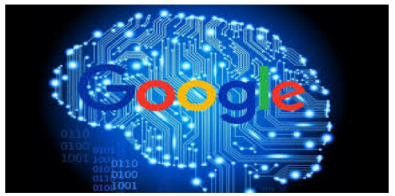
# Optimization for Machine Learning: Beyond Stochastic Gradient Descent

PRINCETON UNIVERSITY

Elad Hazan



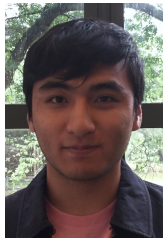

**References and more info:**
**http://www.cs.princeton.edu/~ehazan/tutorial/MLSStutorial.htm**

Based on:
[Agarwal, Bullins, Hazan ICML ’16]
[Agarwal, Allen-Zhu, Bullins, Hazan, Ma STOC ’17]
[Hazan, Singh, Zhang ICML ‘17], [Agarwal, Hazan COLT ‘17]
[Agarwal, Bullins, Chen, Hazan, Singh, Zhang, Zhang ’18]

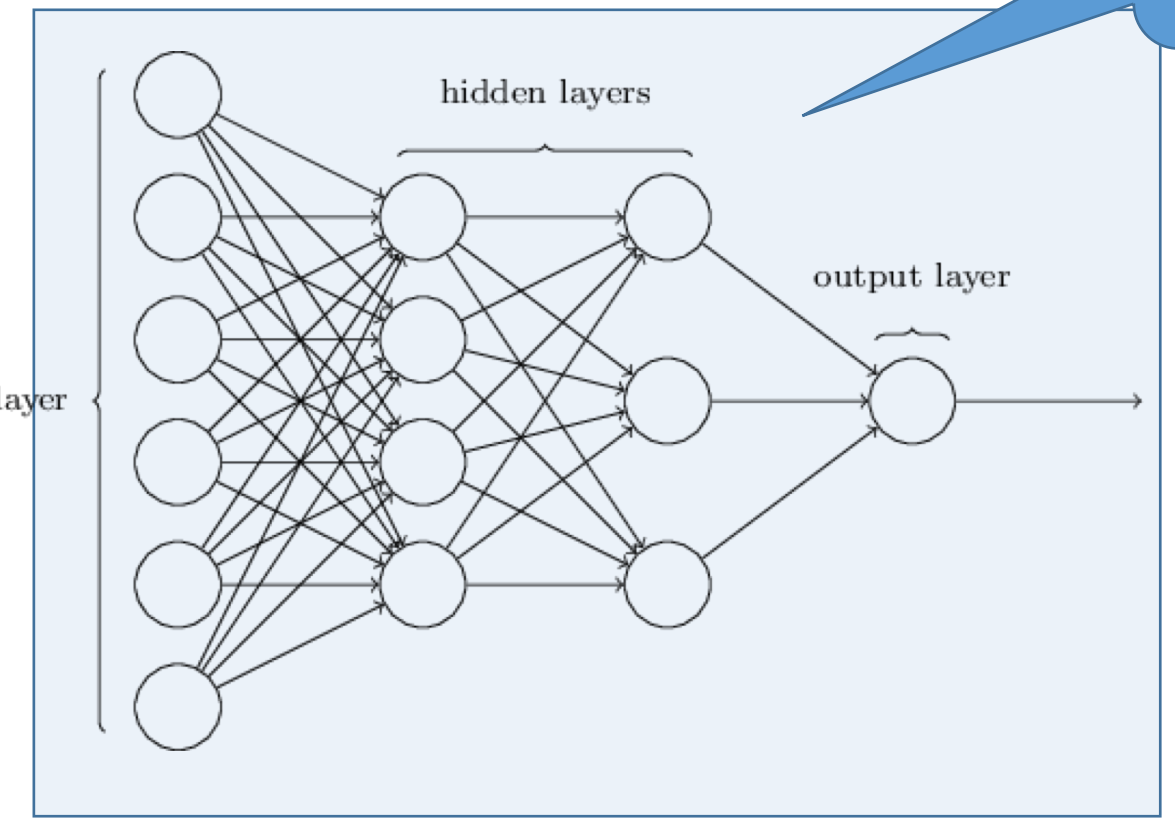
# Princeton-Google Brain team








Naman Agarwal, Brian Bullins, Xinyi Chen,
Karan Singh, Cyril Zhang, Yi Zhang




Deep net, SVM, boosted decision stump,…

Chair/car

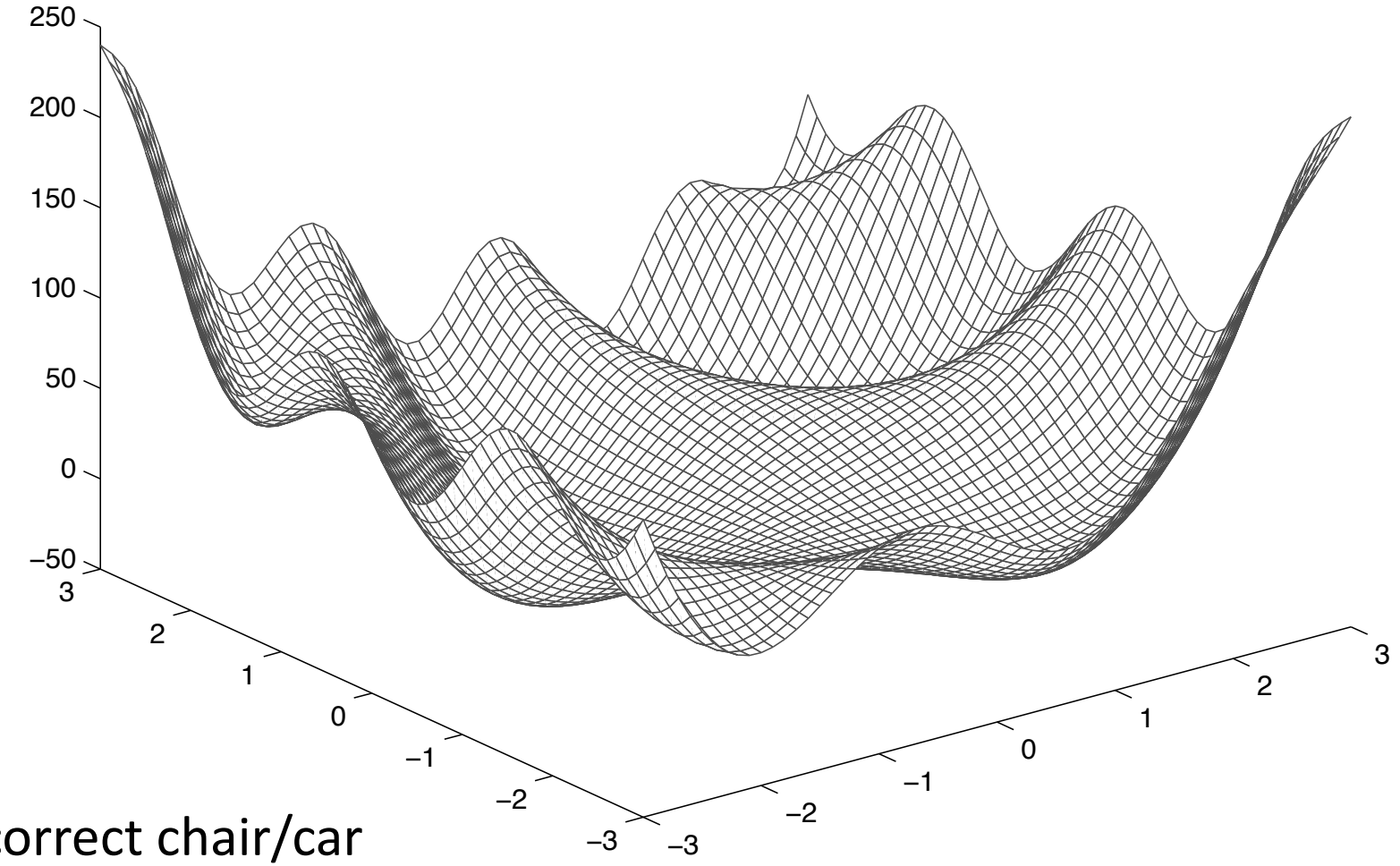
Function of vectors

$f_{weights}(a)$

Model parameters

Distribution over vectors

$\{a\} \in R^d$




Minimize incorrect chair/car predictions on training set

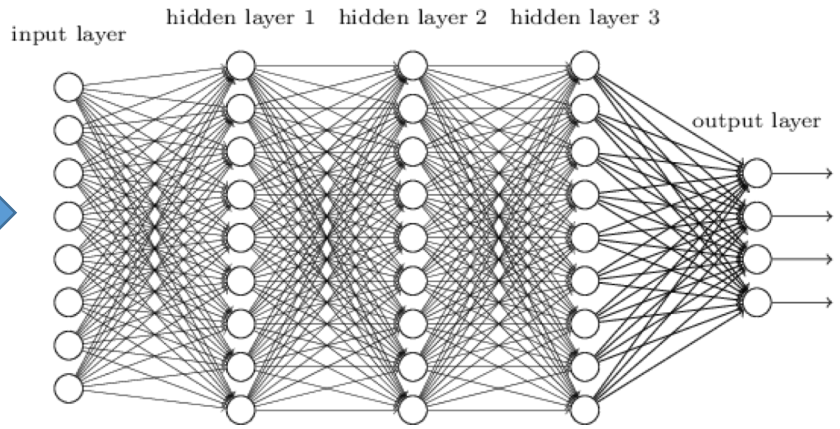
This talk: faster optimization

1. second order methods
2. adaptive regularization

# (Non-Convex) Optimization in ML

Model

Distribution over $\{a\} \in \mathbb{R}^n$




Label

$$\underset{x \in \mathbb{R}^d}{\text{minimize}}\ f(x), \qquad f(x) = \frac{1}{m}\sum_{i=1}^{m} \ell_i(x, a_i, b_i)$$

