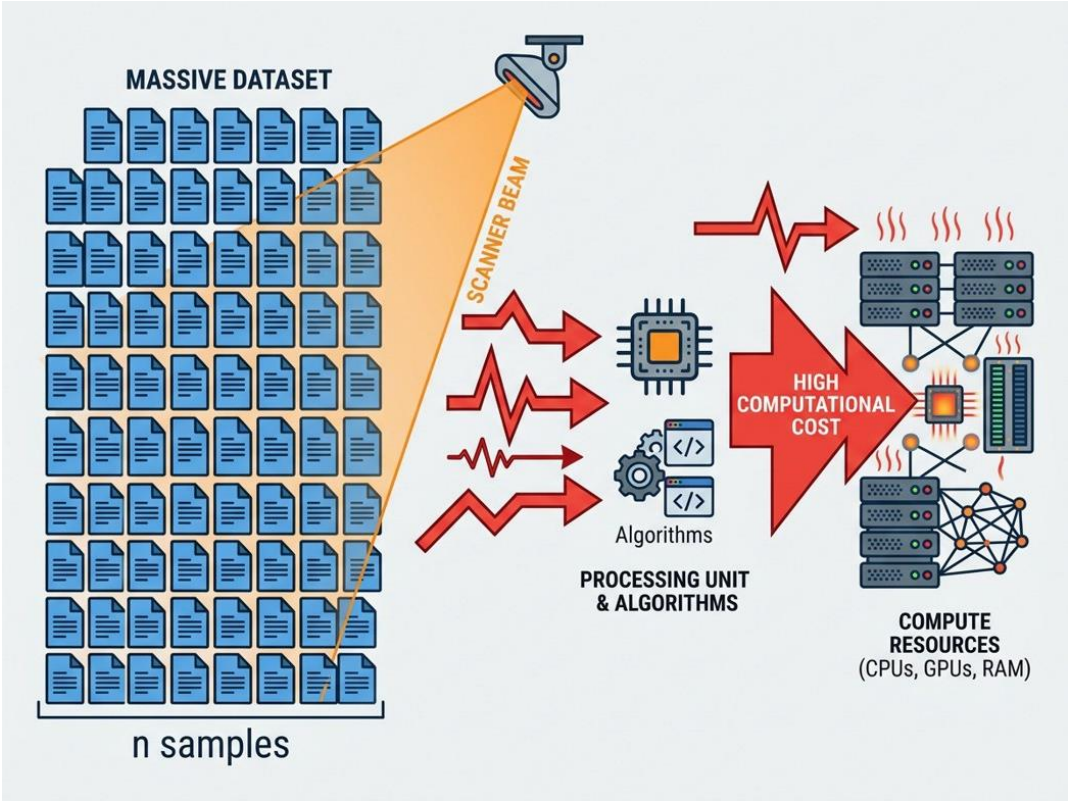
Problem with Gradient Descent

Costly Step

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

Problem

- Must scan entire dataset
- Very expensive for large $N = |\mathcal{D}_{train}|$

