

# NYU CS-GY 6923

# Machine Learning

Prof. Pavel Izmailov

Guest lecturer: Timur Garipov

# Introduction

## Timur Garipov

- Collaborating with Pavel since 2016
- PhD, MIT CSAIL, 2024
- Now AI research in industry

## Research Interests

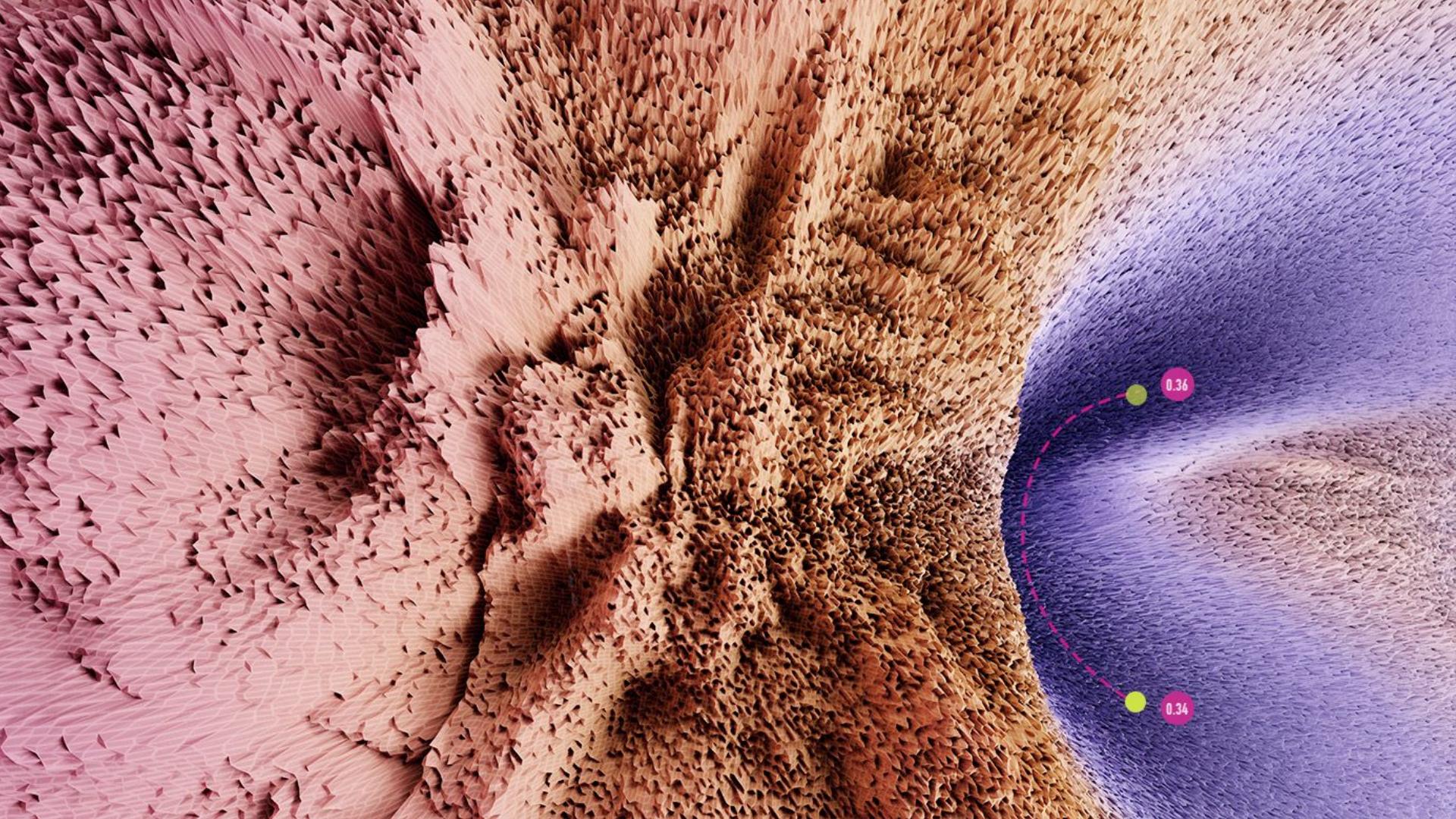
- Deep Learning
- Deep Probabilistic Models / Generative Models
- LLM reasoning



# Today

## Optimization in Machine Learning

- (Regularized) Empirical Risk Minimization
- Gradient Descent & Stochastic Gradient Descent
- Theoretical and Practical Aspects
- Non-convex loss landscapes: Deep Neural Networks
- Beyond optimization:
  - Game-theoretic formulations
  - Dynamical systems



0.36

0.34

# Machine Learning

**Given:** training data

**Want:** predictive model  $f(x)$

**Such that:**

- Model “agrees” with observed data
- Model provides useful predictions for previously unobserved inputs

**Learning Algorithm:**

Training Data => [Algorithm] => Model

# Learning Algorithm

## **Learning Algorithm:**

Training Data => [Algorithm] => Model

How do we design and analyse a learning algorithm?

# Empirical Risk Minimization

- Model “agrees” with observed data
- Model provides useful predictions for previously unobserved inputs

Empirical Risk Minimization:

- Pick a model family (e.g. linear models / neural networks / ...)
- Find a model that minimizes error on training data

$$\min_{\text{model}} \sum_{i=1}^N \ell(x_i, y_i, \text{model}) + \lambda \cdot R(\text{model})$$

$\text{model} \in \text{Model Family}$

# Empirical Risk Minimization

Why ERM?

$$\min_{\text{model}} \sum_{i=1}^N \ell(x_i, y_i, \text{model}) + \lambda \cdot R(\text{model})$$

$\text{model} \in \text{Model Family}$

Assume  $(x_i, y_i) \stackrel{iid}{\sim} P(x, y)$

Empirical error :  $\hat{\mathcal{L}}(\text{model}) = \sum_{i=1}^N \ell(x_i, y_i, \text{model})$

Population error :  $\mathcal{L}(\text{model}) = \mathbb{E}_{P(x,y)} [\ell(x, y, \text{model})]$

**In-distribution generalization: test data comes from the same distribution**

# Recap: Linear Regression; Logistic Regression

Linear regression:

$$\min_{w, w_0} \sum_{i=1}^N (\langle w, x_i \rangle + w_0 - y_i)^2 + \lambda \cdot \|w\|^2$$

Logistic regression:

$$\min_{w, w_0} \sum_{i=1}^N \log(1 + \exp(-y_i(\langle w, x_i \rangle + w_0))) + \lambda \cdot \|w\|^2$$

# Recap: Linear Regression; Logistic Regression

Linear regression:

$$\min_{w, w_0} \sum_{i=1}^N (\langle w, x_i \rangle + w_0 - y_i)^2 + \lambda \cdot \|w\|^2$$

Maximum Likelihood Estimate (MLE):

$$\min_{w, w_0} \sum_{i=1}^N -\log p(y_i | x_i, w, w_0)$$

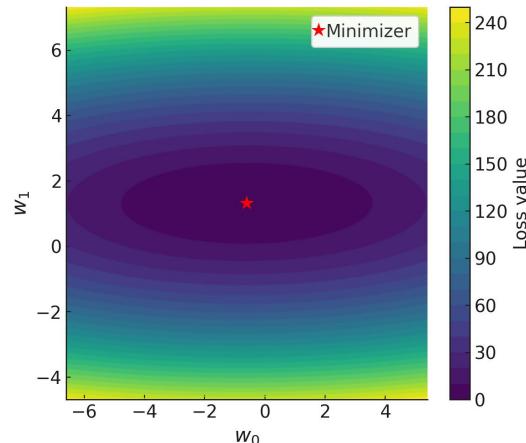
Maximum a Posteriori Estimate(MAE):

$$\min_{w, w_0} \sum_{i=1}^N -\log p(y_i | x_i, w, w_0) - \log p(w, w_0)$$

# Recap: Linear Regression; Logistic Regression

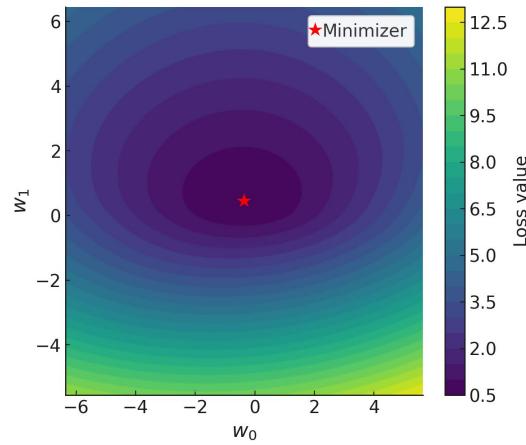
Linear regression:

- Loss function is a quadratic function of  $w$
- Has analytical expression for the optimal solution
- Requires costly inversion of  $D \times D$  matrix



Logistic regression:

- Loss function involves  $\log(1 + \exp(f(x, y, w)))$
- No analytical expression for the optimal solution
- Requires an optimization algorithm



# Today

**ML via optimization.** Find “good” model by

- Minimizing Empirical Risk (Maximizing Likelihood)

Optimization questions:

- Model parameterization + error function → **loss landscape** properties?
- Does the **optimization algorithm** converge to a minimizer?
- What is algorithm’s **convergence rate**?

Machine learning questions:

- **Generalization:** Does the **optimal solution** have useful statistical properties?
- How much **regularization** do we need?
- **Implicit regularization:** Do we want to solve optimization “precisely”?

# Optimization

Logistic regression:

$$\min_w \sum_{i=1}^N \log (1 + \exp (-y_i \langle w, x_i \rangle)) + \lambda \cdot \|w\|^2$$

$$\min_{w \in \mathbb{R}^d} f(w) \text{ — unconstrained optimization problem in } \mathbb{R}^d$$

“Hardness” of the problem depends on:

- Mathematical properties of the function (smoothness, convexity)
- Number of dimensions
- Computational cost of evaluating  $f$ , its gradients, Hessians, ...

For general non-convex functions the problem is NP-hard

# Gradient Descent

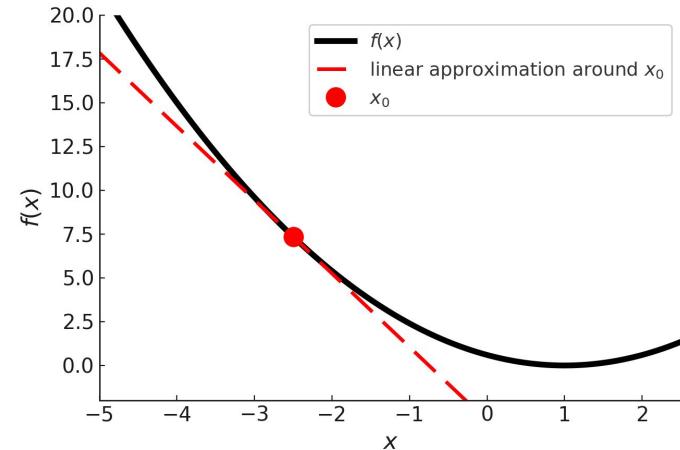
$\min_{w \in \mathbb{R}^d} f(w)$  — unconstrained optimization problem in  $\mathbb{R}^d$

If  $f$  is smooth, then

$$f(x_0 + h) = f(x_0) + \langle \nabla_x f(x_0), h \rangle + O(\|h\|^2)$$

The direction of the fastest descent is

$$h = -\nabla_x f(x_0)$$



Let's take a step in that direction!