Training set size (m) & dimension of data (d) are **very large, days/weeks** to train

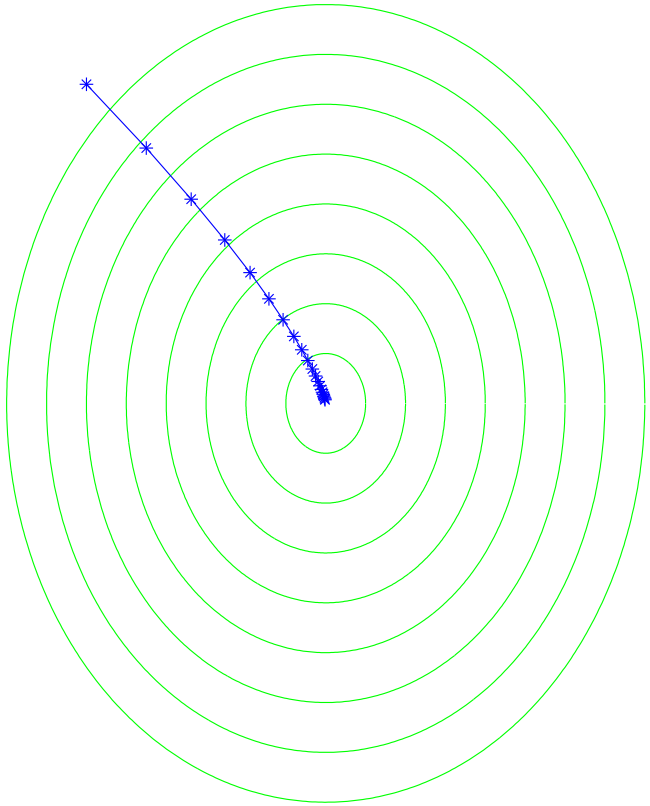
# Gradient Descent

Given first-order oracle: $\nabla f(x), \quad |\nabla f(x)| \leq G$

Iteratively: $x_{t+1} \leftarrow x_t - \eta \nabla f(x_t)$

**Theorem:** for smooth bounded functions, step size $\eta \sim O(1)$ (depends on smoothness),

$$\frac{1}{T}\sum_t \|\nabla f(x_t)\|^2 \sim \frac{1}{T}$$



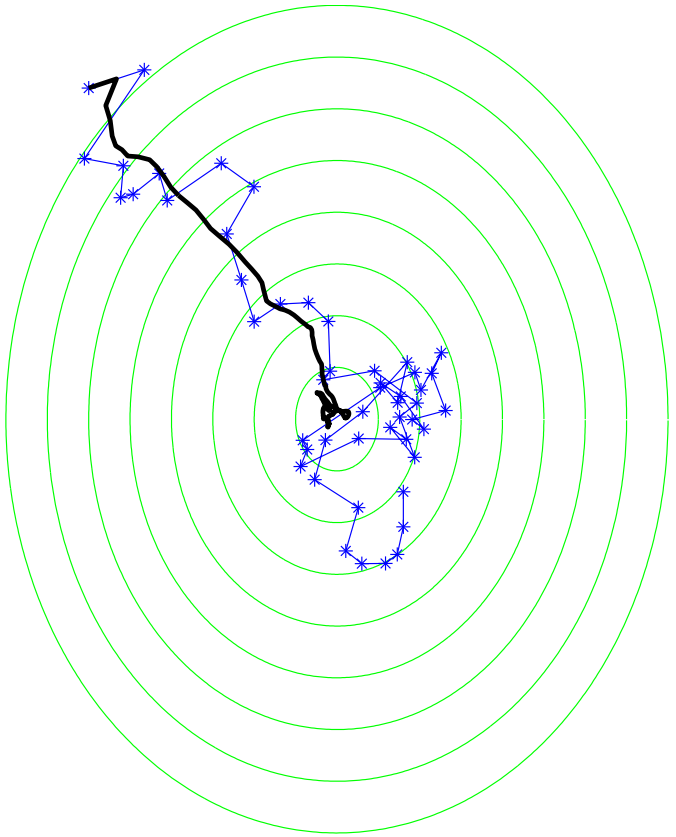
# Stochastic Gradient Descent [Robbins & Monro '51]

Given stochastic first-order oracle: $\mathbb{E}\left[\widetilde{\nabla f}(x)\right] = \nabla f(x), \quad \mathbb{E}\left[\left\|\widetilde{\nabla f}(x)\right\|^2\right] \leq \sigma^2$