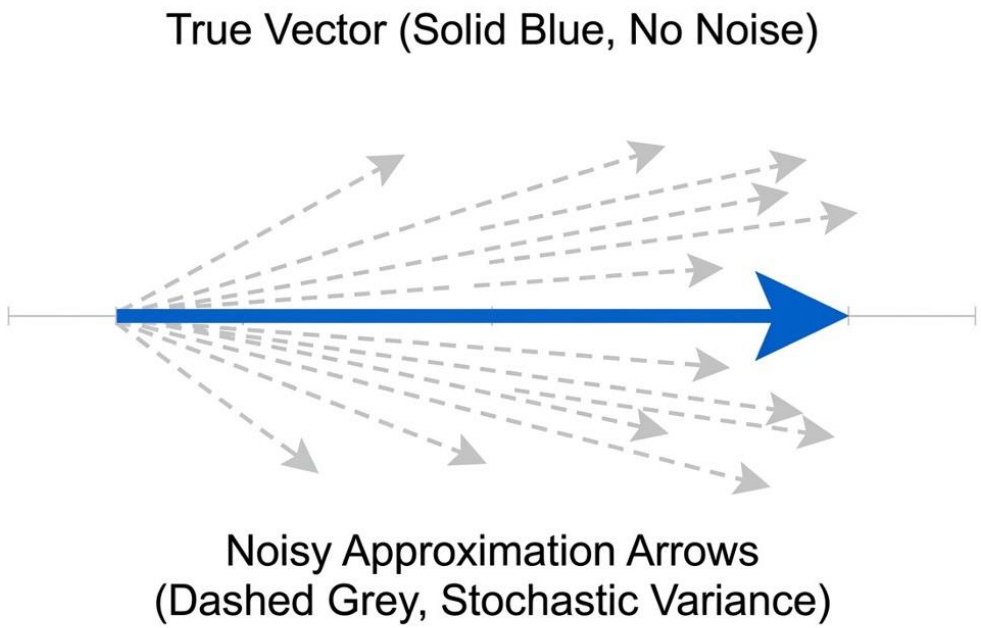


Key Observation

We don’t need the exact gradient

We only need:

- A good estimate of direction
- Approximations are sufficient

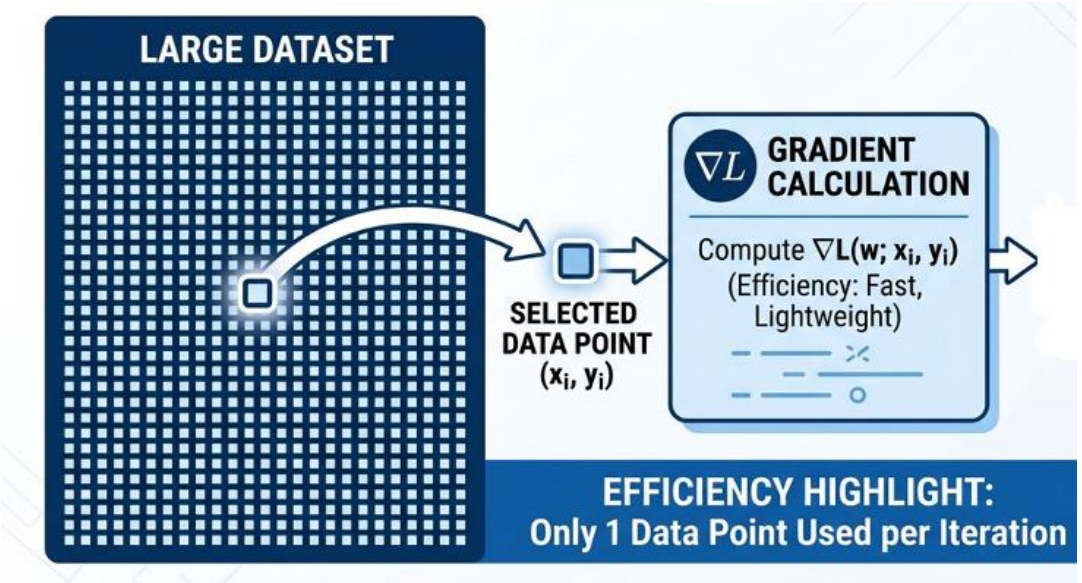


Stochastic Gradient Descent (SGD)

- Idea
  Use one sample instead of full dataset

- Update
  $$w \leftarrow w - \eta \nabla \mathcal{L}(x_i, y_i, w)$$

- Cost
  O(1) per step (independent of N)