# Gradient Descent

$$\min_{w \in \mathbb{R}^d} f(w) \quad \text{— unconstrained optimization problem in } \mathbb{R}^d$$

## Gradient descent

Pick  $x_0$ ; set  $k = 0$

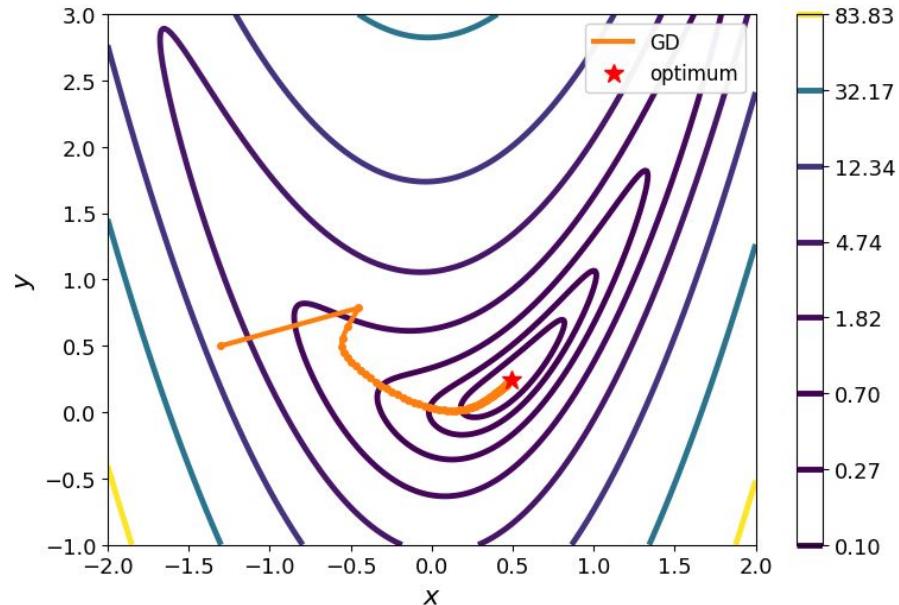
Until `should_stop()`:

$$x_{k+1} = x_k - \alpha_k \cdot \nabla_x f(x_k)$$

$$k = k + 1$$

Algorithm parameters:

- Step-size schedule
- Stopping criterion



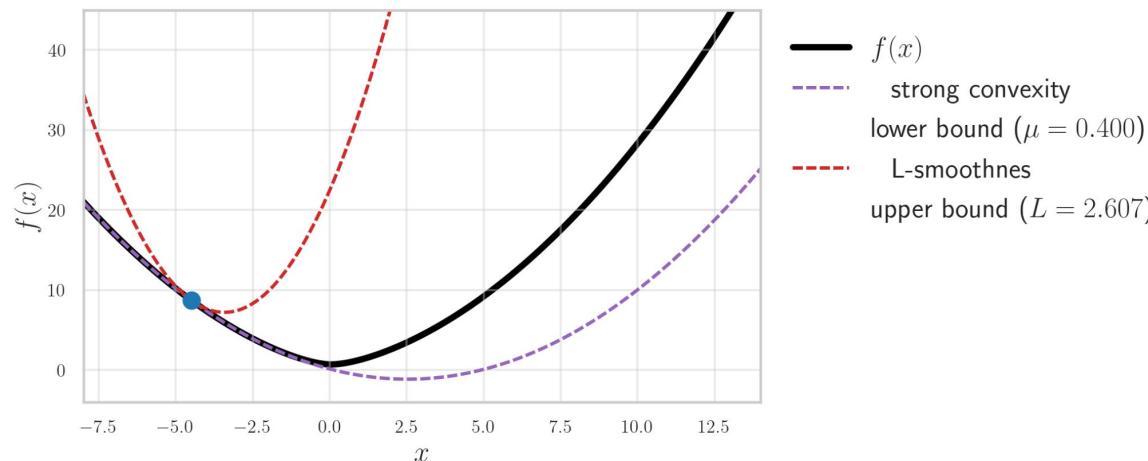
# Gradient Descent: Convergence

**Assume**  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is  $\mu$ -strongly convex and  $L$ -smooth, and let  $x^\star = \arg \min f$ .

**Strong convexity:**  $f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2} \|y - x\|^2, \quad \forall x, y.$

**L-smooth (L-Lipshitz gradient):**  $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \quad \forall x, y.$

Consequence (Descent Lemma):  $f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|^2, \quad \forall x, y.$



# Gradient Descent: Convergence

**Assume**  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is  $\mu$ -strongly convex and  $L$ -smooth, and let  $x^\star = \arg \min f$ .

**Strong convexity:**  $f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2} \|y - x\|^2, \quad \forall x, y.$

**L-smooth (L-Lipshitz gradient):**  $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \quad \forall x, y.$

Consequence (Descent Lemma):  $f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|^2, \quad \forall x, y.$

**Gradient descent with step size**  $\alpha_k = \frac{1}{L}, \quad x_{k+1} = x_k - \frac{1}{L} \nabla f(x_k).$

**Linear convergence:**  $f(x_k) - f(x^\star) \leq \left(1 - \frac{\mu}{L}\right)^k (f(x_0) - f(x^\star)).$

$k(\varepsilon)$  such that  $f(\bar{x}_k) - f(x^\star) \leq \varepsilon \iff k(\varepsilon) = O\left(\log \frac{1}{\varepsilon}\right)$

# Gradient Descent: Convergence (Proof I)

**L-smooth (L-Lipshitz gradient):**  $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \quad \forall x, y.$

Consequence (Descent Lemma):  $f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|^2, \quad \forall x, y.$

**Gradient descent with step size**  $\alpha_k = \frac{1}{L} \quad x_{k+1} = x_k - \frac{1}{L}\nabla f(x_k).$

Apply descent lemma with  $x = x_k, ; y = x_{k+1}$ :

$$\begin{aligned} f(x_{k+1}) &\leq f(x_k) + \left\langle \nabla f(x_k), -\frac{1}{L}\nabla f(x_k) \right\rangle + \frac{L}{2}\left\| -\frac{1}{L}\nabla f(x_k) \right\|^2 \\ &= f(x_k) - \frac{1}{2L}\|\nabla f(x_k)\|^2. \end{aligned}$$

# Gradient Descent: Convergence (Proof II)

**Strong convexity:**  $f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2} \|y - x\|^2, \quad \forall x, y.$

**Strong convexity**  $\Rightarrow$  lower bound on gradient norm

$$\begin{aligned} f(x^*) &\geq f(x) + \langle \nabla f(x), x^* - x \rangle + \frac{\mu}{2} \|x^* - x\|^2 \\ \Rightarrow f(x) - f(x^*) &\leq \langle \nabla f(x), x - x^* \rangle - \frac{\mu}{2} \|x - x^*\|^2 \\ &\leq \|\nabla f(x)\| \|x - x^*\| - \frac{\mu}{2} \|x - x^*\|^2 \end{aligned}$$

Consider quadratic in  $t = \|x - x^*\| : \|\nabla f(x)\| t - \frac{\mu}{2} t^2 \leq \max_{t \geq 0} \left( \|\nabla f(x)\| t - \frac{\mu}{2} t^2 \right)$

Maximum occurs at  $t = \frac{\|\nabla f(x)\|}{\mu}$ , giving value  $\frac{1}{2\mu} \|\nabla f(x)\|^2$

$$f(x) - f(x^*) \leq \frac{1}{2\mu} \|\nabla f(x)\|^2 \Rightarrow \|\nabla f(x)\|^2 \geq 2\mu(f(x) - f(x^*)).$$

# Gradient Descent: Convergence (Proof III)

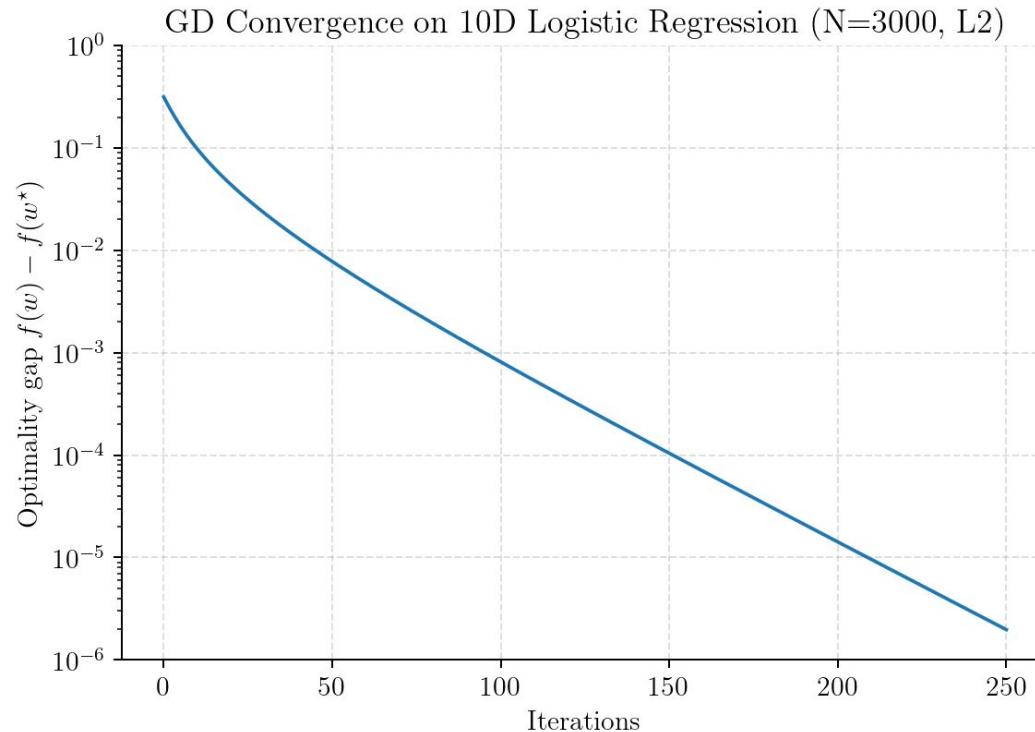
$$f(x_{k+1}) \leq f(x_k) - \frac{1}{2L} \|\nabla f(x_k)\|^2. \quad \|\nabla f(x)\|^2 \geq 2\mu(f(x) - f(x^*)).$$

Combine bounds:

$$\begin{aligned} f(x_{k+1}) - f(x^*) &\leq f(x_k) - \frac{1}{2L} \|\nabla f(x_k)\|^2 - f(x^*) \\ &\leq f(x_k) - f(x^*) - \frac{\mu}{L} (f(x_k) - f(x^*)) \\ &= \left(1 - \frac{\mu}{L}\right) (f(x_k) - f(x^*)). \end{aligned}$$

**Linear convergence:**  $f(x_k) - f(x^*) \leq \left(1 - \frac{\mu}{L}\right)^k (f(x_0) - f(x^*)).$

# Gradient Descent: Convergence. Example



# Gradient Descent: Evaluation Cost

$$f(w) = \frac{1}{N} \sum_{i=1}^N \log \left( 1 + \exp(-y_i \cdot \langle w, x_i \rangle) \right) + \lambda \cdot \|w\|^2$$

$$= \frac{1}{N} \sum_{i=1}^N f_i(w) + R(w)$$

$$\nabla_w f(w) = \frac{1}{N} \sum_{i=1}^N \nabla_w f_i(w) + \nabla_w R(w)$$

One gradient descent step requires accumulating gradient over the entire dataset

How do we handle large datasets?

Well, large dataset is just many small datasets :-)

# Full gradient → batch gradient

**Batch:**  $B_m = [i_1, \dots, i_m]$

- collection of  $m$  indices,  $m \ll N$
- randomly selected (e.g. uniformly with replacement).

$$f(w) = \frac{1}{N} \sum_{i=1}^N f_i(w) + R(w)$$

**Batch loss**

$$f_{B_m}(w) = \frac{1}{m} \sum_{i \in B_m} f_i(w) + R(w)$$

**Batch gradient**

$$g_{B_m}(w) = \nabla_w f_{B_m}(w) = \frac{1}{m} \sum_{i \in B_m} \nabla_w f_i(w) + \nabla_w R(w)$$

$B_m$  is a random variable  $\Rightarrow f_{B_m}(w), g_{B_m}(w)$  are random variables