Iteratively: $x_{t+1} \leftarrow x_t - \eta \widetilde{\nabla f}(x_t)$

**Theorem** [GL'15]: for smooth bounded functions, step size $\eta = \sqrt{\frac{1}{T\sigma^2}}$,

$$\frac{1}{T}\sum_t \|\nabla f(x_t)\|^2 \sim \sqrt{\frac{\sigma^2}{T}}$$

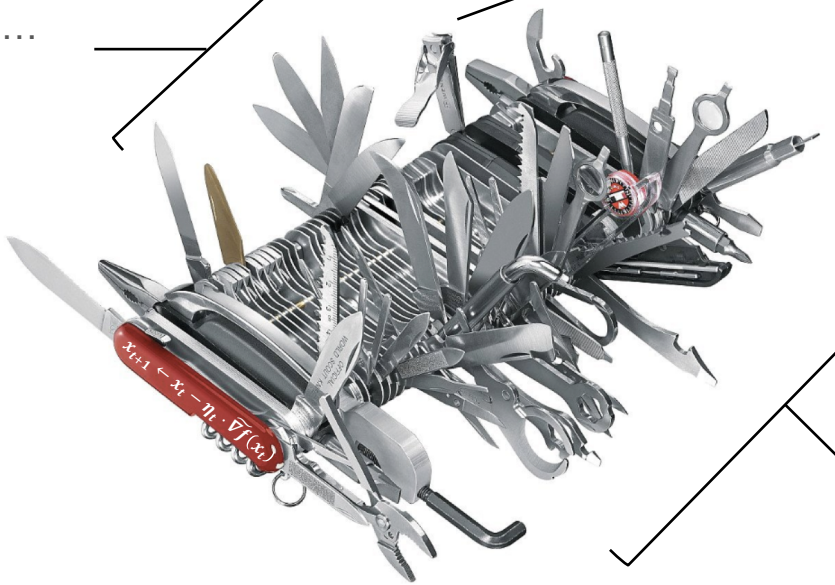


# SGD

# SGD++

**Variance Reduction**
[Le Roux, Schmidt, Bach ‘12]
…

**Momentum**
[Nesterov ‘83],…

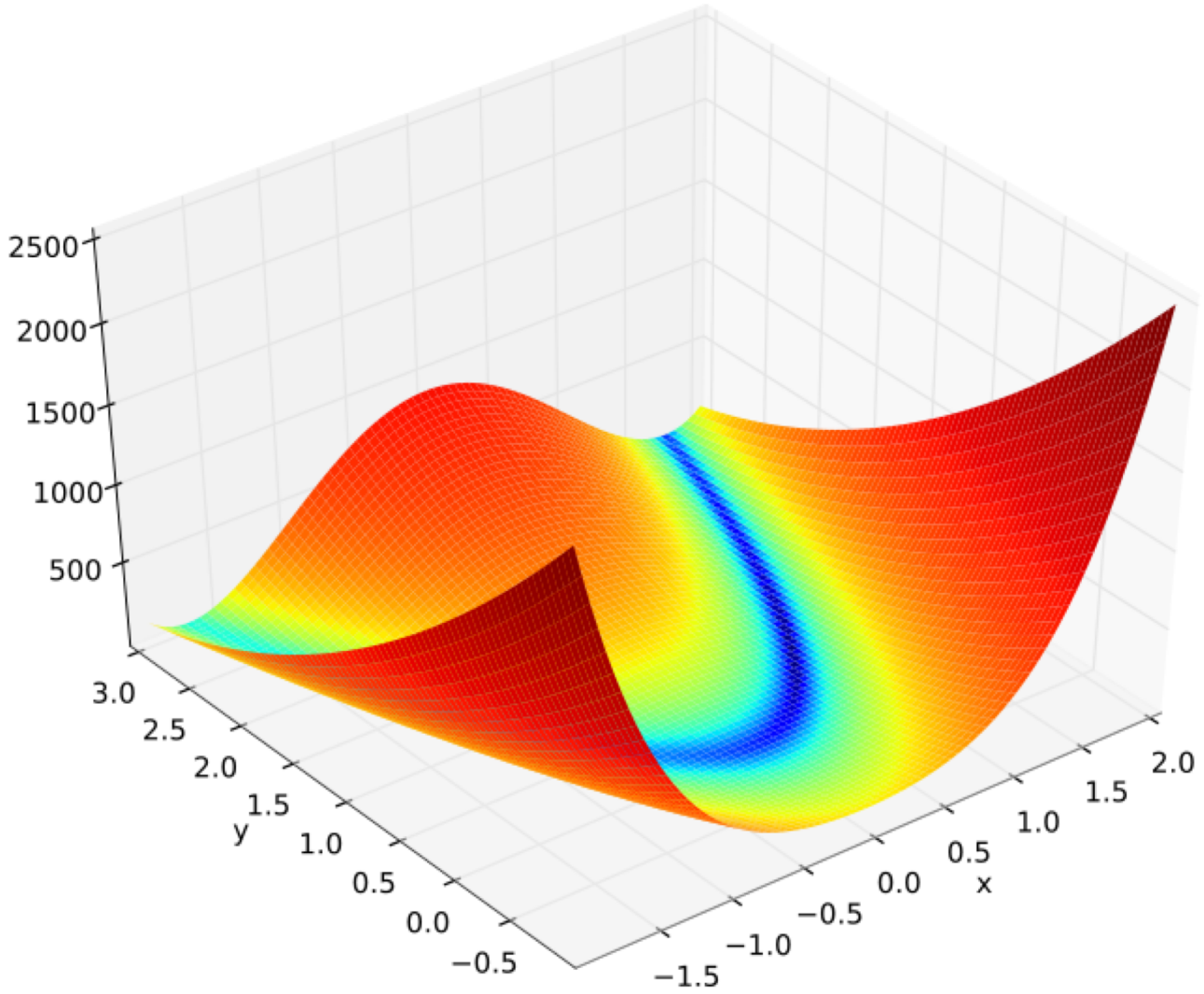



**Adaptive Regularization**
[Duchi, Hazan, Singer ‘10],…

Are we at the limit ?

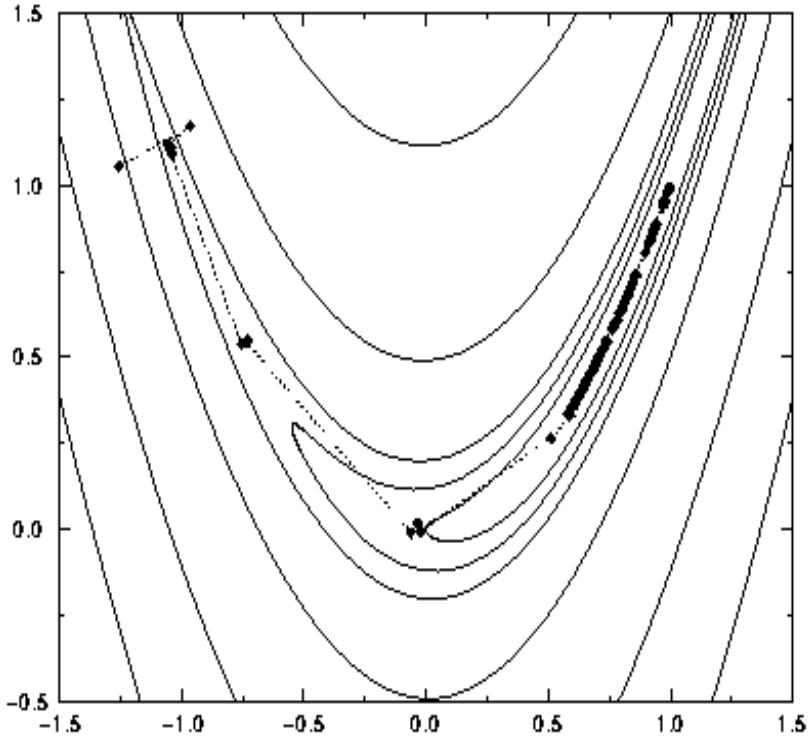
Woodworth,Srebro ‘16: yes!
(gradient methods)

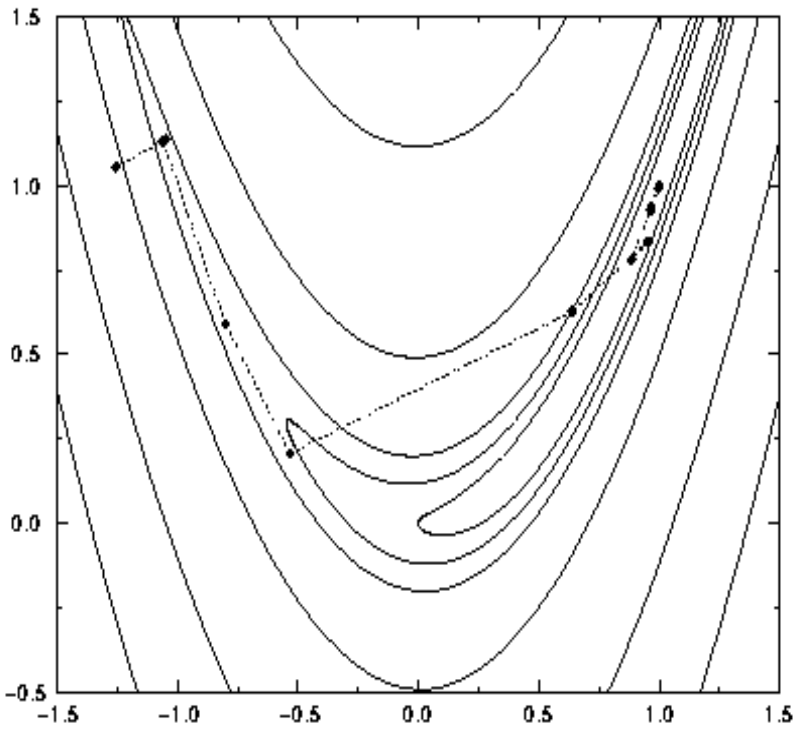
# Rosenbrock function



# Higher Order Optimization

- Gradient Descent – Direction of Steepest Descent
- Second Order Methods – Use Local Curvature

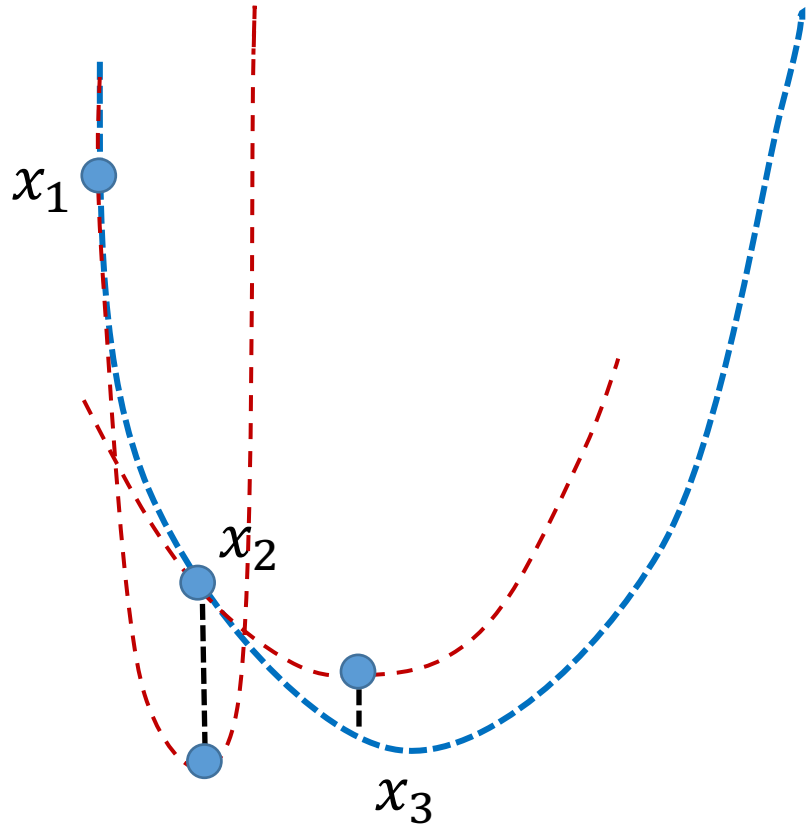


# Newton's method (+ Trust region)

$$x_{t+1} = x_t - \eta \left[\nabla^2 f(x)\right]^{-1} \nabla f(x)$$

For non-convex function: can move to ∞
Solution: solve a quadratic approximation in a local area (trust region)