Algorithm: stochastic gradient descent

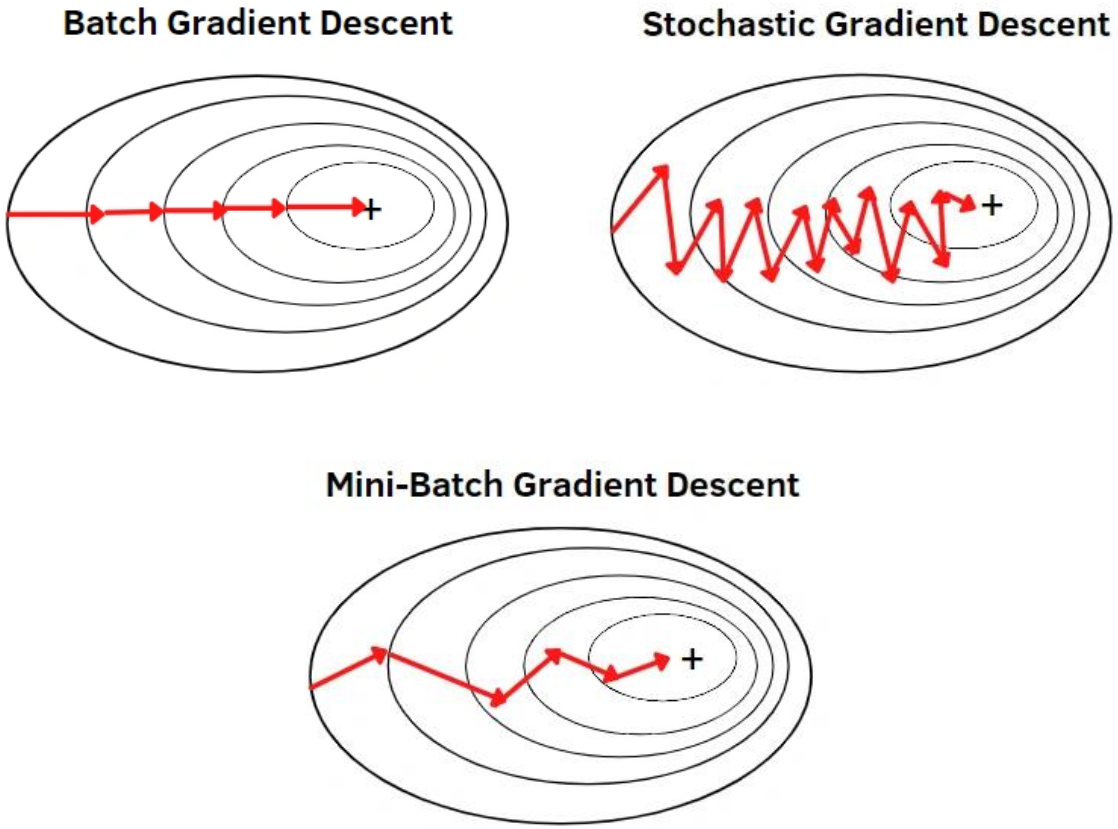
Initialize $\mathbf{w} = [0, \ldots, 0]$
For $t = 1, \ldots, T$:
  For $(x, y) \in \mathcal{D}_{\text{train}}$:
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$