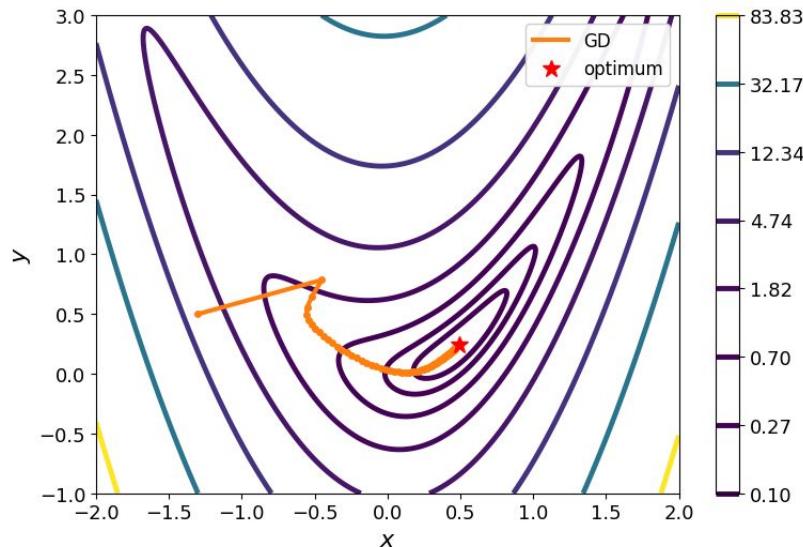
**Unbiased Estimates:**  $\mathbb{E}_{B_m} [f_{B_m}(w)] = f(w), \quad \mathbb{E}_{B_m} [g_{B_m}(w)] = \nabla_w f(w)$

$$g_{B_m}(w) = \nabla_w f(w) + \xi_{B_m}, \quad \mathbb{E} [\xi_{B_m}] = 0$$

# Stochastic Gradient Descent

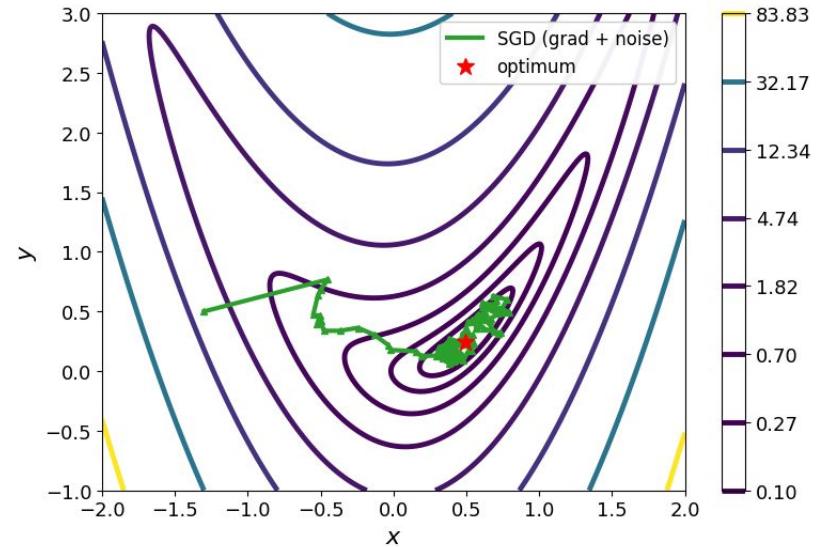
## Gradient descent update

$$x_{i+1} = x_i - \alpha_i \nabla f(x_i)$$



## SGD update: gradient descent with noisy gradients

$$x_{i+1} = x_i - \alpha_i g_i, \quad g_i = \nabla f(x_i) + \xi_i$$



# Stochastic Gradient Descent

**SGD update: gradient descent with noisy gradients**

$$x_{i+1} = x_i - \alpha_i g_i, \quad g_i = \nabla f(x_i) + \xi_i$$

**Assumptions:**

- **strong convexity and smoothness**

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2} \|y - x\|^2,$$

$$\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|$$

- **unbiased gradient, bounded noise variance**

$$\mathbb{E}[\xi_i | x_i] = 0, \quad \mathbb{E}|\xi_i|^2 \leq \sigma^2$$

- **Robbins–Monro stepsizes**

$$\alpha_i > 0, \sum_{i=1}^{\infty} \alpha_i = \infty, \quad \sum_{i=1}^{\infty} \alpha_i^2 < \infty, \quad \text{e.g. } \alpha_i = \frac{c}{i^{\gamma}}, \quad \gamma \in (\frac{1}{2}, 1]$$

**Sublinear convergence of iterate averages**  $\bar{x}_k = \frac{1}{k} \sum_{i=1}^k x_i$

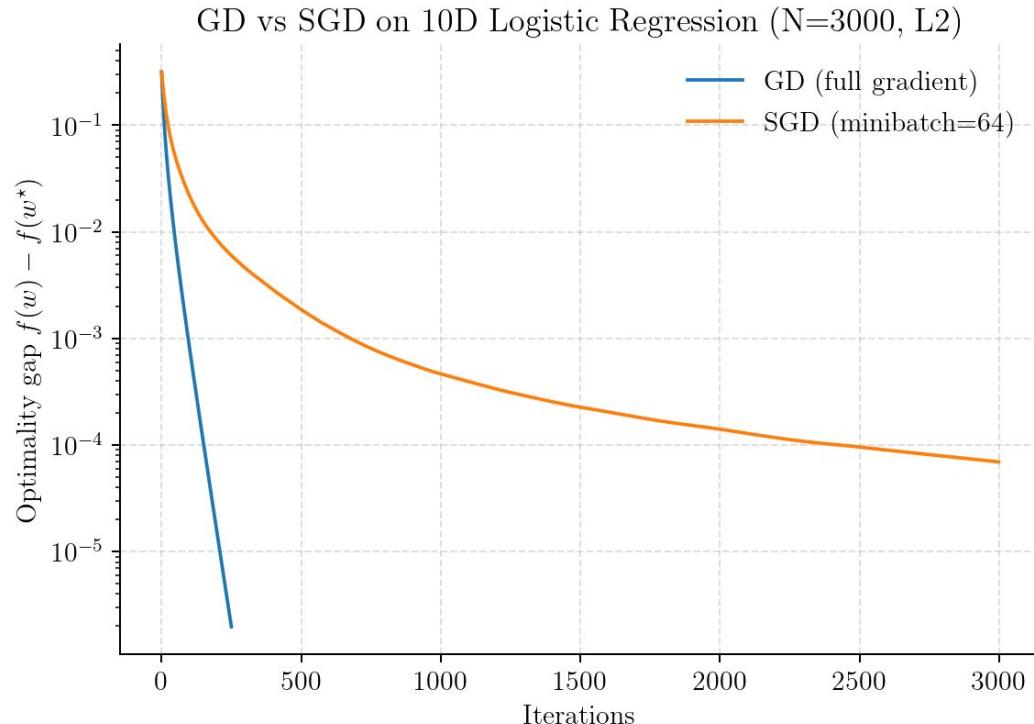
$$C = \text{Tr}(H^{-1} \Sigma H^{-1}), \quad H = \nabla^2 f(x^*), \quad \Sigma = \text{Cov}(\xi_i | x^*)$$

$$\mathbb{E} \|\bar{x}_k - x^*\|^2 \leq \frac{C}{k} + o\left(\frac{1}{k}\right),$$

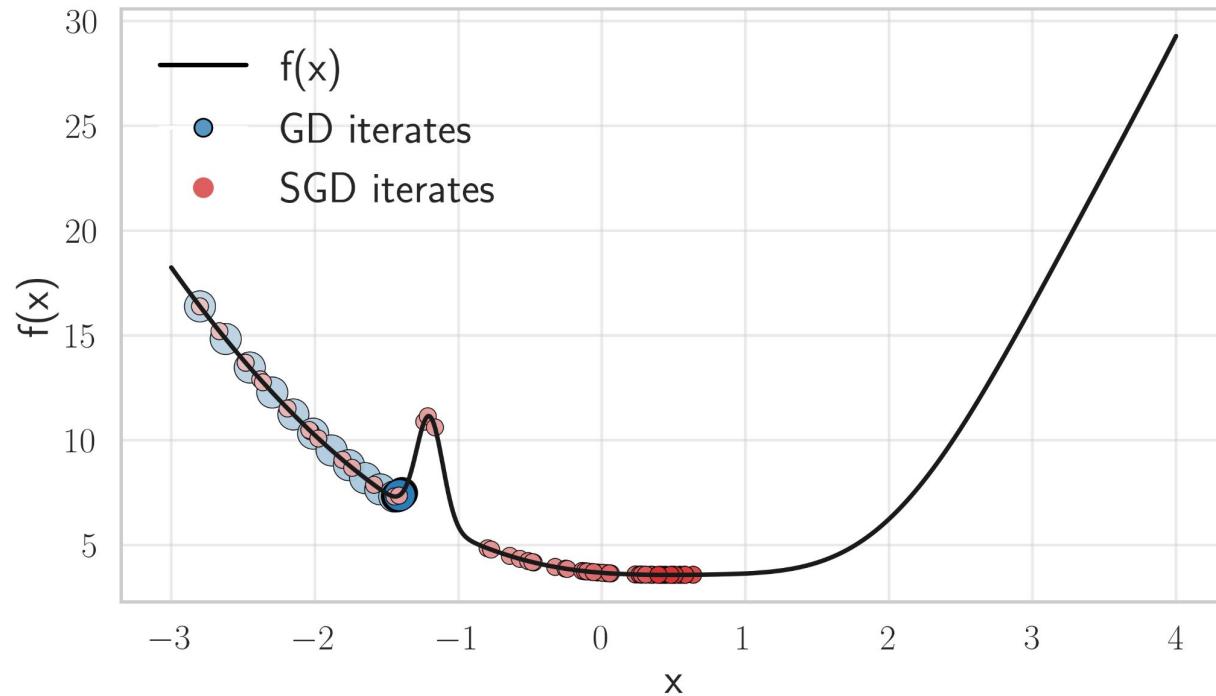
$$\mathbb{E} [f(\bar{x}_k) - f(x^*)] \leq \frac{L}{2} \mathbb{E} \|\bar{x}_k - x^*\|^2 = O\left(\frac{1}{k}\right)$$

$$k(\varepsilon) \text{ such that } \mathbb{E}[f(\bar{x}_k) - f(x^*)] \leq \varepsilon \iff k(\varepsilon) = O\left(\frac{1}{\varepsilon}\right)$$

# Stochastic Gradient Descent. Example



# Is full-batch gradient descent always better?



# Convex Optimization → Non-Convex Optimization

## Convex functions

- Unique optimal solution (global minimum)
- Algorithms with convergence guarantees

## Non-convex functions:

- Might have many optimal solutions (global minima)
- Might have many local minima
- In general, no guarantees, optimization is NP-hard
- Is there hope?
- Should we abandon ML models with non-convex loss?

# Deep Learning & Optimization

Deep Learning millions/billions of parameters; Non-convex loss

Example: Image classification on ImageNet with ResNet-152

- 60 million parameters in the model
- 1 million training examples

Handwavy argument:

- If I have 60M variables and 1M equations
- I can expect to get infinite number of solutions
- Dimension of solution space:  $59M = 60M - 1M$

# Deep Learning & Optimization & Generalization

Deep Learning millions/billions of parameters; Non-convex loss

- “Overparameterized” models
- Large number of local “solutions”
  - Shown both in theory (local optima)
  - Shown in practice (many distinct low train loss points)
- We know that some of the solutions are bad
- It is possible to find a solution where
  - Train set predictions are ~100% accurate
  - Test set predictions are completely off

**Yet, Deep Networks, trained with SGD, generalize to test data in practice.**

**Why?**

# Deep Learning & Optimization & Generalization

**Deep Networks, trained with SGD, generalize to test data in practice. Why?**

Well, we engineered the models, and the training procedures this way

- It took a lot of time and trial and error to get Deep Networks to work
  - network architecture / initialization / learning rate schedule / ...
- We selected for techniques that are likely to lead to good solutions

Model families and learning algorithms that work well have good “**inductive biases**”

- Perhaps there are “more” good solutions than bad solutions?
- Perhaps bad solutions are hard to reach with gradient descent?
  - Sharp optima / wide optima hypothesis
- Effects of overparameterization are not fully understood

# Understanding optimization & generalization?

## Approaches to analysing optimization & generalization in deep learning

### Theoretical:

- Find set of assumption, which lead to phenomena observed in practice.
- Prove theorems about optimization & generalization under assumptions

### Empirical:

- Observe interesting phenomena in practice.
- Ask questions.
- Design and run experiments.

# Visualization

What is a simple thing we can do to understand optimization?

