Machine Learning and Optimization

Lecture 13: Optimization for Neural Networks

CSCI 4140/6960 — Spring 2026

Course website: `cs.rpi.edu/~gittea/.../mlandopt.html`

Learning Objectives

By the end of this lecture, you should be able to:

1. **Identify** the six factors enabling modern deep learning and explain why all are necessary simultaneously
2. **State and prove** GD/SGD convergence to ε-stationary points for L-smooth nonconvex functions, and compare their computational costs
3. **Classify** critical points via the Hessian, distinguish strict from non-strict saddles, and explain why GD escapes strict saddles via spectral instability
4. **Explain** overparameterization, double descent, and why the classical bias–variance tradeoff breaks down for modern NNs

Epistemic Markers

This lecture covers material at different levels of rigor. We use these markers throughout:

Marker	Meaning
✓	**Theorem** — rigorously proved under stated assumptions
◐	**Proven for restricted classes** — not general deep nets
⊙	**Heuristic / empirical** — well-supported, not proved
?	**Open problem** — actively investigated, no consensus

Knowing what we *know* vs. what we *believe* is essential, because **practice has outpaced theory** in NNs.

Lecture Roadmap

History and the Remaining Factors for Success

Nonconvex Optimization for Neural Networks

Why Approximate Stationarity Is Geometrically Meaningful

Why Approximate Stationarity Is Geometrically Meaningful

Overparameterization and the Generalization Puzzle

Summary

A Brief History of Neural Networks

Period	Event
1943–1969	Mathematical neuron models → Perceptron → Minsky & Papert prove limitations; first AI winter
1986–1989	Backpropagation revival; universal approximation theorem; first CNNs (LeNet)
∼1995–2005	Second AI winter: SVMs and kernel methods dominate
2012	**AlexNet wins ImageNet** by a large margin using deep CNN on GPUs; modern deep learning era begins
2017	Transformer architecture (Vaswani et al.); foundation of modern LLMs

A Brief History of Neural Networks

Period	Event
1943–1969	Mathematical neuron models → Perceptron → Minsky & Papert prove limitations; first AI winter
1986–1989	Backpropagation revival; universal approximation theorem; first CNNs (LeNet)
∼1995–2005	Second AI winter: SVMs and kernel methods dominate
2012	**AlexNet wins ImageNet** by a large margin using deep CNN on GPUs; modern deep learning era begins
2017	Transformer architecture (Vaswani et al.); foundation of modern LLMs

Key pattern: Theoretical ideas preceded practical success by *decades*. The gap was closed not by new theory, but by compute, data, and engineering.

Why Now? The Confluence of Factors

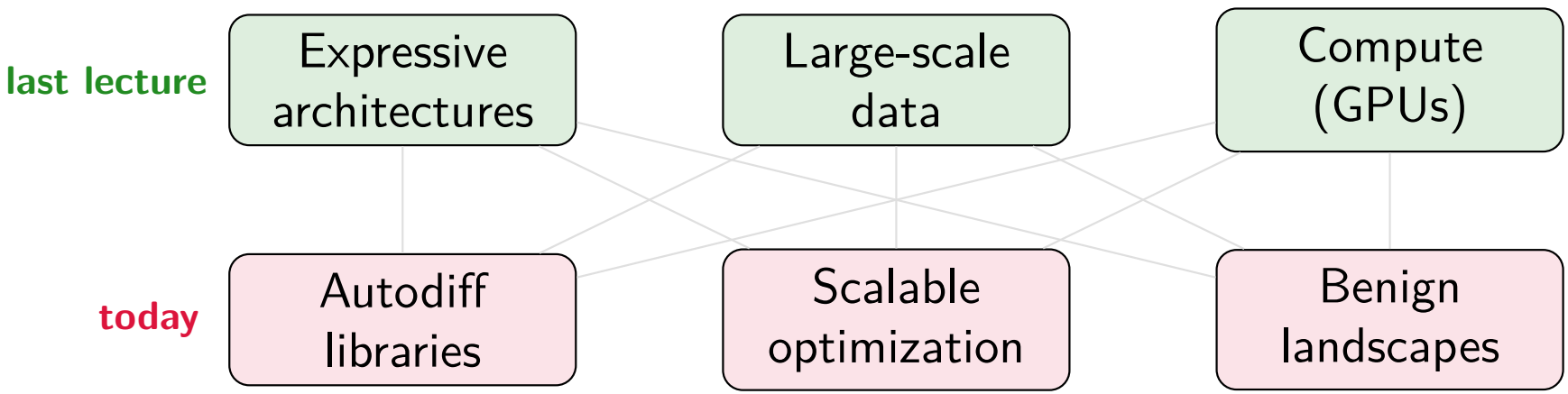
Neural networks were proposed in the 1940s. Backpropagation, CNNs, and universal approximation were all in place by the early 1990s.

Yet NNs only became dominant after $\sim$2012. Why?

Why Now? The Confluence of Factors

Neural networks were proposed in the 1940s. Backpropagation, CNNs, and universal approximation were all in place by the early 1990s.

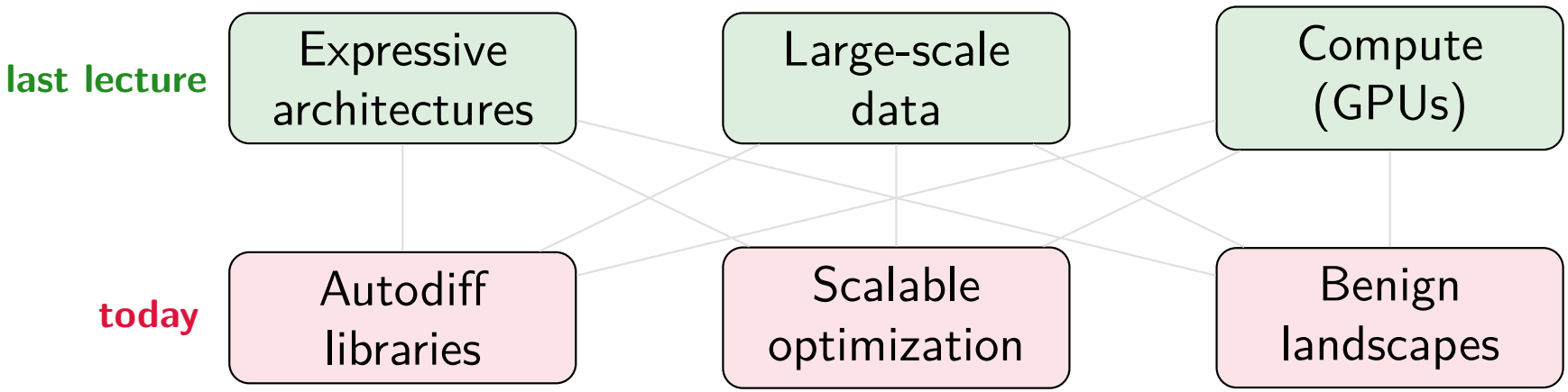
Yet NNs only became dominant after ∼2012. Why?



Why Now? The Confluence of Factors

Neural networks were proposed in the 1940s. Backpropagation, CNNs, and universal approximation were all in place by the early 1990s.

Yet NNs only became dominant after ∼2012. Why?



Remove any of these factors, and the modern NN paradigm collapses. We covered Factors 1–3 last time. Today covers factors 4–6, which are directly about *optimization*.

Factor 4 — Automatic Differentiation Libraries

To train a NN, we need $\nabla_\theta f(\theta)$ for all parameters θ. For architectures with millions of parameters and intricate computational graphs, computing gradients by hand is infeasible.

Factor 4 — Automatic Differentiation Libraries

To train a NN, we need $\nabla_\theta f(\theta)$ for all parameters θ. For architectures with millions of parameters and intricate computational graphs, computing gradients by hand is infeasible.

Automatic differentiation (autodiff) computes exact gradients of arbitrary compositions of differentiable operations via the chain rule.

Library	**Year**	**Key idea**
Theano	2010	One of the earliest; symbolic graphs
TensorFlow	2015	Static computation graphs
PyTorch	2017	Dynamic graphs ("define-by-run"); dominant in research

Factor 4 — Automatic Differentiation Libraries

To train a NN, we need $\nabla_\theta f(\theta)$ for all parameters θ. For architectures with millions of parameters and intricate computational graphs, computing gradients by hand is infeasible.

Automatic differentiation (autodiff) computes exact gradients of arbitrary compositions of differentiable operations via the chain rule.

Library	**Year**	**Key idea**
Theano	2010	One of the earliest; symbolic graphs
TensorFlow	2015	Static computation graphs
PyTorch	2017	Dynamic graphs ("define-by-run"); dominant in research

The core operation is **backpropagation** = reverse-mode autodiff on a DAG. Its cost is O(cost of one forward pass).

We will prove this next lecture.

Factor 5 — Scalable First-Order Optimization

Before $\sim$2010, the standard was **L-BFGS** or conjugate gradient — quasi-Newton methods that do not scale:

- L-BFGS requires full-batch gradients (cost $O(n)$ per step)
- Memory grows with stored gradient pairs
- Mini-batch variants are fragile

Factor 5 — Scalable First-Order Optimization

Before ∼2010, the standard was **L-BFGS** or conjugate gradient — quasi-Newton methods that do not scale:

- L-BFGS requires full-batch gradients (cost $O(n)$ per step)
- Memory grows with stored gradient pairs
- Mini-batch variants are fragile

The shift to **SGD** (championed by LeCun, Bottou, Bengio from the late 1990s) was decisive: $O(1)$-per-iteration cost makes it the right tradeoff for large-scale learning.

Factor 5 — Scalable First-Order Optimization

Before ∼2010, the standard was **L-BFGS** or conjugate gradient — quasi-Newton methods that do not scale:

- L-BFGS requires full-batch gradients (cost $O(n)$ per step)
- Memory grows with stored gradient pairs
- Mini-batch variants are fragile

The shift to **SGD** (championed by LeCun, Bottou, Bengio from the late 1990s) was decisive: $O(1)$-per-iteration cost makes it the right tradeoff for large-scale learning.

Factor 5b — Initialization & normalization: Crucial for *conditioning* the optimization:

- **Xavier/He initialization:** scale weights to preserve gradient variance across layers
- **Batch/Layer normalization:** stabilize training dynamics

This is a *preconditioning* story, just as in the first half of the course.

Factor 6 — Benign Optimization Landscape (Preview)

The theory we develop in this lecture guarantees convergence to **ε-approximate stationary points**. For a general nonconvex function, this is weak: stationary points could be saddle points or terrible local minima.

Factor 6 — Benign Optimization Landscape (Preview)

The theory we develop in this lecture guarantees convergence to **ε-approximate stationary points**. For a general nonconvex function, this is weak: stationary points could be saddle points or terrible local minima.

We need a separate argument that approximate stationarity is *geometrically meaningful* for the problems we care about.

Factor 6 — Benign Optimization Landscape (Preview)

The theory we develop in this lecture guarantees convergence to **ε-approximate stationary points**. For a general nonconvex function, this is weak: stationary points could be saddle points or terrible local minima.

We need a separate argument that approximate stationarity is *geometrically meaningful* for the problems we care about.

Later in this lecture, we will:

- Define strict vs. non-strict saddle points
- Connect saddle escape to **spectral instability** (same eigenvalue analysis from the lecture on accelerated gradient methods!)
- Survey what is rigorously known about NN loss landscapes

The Whole Is Greater Than the Sum of Its Parts

Factor	**Without it...**
Expressive architectures	Generic MLPs cannot encode domain structure; need impractical amounts of data
Large datasets	NNs overfit; no better than classical methods
Compute (GPUs)	Training takes months/years; iteration cycle too slow
Autodiff libraries	Every new architecture requires tedious and error-prone manual gradient derivation
Scalable optimization	L-BFGS/CG cannot handle $n > 10^5$; training doesn't scale
Benign loss landscape	Optimization gets stuck at bad critical points; trained models are useless

The Whole Is Greater Than the Sum of Its Parts

Factor	**Without it...**
Expressive architectures	Generic MLPs cannot encode domain structure; need impractical amounts of data
Large datasets	NNs overfit; no better than classical methods
Compute (GPUs)	Training takes months/years; iteration cycle too slow
Autodiff libraries	Every new architecture requires tedious and error-prone manual gradient derivation
Scalable optimization	L-BFGS/CG cannot handle $n > 10^5$; training doesn't scale
Benign loss landscape	Optimization gets stuck at bad critical points; trained models are useless

The deep learning revolution is the story of all six factors maturing simultaneously. The ideas were there for decades; the conditions to realize them were not.

Shifting Gears

We have identified **six factors** enabling modern deep learning and seen that the theoretical ideas preceded practice by decades.

Shifting Gears

We have identified **six factors** enabling modern deep learning and seen that the theoretical ideas preceded practice by decades.

*Now we shift from **why deep learning works** to **what we can prove about optimizing it**.*

Shifting Gears

We have identified **six factors** enabling modern deep learning and seen that the theoretical ideas preceded practice by decades.