Why SGD Works

- **Each Update**
  Noisy estimate of gradient

- **But**
  On average → correct direction

- **Result**
  Zig-zag path trending downward



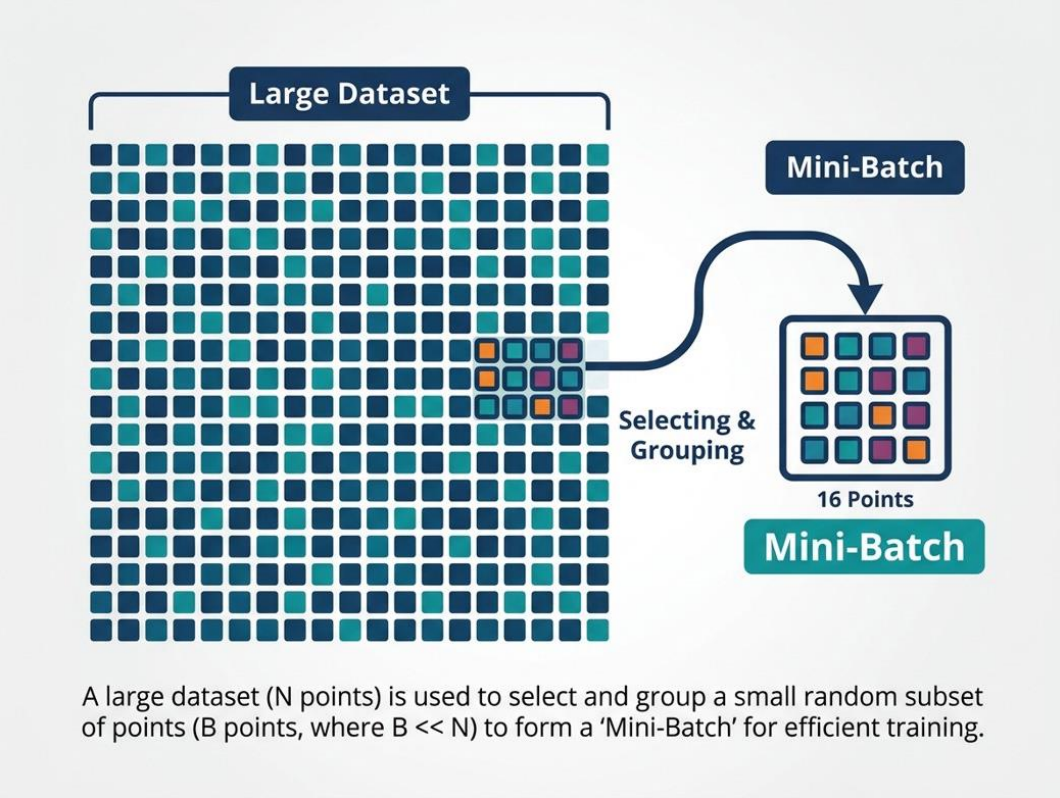
Mini-Batch SGD

Strategy

Use a small batch of b samples

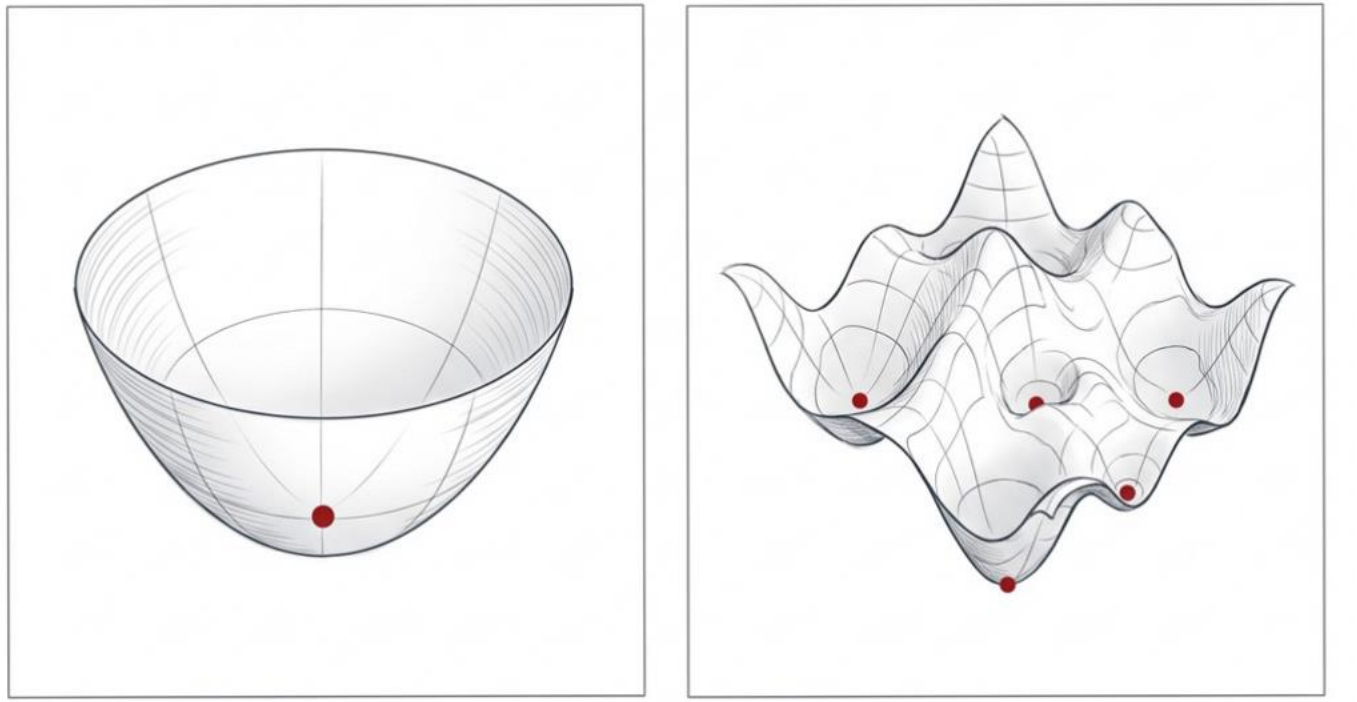
Benefit

- Reduces noise
- Leverages vectorization (GPU)
- Balances stability and speed



Convex vs Non-Convex

- **Convex:** One global minimum (Bowl). Easy.
- **Non-Convex:** Many local minima (Rugged). Hard.