**Let's visualize loss landscapes of deep networks!**

1D visualization:

- Pick a 1D line in D-dimensional space
- Build a grid on the line
- Compute loss in each point on the grid

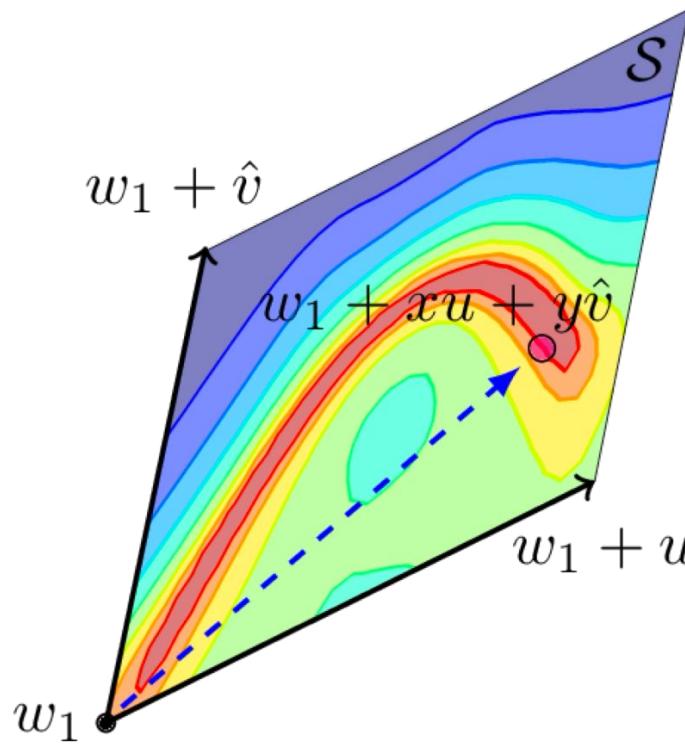
2D visualization:

- Pick a 2D plane in D-dimensional space
- Build a grid on the plane
- Compute loss in each point on the grid

**We can't visualize more than 2 dimensions :-(**

**But we can be creative in our choice of lines/planes :-)**

# LOSS SURFACE VISUALIZATIONS



- ▶ Pick basis vectors  $u, v \in \mathbb{R}^{|\text{net}|}$
- ▶ Pick a shift direction  $w_1 \in \mathbb{R}^{|\text{net}|}$
- ▶ Orthogonalize:  $\hat{v} = v - \frac{\langle v, u \rangle}{\|u\| \|v\|} u$
- ▶ Map  $(x, y) \rightarrow w_1 + xu + y\hat{v}$
- ▶ Define grid in  $(x, y)$  and make a contour plot

We plot loss restricted to a 2D subspace of parameter space

# Loss landscape visualizations

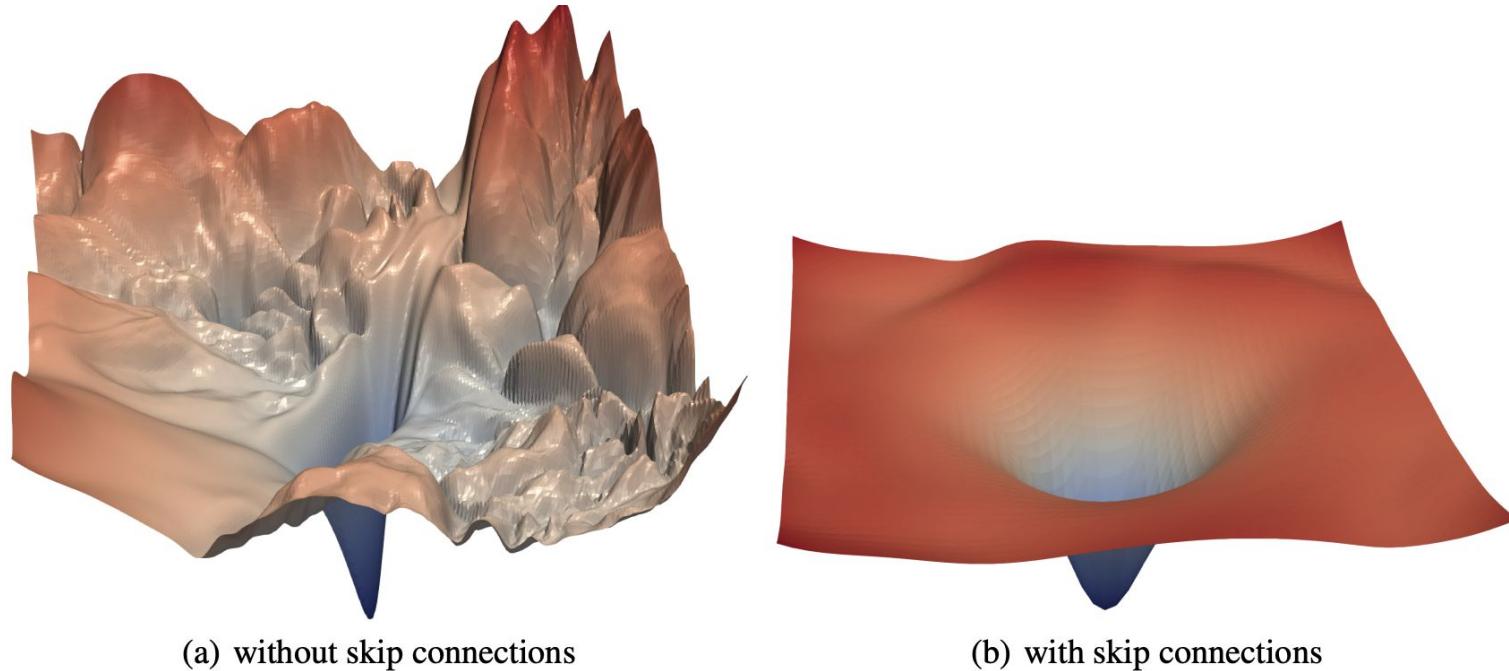
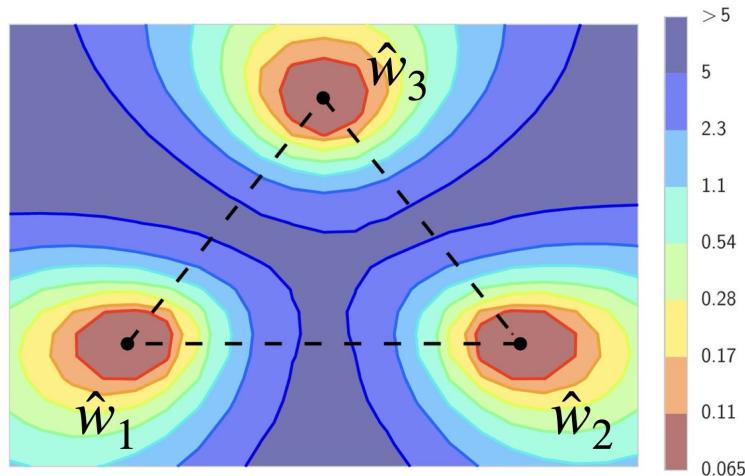


Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

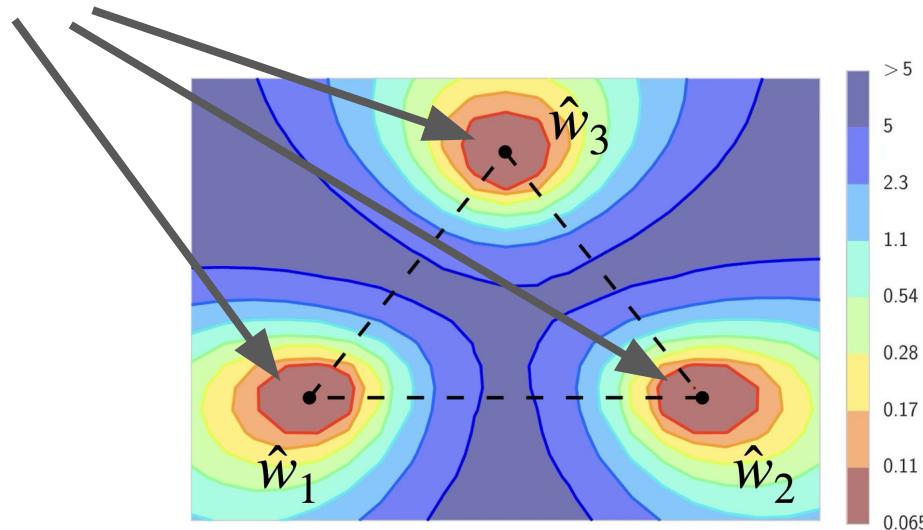
# Multiple local optima



ResNet-164, CIFAR-100; train loss;  $10^6$  parameters

# Multiple local optima

Optima  $\leftrightarrow$  similarly good, but different classifiers



Can be combined  
for improved predictions  
and uncertainty

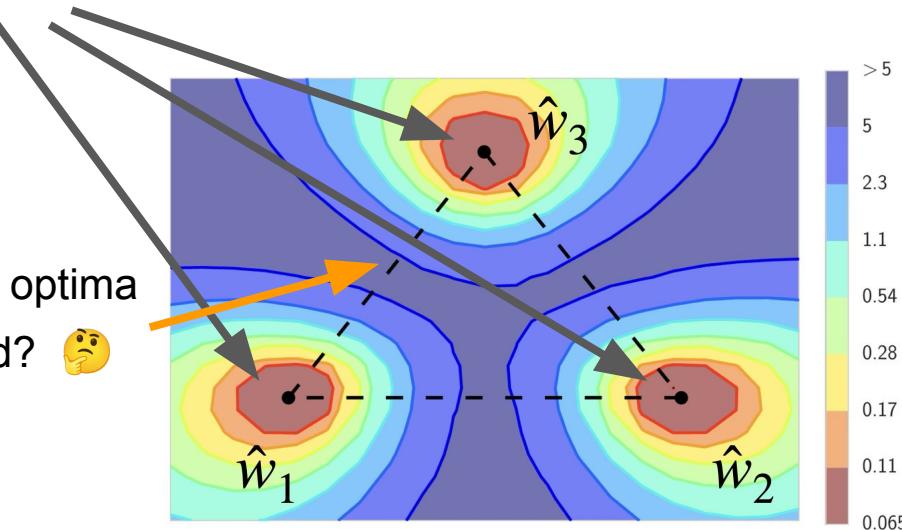
ResNet-164, CIFAR-100; train loss;  $10^6$  parameters

# Multiple local optima

Optima  $\leftrightarrow$  similarly good, but different classifiers

Poor loss between optima

Are optima isolated? 🤔



ResNet-164, CIFAR-100; train loss;  $10^6$  parameters

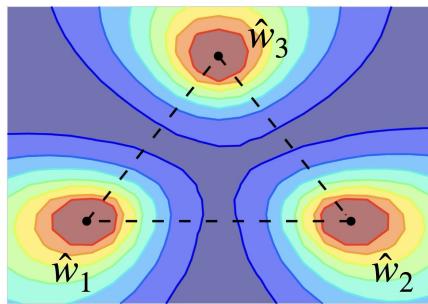
Can be combined  
for improved predictions  
and uncertainty

Do we need to train from scratch for diverse solutions?