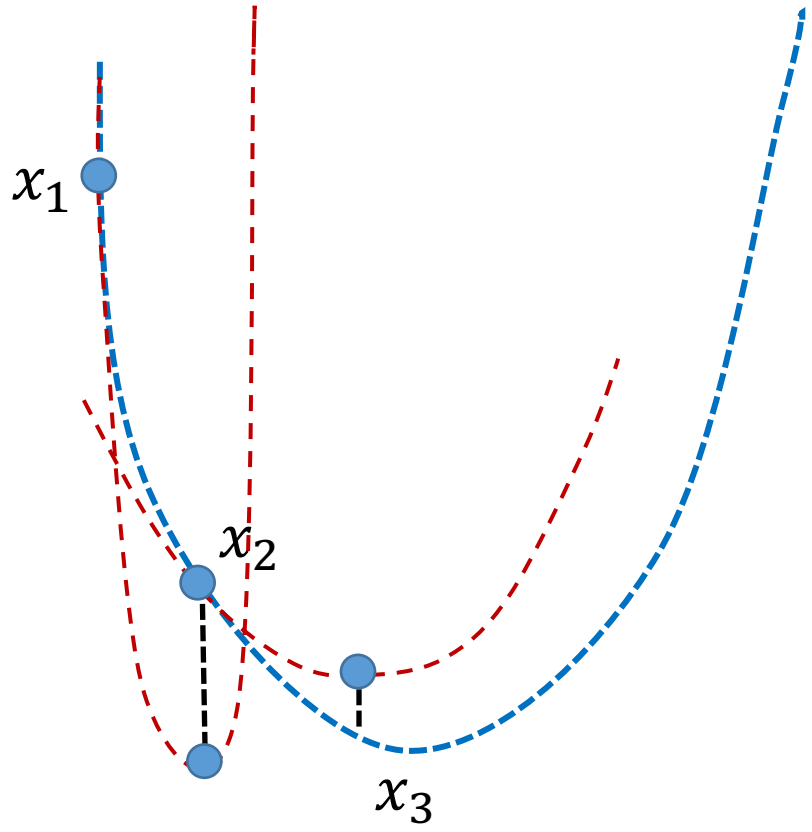


# Newton's method (+ Trust region)

$$x_{t+1} = x_t - \eta \left[\nabla^2 f(x)\right]^{-1} \nabla f(x)$$

1. $d^3$ time per iteration, Infeasible for ML!!
2. Stochastic difference of gradients ≠ hessian

Till recently ☺



# Speed up the Newton direction computation??

- Spielman-Teng ‘04: diagonally dominant systems of equations in linear time!
  - 2015 Godel prize
  - Used by Daitch-Speilman for faster flow algorithms
  - Faster/simpler by Srivasatva, Koutis, Miller, Peng, others…

- Erdogu-Montanari ‘15: low rank approximation & inversion by Sherman-Morisson
  - Allow stochastic information
  - Still prohibitive:  rank * $d^2$

# Our results – Part 1 of talk

- **Natural** Stochastic Newton Method
- Every iteration in **O(d)** time. Linear in **Input Sparsity**
- Couple with Matrix Sampling/ Sketching techniques - **Best known running time** for $m \gg d$ **for both convex and non-convex opt., provably faster than first order methods**

# Stochastic Newton?
# (convex case for illustration)

$$x_{t+1} = x_t - \eta \overbrace{[\nabla^2 f(x)]^{-1} \nabla f(x)}^{\tilde{n}}$$

- ERM, rank-1 loss: $\arg\min_{x} E_{\{i\sim m\}}[\ell(x^T a_i, b_i) + \frac{1}{2}|x|^2]$

- unbiased estimator of the Hessian:

$$\widetilde{\nabla^2} = \mathrm{a_i a_i^T} \cdot \ell'(x^T a_i, \mathrm{b_i}) + I \qquad i \sim U[1, \ldots, m]$$

- clearly $E\left[\widetilde{\nabla^2}\right] = \nabla^2 f$ , but $E\left[\widetilde{\nabla^2}^{-1}\right] \neq \nabla^2 f^{-1}$

# Circumvent Hessian creation and inversion!

- 3 steps:
  - (1) represent Hessian inverse as infinite series

$$\nabla^{-2} = \sum_{i=0\ to\ \infty} (I - \nabla^2)^i$$

For any distribution on naturals $i \sim N$

  - (2) sample from the infinite series (Hessian-gradient product) , ONCE

$$\nabla^2 f^{-1} \nabla f = \sum_i (I - \nabla^2 f)^i \nabla \mathrm{f} = E_{i \sim N} (I - \nabla^2 f)^i \nabla \mathrm{f} \cdot \frac{1}{\Pr[i]}$$

  - (3) estimate Hessian-power by sampling i.i.d. data examples

$$= \mathrm{E}_{i \sim N, k \sim [i]} \left[ \prod_{k=1\ to\ i} (I - \nabla^2 f_k) \nabla \mathrm{f} \cdot \frac{1}{\Pr[i]} \right]$$

Single example Vector-vector products only

# Improved Estimator

- Previously, Estimate a single term in one estimate
- Recursive Reformulation of the series

$$\boldsymbol{M_S^{-1} = I + (I - M)(I + (I - \widetilde{M_{S-1}})(I + \ldots))}$$

$\widetilde{M_{S-1}}$: Recursive estimate $M_{S-1}^{-1}$; $(I - M)$: Ind. Sample; $S \to \infty$

- Truncate after $S$ steps. Typically $S \sim \kappa$ (condition # of f)
- $E\left[\widetilde{M_S^{-1}}\right] \to M^{-1}$ as $S \to \infty$
- Repeat and average to reduce the variance

# LiSSA

## Linear-time Second-order Stochastic Algorithm

$$\arg\min_{x \in R^d} \; E_{\{i \sim m\}}[\ell(x^T a_i, y_i) + \frac{1}{2}|x|^2]$$

- Compute a full (large batch) gradient $\nabla f$
- Use the estimator $\widetilde{\nabla^{-2} f \nabla f}$ defined previously & move there

V is a bound on the variance of the estimator

- In Practice - a small constant (e.g. 1)
- In Theory - $V \leq \kappa^2$

Theorem 1: For large t, LiSSA returns a point in the parameter [illegible] $w_t$ s.t.

$$f(w_t) \leq \; f(w^*) + \epsilon$$

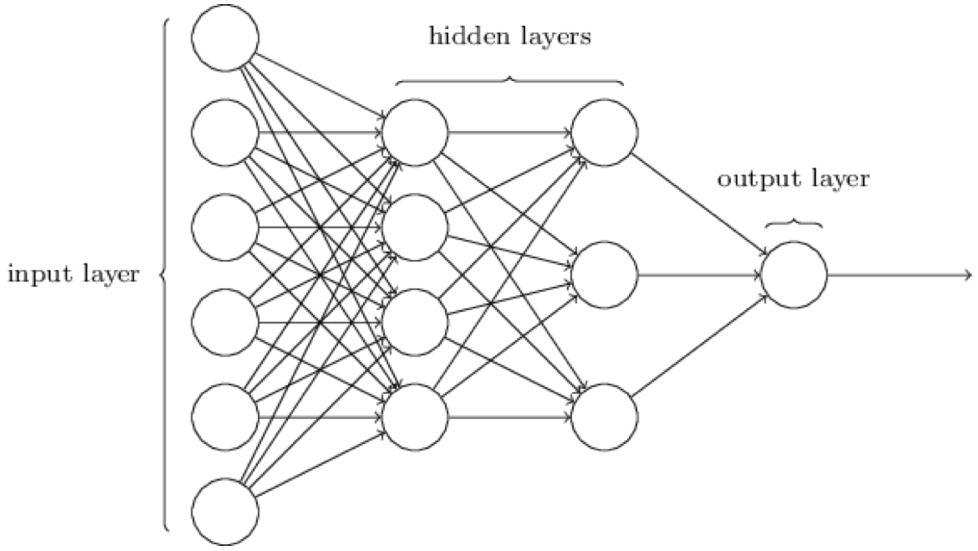
In total time $\log\left(\frac{1}{\epsilon}\right) \; d \; (m + O(\kappa) \; V \;)$

→ (w. more tricks) $\tilde{O}\left(\log^2\left(\frac{1}{\epsilon}\right) \; d \; \left(m + \sqrt{\kappa d}\right)\right)$, fastest known! (& provably faster 1st order WS '16)

# Hessian Vector Products for Neural Networks

in time $O(d)$ $(Perlmutter\ Trick)$

$$\nabla^2 f^{-1} \nabla f = \mathrm{E}_{i\sim N, k\sim[i]}\left[\prod_{k=1\ to\ i} (I - \nabla^2 f_k)\nabla \mathrm{f} \cdot \frac{1}{\Pr[i]}\right]$$




- $f_i$ - computed via a differentiable circuit of size $O(d)$
- $\nabla f_i$ - computed via a differentiable circuit of size O(d) (Backpropagation)
- Define $g_i(h) = \nabla f_i(h)^T v$

$$\nabla\, g(h) = \nabla^2 f_i(h)\, v$$

- There exists a $O(d)$ circuit computing $\nabla^2 f_i(h)v$

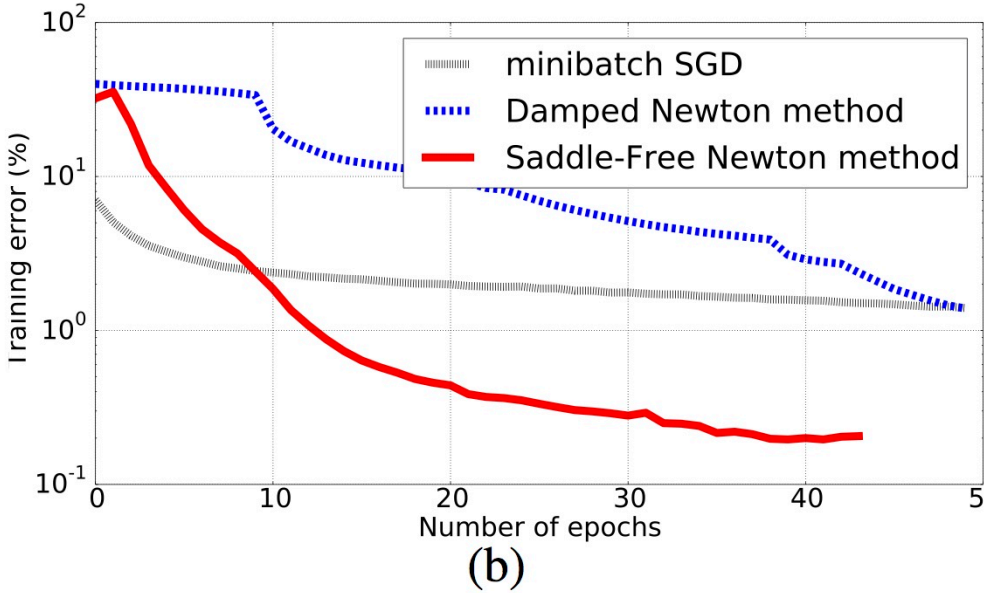
# LiSSA for non-convex (FastCubic)