Convex Non-Convex

What is a Convex Function?

Definition:

- A function $f(x)$ is convex if:

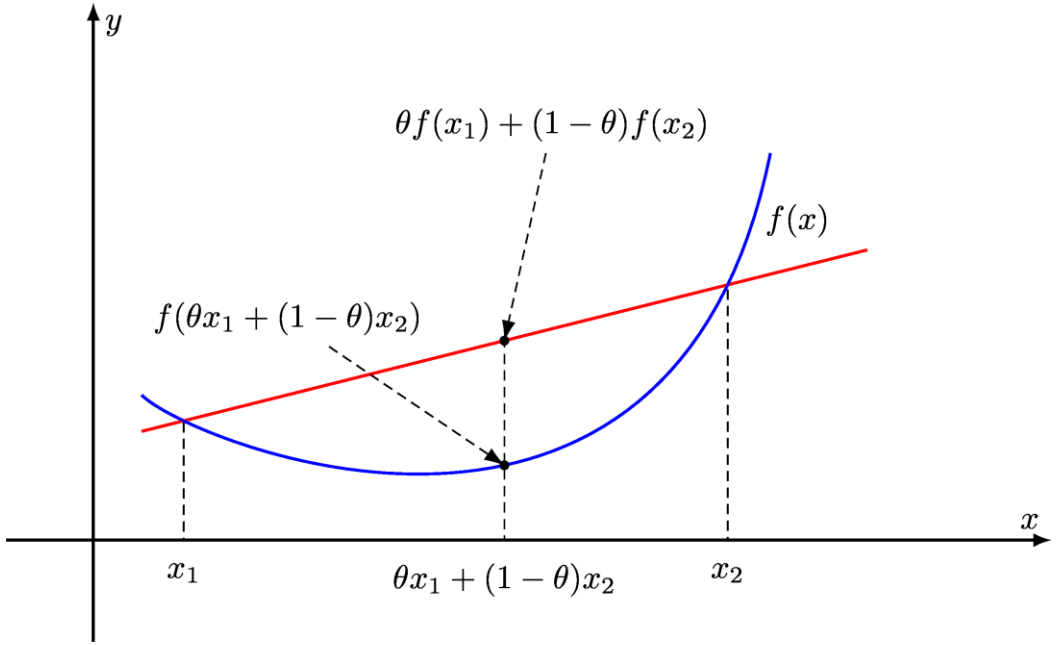
$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

for all x_1, x_2 and $t \in [0,1]$

Intepretation:

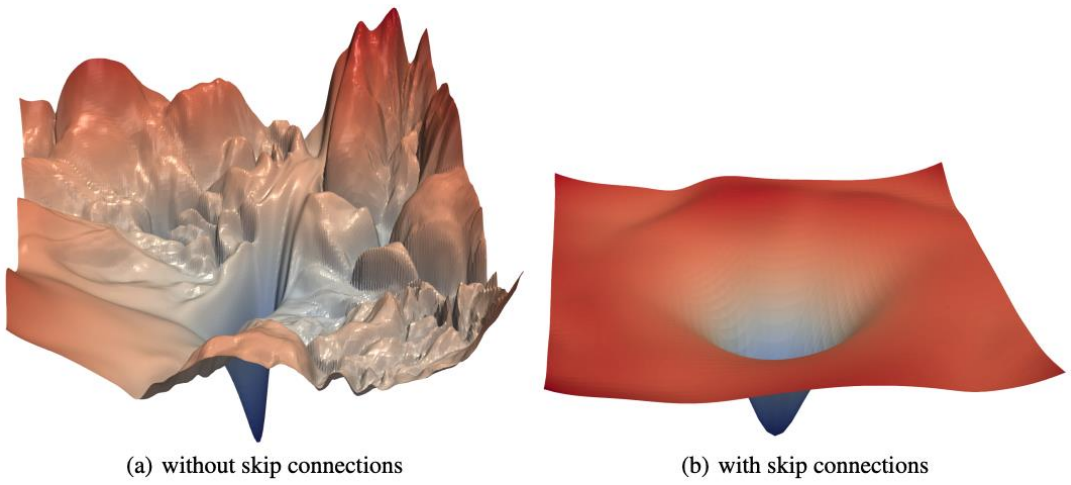
The line segment between any two points on the curve lies above the function

