Computational Mathematics and AI

Lecture 5: Efficient Optimization Methods

Lars Ruthotto

Departments of Mathematics and Computer Science

lruthotto@emory.edu

larsruthotto

slido.com #CBMS25

Reading List

Historical Context: Adaptive methods and structured second-order approximations can accelerate neural network training but generalization remains a challenge.

Key Readings:

1. Kingma and Ba (2015) – Adam: A Method for Stochastic Optimization. *ICLR*
   The modern standard adaptive optimizer
2. Loshchilov and Hutter (2019) – Decoupled Weight Decay Regularization. *ICLR*
   AdamW: fixing weight decay in Adam
3. Chen et al. (2023) – Symbolic Discovery of Optimization Algorithms. *NeurIPS*
   Lion: evolutionary-discovered sign-based optimizer
4. Amari (1998) – Natural Gradient Works Efficiently in Learning. *Neural Comp.*
   Parameterization-invariant optimization
5. Martens and Grosse (2015) – K-FAC: Kronecker-Factored Approximate Curvature. *ICML*
   Practical second-order methods

Lecture Outline: Motivation $\rightarrow$ Momentum $\rightarrow$ Adam/Lion $\rightarrow$ K-FAC

Connection to Lecture 4

What we established in Lecture 4:

- SGD converges to stationary points (theory)
- Implicit regularization: early stopping, minimum norm, flatness preference
- Continuous-time view: Langevin dynamics, temperature $T \propto \eta/b$
- Over-parametrization: NTK and mean field regime create expressive networks

Key conclusion: Lazy regime works reliably!

- SGD in lazy regime performs comparably to Gauss-Newton (Lecture 3)
- Now we understand *why*: benign landscapes + implicit regularization

Today's question:

Can we make optimization more efficient, avoid hyperparameters without hurting generalization?

SGD and GN Pain Points

1. **Learning rate sensitivity:**
   - Too small: slow convergence, limited exploration
   - Too large: divergence or oscillation
   - No guidelines: need careful tuning for each problem
2. **Ill-conditioning:**
   - Loss landscape has different curvatures in different directions
   - Single learning rate can't optimize all directions equally
   - Condition number $\kappa = \lambda_{\max}/\lambda_{\min}$ hurts convergence
3. **No momentum / variance reduction:**
   - Each step independent of history
   - Cannot accelerate in consistent gradient directions
   - Cannot slow down in oscillatory directions

Gauss-Newton solved these via curvature... but is infeasible for large NNs!

Roadmap: Efficient SGD Variants

1. **Momentum Methods**
   - Heavy ball method: accumulate velocity
   - Nesterov acceleration: look-ahead gradient
   - Cost: $O(p)$ memory (one extra vector)
2. **Adaptive Gradient Methods**
   - Per-parameter learning rates from gradient history
   - Adam, AdamW, Lion
   - Cost: $O(2p)$ memory (two moment vectors)
3. **Outlook: Efficient Second-Order**
   - Approximate curvature with structure
   - K-FAC: Kronecker-factored approximation
   - Cost: $O(p + \sum n_\ell^2)$ memory

trade memory for faster convergence and robustness

Momentum Methods

Heavy Ball Method: Adding Memory

Motivation: Ball rolling down a hill accumulates velocity

SGD with Momentum Polyak 1964:

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + \nabla L(\theta_t) \quad \text{(accumulate velocity)}$$
$$\theta_{t+1} = \theta_t - \eta \mathbf{v}_{t+1} \quad \text{(update parameters)}$$

where $\beta \in [0, 1)$ is momentum coefficient (typically $\beta = 0.9$)

Key properties:

- **Acceleration:** Builds speed in consistent gradient directions
- **Damping:** Cancels oscillations in inconsistent directions
- **Memory:** $O(p)$ extra storage for velocity vector

Convergence improvement: Proven for convex quadratics:

- GD: iterations $\propto \kappa$ (condition number)
- Momentum: iterations $\propto \sqrt{\kappa}$ (quadratic speedup!)