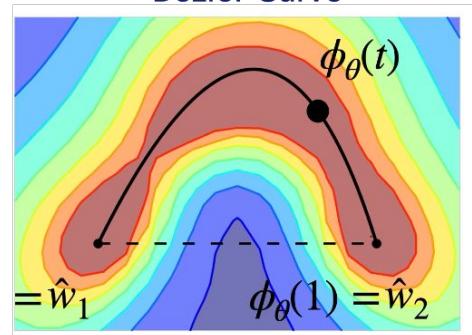
# Mode connectivity

*Optima are connected by simple paths of low loss!*

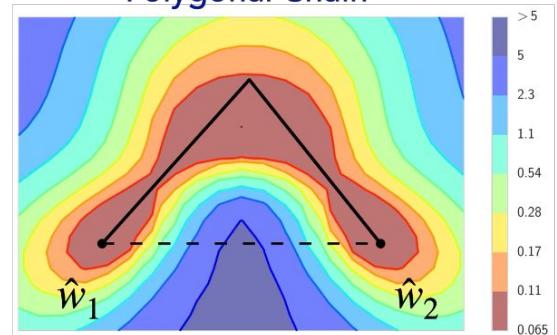
Independent Modes



Bezier Curve



Polygonal Chain

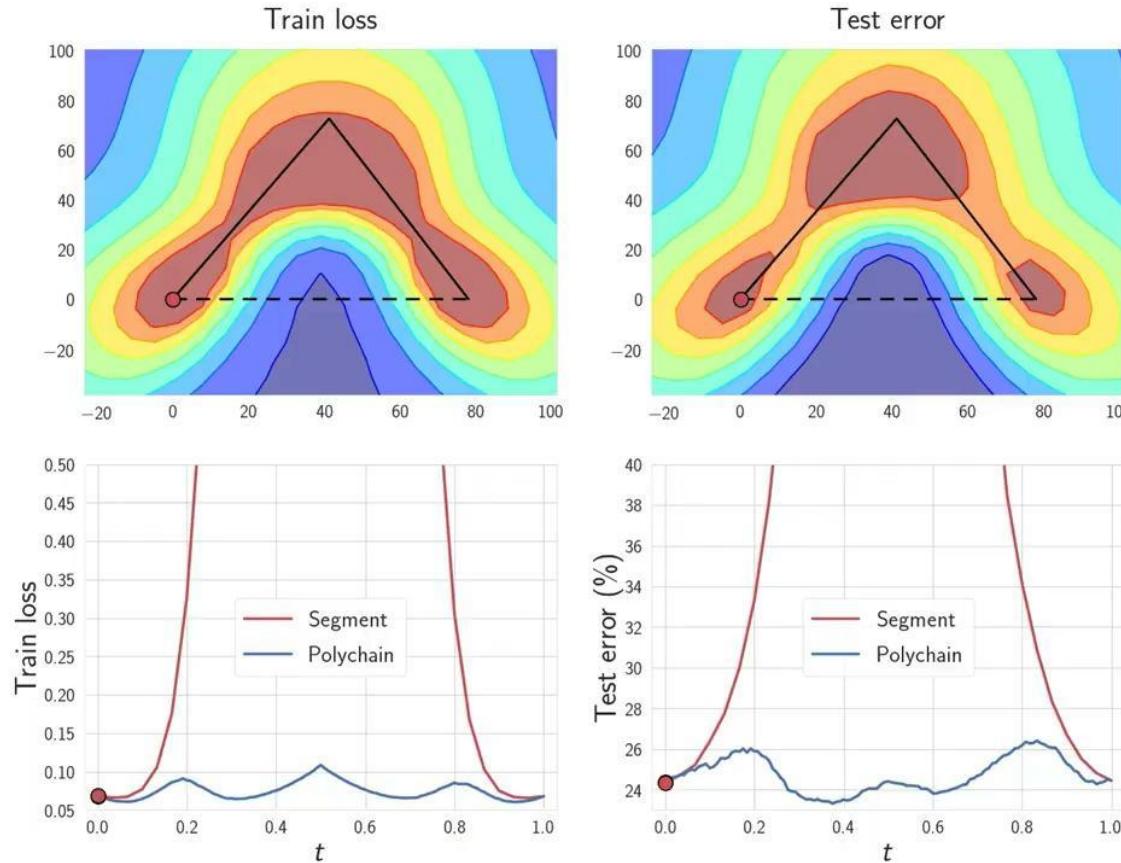


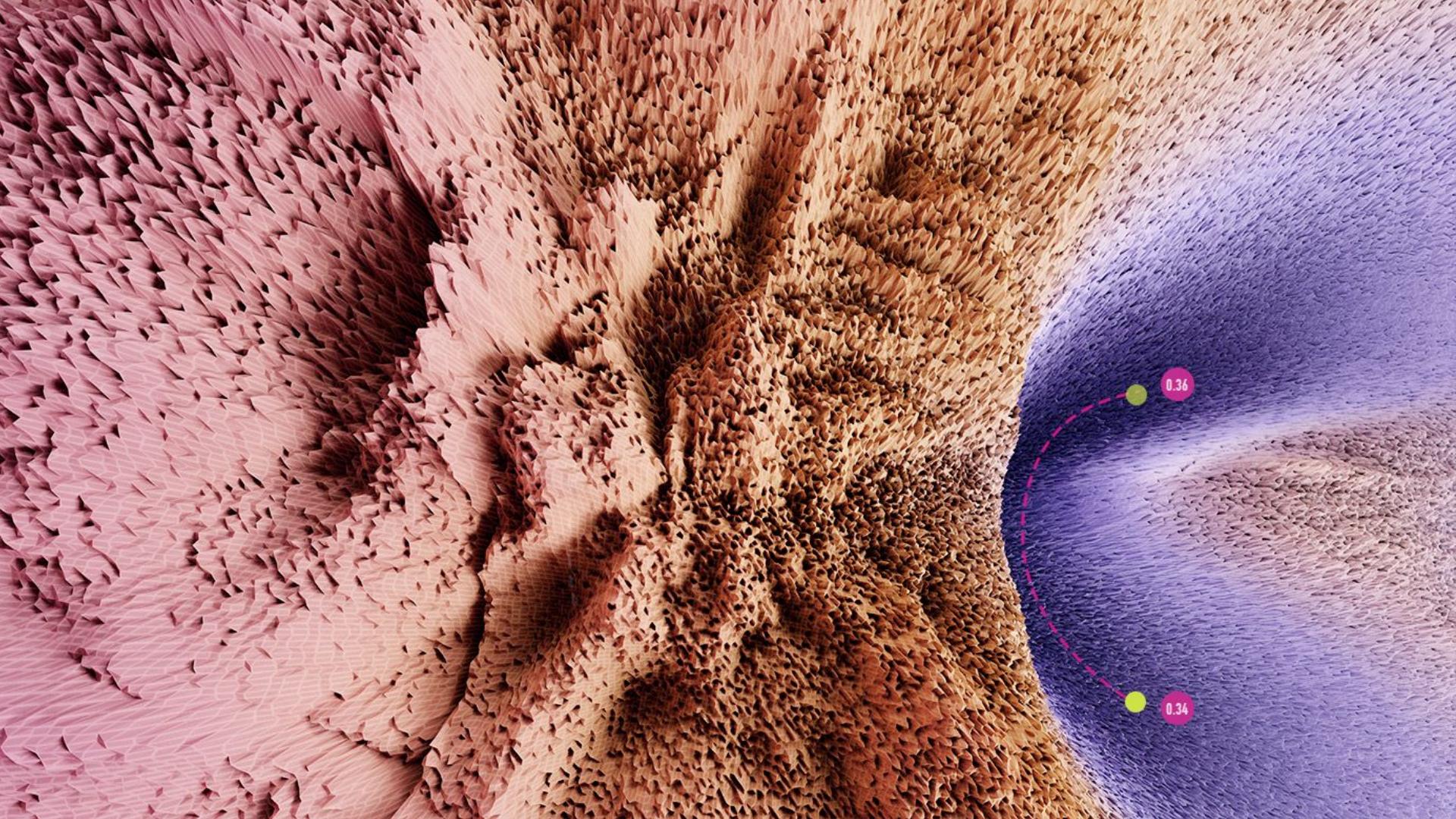
ResNet-164, CIFAR-100; train loss;  $10^6$  parameters

$$\phi_\theta(t) = (1 - t)^2 \cdot \hat{w}_1 + 2t(1 - t) \cdot \theta + t^2 \cdot \hat{w}_2$$

Minimize  $\ell(\theta) = \mathbb{E}_{t \sim U[0,1]} L(\phi_\theta(t)); \quad \nabla_\theta \ell(\theta) = \mathbb{E}_{t \sim U[0,1]} \nabla_\theta L(\phi_\theta(t))$

# Mode connectivity





0.36

0.34



The background of the image is a 3D surface visualization, possibly representing terrain or a complex landscape. The surface is composed of numerous small, vertical, colored spikes. A prominent color gradient runs diagonally across the scene, transitioning from deep purple on the left and top to bright yellow on the right and bottom. The surface shows various geological features like ridges, valleys, and craters, all rendered in a textured, spiky style.

**Visualizations produced in collaboration with Javier Ideami**

**More visualizations at [losslandscape.com](http://losslandscape.com)**

# Flatness & Generalization

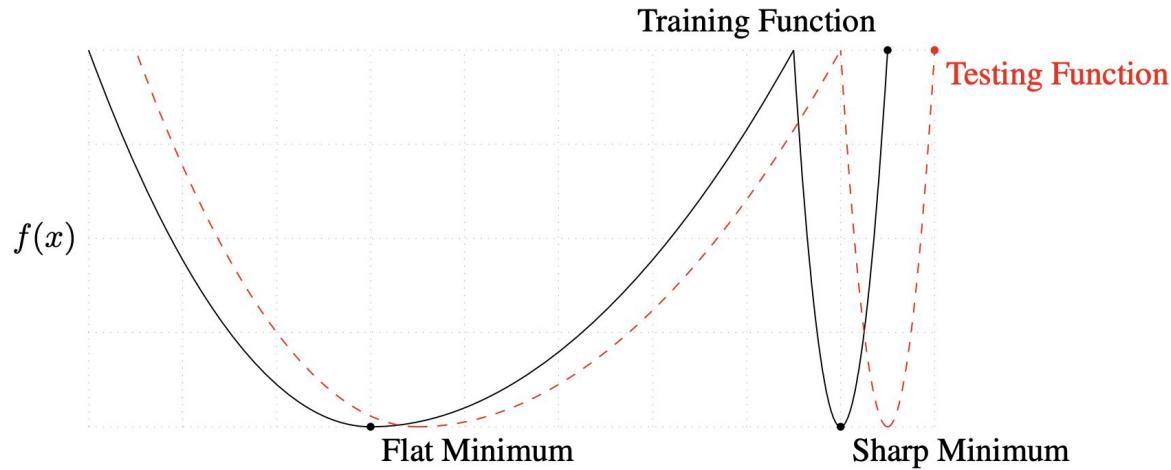
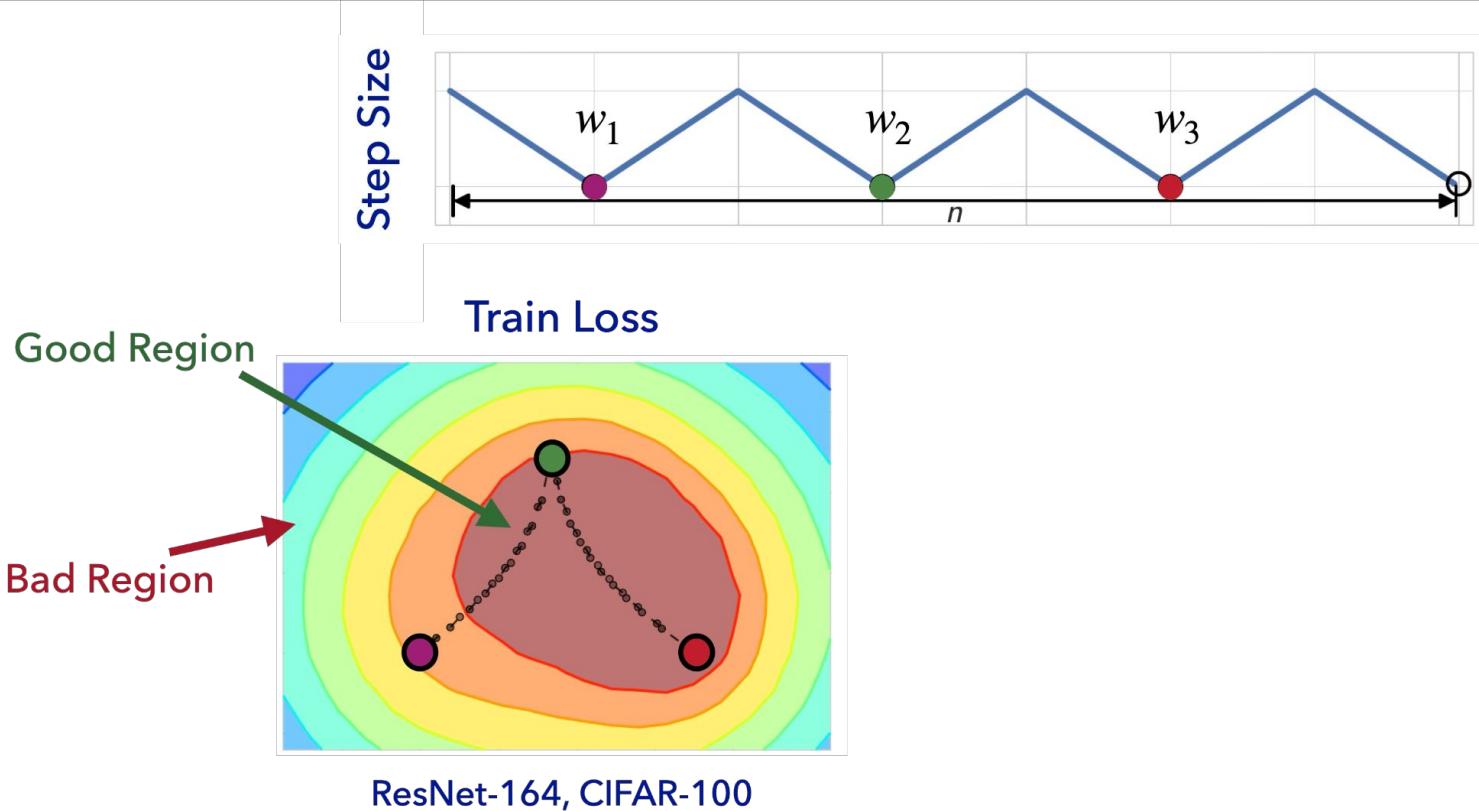
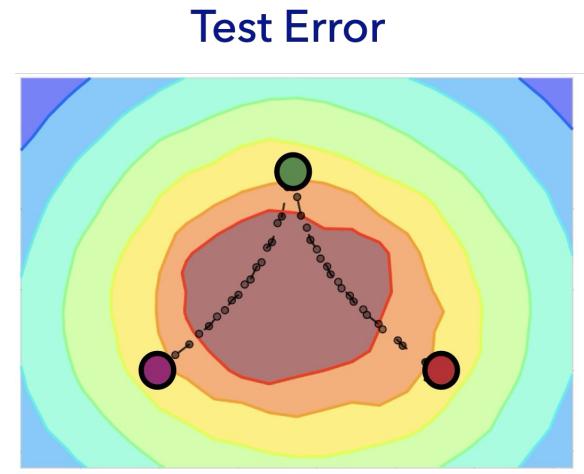
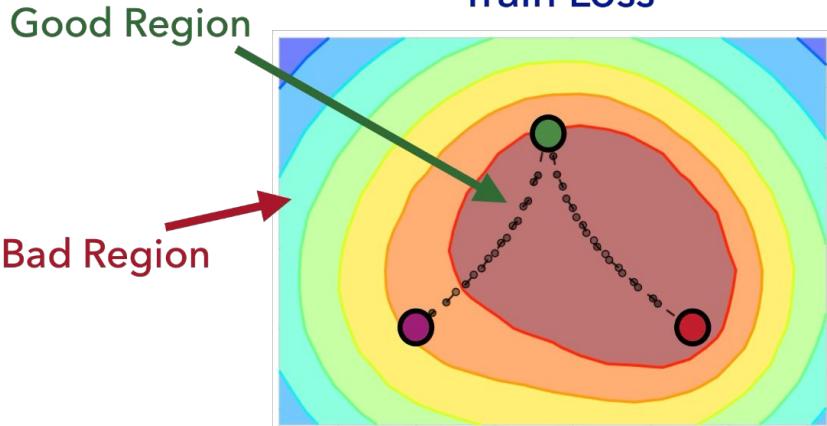
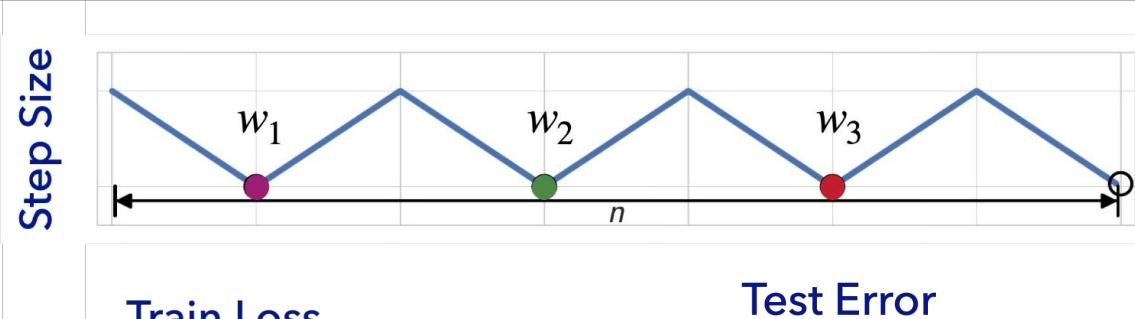