- Convex models: linear/logistic regression
- Non-convex models: neural networks



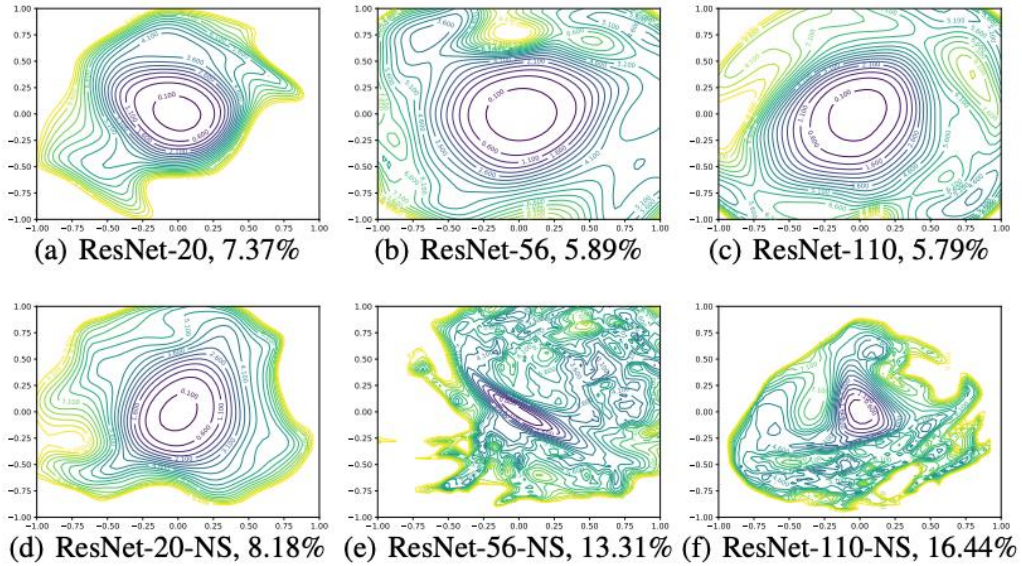
Surprising Fact of Non-Convexity

- **Surprise:** SGD works well for neural networks
- **Reason:** Many local minima are "good enough"



(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.



(a) ResNet-20, 7.37% (b) ResNet-56, 5.89% (c) ResNet-110, 5.79%

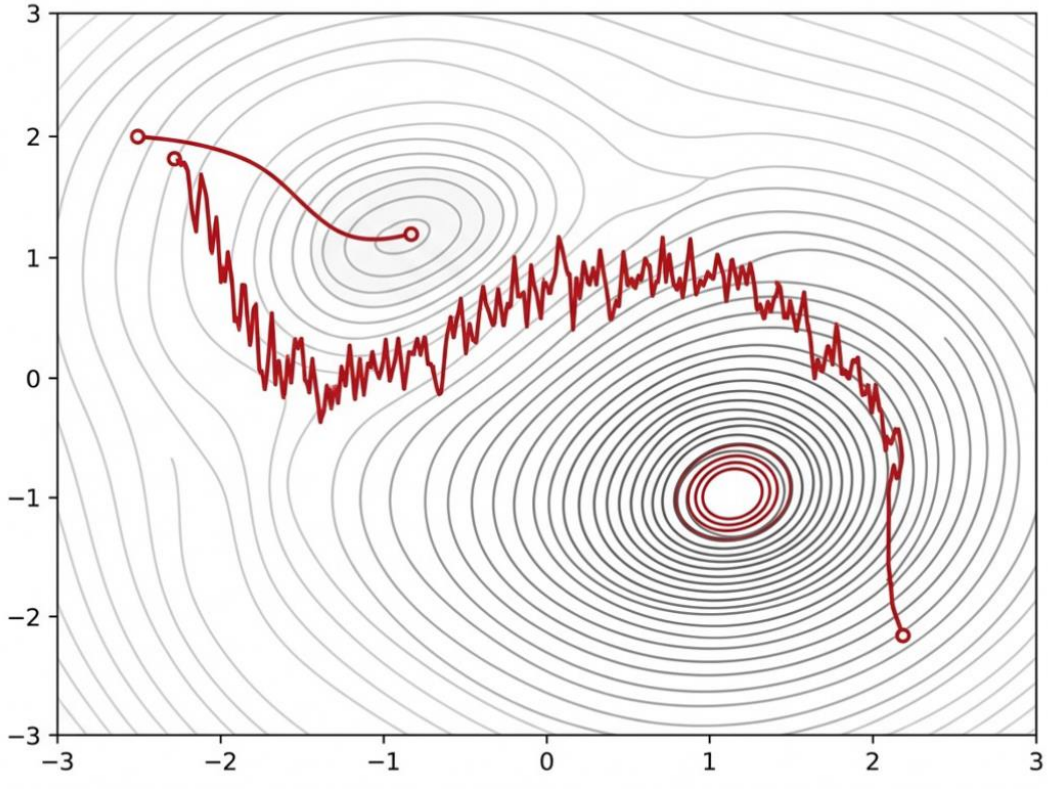
(d) ResNet-20-NS, 8.18% (e) ResNet-56-NS, 13.31% (f) ResNet-110-NS, 16.44%