*Now we shift from **why deep learning works** to **what we can prove about optimizing it**.*

The rest of this lecture looks at the relevant optimization theory:

1. **Convergence:** GD/SGD find ε-stationary points (first-order guarantee)
2. **Landscape:** Why stationarity is geometrically meaningful (second-order structure)
3. **Generalization:** Why interpolating solutions generalize (memorization doesn't hurt learning)

The Training Problem (Recap from RERM)

Once the architecture is fixed, learn parameters via:

$$\theta^* = \underset{\substack{W^\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}} \\ b^\ell \in \mathbb{R}^{n_\ell}}}{\operatorname{argmin}} \quad f(\theta)$$

where

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell\big(o^L(x_i),\, y_i\big) + \lambda \sum_{\ell=1}^{L} R_\ell(W^\ell, b^\ell)$$

The Training Problem (Recap from RERM)

Once the architecture is fixed, learn parameters via:

$$\theta^* = \underset{\substack{W^\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}} \\ b^\ell \in \mathbb{R}^{n_\ell}}}{\operatorname{argmin}} \quad f(\theta)$$

where

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell\big(o^L(x_i),\, y_i\big) + \lambda \sum_{\ell=1}^{L} R_\ell(W^\ell, b^\ell)$$

Common regularizer: **weight decay** on W^ℓ:

$$R_\ell(W^\ell, b^\ell) = \|W^\ell\|_F^2 = \sum_{i \in [n_\ell]} \sum_{j \in [n_{\ell-1}]} (W^\ell)_{ij}^2$$

Why This Is Different from Everything We've Done Before

In the first half of the course, every optimization problem was **convex**. We proved:

- Every local minimum is a global minimum
- GD/SGD converge to ε-suboptimal solutions at known rates
- The solution is essentially unique

Why This Is Different from Everything We've Done Before

In the first half of the course, every optimization problem was **convex**. We proved:

- Every local minimum is a global minimum
- GD/SGD converge to ε-suboptimal solutions at known rates
- The solution is essentially unique

The NN training problem is **nonconvex**: $f(\theta)$ has many local minima, saddle points, and plateaus. The composition $W^L\sigma(\cdots\sigma(W^1x))$ introduces *products of parameters*, which destroy convexity.

Why This Is Different from Everything We've Done Before

In the first half of the course, every optimization problem was **convex**. We proved:

- Every local minimum is a global minimum
- GD/SGD converge to ε-suboptimal solutions at known rates
- The solution is essentially unique

The NN training problem is **nonconvex**: $f(\theta)$ has many local minima, saddle points, and plateaus. The composition $W^L\sigma(\cdots\sigma(W^1x))$ introduces *products of parameters*, which destroy convexity.

The convex guarantees **do not apply**:

1. $f(\theta_T) - f^* \leq \varepsilon$ (convergence to global optimum)
2. $\|\theta_T - \theta_*\| \leq \varepsilon$ (convergence to a global minimizer)

Why This Is Different from Everything We've Done Before

In the first half of the course, every optimization problem was **convex**. We proved:

- Every local minimum is a global minimum
- GD/SGD converge to ε-suboptimal solutions at known rates
- The solution is essentially unique

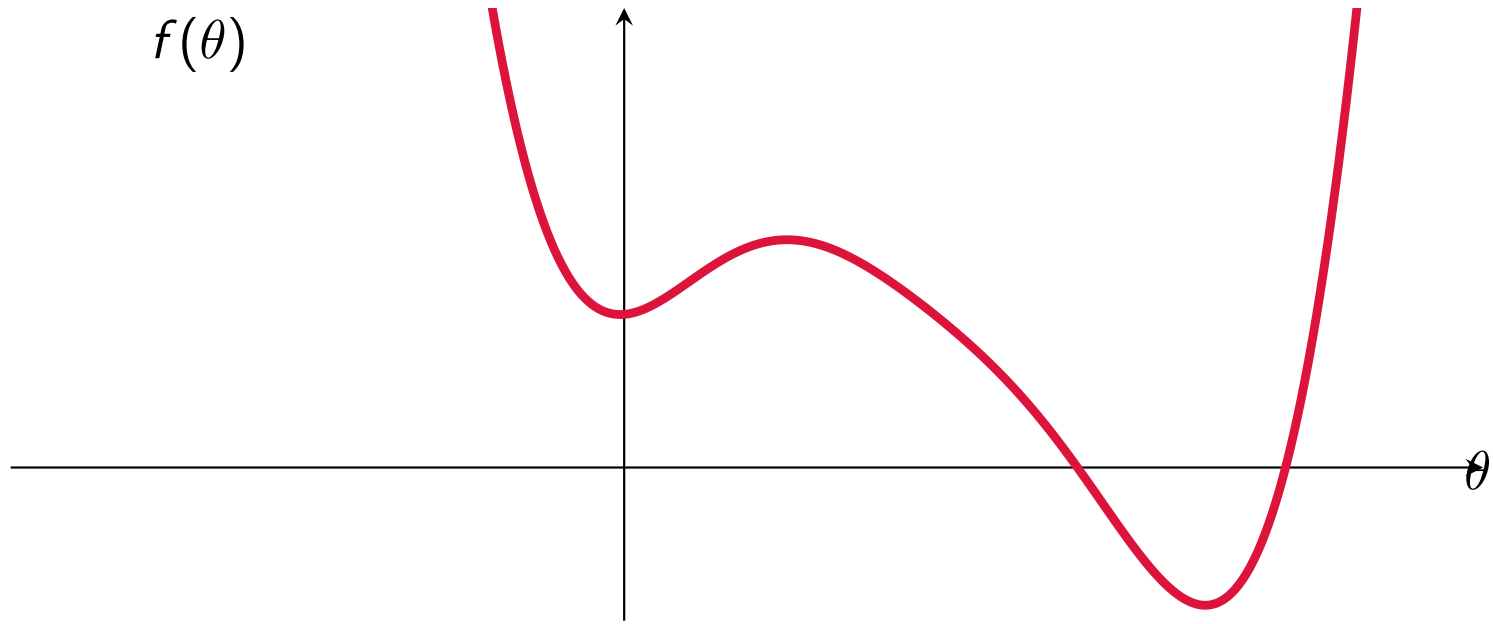
The NN training problem is **nonconvex**: $f(\theta)$ has many local minima, saddle points, and plateaus. The composition $W^L\sigma(\cdots\sigma(W^1x))$ introduces *products of parameters*, which destroy convexity.

The convex guarantees **do not apply**:

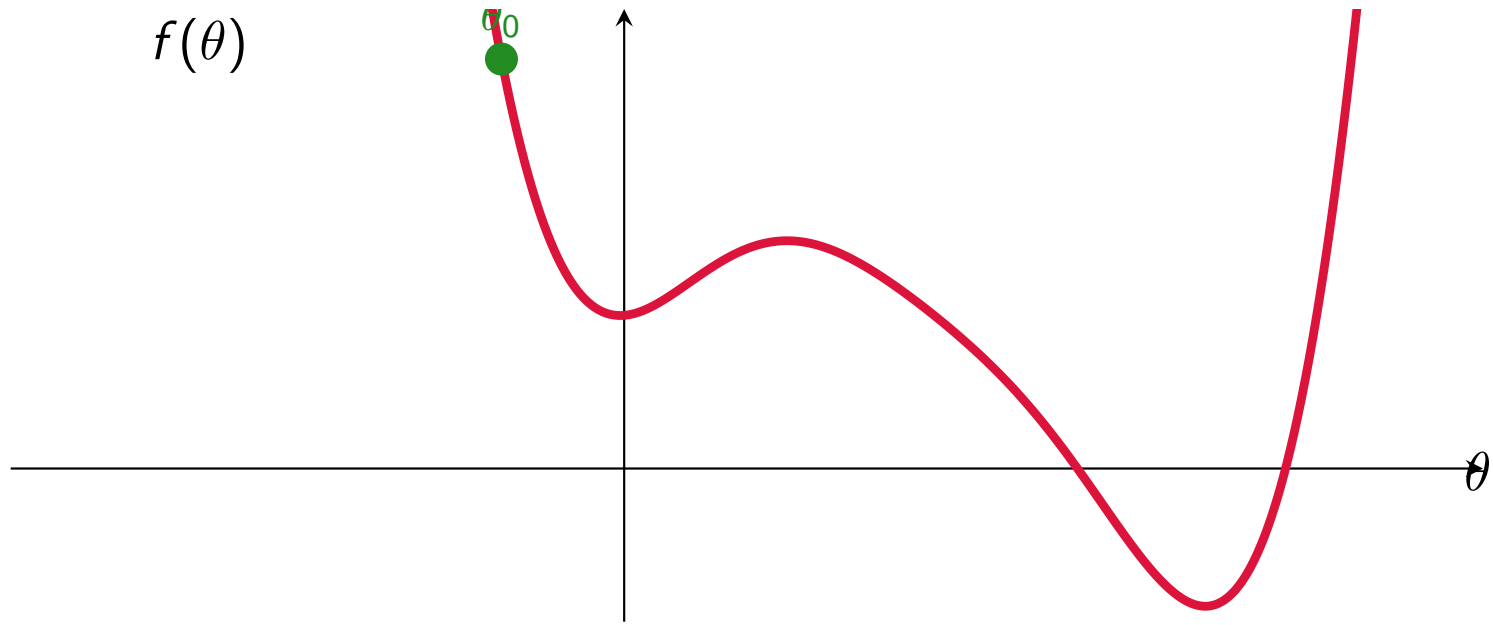
1. $f(\theta_T) - f^* \leq \varepsilon$ (convergence to global optimum)
2. $\|\theta_T - \theta_*\| \leq \varepsilon$ (convergence to a global minimizer)

We need a fundamentally **weaker** notion of success.