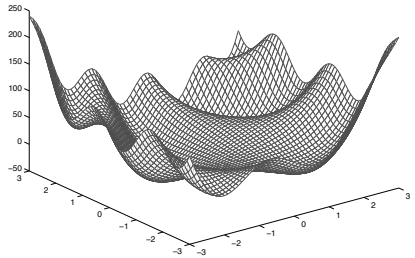
| Method | Time to $\|\| \nabla f(h) \|\| \leq \epsilon$ (Oracle) | Time to $\|\| \nabla f(h) \|\| \leq \epsilon$ (Actual) | Second Order? | Assumption |
|---|---|---|---|---|
| Gradient Descent (Folklore) | $O\left(\frac{T_g}{\epsilon^2}\right)$ | $O\left(\frac{md}{\epsilon^2}\right)$ | N/A | Smoothness |
| Stochastic Gradient Descent (Folklore) | $O\left(\frac{T_{sgd}}{\epsilon^4}\right)$ | $O\left(\frac{d}{\epsilon^4}\right)$ | N/A | Smoothness |
| Noisy SGD (Ge et al) | | $O\left(\frac{d^{C_1}}{\epsilon^4}\right)$ | $\nabla^2 f(h) \succcurlyeq -\epsilon^{\frac{1}{C_2}} I$ | Smoothness |
| Cubic Regularization (Nesterov & Polyak) | | $\tilde{O}\left(\frac{n\, d^{\omega-1} + d^{\omega}}{\epsilon^{1.5}}\right)$ | $\nabla^2 f(h) \succcurlyeq -\epsilon^{\frac{1}{2}} I$ | Smooth and Second Order Lipschitz |
| Fast Cubic | $O\left(\frac{T_g}{\epsilon^{1.5}} + \frac{T_h}{\epsilon^{1.75}}\right)$ | $O\left(\frac{md}{\epsilon^{1.75}}\right)$ | $\nabla^2 f(h) \succcurlyeq -\epsilon^{\frac{1}{2}} I$ | Smooth and Second Order Lipschitz |

# 2nd order information: new phenomena?

- ”Computational lens for deep nets”:
  experiment with 2nd order information…
  - Trust region
  - Cubic regularization, eigenvalue methods….

- Multiple hurdles:
  - Global optimization is NP-hard, even deciding whether you are at a local minimum is NP-hard
  - Goal: local minimum $|| \nabla f(h)|| \leq \epsilon$
    and $\nabla^2 f(h) \succcurlyeq -\sqrt{\epsilon}\, I$

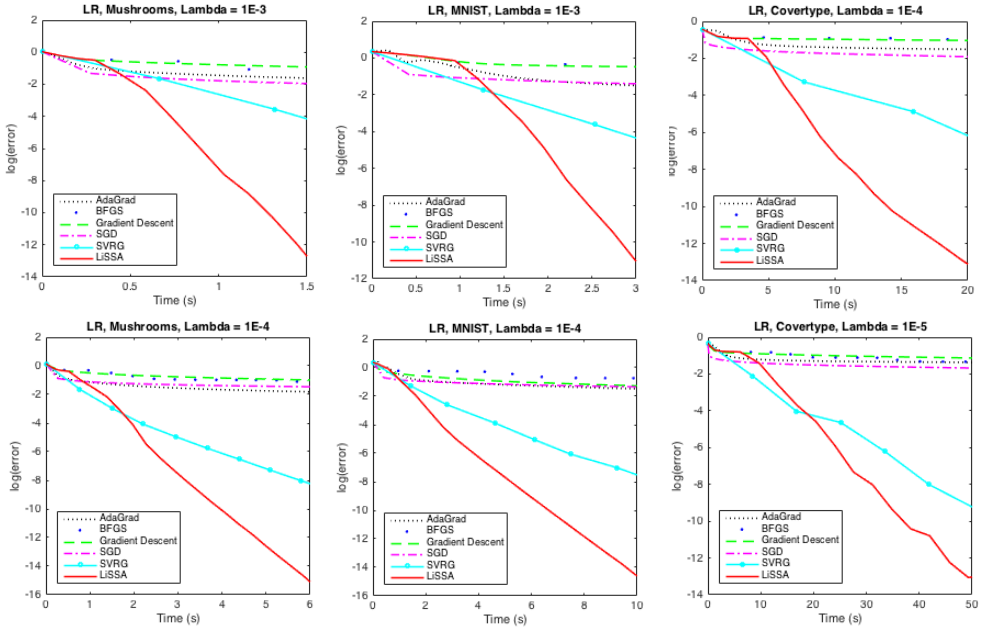


Bengio-group experiment

# Experimental Results

Convex: clear improvements




Neural networks: doesn't improve upon SGD

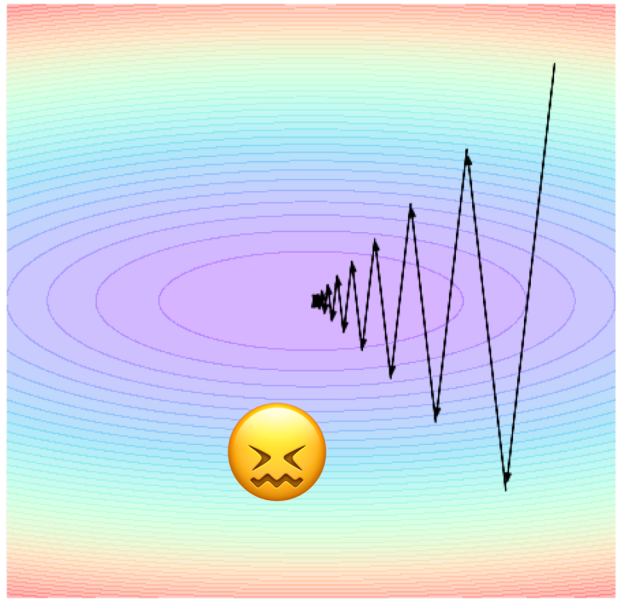
What goes wrong?

# Adaptive Regularization Strikes Back

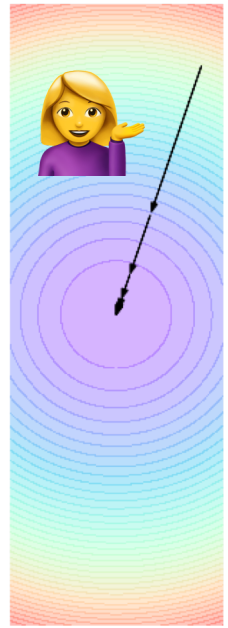

Princeton Google Brain team: Naman Agarwal, Brian Bullins, Xinyi Chen, Elad Hazan, Karan Singh, Cyril Zhang, Yi Zhang

# Adaptive Preconditioning

- Newton's method special case of preconditioning: make loss surface more isotropic