Figure 5: 2D visualization of the loss surface of ResNet and ResNet-noshort with different depth.

[2] Visualizing the Loss Landscape of Neural Nets. Li et al, NIPS 2018. https://arxiv.org/pdf/1712.09913

Noise Can Help

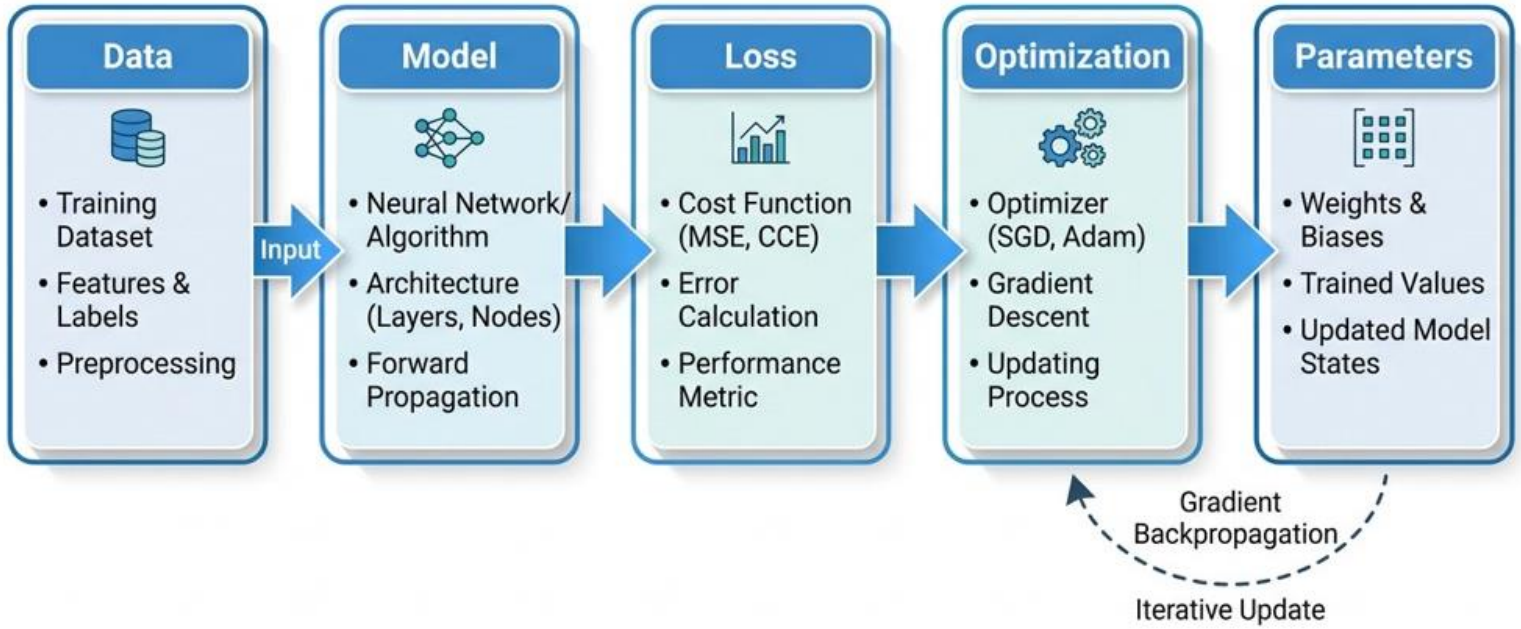
- Gradient over the full dataset is overly-smooth.
- SGD introduces noise that helps escape shallow regions