momentum trades $O(p)$ memory for $\sqrt{\kappa}$ speedup

Nesterov Accelerated Gradient

Key idea Nesterov 1983: Compute gradient at *look-ahead* position

$$\tilde{\theta}_t = \theta_t + \beta(\theta_t - \theta_{t-1}) \quad \text{(look ahead)}$$
$$\theta_{t+1} = \tilde{\theta}_t - \eta \nabla L(\tilde{\theta}_t) \quad \text{(gradient at look-ahead)}$$

Intuition:

- Heavy ball: gradient at current position, then add momentum
- Nesterov: first apply momentum, then compute gradient
- “Correct” the momentum direction before overshooting

Convergence:

- Achieves optimal $O(1/t^2)$ rate for smooth convex functions
- Heavy ball: $O(1/t)$ (worse by factor t)
- Provably optimal among first-order methods (with optimal β)

Nesterov’s look-ahead achieves optimal convergence rate

Adaptive Gradient Methods

The Adaptive Paradigm: Per-Parameter Learning Rates

Core idea: Adapt learning rate per parameter based on gradient history

$$\theta_{t+1} = \theta_t - \eta \ \text{diag}(\sqrt{v_t} + 10^{-8})^{-1} g_t$$

where v_t accumulates information about gradient magnitude

Benefits:

- **Robustness:** Works across wider range of learning rates
- **Sparse features:** Larger updates to infrequent features
- **Ill-conditioning:** Automatically rescales for different curvatures

Connection to preconditioning:

- Adaptive methods = **diagonal preconditioning**
- Approximates diagonal of Fisher or empirical Hessian

Three generations:

1. **AdaGrad** Duchi, Hazan, and Singer 2011: Accumulate all gradients $\Rightarrow$ LR decays too aggressively
2. **RMSprop** Tieleman and Hinton 2012: Exponential moving average $\Rightarrow$ fixes decay
3. **Adam** Kingma and Ba 2015: Add momentum + bias correction

adaptive methods trade $O(2p)$ memory for robustness

Adam: Three Design Principles

Adam = **Ada**ptive **M**oment Estimation Kingma and Ba 2015

Principle 1: Momentum (first moment)

- Exponential moving average of gradients: $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1) g_t$
- Smooths gradient estimates, accelerates in consistent directions
- Hyperparameter: β_1 (typically 0.9)

Principle 2: Adaptive learning rates (second moment)

- Exponential moving average of squared gradients: $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2) g_t^2$
- Scale learning rate inversely to typical gradient magnitude
- Hyperparameter: β_2 (typically 0.999)

Principle 3: Bias correction

- Moving averages initialized at zero $\Rightarrow$ biased toward zero early
- Correct: $\hat{\mathbf{m}}_t = \mathbf{m}_t/(1-\beta_1^t)$, $\hat{\mathbf{v}}_t = \mathbf{v}_t/(1-\beta_2^t)$

Adam = momentum + adaptive rates + bias correction

Adam: Complete Algorithm

Hyperparameters: $\eta = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$

Algorithm:

1. Initialize: $\mathbf{m}_0 = 0$, $\mathbf{v}_0 = 0$, $t = 0$
2. While not converged:
   2.1 $t \leftarrow t + 1$
   2.2 $g_t \leftarrow \nabla_\theta L(\theta_{t-1})$ (gradient)
   2.3 $\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) g_t$ (first moment)
   2.4 $\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) g_t^2$ (second moment)
   2.5 $\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1 - \beta_1^t)$, $\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1 - \beta_2^t)$ (bias correction)
   2.6 $\theta_t \leftarrow \theta_{t-1} - \eta \cdot \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \varepsilon)$ (update)

Memory cost: $O(2p)$ vs $O(p)$ for SGD

Defaults work remarkably well: Often used as-is without tuning

AdamW: Decoupled Weight Decay

Problem with L_2 regularization in Adam:

- Standard: Add $\lambda\|\theta\|^2$ to loss $\Rightarrow$ gradient includes $2\lambda\theta$
- Adam adapts this regularization gradient like any other
- **Issue:** Adaptive scaling interferes with intended regularization strength

AdamW solution Loshchilov and Hutter 2019: Decouple weight decay from gradient

$$\theta_t \leftarrow \theta_{t-1} - \eta \cdot \left(\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \varepsilon} + \lambda\theta_{t-1} \right)$$

- Weight decay $\lambda\theta$ applied *after* adaptive scaling
- Regularization strength independent of gradient magnitude

When to use AdamW:

- Any time you use weight decay (almost always)
- Default for Transformers and language models
- PyTorch: `torch.optim.AdamW`

use AdamW when weight decay is needed (i.e., almost always)

Lion: Evolutionary Discovered Optimizer

Origin Chen et al. 2023: Discovered via AutoML (symbolic program search)

Algorithm: Sign-based update with momentum

$$c_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1) g_t \quad \text{(interpolate)}$$
$$\theta_t = \theta_{t-1} - \eta \cdot \text{sign}(c_t) \quad \text{(sign-based update)}$$
$$\mathbf{m}_t = \beta_2 \mathbf{m}_{t-1} + (1-\beta_2) g_t \quad \text{(momentum for next step)}$$

Key differences from Adam:

- **Sign-based:** Uses sign(c_t) instead of scaled gradient
- **Memory:** $O(p)$ instead of $O(2p)$ (only one momentum vector)
- **Scale invariance:** Update magnitude independent of gradient scale

Typical hyperparameters:

- $\eta = 10^{-4}$ (typically $10\times$ smaller than Adam)
- $\beta_1 = 0.9$, $\beta_2 = 0.99$

Lion = sign-based updates + momentum, memory-efficient alternative

When to Use Adam vs SGD vs Lion

Adam/AdamW advantages:

- **Robustness:** Works across wide LR range
- **Sparse features:** NLP, embeddings
- **Quick prototyping:** Defaults work

SGD (with momentum) advantages:

- **Vision tasks:** Better final accuracy
- **Well-tuned:** Can outperform Adam
- **Memory:** $O(p)$ vs $O(2p)$

Lion advantages:

- **Memory-efficient:** Same as SGD
- **Large-scale:** Competitive on big models
- **Scale-invariant:** Robust to gradient magnitude

Empirical patterns:

Domain	Typical Choice
NLP/Transformers	AdamW
Vision/CNNs	SGD + tuning
Transfer learning	Adam
Large-scale LLMs	AdamW or Lion
Memory-limited	Lion

Theory-practice gaps:

- SGD generalizes better on vision: Flatter minima?
- Adam optimal for sparse gradients: Diagonal preconditioning effective
- Why domain-dependent? Implicit bias differences unclear

Adaptive Methods: Summary

Evolution:

- **AdaGrad (2011):** Accumulate gradients $\rightarrow$ too aggressive decay
- **RMSprop (2012):** Exponential average $\rightarrow$ fixes decay
- **Adam (2015):** + Momentum + bias correction $\rightarrow$ dominant
- **AdamW (2017):** Decoupled weight decay $\rightarrow$ better regularization
- **Lion (2023):** Sign-based $\rightarrow$ memory-efficient alternative

Cost-benefit trade-off:

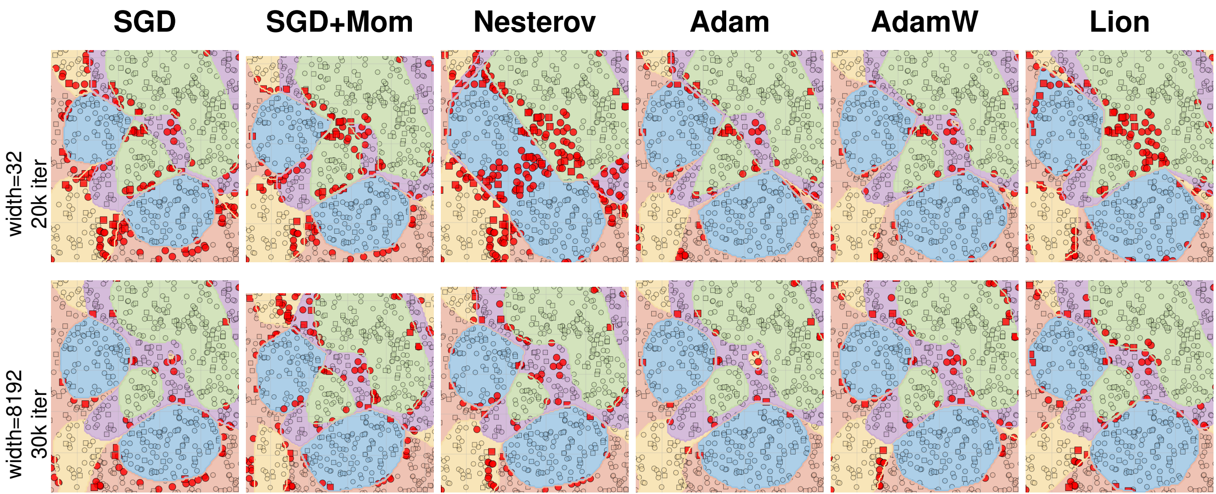
Method	Memory	HP Sensitivity	Best For
SGD	$O(p)$	High	Vision + tuning
SGD + Momentum	$O(2p)$	High	Vision (standard)
Adam/AdamW	$O(2p)$	Low	NLP, default
Lion	$O(p)$	Low	Large-scale, memory

Practical advice: Start with Adam, optimize if needed

adaptive methods trade memory for robustness – start with Adam

Numerical Comparison

Optimizer Comparison: Decision Boundaries



Lazy regime produces smoother boundaries; Adam/AdamW best overall

Optimizer Comparison: Numerical Results

Method	Small (width=32)				Lazy (width=8192)			
	Loss		Accuracy		Loss		Accuracy	
	Train	Test	Train	Test	Train	Test	Train	Test
SGD	0.51	0.50	82.9%	82.5%	0.24	0.42	91.6%	86.0%
SGD+Momentum	0.41	0.48	85.6%	79.5%	0.58	0.84	83.6%	85.5%
SGD+Nesterov	0.89	1.04	68.3%	63.5%	0.24	0.29	90.9%	89.5%
Adam	0.21	0.44	92.9%	90.0%	0.08	0.29	97.2%	91.5%
AdamW	0.13	0.29	95.0%	92.0%	0.24	0.36	90.2%	86.5%
Lion	0.41	0.70	85.6%	81.5%	0.34	0.41	91.4%	85.0%

Key observations:

- **Small regime**: AdamW best (92% test), Nesterov unstable (63.5%)
- **Lazy regime**: Adam best (91.5% test), Nesterov recovers (89.5%)
- Momentum can hurt in small networks but helps in lazy regime

Optimizer choice interacts with network architecture!

Adam Vector Field Theory

Beyond Gaussian Noise: Lévy-Driven Dynamics

Recall from Lecture 4: CLT gives Gaussian noise approximation

$$\hat{g}_S(\theta) \approx \nabla \mathcal{L}(\theta) + \frac{1}{\sqrt{S}} \Delta g, \quad \Delta g \sim \mathcal{N}(0, \Sigma(\theta))$$

Empirical reality [Simsekli et al., 2019]:

- Gradient noise often exhibits **heavy tails**
- Characterized by **symmetric α-stable** (SαS) distributions
- Tail index $\alpha \in (0, 2]$: $p(x) \sim |x|^{-(1+\alpha)}$ for large $|x|$
- $\alpha = 2 \Rightarrow$ Gaussian (CLT special case)
- $\alpha < 2 \Rightarrow$ Heavy tails, infinite variance