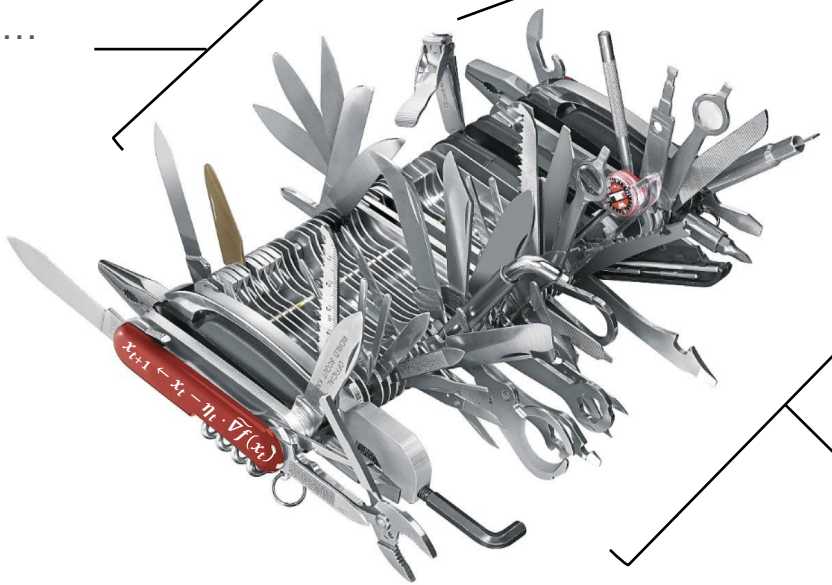
$$f(x) \mapsto f(Dx)$$



# Modern ML is SGD++

**Variance Reduction**
[Le Roux, Schmidt, Bach ‘12]
…

**Momentum**
[Nesterov ‘83],…

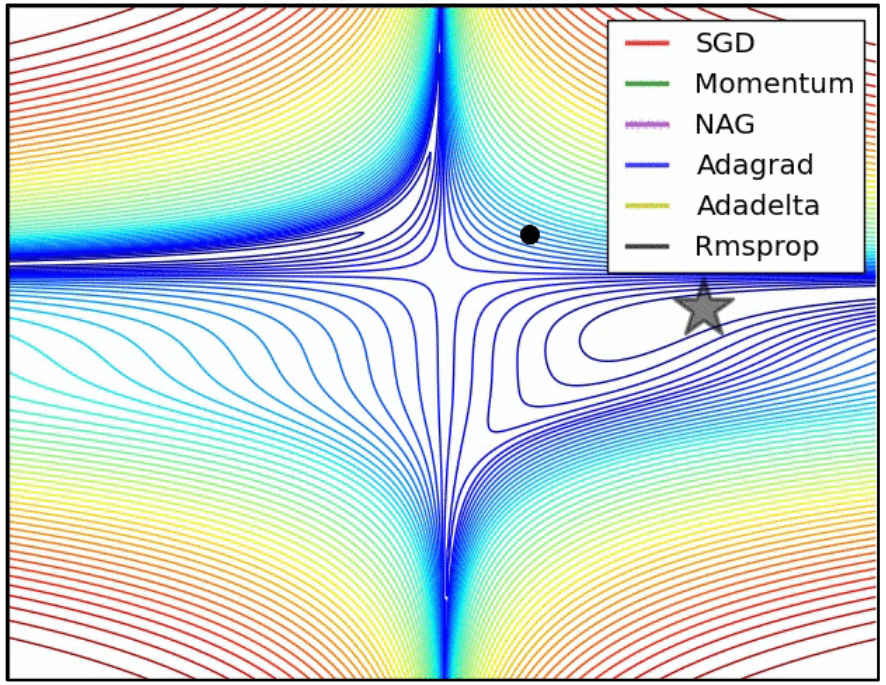



**Adaptive Regularization**
[Duchi, Hazan, Singer ‘10],…

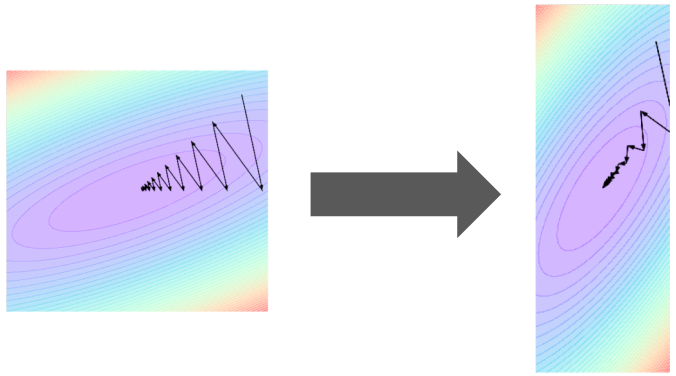
# Adaptive Optimizers

- Each coordinate $x[i]$ gets a learning rate $\mathbf{D}_t[i]$
  $\mathbf{D}_t[i]$ chosen "adaptively" using $\widetilde{\nabla f}(x_{1:t})[i]$ - $g_{1:t}[i]$

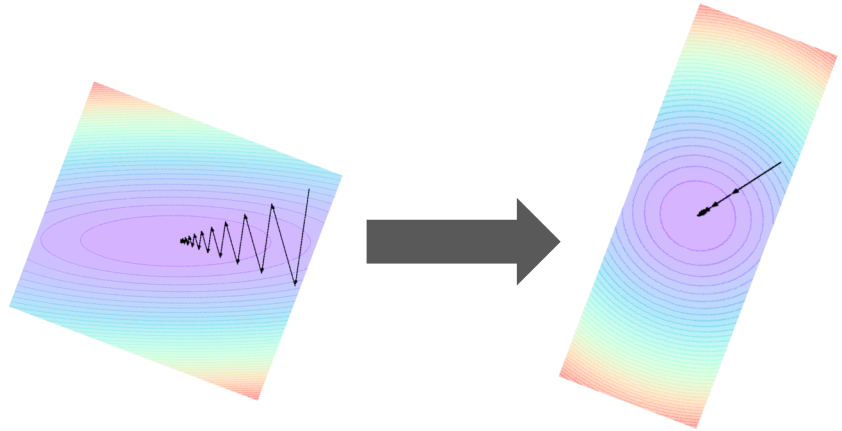
- **AdaGrad:** $D_t[i] \coloneqq \frac{1}{\sqrt{\sum_{s=1}^{t}(g_s[i])^2}}$
- **RMSprop:** $D_t[i] \coloneqq \frac{1}{\sqrt{\sum_{s=1}^{t}\beta^{t-s}\,(g_s[i])^2}}$
- **Adam:** $D_t[i] \coloneqq \frac{1}{(1-\beta^t)\sqrt{\sum_{s=1}^{t}\beta^{t-s}\,(g_s[i])^2}}$



# What about the *other* AdaGrad?



**diagonal** preconditioning
$O(d)$ time per iteration



**full-matrix** preconditioning
$> O(d^2)$ time per iteration

$$x_{t+1} \leftarrow x_t - diag\left[\sum_{s=1}^{t} g_s g_s^\top\right]^{-1/2} \cdot g_t$$

$$x_{t+1} \leftarrow x_t - \left[\sum_{s=1}^{t} g_s g_s^\top\right]^{-1/2} \cdot g_t$$

# What does adaptive regularization even *do*?!

- Convex, full-matrix case: [Duchi-Hazan-Singer '10]: "best regularization in hindsight"

$$\sum_t g_t(x_t - x^*) = O\left(\min_{Tr(A)=d}\left\{\sum_t |g_t|_A^2\right\}\right)$$

- Diagonal version: up to $\frac{1}{\sqrt{d}}$ improvement upon SGD (in optimization AND generalization)

- No analysis for *non-convex* optimization, till recently (still no **speedup** vs. SGD)
  - Convergence: [Li, Orabona '18], [Ward, Wu, Bottou '18]

# The Case for Full-Matrix Adaptive Regularization

- **GGT**, a new adaptive optimizer
- Efficient full-matrix (low-rank) AdaGrad

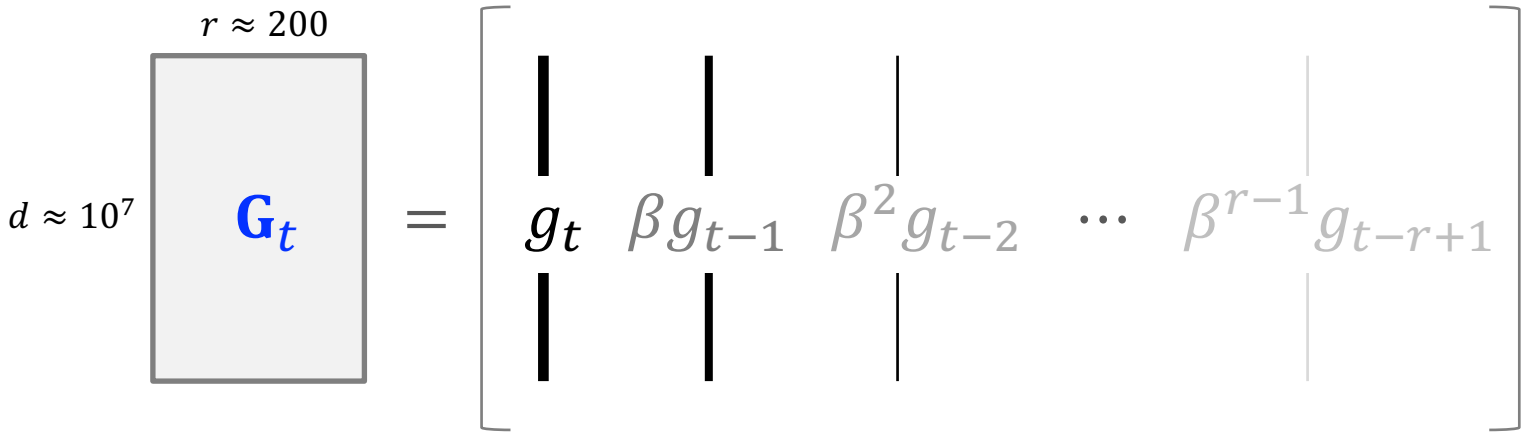
- **Theory:** “Adaptive” convergence rate on convex & non-convex $f$

  **Up to $O\left(\frac{1}{\sqrt{d}}\right)$ faster than SGD!**

- **Experiments:** viable in the deep learning era
- GPU-friendly; not much slower than SGD on deep models
- Accelerates training in deep learning benchmarks
- Empirical insights on anisotropic loss surfaces, real and synthetic