Figure 1: A Conceptual Sketch of Flat and Sharp Minima. The Y-axis indicates value of the loss function and the X-axis the variables (parameters)

# Exploring SGD trajectories

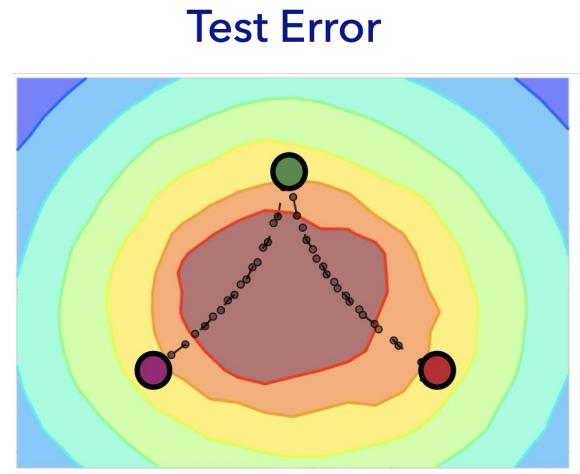
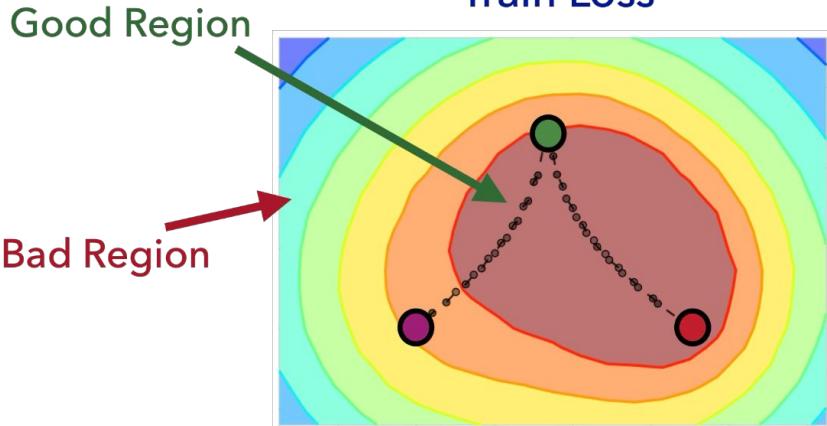
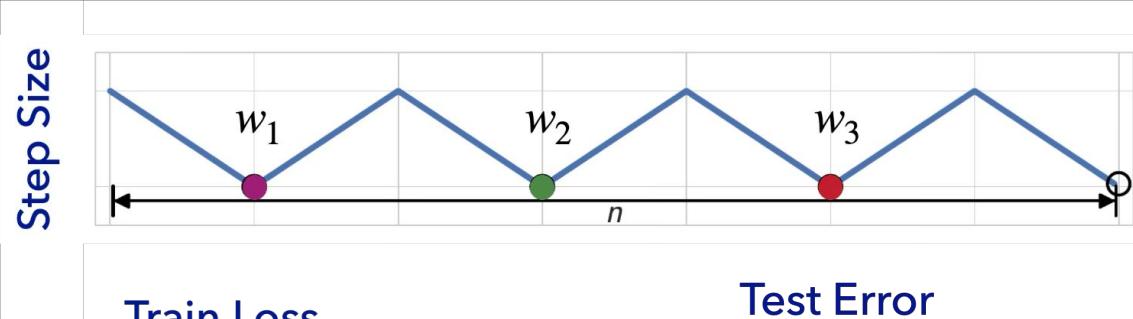


# Exploring SGD trajectories



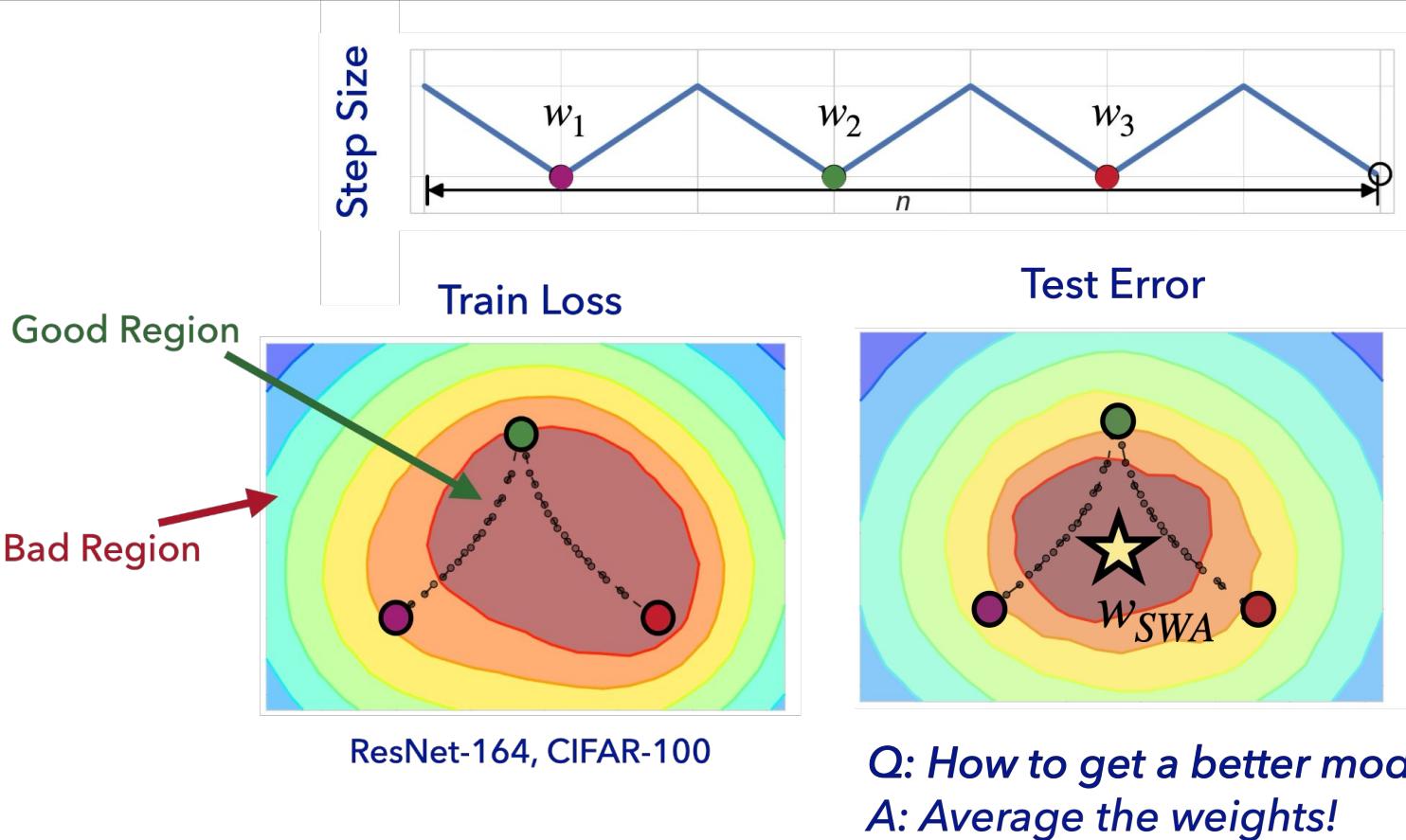
*Q: How to get a better model?*

# Exploring SGD trajectories

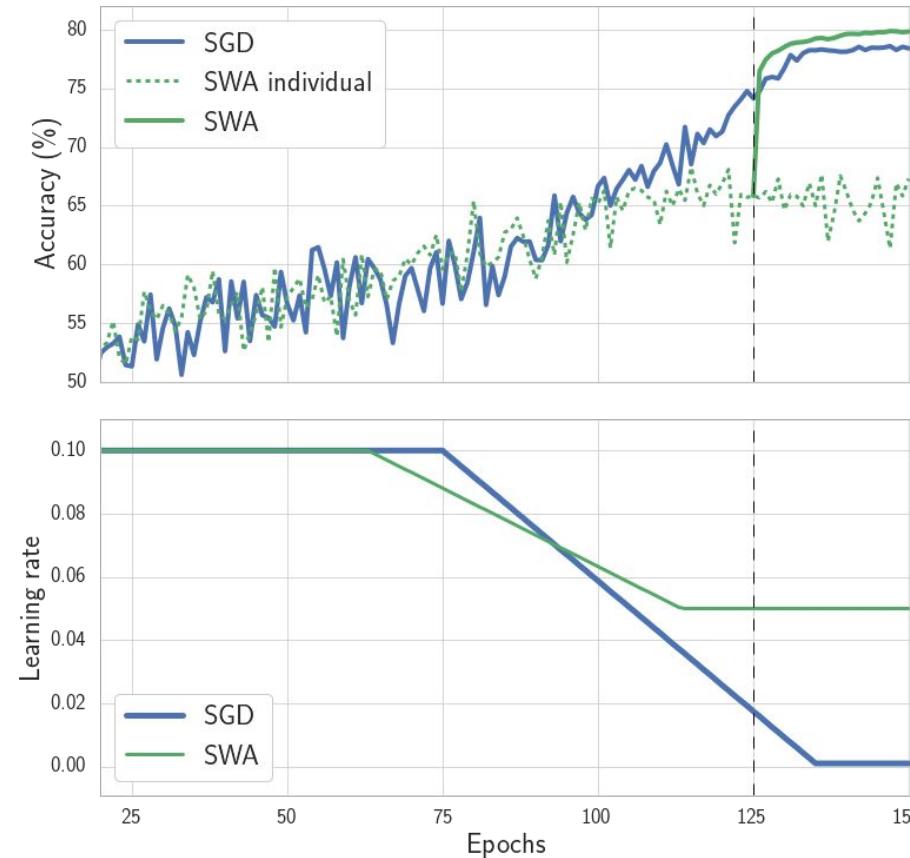


*Q: How to get a better model?*

# Exploring SGD trajectories



# Stochastic Weight Averaging (SWA) [Izmailov et al. 2018]



Model	SGD	SWA
VGG-16	$72.5 \pm 0.10$	$74.3 \pm 0.3$
ResNet-164	$78.5 \pm 0.4$	$80.4 \pm 0.2$
WRN-28-10	$80.8 \pm 0.2$	$82.2 \pm 0.3$
ResNet-50	76.15	$77.0 \pm 0.05$
DenseNet-161	77.65	$78.5 \pm 0.1$