Lévy-driven SDE for SGD:

$$d\theta_t = -\nabla \mathcal{L}(\theta_t)\, dt + \epsilon \Sigma_t\, dL_t, \quad L_t \sim S\alpha S$$

where L_t is a **Lévy motion** with stationary, independent increments.

heavy-tailed noise enables “big jumps” to escape sharp minima

The Adam Vector Field: Mathematical Derivation

Continuous-time limit of Adam yields coupled SDE system:

$$d\theta_t = \mathbf{V}_{\text{Adam}}(\theta_t)\,dt + \epsilon Q_t^{-1}\Sigma_t\,dL_t$$
$$dm_t = \beta_1(\nabla\mathcal{L}(\theta_t) - m_t)\,dt$$
$$dv_t = \beta_2([\nabla\mathcal{L}(\theta_t)]^2 - v_t)\,dt$$

The Adam vector field (deterministic drift):

$$\boxed{\mathbf{V}_{\text{Adam}}(\theta_t) = -\mu_t Q_t^{-1} m_t}$$

Components:

- $Q_t = \text{diag}(\sqrt{\omega_t v_t + \epsilon})$ (adaptive scaling matrix)
- $\mu_t = 1/(1 - e^{-\beta_1 t})$ (first moment bias correction)
- $\omega_t = 1/(1 - e^{-\beta_2 t})$ (second moment bias correction)

Key insight: Adam's fixed points satisfy $\mathbf{V}_{\text{Adam}}(\theta^*) = 0$, **not** $\nabla\mathcal{L}(\theta^*) = 0$!

take away: Adam converges to zeros of its vector field, not the gradient

Why Adam Dampens Noise: Generalization Implications

How Adam modifies the noise structure:

- $Q_t^{-1} = \text{diag}(1/\sqrt{\omega_t v_t + \epsilon})$ scales noise inversely to gradient magnitude
- Large gradients $\Rightarrow$ small effective noise in that coordinate
- **Effect:** Dampens heavy-tailed fluctuations $\Rightarrow$ **lighter tails** (larger α)

Escape time analysis:

Property	**SGD**	**Adam**
Noise tail index α	Heavy ($\alpha < 2$)	Lighter ($\alpha \to 2$)
Anisotropic structure	Preserved	Diminished
Escape time Γ	Smaller	Larger

Consequence for generalization:

- SGD escapes sharp minima **faster** $\Rightarrow$ finds flatter basins
- Adam stays longer in sharp minima $\Rightarrow$ may converge to sharper solutions

Adam's noise dampening explains generalization gap on vision

Reconciling Theory with Practice

If Adam finds sharper minima, why does it work so well?

Domain-dependent effects:

Vision/CNNs: Sharp vs flat strongly correlates with generalization

→ SGD often preferred; generalization gap observed

NLP/Transformers: Sparse gradients, different loss landscape

→ Adam's coordinate-wise adaptation is beneficial

→ Embedding layers have naturally sparse updates

Practical mitigation strategies:

- **AdamW:** Decoupled weight decay restores some regularization
- **Learning rate warmup:** Allows initial exploration before adaptation
- **Lower β_2:** Less aggressive smoothing, more noise preserved

Open questions:

- Precise characterization of Adam's implicit regularization
- When does heavy-tail analysis vs. Gaussian SDE apply?

Efficient Second-Order Methods

Classical Preconditioning Perspective

Preconditioned gradient descent:

$$\theta_{t+1} = \theta_t - \eta \mathbf{M}^{-1} \nabla L(\theta_t)$$

where $\mathbf{M} \succ 0$ is a preconditioner matrix

Benefits of preconditioning:

- Rescales search directions to account for curvature
- Improves conditioning: transforms ill-conditioned $\rightarrow$ well-conditioned
- Faster convergence: Newton converges in 1 step for quadratics

Classical choices:

- **Diagonal:** $\mathbf{M} = \text{diag}(\nabla^2 L) \rightarrow$ cheap but limited
- **Gauss-Newton:** $\mathbf{M} = \frac{1}{N}\sum_{i=1}^{N} \nabla^2 \mathbf{J}_i \mathbf{H}_i \mathbf{J}_i^\top \rightarrow$ effective but $O(p^2)$ memory
- **Fisher:** $\mathbf{M} = \mathbf{F}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla^2 \mathbf{J}_i \mathbf{J}_i^\top \rightarrow$ natural gradient

preconditioning accelerates optimization via curvature information

Second-Order Methods: An Outlook

Why study second-order methods?