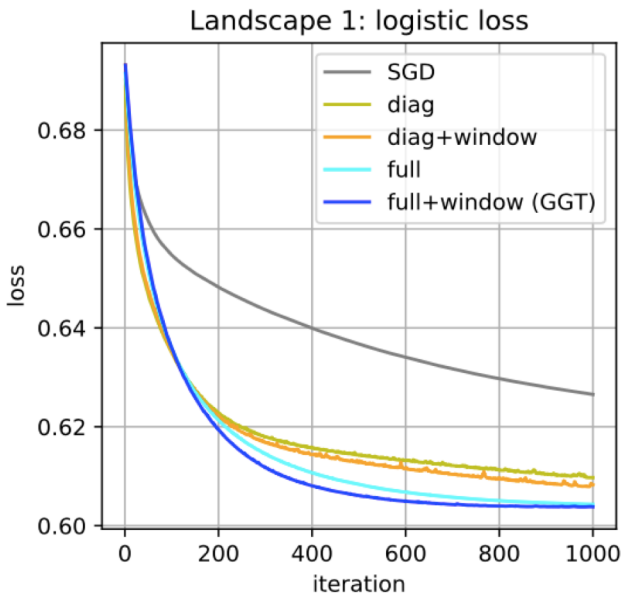
# The GGT Algorithm

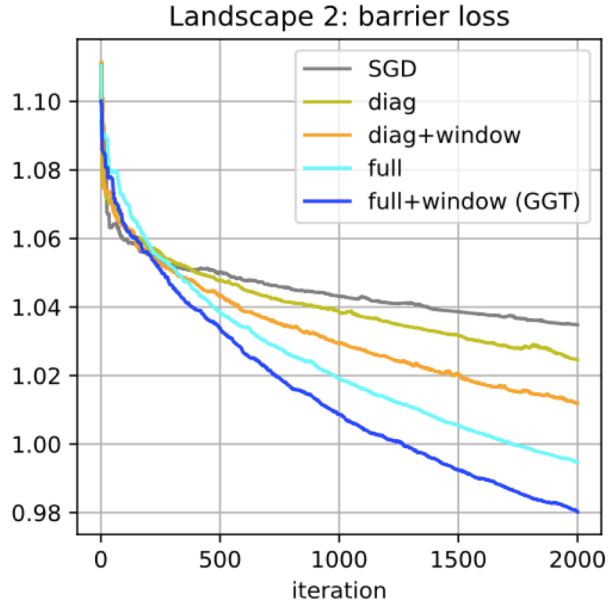
- **SGD:** $x_{t+1} \leftarrow x_t - \eta_t \cdot g_t$
- **AdaGrad:** $x_{t+1} \leftarrow x_t - [\mathrm{diag}(\sum_{s=1}^{t} g_s^2)]^{-1/2} \cdot g_t$
- **Full-Matrix AdaGrad:** $x_{t+1} \leftarrow x_t - [\sum_{s=1}^{t} g_s g_s^\top]^{-1/2} \cdot g_t$
- **GGT:** $x_{t+1} \leftarrow x_t - [\mathbf{G}_t \mathbf{G}_t^\top]^{-1/2} \cdot g_t$



# Why a low-rank preconditioner?

- **Answer 1:** want to forget stale gradients (like Adam)
- **Synthetic experiments:** logistic regression, polytope analytic center

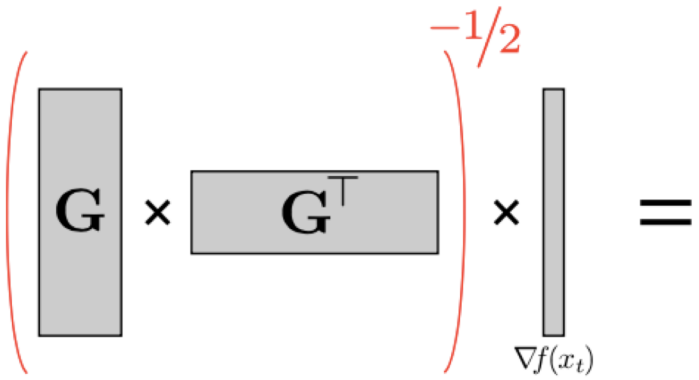


# The GGT speedup

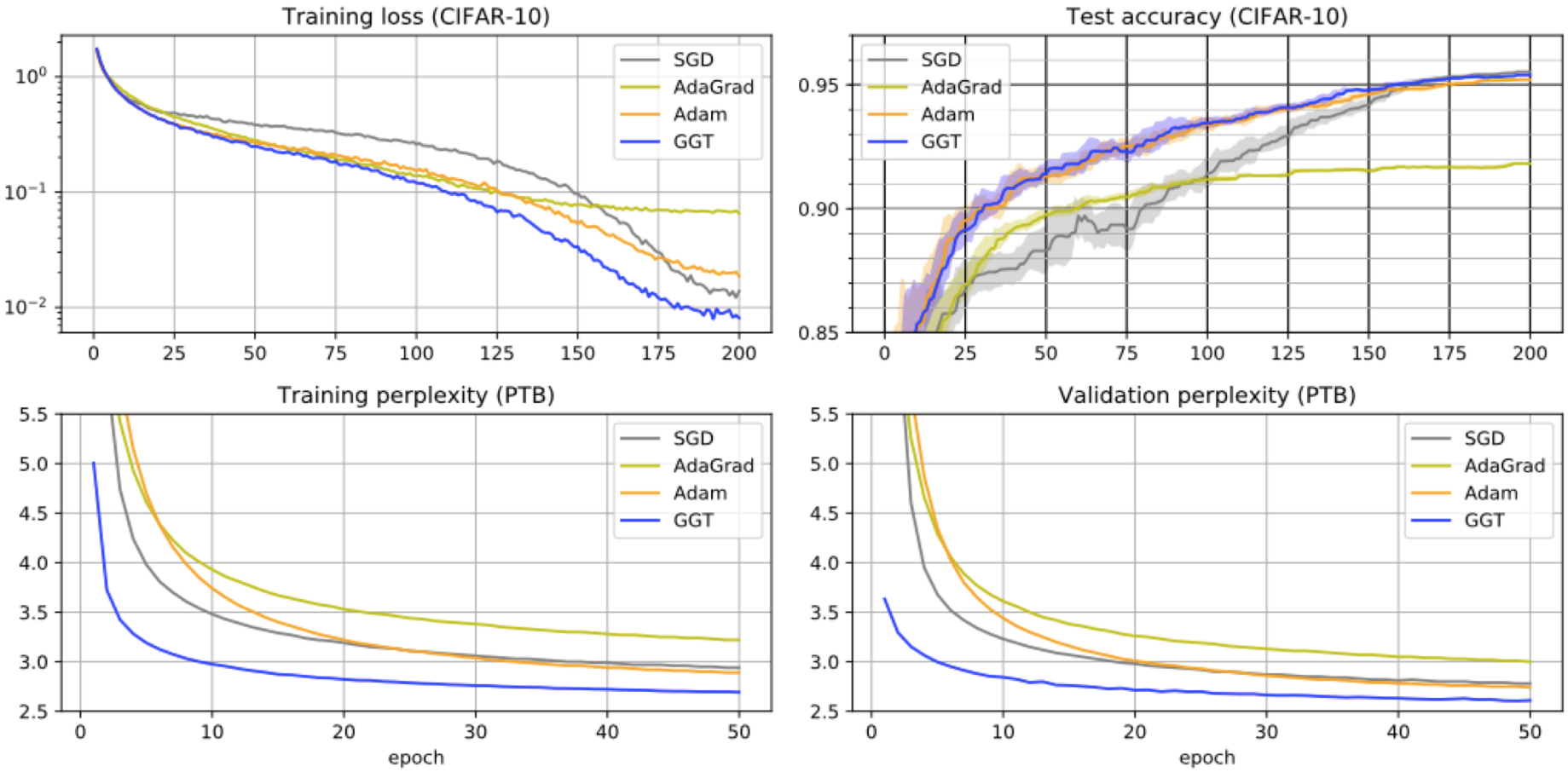
$$(\boldsymbol{a} \times \boldsymbol{a})^{-\frac{\mathbf{1}}{\mathbf{2}}} = \boldsymbol{a} \times (\boldsymbol{a} \times \boldsymbol{a})^{-\frac{\mathbf{3}}{\mathbf{2}}} \times \boldsymbol{a}$$