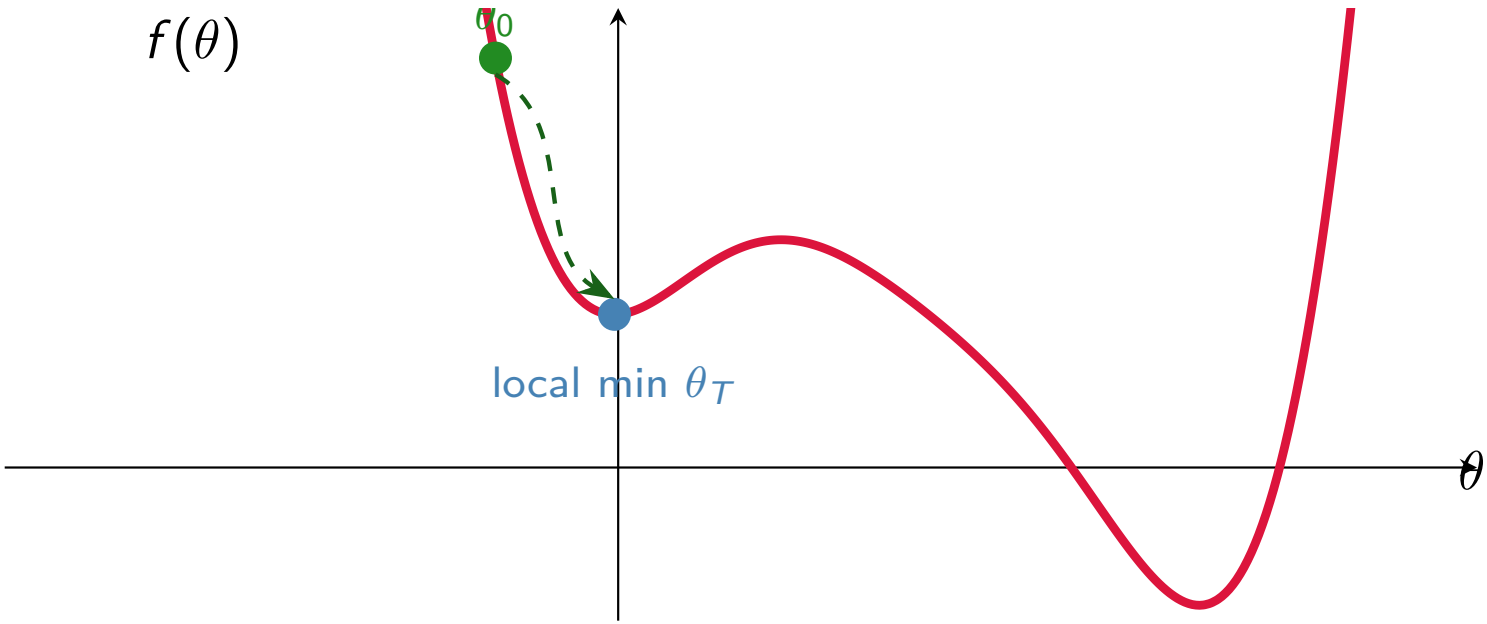
The Nonconvex Landscape



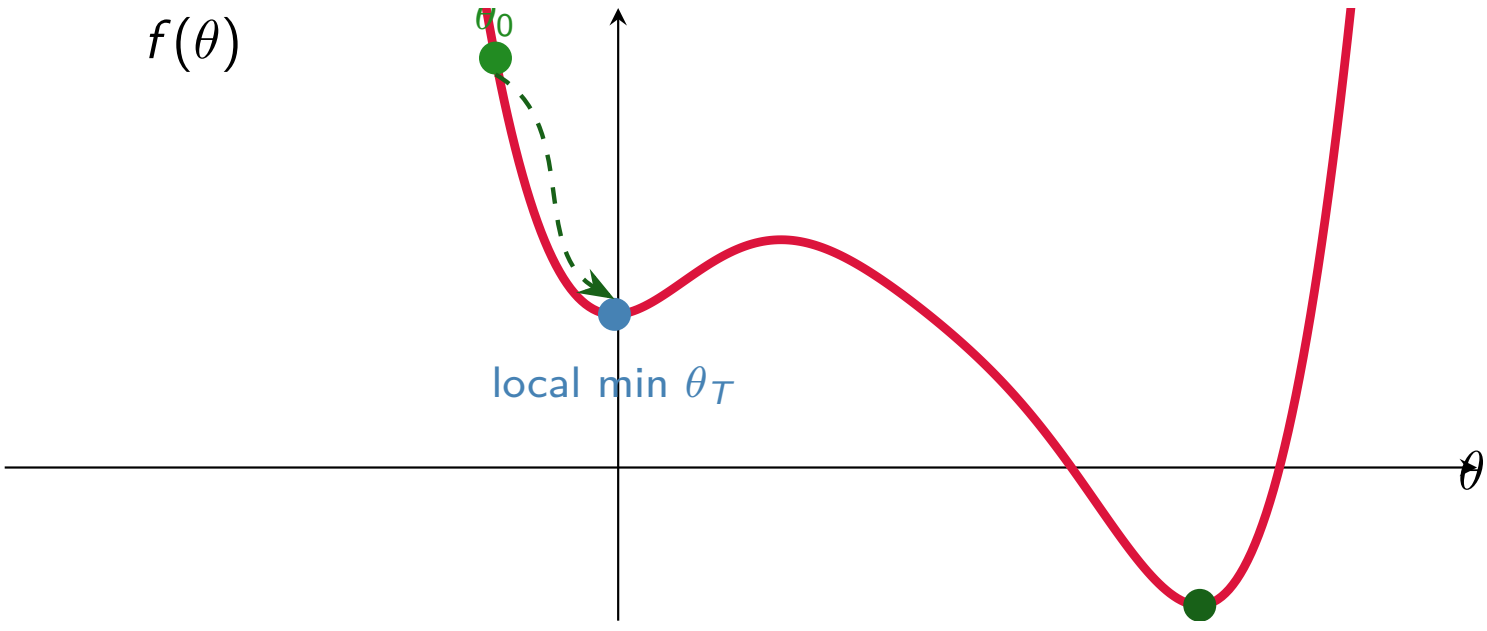
The Nonconvex Landscape



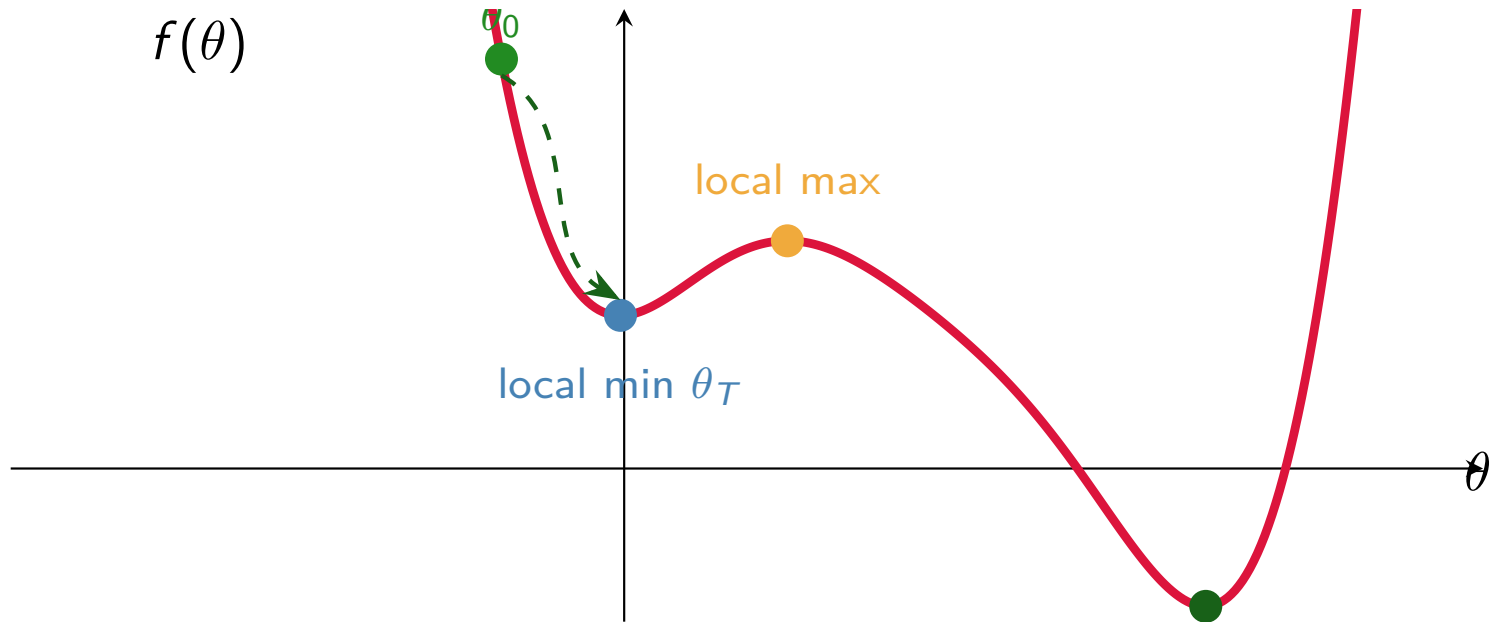
The Nonconvex Landscape



The Nonconvex Landscape



The Nonconvex Landscape



The initial point determines which basin you land in. You may get stuck at saddle points or bad local minima.

What We *Can* Guarantee: Approximate Stationarity

For general nonconvex problems, the best we can guarantee is convergence to ε-**approximate stationary points**:

$$\|\nabla f(\theta)\|_F^2 = \sum_{\ell=1}^{L} \left(\|\nabla_{W^\ell} f(\theta)\|_F^2 + \|\nabla_{b^\ell} f(\theta)\|_2^2 \right) \leq \varepsilon$$

What We *Can* Guarantee: Approximate Stationarity

For general nonconvex problems, the best we can guarantee is convergence to ε-**approximate stationary points**:

$$\|\nabla f(\theta)\|_F^2 = \sum_{\ell=1}^{L} \left(\|\nabla_{W^\ell} f(\theta)\|_F^2 + \|\nabla_{b^\ell} f(\theta)\|_2^2 \right) \leq \varepsilon$$

Comparison with the convex case:

- **Convex:** $\nabla f = 0 \implies$ global minimum. Proving stationarity $\equiv$ proving optimality.
- **Nonconvex:** $\nabla f = 0$ could be a local minimum (good), local maximum (bad), or saddle point (bad). The guarantee is fundamentally weaker.

What We *Can* Guarantee: Approximate Stationarity

For general nonconvex problems, the best we can guarantee is convergence to ε-**approximate stationary points**:

$$\|\nabla f(\theta)\|_F^2 = \sum_{\ell=1}^{L} \left(\|\nabla_{W^\ell} f(\theta)\|_F^2 + \|\nabla_{b^\ell} f(\theta)\|_2^2 \right) \leq \varepsilon$$

Comparison with the convex case:

- **Convex:** $\nabla f = 0 \implies$ global minimum. Proving stationarity $\equiv$ proving optimality.
- **Nonconvex:** $\nabla f = 0$ could be a local minimum (good), local maximum (bad), or saddle point (bad). The guarantee is fundamentally weaker.

So is approximate stationarity geometrically meaningful, or are we proving convergence to useless points? We'll return to this important question shortly.

GD Convergence for L-Smooth Nonconvex Functions ✓

Same smoothness machinery as the first half — descent lemma and telescoping. The key difference: we bound *gradient norms*, not *suboptimality gaps*.

GD Convergence for L-Smooth Nonconvex Functions ✓

Same smoothness machinery as the first half — descent lemma and telescoping. The key difference: we bound *gradient norms*, not *suboptimality gaps*.

Assumptions (no convexity required):

- f is L-smooth (i.e., ∇f is L-Lipschitz)
- f is bounded below: $f^* = \inf_x f(x) > -\infty$

GD Convergence for L-Smooth Nonconvex Functions ✓

Same smoothness machinery as the first half — descent lemma and telescoping. The key difference: we bound *gradient norms*, not *suboptimality gaps*.

Assumptions (no convexity required):

- f is L-smooth (i.e., ∇f is L-Lipschitz)
- f is bounded below: $f^* = \inf_x f(x) > -\infty$

Theorem (GD, nonconvex)

If f is L-smooth and bounded below, GD with stepsize $\alpha = 1/L$ satisfies:

$$\frac{1}{T}\sum_{t=1}^{T} \|\nabla f(x_{t-1})\|_2^2 \leq \frac{2LR}{T}$$

where $R = f(x_0) - f^*$.

GD Convergence for L-Smooth Nonconvex Functions ✓

Same smoothness machinery as the first half — descent lemma and telescoping. The key difference: we bound *gradient norms*, not *suboptimality gaps*.

Assumptions (no convexity required):

- f is L-smooth (i.e., ∇f is L-Lipschitz)
- f is bounded below: $f^* = \inf_x f(x) > -\infty$

Theorem (GD, nonconvex)

If f is L-smooth and bounded below, GD with stepsize $\alpha = 1/L$ satisfies:

$$\frac{1}{T}\sum_{t=1}^{T} \|\nabla f(x_{t-1})\|_2^2 \leq \frac{2LR}{T}$$

where $R = f(x_0) - f^*$.

Interpretation: $\min_{i\in[T]} \|\nabla f(x_{i-1})\|_2^2 \leq \frac{2LR}{T} \leq \varepsilon$ for $T = O(1/\varepsilon)$.

Proof: Per-Step Descent

By L-smoothness gives a quadratic upper bound:

$$f(x_t) \leq f(x_{t-1}) + \langle \nabla f(x_{t-1}), x_t - x_{t-1} \rangle + \frac{L}{2} \|x_t - x_{t-1}\|_2^2$$

Proof: Per-Step Descent

By L-smoothness gives a quadratic upper bound:

$$f(x_t) \leq f(x_{t-1}) + \langle \nabla f(x_{t-1}), x_t - x_{t-1} \rangle + \frac{L}{2} \|x_t - x_{t-1}\|_2^2$$

As in the midterm (and lecture notes back when we discussed the descent lemma), this implies that with $\alpha = 1/L$ we get **per-step descent**:

$$\boxed{\frac{1}{2L} \|\nabla f(x_{t-1})\|_2^2 \leq f(x_{t-1}) - f(x_t)}$$

Proof: Telescoping Sum

Accumulate the descents over $t = 1, \ldots, T$:

$$\frac{1}{2L} \sum_{t=1}^{T} \|\nabla f(x_{t-1})\|_2^2 \leq \sum_{t=1}^{T} \big[f(x_{t-1}) - f(x_t) \big]$$

Proof: Telescoping Sum

Accumulate the descents over $t = 1, \ldots, T$:

$$\frac{1}{2L} \sum_{t=1}^{T} \|\nabla f(x_{t-1})\|_2^2 \leq \sum_{t=1}^{T} \big[f(x_{t-1}) - f(x_t) \big]$$

The right side telescopes:

$$\begin{aligned}
&= \big(f(x_0) - f(x_1) \big) + \big(f(x_1) - f(x_2) \big) + \cdots + \big(f(x_{T-1}) - f(x_T) \big) \\
&= f(x_0) - f(x_T) \\
&\leq f(x_0) - f^*
\end{aligned}$$

Proof: Telescoping Sum

Accumulate the descents over $t = 1, \ldots, T$:

$$\frac{1}{2L} \sum_{t=1}^{T} \|\nabla f(x_{t-1})\|_2^2 \leq \sum_{t=1}^{T} \big[f(x_{t-1}) - f(x_t) \big]$$

The right side telescopes:

$$\begin{aligned}
&= \big(f(x_0) - f(x_1) \big) + \big(f(x_1) - f(x_2) \big) + \cdots + \big(f(x_{T-1}) - f(x_T) \big) \\
&= f(x_0) - f(x_T) \\
&\leq f(x_0) - f^*
\end{aligned}$$

Thus

$$\frac{1}{2L} \sum_{t=1}^{T} \|\nabla f(x_{t-1})\|_2^2 \leq f(x_0) - f^* = R$$

Proof: Telescoping Sum

Accumulate the descents over $t = 1, \ldots, T$:

$$\frac{1}{2L} \sum_{t=1}^{T} \|\nabla f(x_{t-1})\|_2^2 \leq \sum_{t=1}^{T} \big[f(x_{t-1}) - f(x_t) \big]$$

The right side telescopes:

$$\begin{aligned} &= \big(f(x_0) - f(x_1)\big) + \big(f(x_1) - f(x_2)\big) + \cdots + \big(f(x_{T-1}) - f(x_T)\big) \\ &= f(x_0) - f(x_T) \\ &\leq f(x_0) - f^* \end{aligned}$$

Thus

$$\frac{1}{2L} \sum_{t=1}^{T} \|\nabla f(x_{t-1})\|_2^2 \leq f(x_0) - f^* = R$$

Multiply both sides by $\frac{2L}{T}$:

$$\boxed{\frac{1}{T} \sum_{t=1}^{T} \|\nabla f(x_{t-1})\|_2^2 \leq \frac{2LR}{T}} \qquad \square$$

SGD: Constant Stepsize ✓

In NN training, $f(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta)$ is a finite sum. SGD uses a stochastic gradient g_t satisfying:

- $\mathbb{E}[g_t \mid x_t] = \nabla f(x_t)$ (unbiasedness)
- $\mathbb{E}\|g_t - \nabla f(x_t)\|_2^2 \leq \sigma^2$ (bounded variance)

SGD: Constant Stepsize ✓

In NN training, $f(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta)$ is a finite sum. SGD uses a stochastic gradient g_t satisfying:

- $\mathbb{E}[g_t \mid x_t] = \nabla f(x_t)$ (unbiasedness)
- $\mathbb{E}\|g_t - \nabla f(x_t)\|_2^2 \leq \sigma^2$ (bounded variance)

Theorem (SGD, constant stepsize, nonconvex)

With $\alpha = \frac{1}{2L}$ and $R = f(x_0) - f^*$:

$$\mathbb{E}\left(\frac{1}{T}\sum_{t=1}^{T} \|\nabla f(x_{t-1})\|_2^2\right) \leq \frac{2LR}{T} + \sigma^2$$

SGD: Constant Stepsize ✓

In NN training, $f(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta)$ is a finite sum. SGD uses a stochastic gradient g_t satisfying:

- $\mathbb{E}[g_t \mid x_t] = \nabla f(x_t)$ (unbiasedness)
- $\mathbb{E}\|g_t - \nabla f(x_t)\|_2^2 \leq \sigma^2$ (bounded variance)

Theorem (SGD, constant stepsize, nonconvex)

With $\alpha = \frac{1}{2L}$ and $R = f(x_0) - f^*$:

$$\mathbb{E}\left(\frac{1}{T}\sum_{t=1}^{T} \|\nabla f(x_{t-1})\|_2^2\right) \leq \frac{2LR}{T} + \sigma^2$$

The bound has a σ^2 **noise floor**: no matter how long we run, constant-stepsize SGD cannot guarantee to visit points with gradient norm smaller than σ.