- **Mathematical elegance:** Natural gradient, information geometry
- **Theoretical insights:** Understanding curvature structure
- **Specialized applications:** Small networks, scientific computing

Current status:

- **Not mainstream:** Adam/AdamW dominate in practice
- **Implementation complexity:** Requires architecture-specific code
- **Computational overhead:** $O(n^3)$ per layer adds up
- **Niche success:** Large-batch training, small models

Our approach:

- Explain **mathematical ideas** (natural gradient, Kronecker structure)
- Show **what's possible** (K-FAC worked example)
- Understand **why not mainstream** (computational cost vs. benefit)

second-order methods are effective but not (yet) practical at scale

Natural Gradient Descent

Motivation Amari 1998: Parameterization-invariant optimization

Fisher information matrix: Measures parameter space curvature

$$\mathbf{F}(\theta) = \mathbb{E}_{x,y}\left[\nabla_\theta \log p(y|x,\theta)\nabla_\theta \log p(y|x,\theta)^T\right]$$

Natural gradient: Steepest descent in Fisher metric

$$\theta_{t+1} = \theta_t - \eta \mathbf{F}(\theta_t)^{-1}\nabla L(\theta_t)$$

Properties:

- Parameterization-invariant: reparameterizing doesn't change trajectory
- Accounts for parameter correlations
- Faster convergence in function space

The problem: Same infeasibility as Hessian

- Fisher matrix: $O(p^2)$ memory
- Inversion: $O(p^3)$ computation
- **Need structured approximations!**

natural gradient is ideal but needs approximation

K-FAC: Kronecker-Factored Approximation

K-FAC = **K**ronecker-**F**actored **A**pproximate **C**urvature Martens and Grosse 2015

Key insight: Exploit neural network **layer structure**

For one layer: $h^{(\ell+1)} = \sigma(\mathbf{W}^{(\ell)} h^{(\ell)} + b^{(\ell)})$

- Weight matrix: $\mathbf{W}^{(\ell)} \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$
- Fisher block for this layer: $\mathbf{F}_W \in \mathbb{R}^{(n_{\text{out}} \cdot n_{\text{in}}) \times (n_{\text{out}} \cdot n_{\text{in}})}$

K-FAC approximation: Factor Fisher block as Kronecker product

$$\mathbf{F}_W \approx \mathbf{A} \otimes \mathbf{S}$$

where:

- $\mathbf{A} = \mathbb{E}[hh^T] \in \mathbb{R}^{n_{\text{in}} \times n_{\text{in}}}$: Activation correlation
- $\mathbf{S} = \mathbb{E}[\delta\delta^T] \in \mathbb{R}^{n_{\text{out}} \times n_{\text{out}}}$: Error correlation

Memory savings:

- Full block: $O((n_{\text{in}} \cdot n_{\text{out}})^2) \to$ K-FAC: $O(n_{\text{in}}^2 + n_{\text{out}}^2)$

Kronecker factorization exploits layer structure for massive savings

K-FAC: Why Not Mainstream?