Full ML Pipeline

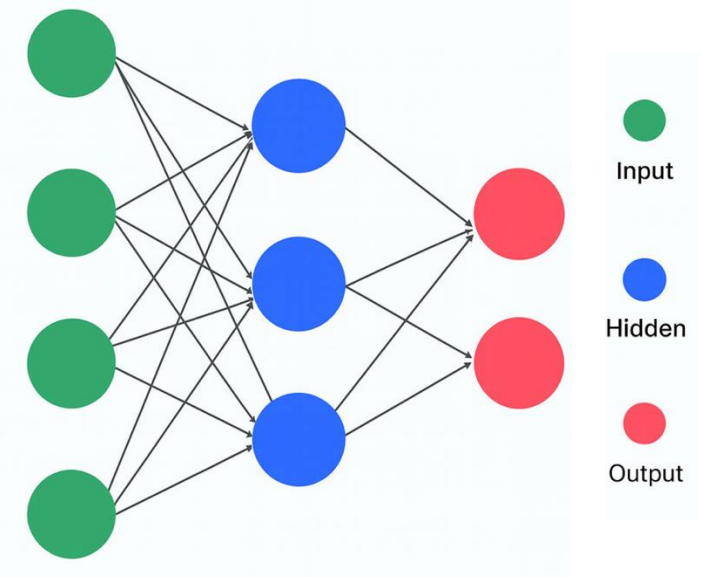
Workflow:

- Data → Model → Loss → Optimization → Parameters
- We now have all components to build AI.



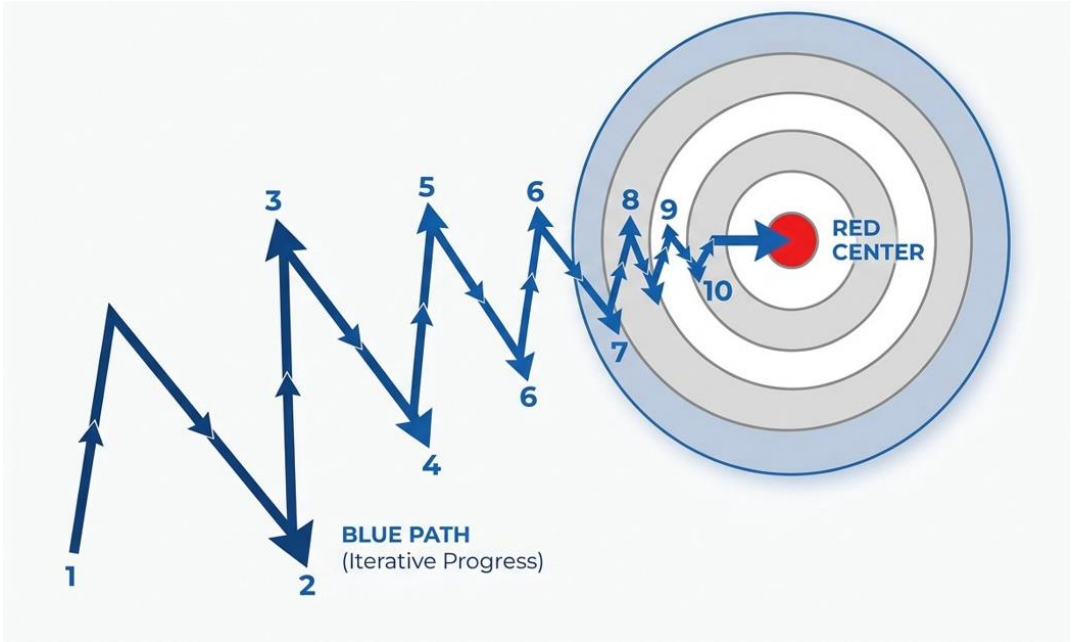
From Linear Optimization to Deep Learning

- **Scaling Up:**
  To complex non-linear models
- **Neural Networks:**
  Architecture
- **Backpropagation:**
  Computing gradients efficiently



Takeaway

- We cannot solve most ML problems analytically
- We solve them by:
  iterative optimization (gradient descent)
- SGD
  is the engine of modern AI



Next Week

Classical ML Models

- Support Vector Machines
- Decision Trees & Ensemble Methods
- Model Evaluation