Decaying Stepsizes Remove the Noise Floor ✓

To eliminate the noise floor, decrease α_t so the variance term vanishes asymptotically.

Decaying Stepsizes Remove the Noise Floor ✓

To eliminate the noise floor, decrease α_t so the variance term vanishes asymptotically.

Theorem (Nonconvex SGD, decaying stepsize)

Let $R := f(x_0) - f^*$ and choose $\alpha_t = \sqrt{\frac{R}{L\sigma^2 t}}$. If $T \geq 4RL/\sigma^2$, then

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(x_{t-1})\|^2\right] \leq 4\sqrt{\frac{RL\sigma^2}{T}}.$$

Decaying Stepsizes Remove the Noise Floor ✓

To eliminate the noise floor, decrease α_t so the variance term vanishes asymptotically.

Theorem (Nonconvex SGD, decaying stepsize)

Let $R := f(x_0) - f^*$ and choose $\alpha_t = \sqrt{\frac{R}{L\sigma^2 t}}$. If $T \geq 4RL/\sigma^2$, then

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(x_{t-1})\|^2\right] \leq 4\sqrt{\frac{RL\sigma^2}{T}}.$$

Thus $\mathbb{E}\min_{i\in[T]}\|\nabla f(x_t)\|^2 = O\left(\frac{1}{\sqrt{T}}\right)$. To obtain $\mathbb{E}\min_{i\in[T]}\|\nabla f(x_t)\| \leq \varepsilon$, we need

$$\frac{1}{\sqrt{T}} = O(\varepsilon^2) \quad \Longrightarrow \quad T = O(\varepsilon^{-4}).$$

Computational Cost: GD vs. SGD

Convergence rates alone don't tell the full story — we must account for **per-iteration cost**.

	Iterations to ε	**Cost/iteration**	**Total work**
GD	$O(1/\varepsilon^2)$	$O(n)$	$O(n/\varepsilon^2)$
SGD	$O(1/\varepsilon^4)$	$O(1)$	$O(1/\varepsilon^4)$

Computational Cost: GD vs. SGD

Convergence rates alone don't tell the full story — we must account for **per-iteration cost**.

	Iterations to ε	**Cost/iteration**	**Total work**
GD	$O(1/\varepsilon^2)$	$O(n)$	$O(n/\varepsilon^2)$
SGD	$O(1/\varepsilon^4)$	$O(1)$	$O(1/\varepsilon^4)$

SGD wins when $n > 1/\varepsilon^2$, i.e., when the dataset is large relative to the desired accuracy. This is *essentially always the case* for modern deep learning.

Computational Cost: GD vs. SGD

Convergence rates alone don't tell the full story — we must account for **per-iteration cost**.

	Iterations to ε	**Cost/iteration**	**Total work**
GD	$O(1/\varepsilon^2)$	$O(n)$	$O(n/\varepsilon^2)$
SGD	$O(1/\varepsilon^4)$	$O(1)$	$O(1/\varepsilon^4)$

SGD wins when $n > 1/\varepsilon^2$, i.e., when the dataset is large relative to the desired accuracy. This is *essentially always the case* for modern deep learning.

Caveat (Bottou & Bousquet, 2008): In practice, we rarely optimize to a target ε. We optimize until the *statistical* estimation error dominates — the **statistical error floor**. The comparison is about *scaling behavior*: SGD scales better with n.

Practical Stepsize Schedules

The theory prescribes $\alpha_t = c/\sqrt{T}$ (requires knowing T in advance) or $\alpha_t = c/\sqrt{t}$. In practice:

Practical Stepsize Schedules

The theory prescribes $\alpha_t = c/\sqrt{T}$ (requires knowing T in advance) or $\alpha_t = c/\sqrt{t}$. In practice:

- **Cosine annealing** (Loshchilov & Hutter, 2017): smooth decay following a cosine curve
- **Linear warmup + decay:** start small, increase, then decrease
- **Adaptive methods** (Adam, AdaGrad): per-coordinate effective stepsizes that implicitly decay

Practical Stepsize Schedules

The theory prescribes $\alpha_t = c/\sqrt{T}$ (requires knowing T in advance) or $\alpha_t = c/\sqrt{t}$. In practice:

- **Cosine annealing** (Loshchilov & Hutter, 2017): smooth decay following a cosine curve
- **Linear warmup + decay:** start small, increase, then decrease
- **Adaptive methods** (Adam, AdaGrad): per-coordinate effective stepsizes that implicitly decay

All are refinements of the same principle: *start with larger steps for exploration, shrink steps for convergence*.

Practical Stepsize Schedules

The theory prescribes $\alpha_t = c/\sqrt{T}$ (requires knowing T in advance) or $\alpha_t = c/\sqrt{t}$. In practice:

- **Cosine annealing** (Loshchilov & Hutter, 2017): smooth decay following a cosine curve
- **Linear warmup + decay:** start small, increase, then decrease
- **Adaptive methods** (Adam, AdaGrad): per-coordinate effective stepsizes that implicitly decay

All are refinements of the same principle: *start with larger steps for exploration, shrink steps for convergence*.

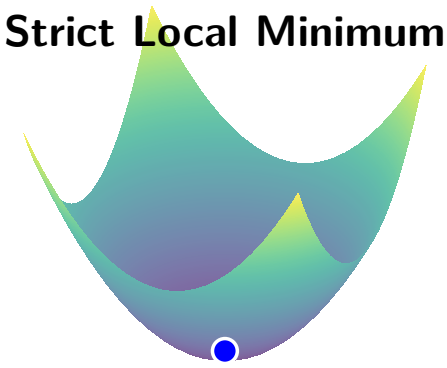
By AlexNet (2012), **SGD with momentum** was the default. Adam, learning rate schedules, and gradient clipping are refinements of this first-order stochastic paradigm.

How meaningful is ϵ-stationarity?

The Hessian $\nabla^2 f(x^*)$ classifies critical points where $\nabla f(x^*) = 0$:

How meaningful is ϵ-stationarity?

The Hessian $\nabla^2 f(x^*)$ classifies critical points where $\nabla f(x^*) = 0$:

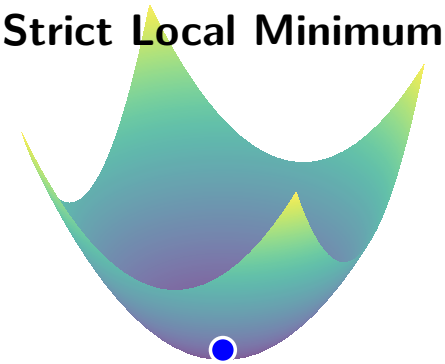


$\nabla^2 f \succ 0$

How meaningful is ϵ-stationarity?

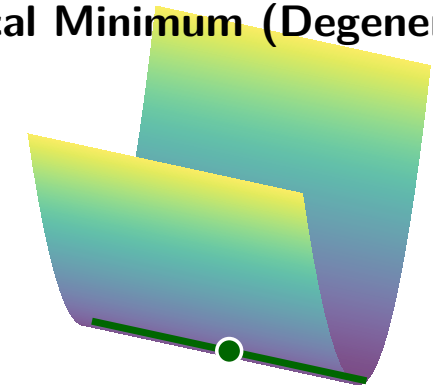
The Hessian $\nabla^2 f(x^*)$ classifies critical points where $\nabla f(x^*) = 0$:

Strict Local Minimum



$\nabla^2 f \succ 0$

Local Minimum (Degenerate)

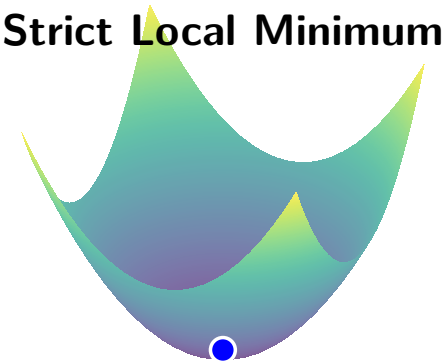


$\nabla^2 f \succeq 0$

How meaningful is ϵ-stationarity?

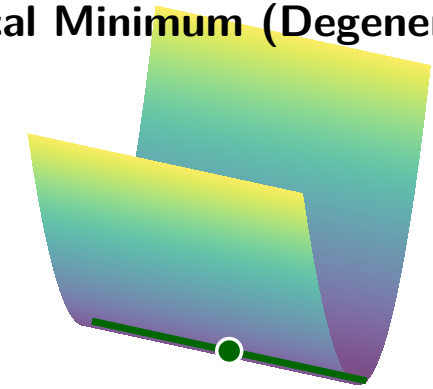
The Hessian $\nabla^2 f(x^*)$ classifies critical points where $\nabla f(x^*) = 0$:

Strict Local Minimum



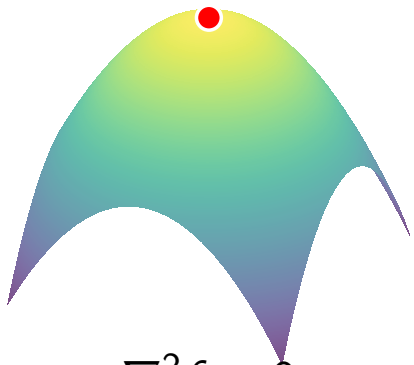
$\nabla^2 f \succ 0$

Local Minimum (Degenerate)



$\nabla^2 f \succeq 0$

Strict Local Maximum

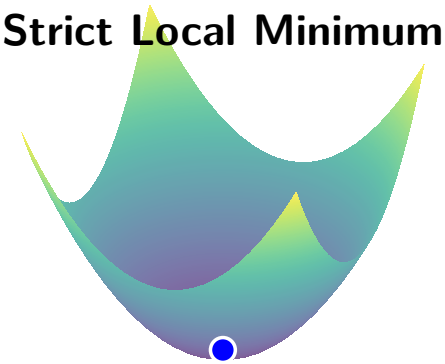


$\nabla^2 f \prec 0$

How meaningful is ϵ-stationarity?

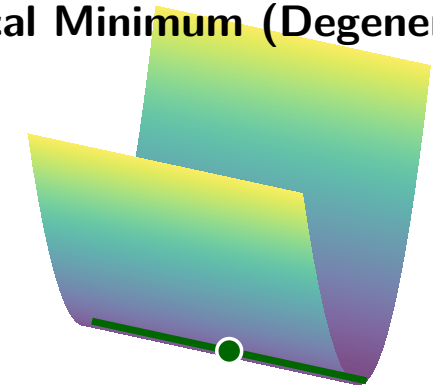
The Hessian $\nabla^2 f(x^*)$ classifies critical points where $\nabla f(x^*) = 0$:

Strict Local Minimum



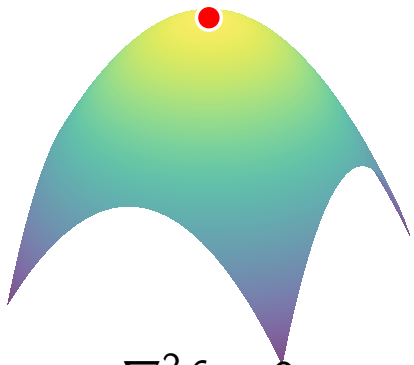
$\nabla^2 f \succ 0$

Local Minimum (Degenerate)



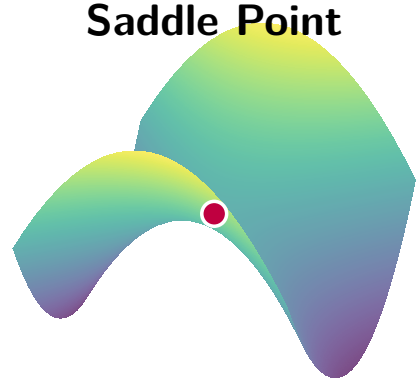
$\nabla^2 f \succeq 0$

Strict Local Maximum



$\nabla^2 f \prec 0$

Saddle Point



Indefinite

How meaningful is ϵ-stationarity?

Recall: at a critical point x^* with $\nabla f(x^*) = 0$, the Hessian gives a *second-order test*:

Hessian condition	**Conclusion**
$\nabla^2 f(x^*) \succ 0$ (positive definite)	Strict local minimum
$\nabla^2 f(x^*) \prec 0$ (negative definite)	Strict local maximum
$\nabla^2 f(x^*)$ indefinite	Saddle point
$\nabla^2 f(x^*) \succeq 0$ but not $\succ 0$	Inconclusive: may be min or saddle
$\nabla^2 f(x^*) \preceq 0$ but not $\prec 0$	Inconclusive: may be max or saddle

In the convex case, $\nabla^2 f \succeq 0$ everywhere, so the last three cases never arose.