CIFAR-100  
ImageNet

**Learning rate is important**

Too low learning rate:

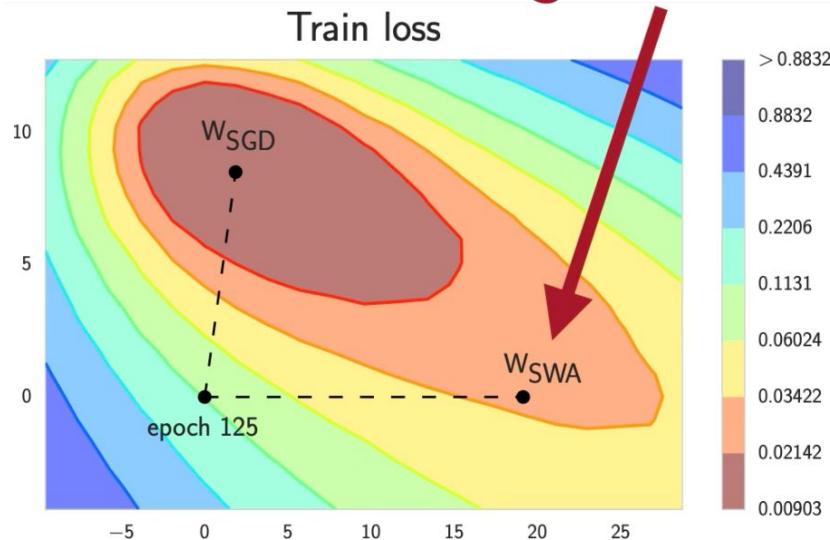
- little exploration

Too high learning rate:

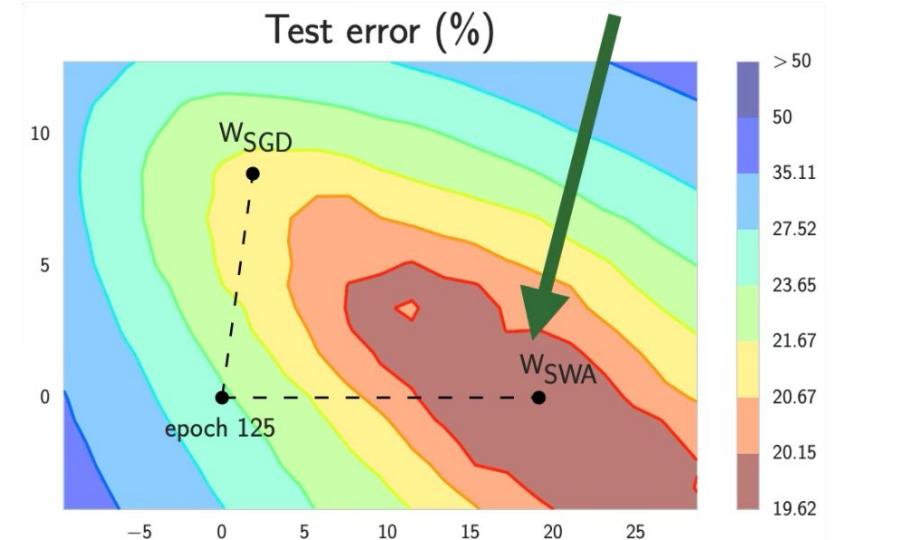
- poor individual model accuracy

# Stochastic Weight Averaging (SWA) [Izmailov et al. 2018]

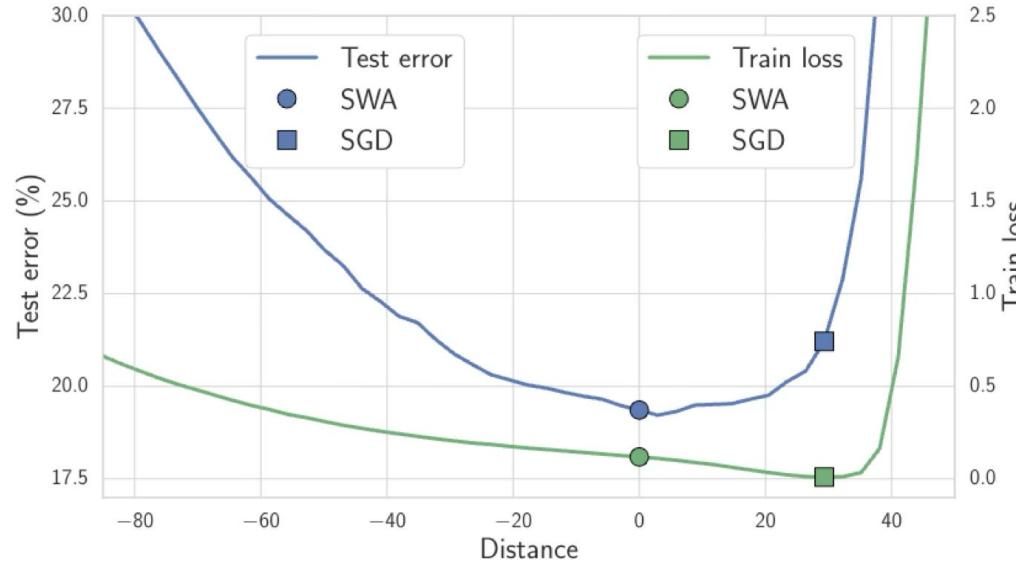
Higher Train Loss



Lower Test error



# Stochastic Weight Averaging (SWA) [Izmailov et al. 2018]



- ▶ Train loss landscape is a **perturbed version** of test loss landscape
- ▶ SGD **overoptimizes train loss** and finds solution that is **not robust** to weight perturbations ("**sharp optima**")
- ▶ Cyclical learning rate SGD **explores a local region of good solutions**
- ▶ SWA finds a **robust solution ("wide optima")**

August 18, 2020

# PyTorch 1.6 now includes Stochastic Weight Averaging

```
from torch.optim import swa_utils
```

The screenshot shows the TensorFlow API documentation for the module `tfa.optimizers.SWA`. The page is titled "TensorFlow > Resources > Addons > API". Below the title is the module name `tfa.optimizers.SWA`. At the bottom of the page, there are links for "[How to Use]", "[Suggested Hyperparameters]", "[Attribution]", and "[API Reference]".

The screenshot shows the INA composer library documentation for "Stochastic Weight Averaging". The page features the INA logo and the word "composer" in red. Below the title is a green gear icon followed by the text "Stochastic Weight Averaging". At the bottom, there are links for "Computer Vision" and "Natural Language Processing".

This block contains a screenshot of a research paper from the journal "Computer Methods and Programs in Biomedicine". The journal cover features a tree and the Elsevier logo. The paper title is "Efficient skin lesion segmentation using separable-Unet with stochastic weight averaging". It lists authors Tang Peng, Liang Qiaokang, Yan Xintong, Xiang Shao, Sun Wei, Zhang Dan, and Gianmarc Coppola. The paper was published in Volume 178, September 2019, Pages 289-301.

This block contains a screenshot of a research paper titled "Benefits of Stochastic Weight Averaging in Developing Neural Network Radiation Scheme for Numerical Weather Prediction". The authors listed are Hwan-Jin Song, Soonyoung Roh, Juho Lee, Giung Nam, Eunggu Yun, Jongmin Yoon, and Park Sa Kim.

This block contains a screenshot of a research paper titled "Improving Generalization of Pre-trained Language Models via Stochastic Weight Averaging". The authors listed are Peng Lu, Ivan Kobyzev, Mehdi Rezagholizadeh, Ahmad Rashid, Ali Ghodsi, and Philippe Langlais.

Model Soups   
ImageNet SOTA (2022)

Git Re-Basin   
Model Fusion

SWAD

SWAP

SWAG

SWALP

TSWA

# Branch-Train-Merge: Embarrassingly Parallel Training of Expert Language Models

Margaret Li<sup>\*†</sup>

Suchin Gururangan<sup>\*†○</sup>

Tim Dettmers<sup>†</sup>

Mike Lewis<sup>△</sup>

Tim Althoff<sup>†</sup>

Noah A. Smith<sup>†◆</sup>

Luke Zettlemoyer<sup>†○</sup>

<sup>†</sup>Paul G. Allen School of Computer Science & Engineering, University of Washington

<sup>◆</sup>Allen Institute for AI

<sup>○</sup>Meta AI

## Early Weight Averaging meets High Learning Rates for LLM Pre-training

Sunny Sanyal\*

UT Austin

Atula Neerkaje\*

UT Austin

Jean Kaddour

UCL

Abhishek Kumar

Google DeepMind

Sujay Sanghavi

UT Austin

Google DeepMind

## WARM: On the Benefits of Weight Averaged Reward Models

Alexandre Ramé, Nino Vieillard, Léonard Hussenot, Robert Dadashi, Geoffrey Cideron, Olivier Bachem, Johan Ferret  
Google DeepMind



Sebastian Raschka

@rasbt

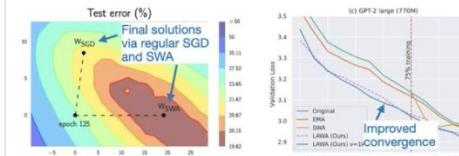
...

Weight averaging and model merging for LLMs seem to be the most interesting themes in 2024 so far. What are the benefits? Combining multiple models (or checkpoints) into a single one can improve training convergence, overall performance, and also robustness.

I will probably do a deeper dive in the upcoming weeks, but here are at least 3 interesting papers. To get started, here's a selection of papers on this learning trajectory (no pun intended).

### Classic Stochastic Weight Averaging (SWA)

Average the weights (model checkpoints) towards the end of training when the learning rate is low.



### Early Weight Averaging + High learning Rates (LAWA)

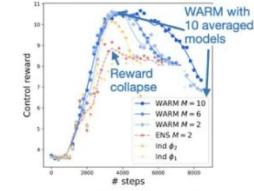
Average the checkpoints with high learning rate (early and considerably spaced) during the training run for higher gains

Early Weight Averaging meets High Learning Rates for LLM Pre-training, Sanyal et al. (2018), <https://arxiv.org/abs/1806.03241>

### Weight Averaging of LLM Reward Models (WARM)

Average the weights of multiple finetuned reward models to improve efficiency for reinforcement learning for human feedback.

How? Start with (1) averaging SFT models throughout the trajectory, (2) finetune reward models, then (3) average the final reward models.



10:34 AM · Jan 24, 2024 · 151.8K Views

21

178

1K

796



# Deep Learning Loss Landscapes

“Global structure”:

- Multiple distinct local optima
- Optima are connected by non-linear paths of low loss

“Local structure”:

- SGD with non-zero LR doesn't seem to exit linearly-connected region
- Medium-LR SGD bounces around in a wide region of low loss points
- Averaging SGD iterates helps to land in the interior of good region
- Empirically averaging finds a model that generalizes better

Still active area of research

# Gradient Descent: Variations

## Gradient Descent Update

$$x_{k+1} = x_k - \alpha \nabla_x f(x_k)$$

## Change notation

$$\frac{x(t + dt) - x(t)}{dt} = -\nabla_x f(x(t))$$

**Looks like discrete-time numerical integration of continuous time ODE**

$$\dot{x}(t) = -\nabla_x f(x(t)) \quad (\text{a.k.a. gradient flow})$$

## Physical interpretation

$x(t)$  is a trajectory of an object which moves with velocity  $\dot{x}(t)$

given by the the anti-gradient vector field  $v(x) = -\nabla_x f(x)$

**What would change if gradient influenced acceleration instead of velocity?**

# Gradient Descent with Momentum

## Gradient Descent Update

$$x_{k+1} = x_k - \alpha \nabla_x f(x_k)$$

**What would change if gradient influenced acceleration instead of velocity?**

Object with a unit mass ( $m = 1$ ) moves under  $a = \frac{F}{m}$  in a force field  $F(x) = -\nabla_x f(x)$ .

$$\ddot{x}(t) = -\nabla_x f(x(t))$$

state of the object is described by position  $x(t)$  and velocity  $v(t)$ .