Computational cost analysis:

- Forward-backward pass: $O(p)$ (same as Adam)
- Accumulate $\mathbf{A}$, $\mathbf{S}$: $O(n^2)$ per layer per step
- Invert factors: $O(n^3)$ per layer (every 10-100 steps)
- **Example:** Transformer with 1024-dim layers $\rightarrow$ 1B FLOPs/inversion

Empirical benefits of K-FAC:

- **Large-batch regime:** Better curvature estimates (batch $\geq$ 512)
- **Small-medium networks:** Overhead manageable ($n \leq 1024$)
- **Fully-connected or Conv layers:** Kronecker structure exact
- **Wall-clock matters:** Willing to pay per-iteration cost for fewer iterations

Why not mainstream:

- **Memory:** $O(\sum n_\ell^2)$ overhead significant for wide networks
- **Implementation complexity:** Architecture-specific code needed
- **Attention mechanisms:** Kronecker approximation less natural
- **Cost-benefit:** Adam improvements usually sufficient

K-FAC effective in specialized settings, not general-purpose

Trilogy Synthesis

The Optimization Trilogy: Summary

Lecture 3: Foundations

- SA vs SAA framework for stochastic optimization
- Backpropagation enables $O(p)$ gradient computation
- Second-order methods infeasible: $O(p^2)$ memory, $O(p^3)$ computation
- SGD as prototype: simple, scalable, surprisingly effective

Lecture 4: Why SGD Works

- Convergence theory: stationary points, not global minima
- Implicit regularization: early stopping, minimum norm, batch size
- Continuous perspectives: gradient flow, Langevin, edge of stability
- Landscape structure: over-parametrization creates benign landscapes

Lecture 5: Efficient Methods

- Momentum: $O(p)$ memory for $\sqrt{\kappa}$ speedup
- Adaptive methods: Adam/Lion trade $O(2p)$ for robustness
- K-FAC: structured second-order approximation
- Computational comparison: optimizer choice depends on constraints

trilogy theme: foundations $\to$ understanding $\to$ accelerations

Key Insights Across Three Lectures

1. **Why neural network optimization works**:
   - Over-parametrization ($p \gg n$) creates favorable landscape
   - SGD noise helps: escapes saddles, prefers flat minima
   - Mode connectivity: good solutions are connected
2. **Optimization is more than minimization**
   - *Which* minimum matters for generalization
   - SGD implicitly regularizes: early stopping, flatness bias
   - Algorithm choice affects solution quality, not just speed
3. **Momentum vs. adaptive: different mechanisms**
   - **Momentum:** Numerical acceleration via ODE discretization
   - **Adaptive:** Statistical diagonal preconditioning
   - **Complementary:** Adam combines both (momentum + adaptive rates)
4. **Trade-offs are fundamental**:
   - Memory vs robustness: Adam ($O(2p)$) vs SGD ($O(p)$)
   - Compute vs iterations: K-FAC (expensive) vs SGD (cheap)
   - Tuning vs convenience: tuned SGD vs out-of-box Adam

References I

- Amari, S.-I. (1998). "Natural Gradient Works Efficiently in Learning". In: *Neural Computation* 10.2, pp. 251–276.
- Chen, X., C. Liang, D. Huang, E. Real, K. Wang, Y. Liu, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le (2023). "Symbolic Discovery of Optimization Algorithms". In: *arXiv preprint arXiv:2302.06675*.
- Duchi, J., E. Hazan, and Y. Singer (2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12, pp. 2121–2159.
- Kingma, D. P. and J. Ba (2015). "Adam: A Method for Stochastic Optimization". In: *arXiv preprint arXiv:1412.6980*.
- Loshchilov, I. and F. Hutter (2019). "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations (ICLR)*.
- Martens, J. and R. Grosse (2015). "Optimizing Neural Networks with Kronecker-Factored Approximate Curvature". In: *International Conference on Machine Learning (ICML)*, pp. 2408–2417.

References II

- Nesterov, Yurii E. (1983). "A Method for Solving the Convex Programming Problem with Convergence Rate $O(1/k^2)$". In: *Doklady Akademii Nauk SSSR* 269.3, pp. 543–547.
- Polyak, Boris T. (1964). "Some Methods of Speeding Up the Convergence of Iteration Methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5, pp. 1–17.
- Tieleman, T. and G. Hinton (2012). *Lecture 6.5-RMSProp: Divide the Gradient by a Running Average of Its Recent Magnitude*. COURSERA: Neural Networks for Machine Learning.