How meaningful is ϵ-stationarity?

Recall: at a critical point x^* with $\nabla f(x^*) = 0$, the Hessian gives a *second-order test*:

Hessian condition	**Conclusion**
$\nabla^2 f(x^*) \succ 0$ (positive definite)	Strict local minimum
$\nabla^2 f(x^*) \prec 0$ (negative definite)	Strict local maximum
$\nabla^2 f(x^*)$ indefinite	Saddle point
$\nabla^2 f(x^*) \succeq 0$ but not $\succ 0$	Inconclusive: may be min or saddle
$\nabla^2 f(x^*) \preceq 0$ but not $\prec 0$	Inconclusive: may be max or saddle

In the convex case, $\nabla^2 f \succeq 0$ everywhere, so the last three cases never arose.

Local maxima are not a concern: the descent lemma guarantees $f(x_t) \leq f(x_{t-1})$, so GD cannot converge to a local maximum.

How meaningful is ϵ-stationarity?

Recall: at a critical point x^* with $\nabla f(x^*) = 0$, the Hessian gives a *second-order test*:

Hessian condition	**Conclusion**
$\nabla^2 f(x^*) \succ 0$ (positive definite)	Strict local minimum
$\nabla^2 f(x^*) \prec 0$ (negative definite)	Strict local maximum
$\nabla^2 f(x^*)$ indefinite	Saddle point
$\nabla^2 f(x^*) \succeq 0$ but not $\succ 0$	Inconclusive: may be min or saddle
$\nabla^2 f(x^*) \preceq 0$ but not $\prec 0$	Inconclusive: may be max or saddle

In the convex case, $\nabla^2 f \succeq 0$ everywhere, so the last three cases never arose.

Local maxima are not a concern: the descent lemma guarantees $f(x_t) \leq f(x_{t-1})$, so GD cannot converge to a local maximum.

The real concern is saddle points. In high dimensions, saddle points can vastly outnumber local minima.

Strict vs. Non-Strict Saddles

Definition (Strict Saddle)

A critical point x^* is a *strict saddle* if $\nabla f(x^*) = 0$ and

$$\lambda_{\min}(\nabla^2 f(x^*)) < 0.$$

Strict vs. Non-Strict Saddles

Definition (Strict Saddle)

A critical point x^* is a *strict saddle* if $\nabla f(x^*) = 0$ and

$$\lambda_{\min}(\nabla^2 f(x^*)) < 0.$$

There exists a direction v (eigenvector for $\lambda_{\min}$) along which f curves *downward*:

$$f(x^* + \epsilon v) \approx f(x^*) + \tfrac{1}{2}\epsilon^2 \lambda_{\min} < f(x^*)$$

So x^* is not a local minimum, and there is a **detectable escape direction**.

Strict vs. Non-Strict Saddles

Definition (Strict Saddle)

A critical point x^* is a *strict saddle* if $\nabla f(x^*) = 0$ and

$$\lambda_{\min}(\nabla^2 f(x^*)) < 0.$$

There exists a direction v (eigenvector for $\lambda_{\min}$) along which f curves *downward*:

$$f(x^* + \epsilon v) \approx f(x^*) + \tfrac{1}{2}\epsilon^2 \lambda_{\min} < f(x^*)$$

So x^* is not a local minimum, and there is a **detectable escape direction**.

Definition (Non-Strict / Degenerate Saddle)

A critical point with $\nabla^2 f(x^*) \succeq 0$ (PSD) that is *not* a local minimum. The saddle structure is visible only in higher-order terms.

Strict vs. Non-Strict Saddles

Definition (Strict Saddle)

A critical point x^* is a *strict saddle* if $\nabla f(x^*) = 0$ and

$$\lambda_{\min}(\nabla^2 f(x^*)) < 0.$$

There exists a direction v (eigenvector for $\lambda_{\min}$) along which f curves *downward*:

$$f(x^* + \epsilon v) \approx f(x^*) + \tfrac{1}{2}\epsilon^2 \lambda_{\min} < f(x^*)$$

So x^* is not a local minimum, and there is a **detectable escape direction**.

Definition (Non-Strict / Degenerate Saddle)

A critical point with $\nabla^2 f(x^*) \succeq 0$ (PSD) that is *not* a local minimum. The saddle structure is visible only in higher-order terms.

Strict Saddles Are Unstable Fixed Points ✓

This connects directly to spectral analysis from the first half.

Strict Saddles Are Unstable Fixed Points ✓

This connects directly to spectral analysis from the first half.

Near a strict saddle x^*, the GD update $x_{t+1} = x_t - \alpha \nabla f(x_t)$ linearizes to:

$$e_{t+1} = x_{t+1} - x^* \approx (I - \alpha \underbrace{\nabla^2 f(x^*)}_{H}) e_t$$

Strict Saddles Are Unstable Fixed Points ✓

This connects directly to spectral analysis from the first half.

Near a strict saddle x^*, the GD update $x_{t+1} = x_t - \alpha \nabla f(x_t)$ linearizes to:

$$e_{t+1} = x_{t+1} - x^* \approx (I - \alpha \underbrace{\nabla^2 f(x^*)}_{H}) e_t$$

Eigenbasis decomposition: If $Hv_j = \lambda_j v_j$ and $e_0 = \sum_j c_j v_j$, then $e_t = \sum_{j=1}^{d} c_j (1 - \alpha\lambda_j)^t v_j$.

Strict Saddles Are Unstable Fixed Points ✓

This connects directly to spectral analysis from the first half.

Near a strict saddle x^*, the GD update $x_{t+1} = x_t - \alpha \nabla f(x_t)$ linearizes to:

$$e_{t+1} = x_{t+1} - x^* \approx (I - \alpha \underbrace{\nabla^2 f(x^*)}_{H}) e_t$$

Eigenbasis decomposition: If $Hv_j = \lambda_j v_j$ and $e_0 = \sum_j c_j v_j$, then $e_t = \sum_{j=1}^{d} c_j (1 - \alpha\lambda_j)^t v_j$.

Convex case: All $\lambda_j > 0$.
$|1 - \alpha\lambda_j| < 1$
$\Rightarrow$ all components **decay**

Strict saddle: Some $\lambda_j < 0$.
$|1 - \alpha\lambda_j| > 1$
$\Rightarrow$ component along v_j **grows exponentially**

Strict Saddles Are Unstable Fixed Points ✓

This connects directly to spectral analysis from the first half.

Near a strict saddle x^*, the GD update $x_{t+1} = x_t - \alpha \nabla f(x_t)$ linearizes to:

$$e_{t+1} = x_{t+1} - x^* \approx (I - \alpha \underbrace{\nabla^2 f(x^*)}_{H}) e_t$$

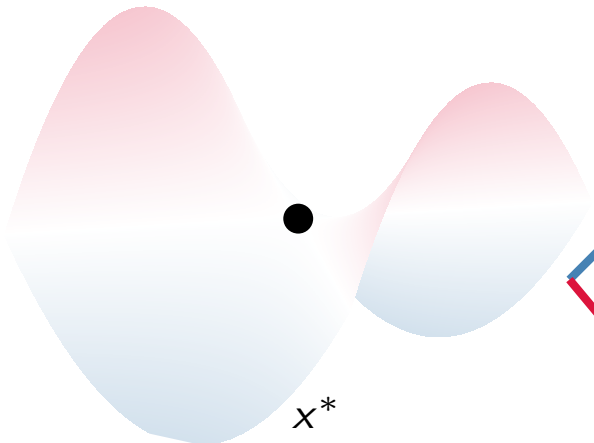
Eigenbasis decomposition: If $Hv_j = \lambda_j v_j$ and $e_0 = \sum_j c_j v_j$, then $e_t = \sum_{j=1}^{d} c_j (1 - \alpha\lambda_j)^t v_j$.

Convex case: All $\lambda_j > 0$.
$|1 - \alpha\lambda_j| < 1$
$\Rightarrow$ all components **decay**

Strict saddle: Some $\lambda_j < 0$.
$|1 - \alpha\lambda_j| > 1$
$\Rightarrow$ component along v_j **grows exponentially**

Same spectral analysis, opposite conclusion: $\rho(I - \alpha H) > 1$ means the saddle is an **unstable fixed point** of the GD dynamical system.

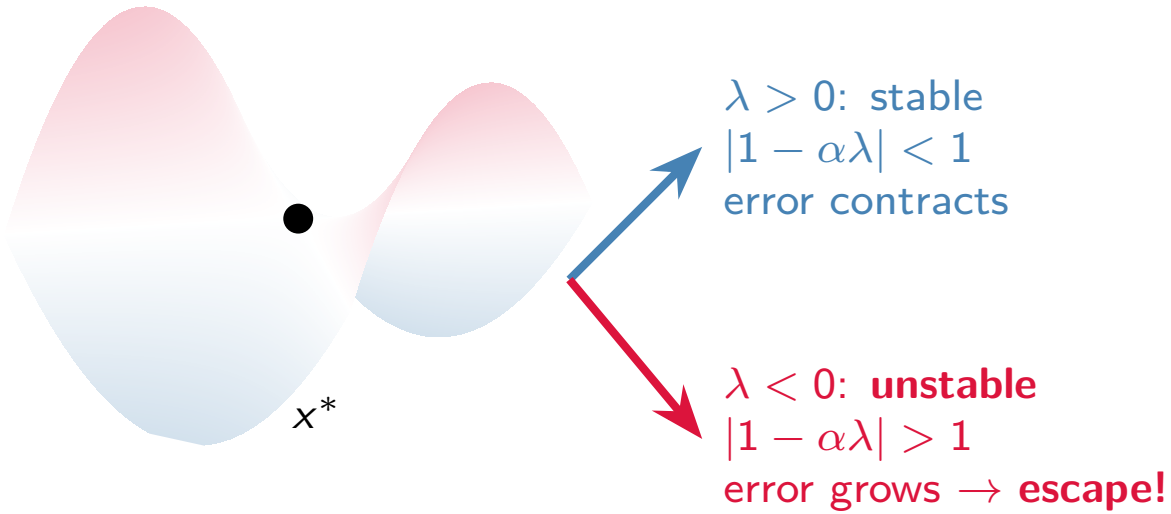
Spectral Instability — Visualization



$\lambda > 0$: stable
$|1 - \alpha\lambda| < 1$
error contracts

$\lambda < 0$: **unstable**
$|1 - \alpha\lambda| > 1$
error grows $\rightarrow$ **escape!**

Spectral Instability — Visualization



GD on a convex quadratic converges because $\rho(I - \alpha H) < 1$. GD near a strict saddle *diverges* (escapes) because $\rho(I - \alpha H) > 1$.

Formal Escape Guarantees

The spectral instability argument shows GD *wants* to escape strict saddles. The formal results:

Formal Escape Guarantees

The spectral instability argument shows GD *wants* to escape strict saddles. The formal results:

Deterministic GD ✓ (Lee et al., 2016): GD converges to a strict saddle only if initialized on the *stable manifold* — a set of dimension $< d$, hence **measure zero**. Under random initialization, GD avoids strict saddles *almost surely*.

Formal Escape Guarantees

The spectral instability argument shows GD *wants* to escape strict saddles. The formal results:

Deterministic GD ✓ (Lee et al., 2016): GD converges to a strict saddle only if initialized on the *stable manifold* — a set of dimension $< d$, hence **measure zero**. Under random initialization, GD avoids strict saddles *almost surely*.

Perturbed GD ✓ (Jin et al., 2017): Adding small isotropic noise, the algorithm escapes all strict saddles in **polynomial time** (poly in d, L, $1/\varepsilon$).

Formal Escape Guarantees

The spectral instability argument shows GD *wants* to escape strict saddles. The formal results:

Deterministic GD ✓ (Lee et al., 2016): GD converges to a strict saddle only if initialized on the *stable manifold* — a set of dimension $< d$, hence **measure zero**. Under random initialization, GD avoids strict saddles *almost surely*.

Perturbed GD ✓ (Jin et al., 2017): Adding small isotropic noise, the algorithm escapes all strict saddles in **polynomial time** (poly in d, L, $1/\varepsilon$).

SGD ◐: Mini-batch noise plays a similar role to explicit perturbation, but with a key difference: SGD noise is *not isotropic*. It depends on the loss landscape geometry at the current iterate. Jin et al.'s guarantees don't directly apply.

Formal Escape Guarantees

The spectral instability argument shows GD *wants* to escape strict saddles. The formal results:

Deterministic GD ✓ (Lee et al., 2016): GD converges to a strict saddle only if initialized on the *stable manifold* — a set of dimension $< d$, hence **measure zero**. Under random initialization, GD avoids strict saddles *almost surely*.

Perturbed GD ✓ (Jin et al., 2017): Adding small isotropic noise, the algorithm escapes all strict saddles in **polynomial time** (poly in d, L, $1/\varepsilon$).

SGD ◐: Mini-batch noise plays a similar role to explicit perturbation, but with a key difference: SGD noise is *not isotropic*. It depends on the loss landscape geometry at the current iterate. Jin et al.'s guarantees don't directly apply.

In practice, SGD escapes strict saddles reliably and tends toward flatter minima that generalize better.

Why Non-Strict Saddles Are Harder

At a non-strict (degenerate) saddle, $\nabla^2 f(x^*) \succeq 0$ and $\lambda_{\min} = 0$.

Why Non-Strict Saddles Are Harder

At a non-strict (degenerate) saddle, $\nabla^2 f(x^*) \succeq 0$ and $\lambda_{\min} = 0$.

The spectral instability argument **breaks down**: the eigenvalue of the iteration matrix is

$$1 - \alpha \cdot 0 = 1$$

The linearized dynamics are **neutral** in the flat direction — neither escaping nor converging.