(dynamical system with 2D state)

$$\frac{d}{dt} \begin{bmatrix} v(t) \\ x(t) \end{bmatrix} = \begin{bmatrix} -\nabla_x f(x(t)) \\ v(t) \end{bmatrix}$$

## Gradient Descent with Momentum

$$v_{k+1} = \beta v_k - \nabla_x f(x_k) \quad (\beta \in [0, 1] — \text{dampening})$$

$$x_{k+1} = x_k + \alpha_k v_{k+1}$$

# Gradient Descent with Momentum

## Gradient Descent Update

$$x_{k+1} = x_k - \alpha \nabla_x f(x_k)$$

## Gradient Descent with Momentum

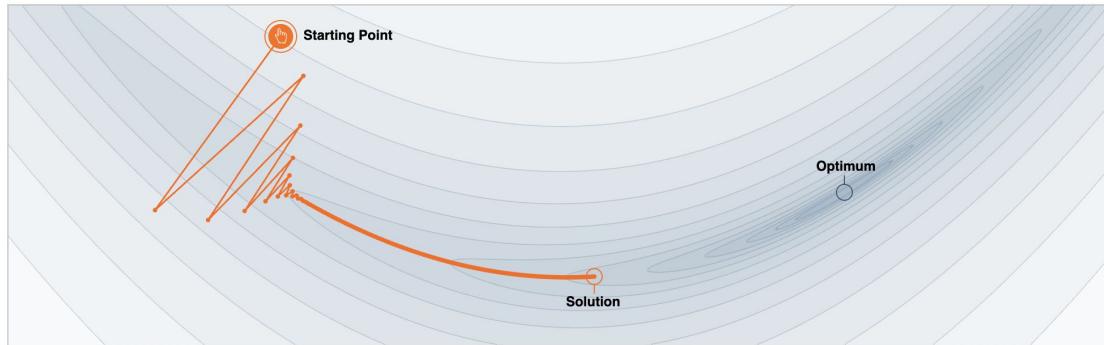
$$v_{k+1} = \beta v_k - \nabla_x f(x_k) \quad (\beta \in [0, 1] \text{ — dampening})$$

$$x_{k+1} = x_k + \alpha_k v_{k+1}$$

For strongly-convex and L-smooth functions

**Linear convergence rate**, but **better constant**

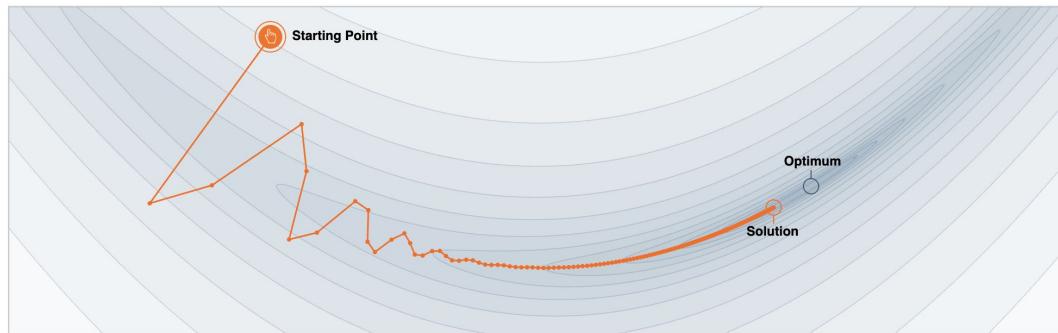
Visualizations by Gabriel Goh, "Why Momentum Really Works", Distill, 2017.  
<https://distill.pub/2017/momentum/>



Step-size  $\alpha = 0.0032$



Momentum  $\beta = 0.0$



Step-size  $\alpha = 0.0032$



Momentum  $\beta = 0.66$



# ADAM

ADAM [Kingma et al. 2015]

- Go-To Optimization Method for Deep Learning
- Typically optimizes loss faster than SGD
- Two ideas
  - Stochastic Gradient Descent with Momentum
  - + adaptive learning rate scale for each parameter
- Typically requires minimal hyperparameter tuning
  - Handy when experimenting with new models

# ADAM

**for**  $t = 1$  **to** ... **do**

$$g_t \leftarrow -\nabla_{\theta} f_t(\theta_{t-1})$$

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$$

$$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$$

$$\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$$

## Notation

- $t$  — step number
  - $\theta_t$  — model parameters at step  $t$
  - $g_t$  — gradient
  - $m_t$  — moving average of gradients (velocity)
  - $v_t$  — moving average of squared gradients
- interpretation: average (squared) gradient magnitude per parameter
- $\widehat{m}_t, \widehat{v}_t$  — “bias corrected”  $m_t$  and  $v_t$
  - $\beta_1, \beta_2$  — exponential decay rates for moving averages
  - $\gamma$  — learning rate
  - $\epsilon$  — small number to prevent division by zero

## Update rule

$$\theta_t = \theta_{t-1} - \gamma \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon} \quad (\text{element-wise vector division})$$

# ADAM

## Update rule

$$\theta_t = \theta_{t-1} - \gamma \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t + \varepsilon}}$$

(element-wise vector division)

### Notation

- $m_t$  — moving average of gradients (velocity)
- $v_t$  — moving average of squared gradients
  - interpretation: average (squared) gradient magnitude per parameter
- $\widehat{m}_t, \widehat{v}_t$  — “bias corrected”  $m_t$  and  $v_t$
- $\gamma$  — learning rate
- $\varepsilon$  — small number to prevent division by zero

### Intuition:

- SGD w/momentum + adaptive per-parameter step size scaling
- Consider update for the  $i$ -th parameter  $\theta[i]$
- **Large**  $v_t[i] \implies$  loss is sensitive to the value of  $\theta[i]$ 
  - use **smaller step size** for  $\theta[i]$  to be safe
- **Small**  $v_t[i] \implies$  loss is not too sensitive to the value of  $\theta[i]$ 
  - use **larger step size** for  $\theta[i]$  to move faster

# Second Order Methods

$\min_{w \in \mathbb{R}^d} f(w)$  — unconstrained optimization problem in  $\mathbb{R}^d$

If  $f$  is smooth, then

$$f(x_0 + h) = f(x_0) + \langle \nabla_x f(x_0), h \rangle + O(\|h\|^2)$$

The direction of the fastest descent is  $h = -\nabla_x f(x_0)$

**What if we use second-order approximation?**

# Newton's Method

If  $f$  is smooth, then

$$f(x_0 + h) = f(x_0) + \langle \nabla_x f(x_0), h \rangle + \langle h, [\nabla_x^2 f(x_0)] h \rangle + O(\|h\|^3)$$

$[\nabla_x^2 f(x_0)]$  – Hessian;  $d \times d$  matrix of second derivatives

The direction of the fastest descent is  $h = -[\nabla_x^2 f(x_0)]^{-1} \nabla_x f(x_0)$

# Newton's Method

If  $f$  is smooth, then

$$f(x_0 + h) = f(x_0) + \langle \nabla_x f(x_0), h \rangle + \langle h, [\nabla_x^2 f(x_0)] h \rangle + O(\|h\|^3)$$

$[\nabla_x^2 f(x_0)]$  – Hessian;  $d \times d$  matrix of second derivatives

The direction of the fastest descent is  $h = -[\nabla_x^2 f(x_0)]^{-1} \nabla_x f(x_0)$

# Newton's Method

$$\min_{w \in \mathbb{R}^d} f(w) \quad \text{— unconstrained optimization problem in } \mathbb{R}^d$$

Pick  $x_0$ ; set  $k = 0$

Until `should_stop()` :

$$x_{k+1} = x_k - \alpha_k \cdot [\nabla_x^2 f(x_k)]^{-1} \nabla_x f(x_k)$$
$$k = k + 1$$

In convex optimization

- + Super-linear convergence rate
- + Under some conditions:  
quadratic convergence rate  
 $k(\varepsilon) = O\left(\log \log\left(\frac{1}{\varepsilon}\right)\right)$
- Each step requires  $O(d^3)$  time  
due to matrix inversion

Quasi-Newton Methods

- Approximate Newton's direction
- e.g. L-BFGS

# Second Order Methods for Large ML Models

For models with large number of parameters

- Just computing the Hessian takes  $O(d^2)$  time and memory
- Many quasi-second-order methods use cheap approximations of the Hessian
- Diagonal approximation
  - Compute only the diagonal of the Hessian
  - Takes  $O(d)$  time and memory
  - Results in per-parameter learning rate scaling
    - as in ADAM; but ADAM uses gradient history instead of Hessian
- Compressed Hessian approximations
  - Low-rank / matrix factorizations / sparse (block-sparse) / ...
- Recent methods for Deep Networks
  - Shampoo [Gupta et al., 2018]
  - SOAP [Vyas et al., 2024]

# Optimization in ML: Overview

Optimization is one of the tools in ML theory and practice

Theory:

- Learning as optimization
- Design & analysis of optimization algorithms, convergence guarantees
- Theoretically optimal solution → predictive model properties
  - Assuming infinite data and infinite compute, do we learn the right distribution?

$$\min_{f(\cdot)} \mathbb{E}_{p(y|x)} [(f(x) - y)^2] \implies f^*(x) = \mathbb{E}[y | x]$$

$$\min_{f(\cdot)} \mathbb{E}_{p(y|x)} [\log(1 + \exp(-yf(x)))] \implies f^*(x) = \text{logit}(p(y | x))$$

Practice:

- Guidelines for parameter tuning
- Inspiration for improved methods and empirical analysis

ML is not just optimization, we care about generalization too!

# Beyond Optimization

**Given:** training data

**Want:** predictive model  $f(x)$

**Learning Algorithm:**

Training Data => [Algorithm] => Model

- Our primary interest is in solving learning
- Optimization is one way to look at learning algorithms
- Are there other ways?

# Beyond Optimization

## Learning Algorithm:

Training Data => [Algorithm] => Model

“Most general view”: dynamical systems

An algorithm running on a computer is a dynamical system

- Memory = state
- Program = update rule

$$s_{t+1} = F(s_t)$$

Cannot get more general than that :-)

Instead of thinking about design of loss function and optimization algorithm

We can think about design of dynamical system

# Optimization $\subset$ Dynamical Systems

## Continuous optimization

Model Parameters & **Loss Function**  $\rightarrow$  **Gradient** Vector Field  $\rightarrow$  Update Rule

$$x_{t+1} = x_t + V(x_t), \quad V(x_t) = -\nabla_x \text{Loss}(x_t)$$

- $V(x)$  is called a “conservative” vector field in calculus / physics
- $-\text{Loss}(x)$  is the “potential” of  $V(x)$

## Continuous dynamical system

Model Parameters  $\rightarrow$  Vector Field  $\rightarrow$  Update Rule

$$x_{t+1} = x_t + V(x_t)$$

Let's come up with interesting non-conservative vector fields!

# Optimization $\subset$ Game Theory

## Optimization:

- One group of parameters
- Minimizing one loss function
- Solution: minimizer

$$\min_x U(x) \Rightarrow V(x) = -\nabla_x U(x) \Rightarrow x_t = x_t - \alpha_t \nabla_x U(x_t)$$

## Game:

- Multiple groups of parameters (players)
- Each player minimizes its own loss function (other terminology: maximizing utility fn)
- Solution: equilibrium point
  - E.g. Nash equilibrium

$$\begin{aligned} \min_{x_1} U_1(x_1, x_2) \\ \min_{x_2} U_2(x_1, x_2) \end{aligned} \Rightarrow V \left( \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} -\nabla_{x_1} U_1(x_1, x_2) \\ -\nabla_{x_2} U_2(x_1, x_2) \end{bmatrix} \Rightarrow$$

## Simultaneous Gradient Descent

$$\begin{aligned} x_{1,t+1} &= x_{1,t} - \alpha_t \nabla_{x_1} U_1(x_{1,t}, x_{2,t}) \\ x_{2,t+1} &= x_{2,t} - \alpha_t \nabla_{x_2} U_2(x_{1,t}, x_{2,t}) \end{aligned}$$

# Games in ML. Example: GANs

## Generative Adversarial Networks (GANs)

- Generative model
- Goal:
  - Given a dataset of examples (e.g. images)
  - Learn to generate new examples (e.g. new realistic images)

## GAN: 2-player Game

- Generator Network:
  - Takes random noise as an input, produces an image
- Discriminator Network:
  - Takes image as an input; outputs a probability score
  - $D(x) = \text{probability of } x \text{ being a real image}$

# Games in ML. Example: GANs

## GAN: 2-player Game

- Generator Network:
  - Takes random noise as an input, produces an image
- Discriminator Network:
  - Takes image as an input; outputs a probability score
  - $D(x)$  = probability of  $x$  being a real image

Loss functions:

- Discriminator loss: classification loss; classify if input  $x$  is a real or fake image
- Generator loss: opposite of discriminator loss
  - Generator gets low loss if discriminator thinks that generated image is “real”

Nash equilibrium:

- $G^*$  – Generator that perfectly learns the training data distribution
- $D^*$  – Discriminator that outputs 0.5 for any input