# The GGT speedup

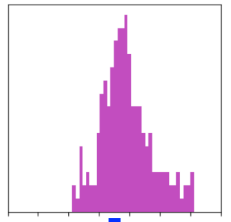



**Matrix ops:** $O(rd^2)$
**Huge SVD:** $O(d^3)$

# Large-Scale Experiments (CIFAR-10, PTB)

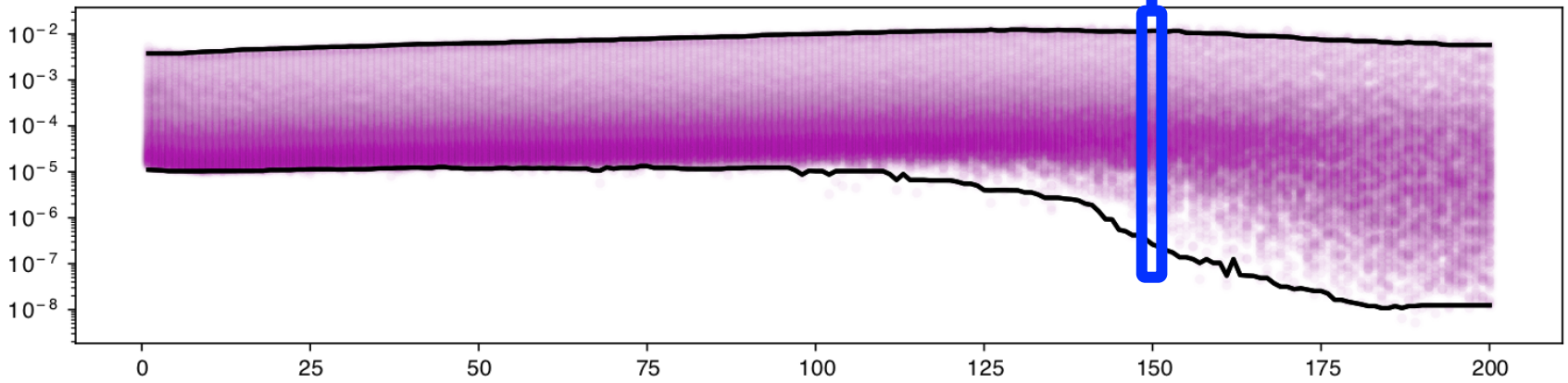


# Visualizing Gradient Spectra

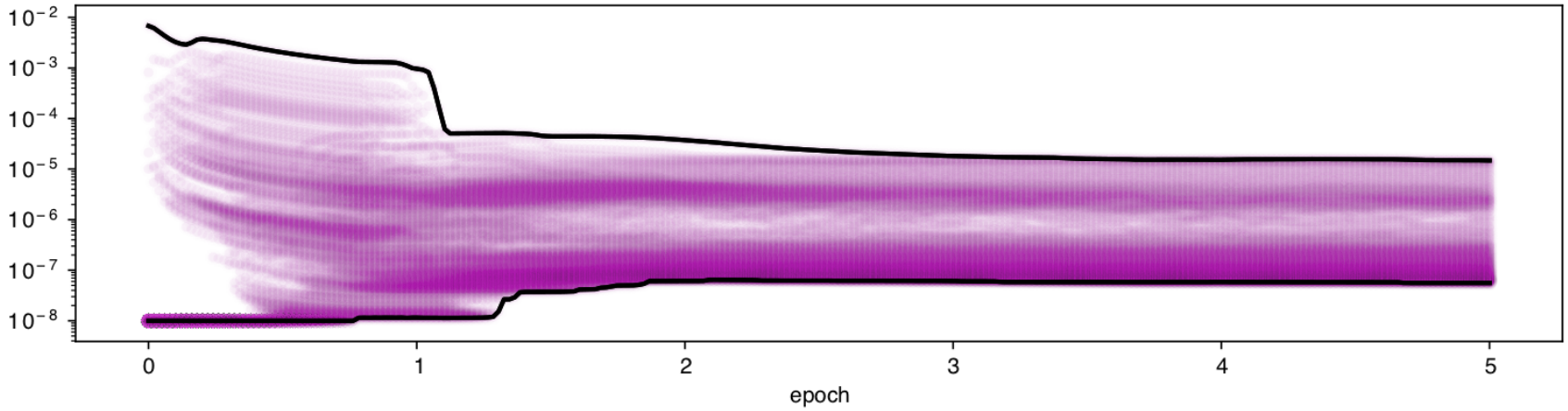


$\text{eigs}(\mathbf{G}_t^\top \mathbf{G}_t)$
$@\ t = 150$

**26-layer ResNet**
CIFAR-10

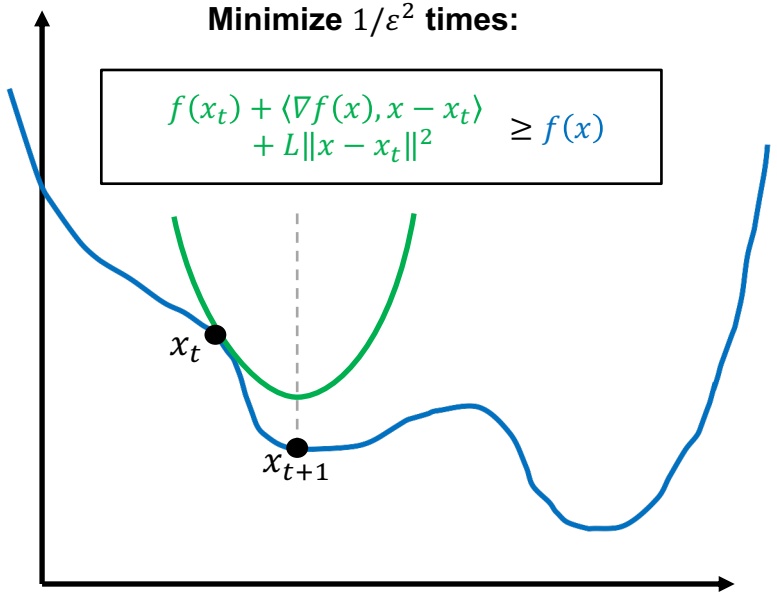


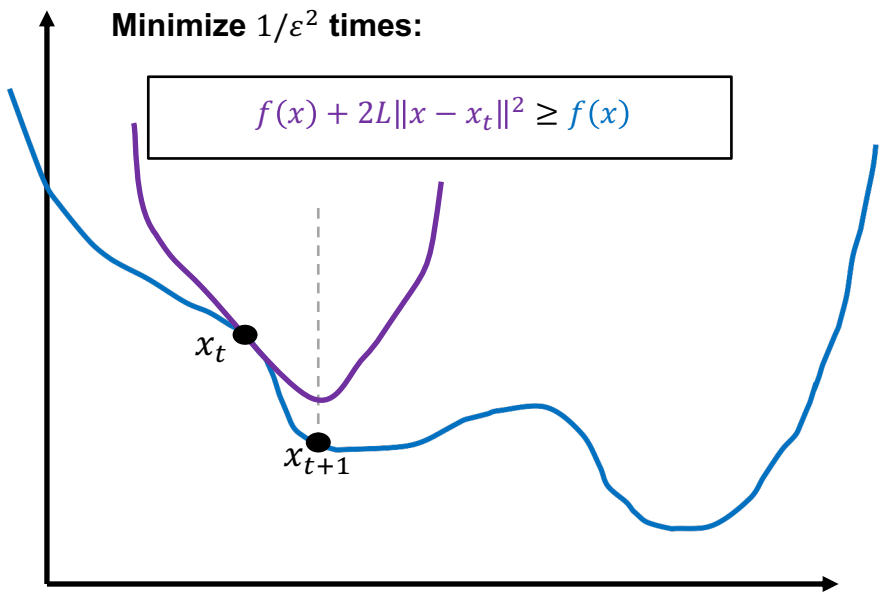

**3-layer LSTM**
Penn Treebank
(char-level)



# Theory: faster convergence vs. non-convex SGD

- **Convex:** $f(x_T) \leq \operatorname{argmin}_x f(x) + \varepsilon$ in $O\left(\frac{\sigma^2}{\varepsilon^2}\right)$ steps
- **Non-convex:** $\exists t: \|\nabla f(x_t)\| \leq \varepsilon$ within $O\left(\frac{1}{\varepsilon^2}\right)$ **convex epochs**
- **Reduction via modified descent lemma:**



# The Ratio of Adaptivity

- Define the *adaptivity ratio* $\mu$:

$$\mu^2 \coloneqq \frac{\sum_{t=1}^{T}\left\|g_t^{AG}\right\|^2}{\sum_{t=1}^{T}\left\|g_t^{SGD}\right\|^2} = \frac{\text{AdaGrad regret}}{\text{worst-case OGD regret}}$$

- [DHS10]: $\mu \leq \left[\frac{1}{\sqrt{d}}, \sqrt{d}\right]$ for diag-AdaGrad, sometimes smaller for full AdaGrad
- **Strongly convex losses:** GGT* converges in $\tilde{O}\left(\frac{\mu^2\sigma^2}{\varepsilon}\right)$ steps
- **Non-convex reduction:** GGT* converges in $\tilde{O}\left(\frac{\mu^2\sigma^2}{\varepsilon^4}\right)$ steps
- First step towards analyzing adaptive methods in non-convex optimization

# A note on the important parameters

- *A lot* of work on improving dependence on $\varepsilon$

- Recent state-of-the-art in SGD++: $\frac{1}{\varepsilon^4} \to \frac{1}{\varepsilon^{3.5}}$

- In practice: $\epsilon \sim 0.1$, improvement amounts to factor $\sqrt{10} \sim 3.1$

- Our improvement $\frac{1}{\epsilon^4} \to \frac{\mu^2}{\epsilon^4}$ : **can be as large as $d$**
  ($d \sim 10^7$ for language models!)

- **Huge untapped potential: characterize the ratio of adaptivity!**

# Summary

1. Special characteristics of stochastic optimization in ML
2. Second order methods in linear time
   - LiSSA : fastest running time for convex ML
   - Non-convex – different solution concept
     FastCubic: faster than gradient descent!
3. Adaptive regularization strikes again:
   - full-matrix AR in linear time
   - Dimension-scale improvements possible, visible in experiments
4. Opportunity to improve factors of **dimension** rather than **approximation**

Thank you !