Why Non-Strict Saddles Are Harder

At a non-strict (degenerate) saddle, $\nabla^2 f(x^*) \succeq 0$ and $\lambda_{\min} = 0$.

The spectral instability argument **breaks down**: the eigenvalue of the iteration matrix is

$$1 - \alpha \cdot 0 = 1$$

The linearized dynamics are **neutral** in the flat direction — neither escaping nor converging.

Escape depends on higher-order terms (third derivatives and beyond) and can be *arbitrarily slow*. No general polynomial-time escape guarantees exist.

Why Non-Strict Saddles Are Harder

At a non-strict (degenerate) saddle, $\nabla^2 f(x^*) \succeq 0$ and $\lambda_{\min} = 0$.

The spectral instability argument **breaks down**: the eigenvalue of the iteration matrix is

$$1 - \alpha \cdot 0 = 1$$

The linearized dynamics are **neutral** in the flat direction — neither escaping nor converging.

Escape depends on higher-order terms (third derivatives and beyond) and can be *arbitrarily slow*. No general polynomial-time escape guarantees exist.

Important for practice: Do NN loss landscapes have non-strict saddles, or are all their saddles strict?

Landscape Results — What We Know Rigorously ◐

The strongest results apply to two important model classes:

Landscape Results — What We Know Rigorously ◐

The strongest results apply to two important model classes:

Deep linear networks. Model: $f_\theta(x) = W_L \cdots W_1 x$. Linear in x but **nonconvex in** $(W_1, \ldots, W_L)$ due to matrix products. A theoretical laboratory for depth-induced nonconvexity.

✓ **Result** (Kawaguchi, 2016): Every local minimum is *global*, and every saddle is *strict*.

Landscape Results — What We Know Rigorously ◐

The strongest results apply to two important model classes:

Deep linear networks. Model: $f_\theta(x) = W_L \cdots W_1 x$. Linear in x but **nonconvex in** $(W_1, \ldots, W_L)$ due to matrix products. A theoretical laboratory for depth-induced nonconvexity.

✓ **Result** (Kawaguchi, 2016): Every local minimum is *global*, and every saddle is *strict*.

Matrix sensing. Recover low-rank M from measurements $y_i = \langle A_i, M \rangle$ by factoring $M = UV^\top$ and minimizing $\sum_i (\langle A_i, UV^\top \rangle - y_i)^2$. Nonconvex in (U, V).

✓ **Result** (Bhojanapalli, Neyshabur & Srebro, 2016): No spurious local minima; all saddles strict.

Landscape Results — What We Know Rigorously ◐

The strongest results apply to two important model classes:

Deep linear networks. Model: $f_\theta(x) = W_L \cdots W_1 x$. Linear in x but **nonconvex in** $(W_1, \ldots, W_L)$ due to matrix products. A theoretical laboratory for depth-induced nonconvexity.

✓ **Result** (Kawaguchi, 2016): Every local minimum is *global*, and every saddle is *strict*.

Matrix sensing. Recover low-rank M from measurements $y_i = \langle A_i, M \rangle$ by factoring $M = UV^\top$ and minimizing $\sum_i (\langle A_i, UV^\top \rangle - y_i)^2$. Nonconvex in (U, V).

✓ **Result** (Bhojanapalli, Neyshabur & Srebro, 2016): No spurious local minima; all saddles strict.

Why these matter: They show that product-of-parameters nonconvexity does *not automatically* create bad landscapes. Problem structure (squared loss, overparameterization) can ensure all critical points are global minima or strict saddles.

What the Landscape Results Do *Not* Cover

- Deep networks with **nonlinear activations** (e.g., ReLU). The Hessian doesn't even exist at activation switching points.
- **Arbitrary loss functions** (results above use squared loss)
- Networks near the **interpolation threshold** ($p \approx n$), where overparameterization is absent

What the Landscape Results Do *Not* Cover

- Deep networks with **nonlinear activations** (e.g., ReLU). The Hessian doesn't even exist at activation switching points.
- **Arbitrary loss functions** (results above use squared loss)
- Networks near the **interpolation threshold** ($p \approx n$), where overparameterization is absent

Honest summary:

- ◐ For deep linear nets and matrix sensing, ε-stationarity $\implies$ ε-global optimality.
- **?** For general deep nonlinear networks, this is a well-supported *empirical observation* but **not a theorem**.

Putting It Together — Why ε-Stationarity Suffices

The argument has three legs, each with different epistemic status:

Step	Claim	Status
1	GD/SGD reach ε-stationary points in $O(1/\varepsilon)$ / $O(1/\varepsilon^2)$	✓ Theorem
2	Strict saddles are escaped via spectral instability	✓ (perturbed GD); ◐ (SGD)
3	In overparameterized NNs, local minima are nearly global	◐ deep linear; ⊙ general

Putting It Together — Why ε-Stationarity Suffices

The argument has three legs, each with different epistemic status:

Step	Claim	Status
1	GD/SGD reach ε-stationary points in $O(1/\varepsilon)$ / $O(1/\varepsilon^2)$	✓ Theorem
2	Strict saddles are escaped via spectral instability	✓ (perturbed GD); ◐ (SGD)
3	In overparameterized NNs, local minima are nearly global	◐ deep linear; ⊙ general

Big picture: Deep learning works because (1) the landscape eliminates bad critical points, and (2) SGD's stochastic dynamics are spectrally unstable at remaining saddles.

Putting It Together — Why ε-Stationarity Suffices

The argument has three legs, each with different epistemic status:

Step	Claim	Status
1	GD/SGD reach ε-stationary points in $O(1/\varepsilon)$ / $O(1/\varepsilon^2)$	✓ Theorem
2	Strict saddles are escaped via spectral instability	✓ (perturbed GD); ◐ (SGD)
3	In overparameterized NNs, local minima are nearly global	◐ deep linear; ⊙ general

Big picture: Deep learning works because (1) the landscape eliminates bad critical points, and (2) SGD's stochastic dynamics are spectrally unstable at remaining saddles.

Caveat: This rests on overparameterization, appropriate losses, and good initialization. It is *not inevitable*.

What Convergence Does and Does Not Mean

A potential source of confusion, so let us be precise:

Statement	**Status**	**What it says**

What Convergence Does and Does Not Mean

A potential source of confusion, so let us be precise:

Statement	**Status**	**What it says**
$\|\nabla f(\theta_T)\|^2 \leq \varepsilon$	✓ GD/SGD	Gradient is small; first-order necessary condition only

What Convergence Does and Does Not Mean

A potential source of confusion, so let us be precise:

Statement	**Status**	**What it says**
$\|\nabla f(\theta_T)\|^2 \leq \varepsilon$	✓ GD/SGD	Gradient is small; first-order necessary condition only
θ_T is a local minimum	Requires 2nd order	Need $\nabla^2 f(\theta_T) \succeq 0$ in addition

What Convergence Does and Does Not Mean

A potential source of confusion, so let us be precise:

Statement	**Status**	**What it says**
$\|\nabla f(\theta_T)\|^2 \leq \varepsilon$	✓ GD/SGD	Gradient is small; first-order necessary condition only
θ_T is a local minimum	Requires 2nd order	Need $\nabla^2 f(\theta_T) \succeq 0$ in addition
θ_T is a good local minimum	◐ Restricted	Landscape-dependent; need no bad local minima

What Convergence Does and Does Not Mean

A potential source of confusion, so let us be precise:

Statement	**Status**	**What it says**
$\|\nabla f(\theta_T)\|^2 \leq \varepsilon$	✓ GD/SGD	Gradient is small; first-order necessary condition only
θ_T is a local minimum	Requires 2nd order	Need $\nabla^2 f(\theta_T) \succeq 0$ in addition
θ_T is a good local minimum	◐ Restricted	Landscape-dependent; need no bad local minima
θ_T generalizes well	⊙ Empirical	Not implied by optimization alone; depends on SGD implicit bias

What Convergence Does and Does Not Mean

A potential source of confusion, so let us be precise:

Statement	**Status**	**What it says**
$\|\nabla f(\theta_T)\|^2 \leq \varepsilon$	✓ GD/SGD	Gradient is small; first-order necessary condition only
θ_T is a local minimum	Requires 2nd order	Need $\nabla^2 f(\theta_T) \succeq 0$ in addition
θ_T is a good local minimum	◐ Restricted	Landscape-dependent; need no bad local minima
θ_T generalizes well	⊙ Empirical	Not implied by optimization alone; depends on SGD implicit bias

Each row requires a *strictly stronger* argument than the one above. We just covered the first three rows.

From Optimization to Generalization

Optimization finds stationary points.

*But why do these solutions **generalize**?*

From Optimization to Generalization

Optimization finds stationary points.

*But why do these solutions **generalize**?*

We have shown:

1. GD/SGD converge to ε-stationary points ✓
2. Strict saddles are escaped ✓
3. For certain problem classes, all local minima are global ◐

From Optimization to Generalization

Optimization finds stationary points.

*But why do these solutions **generalize**?*

We have shown:

1. GD/SGD converge to ε-stationary points ✓
2. Strict saddles are escaped ✓
3. For certain problem classes, all local minima are global ◐

But even a global minimum of the *training* loss could generalize badly. Classical theory says overparameterized models that memorize the data *should* fail on new data.

From Optimization to Generalization

Optimization finds stationary points.

*But why do these solutions **generalize**?*

We have shown:

1. GD/SGD converge to ε-stationary points ✓
2. Strict saddles are escaped ✓
3. For certain problem classes, all local minima are global ◐

But even a global minimum of the *training* loss could generalize badly. Classical theory says overparameterized models that memorize the data *should* fail on new data.

The puzzle: Modern NNs violate this prediction. Understanding why requires rethinking the bias–variance tradeoff.

What Is Overparameterization?

In the first half of the course, our models had $p \ll n$ (far fewer parameters than data points). For neural networks, the situation is **reversed**.

What Is Overparameterization?

In the first half of the course, our models had $p \ll n$ (far fewer parameters than data points). For neural networks, the situation is **reversed**.

Definition

A model is **overparameterized** if the number of trainable parameters p significantly exceeds the number of training examples n.

What Is Overparameterization?

In the first half of the course, our models had $p \ll n$ (far fewer parameters than data points). For neural networks, the situation is **reversed**.

Definition

A model is **overparameterized** if the number of trainable parameters p significantly exceeds the number of training examples n.

Model	**Parameters** p	**Training data** n
3-layer MLP (MNIST, 512 hidden)	$\sim 670{,}000$	60,000
ResNet-50 (ImageNet)	$\sim 25 \times 10^6$	$\sim 1.2 \times 10^6$
BERT-Large (SST-2 fine-tune)	$\sim 340 \times 10^6$	67,349

What Is Overparameterization?

In the first half of the course, our models had $p \ll n$ (far fewer parameters than data points). For neural networks, the situation is **reversed**.

Definition

A model is **overparameterized** if the number of trainable parameters p significantly exceeds the number of training examples n.

Model	**Parameters p**	**Training data n**
3-layer MLP (MNIST, 512 hidden)	$\sim 670{,}000$	60,000
ResNet-50 (ImageNet)	$\sim 25 \times 10^6$	$\sim 1.2 \times 10^6$
BERT-Large (SST-2 fine-tune)	$\sim 340 \times 10^6$	67,349

Consequence: An overparameterized model can **perfectly fit (memorize)** the training data, including noise. Classical theory says this should be a disaster.

Overparameterization and the Landscape

When $p \gg n$, the set of global minimizers $\{\theta : f(\theta) = 0\}$ is typically a *high-dimensional manifold*, not an isolated point.

Overparameterization and the Landscape

When $p \gg n$, the set of global minimizers $\{\theta : f(\theta) = 0\}$ is typically a *high-dimensional manifold*, not an isolated point.

Structural implications:

- ⊙ **Many global minima exist**, increasing the probability that SGD's basin leads to a global minimizer
- ⊙ **Spurious local minima become rare** (Choromanska et al., 2015): as p increases, loss at non-global local minima approaches the global minimum value

Overparameterization and the Landscape

When $p \gg n$, the set of global minimizers $\{\theta : f(\theta) = 0\}$ is typically a *high-dimensional manifold*, not an isolated point.

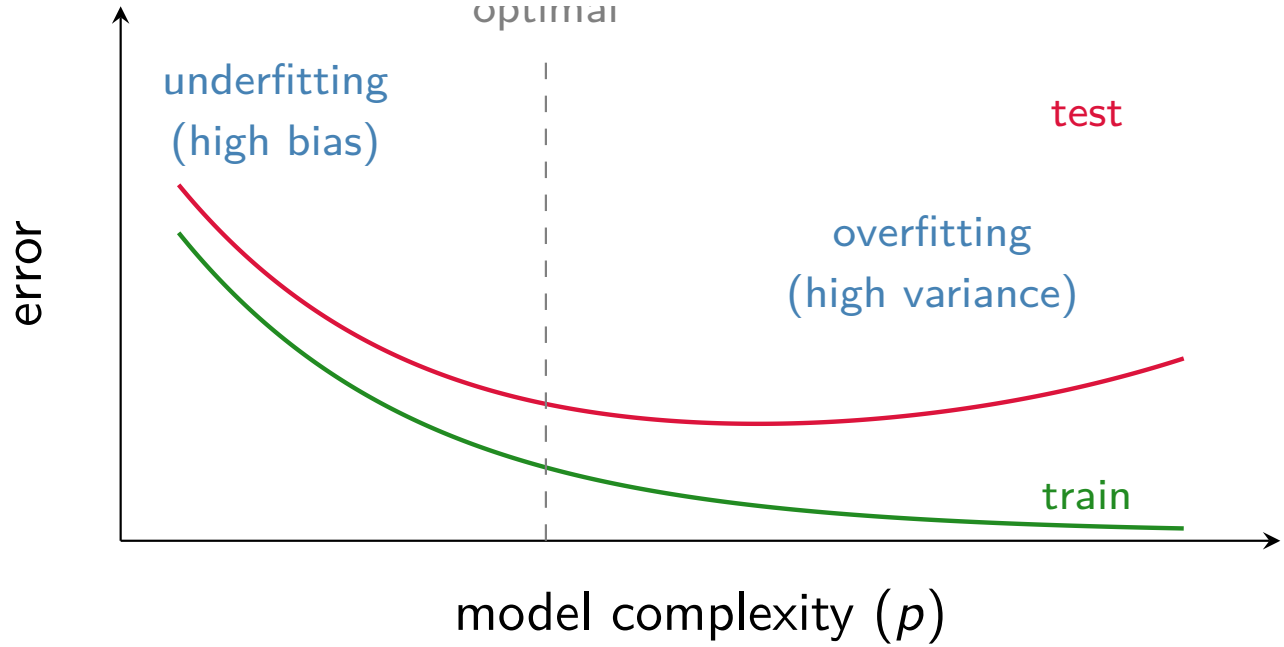
Structural implications:

- ⊙ **Many global minima exist**, increasing the probability that SGD's basin leads to a global minimizer
- ⊙ **Spurious local minima become rare** (Choromanska et al., 2015): as p increases, loss at non-global local minima approaches the global minimum value

This is the optimization-geometric explanation for Factor 6: overparameterization *reshapes the landscape* to eliminate bad critical points.

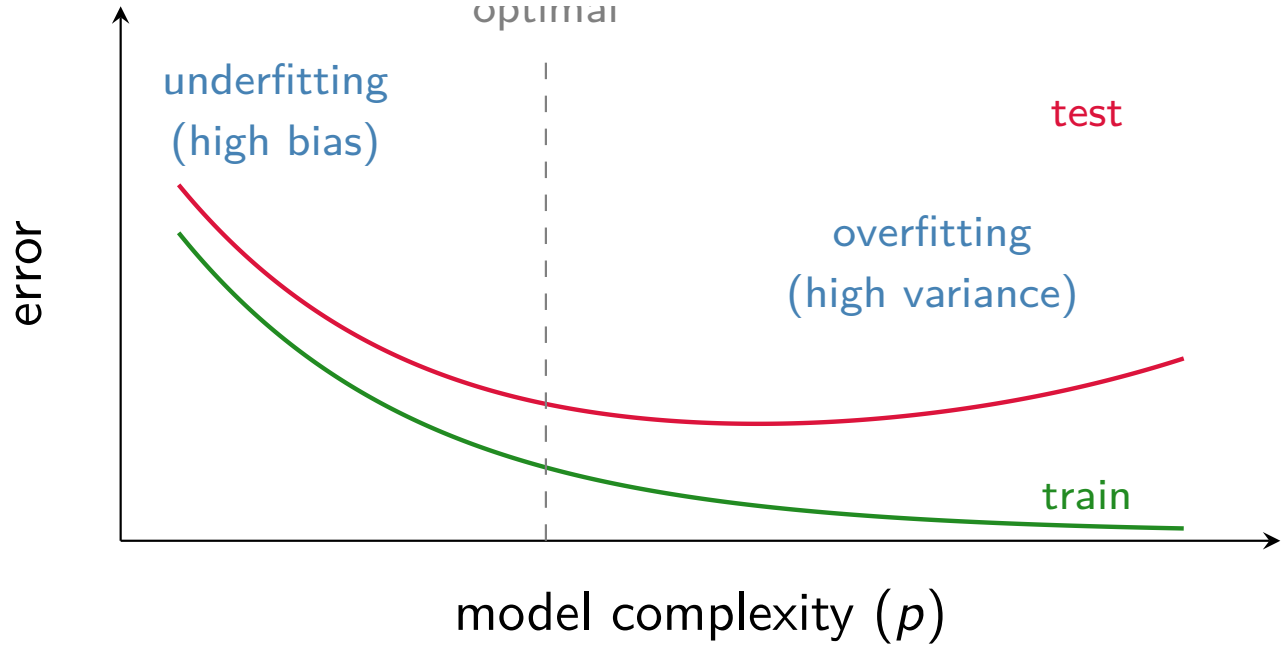
The Classical U-Curve and Why It Breaks

Classical statistical learning theory: test error follows a **U-curve** (bias–variance tradeoff).



The Classical U-Curve and Why It Breaks

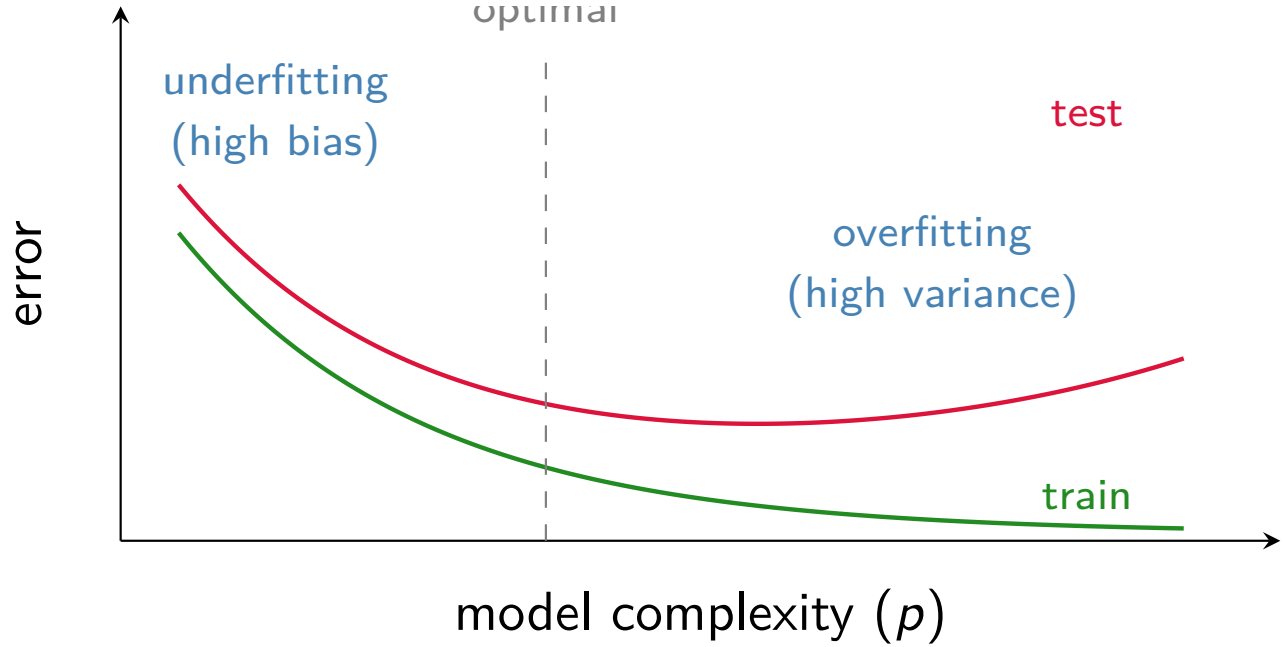
Classical statistical learning theory: test error follows a **U-curve** (bias–variance tradeoff).



Classical prescription: choose complexity to minimize test error (cross-validation). **Stop before the model can memorize.**

The Classical U-Curve and Why It Breaks

Classical statistical learning theory: test error follows a **U-curve** (bias–variance tradeoff).

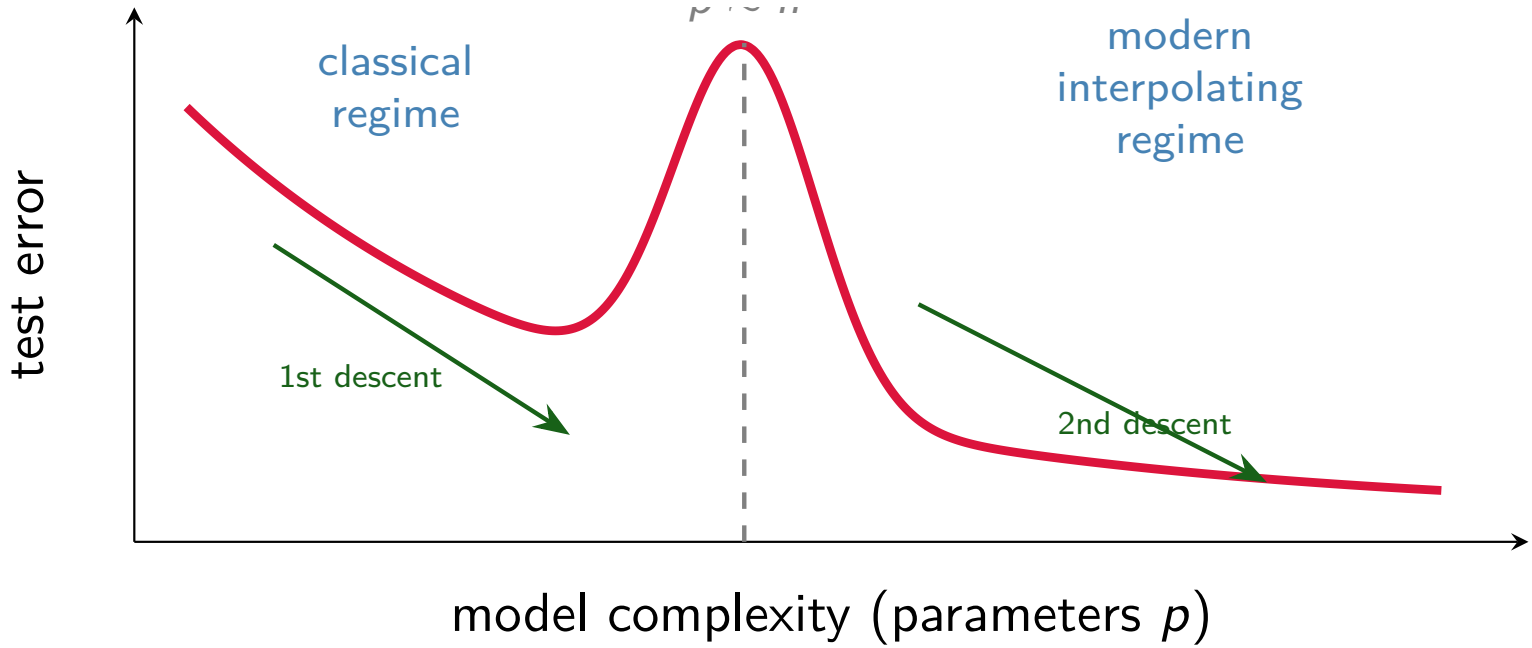


Classical prescription: choose complexity to minimize test error (cross-validation). **Stop before the model can memorize.**

But modern NNs *violate* this: they memorize the data and *still generalize well*.

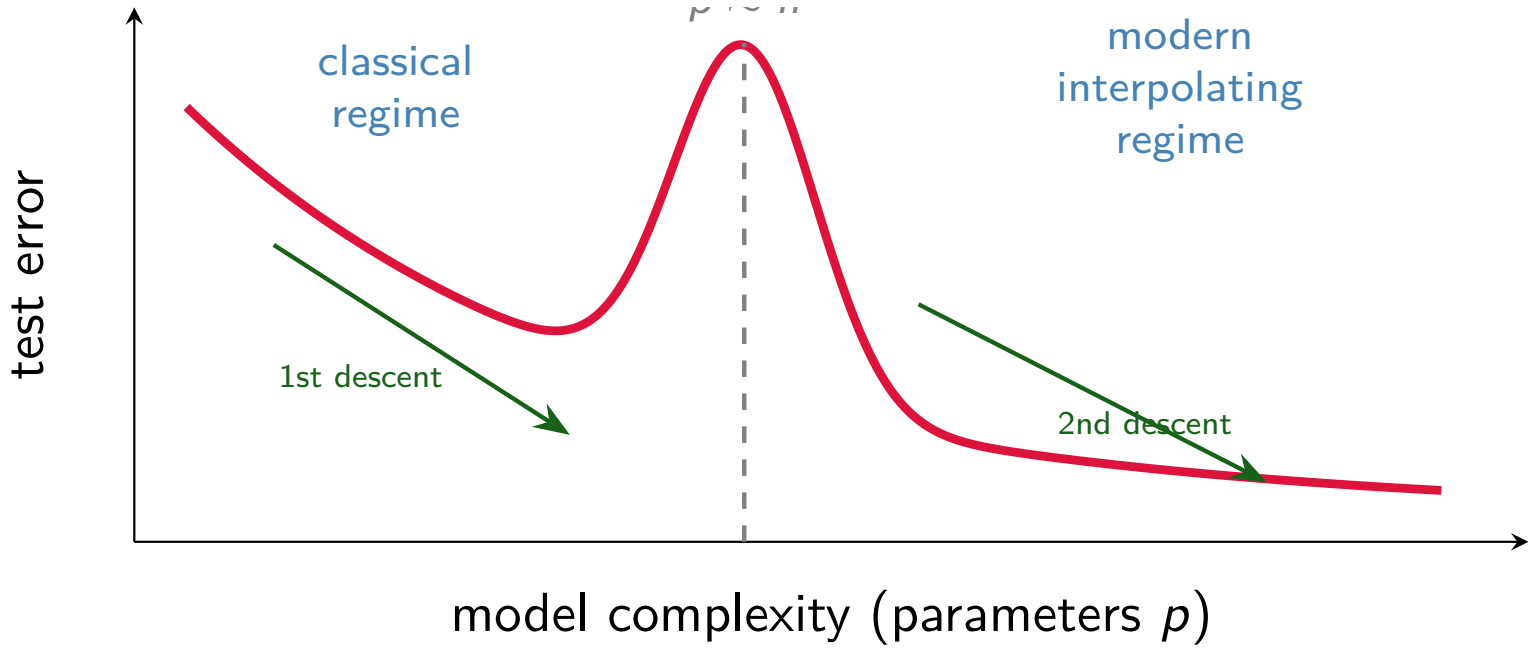
Double Descent ⊙

Belkin et al. (2019): the U-curve is only part of the picture. When complexity is pushed *far beyond* the interpolation threshold, test error **decreases again**:



Double Descent ⊙

Belkin et al. (2019): the U-curve is only part of the picture. When complexity is pushed *far beyond* the interpolation threshold, test error **decreases again**:



At the peak ($p \approx n$): the model barely fits the data; the unique interpolator is irregular. With $p \gg n$, many interpolators exist; SGD finds "smooth" ones that generalize.

NNs Can Memorize Random Labels ⊙

Landmark experiment (Zhang et al., 2017): Train a standard architecture (ResNet) on data with **random labels**.

NNs Can Memorize Random Labels ⊙

Landmark experiment (Zhang et al., 2017): Train a standard architecture (ResNet) on data with **random labels**.

Result: The network achieves **zero training error**. It memorizes all n random label assignments perfectly.

NNs Can Memorize Random Labels ⊙

Landmark experiment (Zhang et al., 2017): Train a standard architecture (ResNet) on data with **random labels**.

Result: The network achieves **zero training error**. It memorizes all n random label assignments perfectly.

Implications:

1. The model class is expressive enough to fit *arbitrary* label assignments
2. Classical complexity measures (VC dimension, Rademacher complexity) give **vacuous** generalization bounds for these networks
3. What matters is the interaction between the architecture, data, initialization, and optimizer— aka **implicit regularization**. The model *could* memorize, but on real data with SGD, it finds generalizing solutions

How This Changes Practice

	Classical (convex)	Modern (overparameterized)

How This Changes Practice

	Classical (convex)	**Modern (overparameterized)**
Model selection	Choose complexity via U-curve	Use large models; implicit regularization

How This Changes Practice

	Classical (convex)	**Modern (overparameterized)**
Model selection	Choose complexity via U-curve	Use large models; implicit regularization
Training goal	Minimize to good (not zero) level	Often train to zero loss (interpolation)

How This Changes Practice

	Classical (convex)	**Modern (overparameterized)**
Model selection	Choose complexity via U-curve	Use large models; implicit regularization
Training goal	Minimize to good (not zero) level	Often train to zero loss (interpolation)
Regularization	Explicit (ℓ_1, ℓ_2) essential	Explicit helps; **implicit** dominates

How This Changes Practice

	Classical (convex)	**Modern (overparameterized)**
Model selection	Choose complexity via U-curve	Use large models; implicit regularization
Training goal	Minimize to good (not zero) level	Often train to zero loss (interpolation)
Regularization	Explicit (ℓ_1, ℓ_2) essential	Explicit helps; **implicit** dominates
Scaling		

How This Changes Practice

	Classical (convex)	**Modern (overparameterized)**
Model selection	Choose complexity via U-curve	Use large models; implicit regularization
Training goal	Minimize to good (not zero) level	Often train to zero loss (interpolation)
Regularization	Explicit (ℓ_1, ℓ_2) essential	Explicit helps; **implicit** dominates
Scaling	Diminishing returns	Power law in data, compute, parameters

Takeaway: The modern NN training recipe is to *use a large model, train with SGD until interpolation, and rely on implicit regularization*

Summary

Summary

- ✓ GD/SGD converge to ε-stationary points ($O(1/\varepsilon)$ / $O(1/\varepsilon^2)$ work)

Summary

- ✓ GD/SGD converge to ε-stationary points ($O(1/\varepsilon)$ / $O(1/\varepsilon^2)$ work)
- ◐ Stationarity is meaningful: strict saddles escaped via spectral instability; overparameterization eliminates bad local minima

Summary

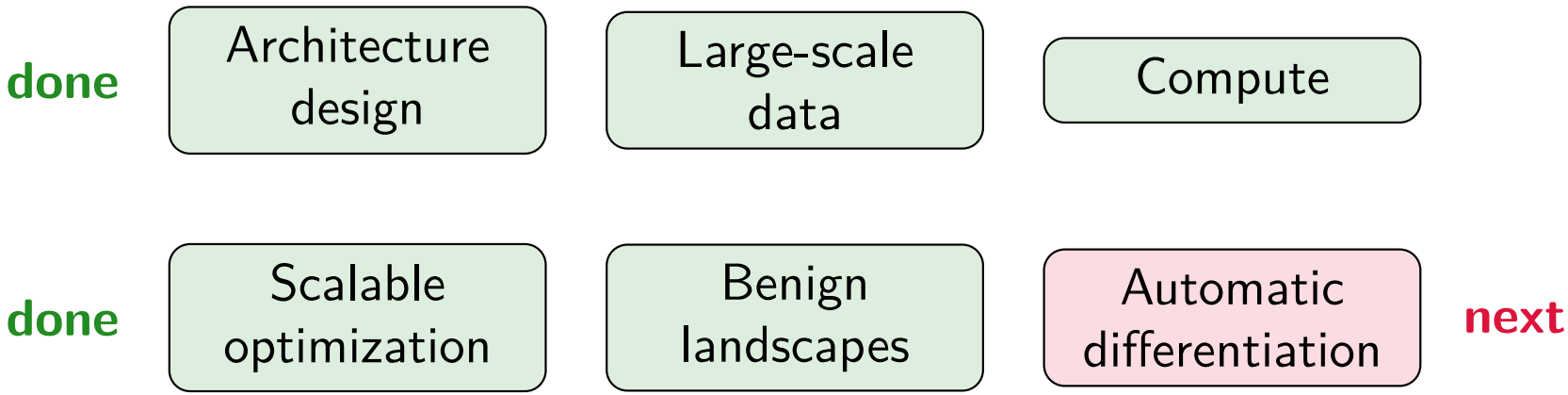
- ✓ GD/SGD converge to ε-stationary points ($O(1/\varepsilon)$ / $O(1/\varepsilon^2)$ work)
- ◐ Stationarity is meaningful: strict saddles escaped via spectral instability; overparameterization eliminates bad local minima
- ⊙ Classical bias–variance tradeoff gives way to double descent; the training recipe is fundamentally different from convex ML

What's Next

The practical question left: How to compute $\nabla_\theta f(\theta)$ efficiently?

What's Next

The practical question left: How to compute $\nabla_\theta f(\theta)$ efficiently?



Next lecture: Backpropagation — an efficient algorithm for computing gradients of the loss w.r.t. all parameters via the chain rule, in a single backward pass through the network DAG.

Questions?

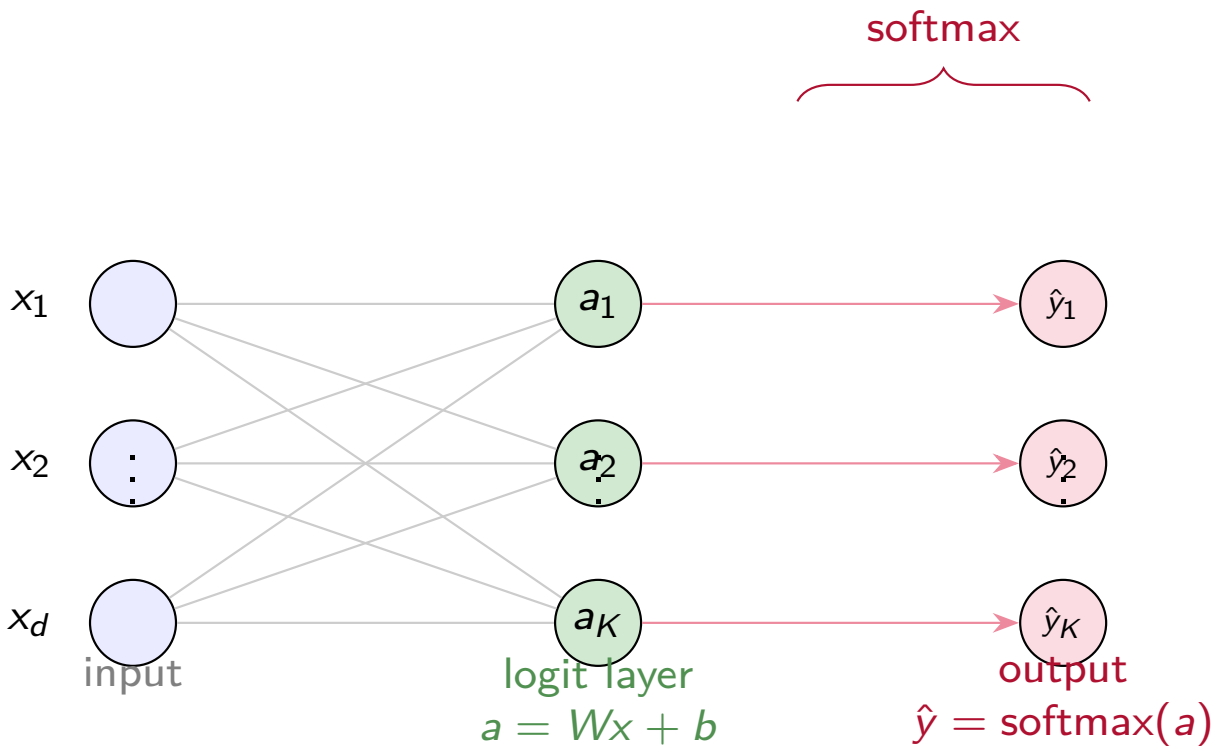
Next time: backpropagation and automatic differentiation

Appendix: Full History of Neural Networks

Year	Event	Reference
1943	McCulloch–Pitts neuron: first mathematical model (binary threshold logic)	McCulloch & Pitts, *Bull. Math. Biophys.*
1958	Rosenblatt's Perceptron: learnable weights, convergence theorem	Rosenblatt, *Psych. Rev.*
1969	Minsky & Papert prove single-layer limitations; first AI winter	Minsky & Papert, *Perceptrons*
1986	Backpropagation (re)discovered; multi-layer network revival	Rumelhart, Hinton & Williams, *Nature*
1989	Cybenko's universal approximation theorem (sigmoidal)	Cybenko, *Math. Control Signals Syst.*
1989	LeNet: CNNs for handwritten digit recognition	LeCun et al.
1993	Leshno et al.: UAT iff non-polynomial activation	Leshno et al., *Neural Networks*
1997	LSTM: gated recurrence for long-range dependencies	Hochreiter & Schmidhuber, *Neural Comput.*
$\sim$95–05	**Second AI winter:** SVMs dominate	
2006	Deep Belief Networks; greedy layer-wise pretraining	Hinton et al., *Neural Comput.*
2012	**AlexNet wins ImageNet** by $>$10pp; deep learning era begins	Krizhevsky et al., *NeurIPS*
2014	GANs / VAEs: deep generative models	Goodfellow et al.; Kingma & Welling
2017	Transformer: attention replaces recurrence; foundation of LLMs	Vaswani et al., *NeurIPS*

Pattern: Theoretical ideas preceded practical success by decades. Backpropagation was known in the 1960s–70s but only became practical with compute + data.

Appendix: Multiclass Logistic Regression as a NN



Multiclass logistic regression is a depth-1 FCFFN with K output neurons and softmax activation. The only difference from deeper NNs is the number of layers.

Appendix: SGD with Tuned Constant Stepsize

If the total number of iterations T is known in advance, one can use a single constant stepsize tuned to T.

With $\alpha = \sqrt{\frac{R}{TL\sigma^2}}$ and $T \geq \frac{4RL}{\sigma^2}$:

$$\mathbb{E}\left(\frac{1}{T}\sum_{t=1}^{T}\|\nabla f(x_{t-1})\|_2^2\right) \leq 4\sqrt{\frac{RL\sigma^2}{T}}$$

This achieves the same $O(1/\sqrt{T})$ rate as the decaying schedule, but requires knowing T in advance. In practice this is done via "budget-aware" training where total compute is fixed.

Appendix: Additional Landscape Results

Beyond deep linear networks and matrix sensing:

Matrix completion. Observe a sparse subset of entries of low-rank M. Under incoherence conditions, the landscape is benign ✓ (Ge, Jin & Zheng, 2017). With very sparse observations, spurious local minima can appear.

Phase retrieval. Recover $x \in \mathbb{R}^d$ from $y_i = |a_i^\top x|^2$ (phase lost). With spectral initialization, the landscape is locally benign ✓ (Sun, Qu & Wright, 2018). Without good initialization, gradient methods can get trapped.

Symmetric matrix factorization. Given PSD M, find U with $UU^\top \approx M$. The strict saddle property holds ✓ (Ge et al., 2017).

Appendix: When Does the Benign Landscape Story Fail?

Failure mode	**What breaks**	**Lesson**
$p \approx n$ (interpolation threshold)	Loss of overparameterization	Landscape is benign *because of* overparameterization
Sparse observations	Problem instance structure	Landscape depends on data, not just the model
Bad initialization	Local vs. global benignity	Some landscapes are only locally benign
Sharp minima (large-batch SGD)	Wrong basin	*Which* minimum matters for generalization
No normalization	Never reaching relevant region	Landscape theory is local; conditioning matters

The benign landscape narrative is not universal. Understanding *where* it breaks sharpens our understanding of when it works.