Optimization for Machine Learning: Beyond Stochastic Gradient Descent

References and more info:

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
Function of vectors $f_{weights}(a)$

Distribution over vectors $\{a\} \in \mathbb{R}^d$

Deep net, SVM, boosted decision stump,...
What is Optimization

But generally speaking...

We're screwed.

▶ Local (non global) minima of $f$

▶ All kinds of constraints (even restricting to continuous functions):
  
  $h(x) = \sin(2\pi x) = 0$

Minimize incorrect chair/car predictions on training set

This talk: faster optimization

1. second order methods
2. adaptive regularization
(Non-Convex) Optimization in ML

\[ \text{minimize } f(x), \quad f(x) = \frac{1}{m} \sum_{i=1}^{m} \ell_i(x, a_i, b_i) \]

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

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

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: $\mathbb{E}[\nabla f(x)] = \nabla f(x), \quad \mathbb{E}\left[\|\nabla f(x)\|^2\right] \leq \sigma^2$

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

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

$$\frac{1}{T} \sum_t \|\nabla f(x_t)\|^2 \sim \frac{\sigma^2}{\sqrt{T}}$$
$x_{t+1} \leftarrow x_t - \eta_t \cdot \nabla f(x_t)$
SGD++

Are we at the limit?

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

Variance Reduction
[Le Roux, Schmidt, Bach ‘12]
...

Momentum
[Nesterov ‘83], ...

Adaptive Regularization
[Duchi, Hazan, Singer ‘10], ...

Are we at the limit?
Rosenbrock function
Higher Order Optimization

• Gradient Descent – Direction of Steepest Descent
• Second Order Methods – Use Local Curvature
Newton’s method (+ Trust region)

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

For non-convex function: can move to \( \infty \)
Solution: solve a quadratic approximation in a local area (trust region)
Newton’s method (+ Trust region)

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

1. \( d^3 \) time per iteration, Infeasible for ML!!
2. Stochastic difference of gradients \( \neq \) 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-Spielman 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 * d²
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)

• 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:

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

• clearly \( E[\tilde{\nabla}^2] = \nabla^2 f \), but \( E\left[\tilde{\nabla}^2^{-1}\right] \neq \nabla^2 f^{-1} \)
Circumvent Hessian creation and inversion!

• 3 steps:
  • (1) represent Hessian inverse as infinite series
    \[ \nabla^{-2} = \sum_{i=0 \text{ to } \infty} (I - \nabla^2)^i \nabla^{-1} = E_{i \sim N} (I - \nabla^2)^i \nabla \cdot \frac{1}{\Pr[i]} \]

  • (2) sample from the infinite series (Hessian-gradient product), ONCE
    \[ \nabla^2 f^{-1} \nabla f = \sum_i (I - \nabla^2 f)^i \nabla f = E_{i \sim N} (I - \nabla^2 f)^i \nabla f \cdot \frac{1}{\Pr[i]} \]

  • (3) estimate Hessian-power by sampling i.i.d. data examples
    \[ = E_{i \sim N, k \sim [i]} \left[ \prod_{k=1 \text{ to } i} (I - \nabla^2 f_k) \nabla f \cdot \frac{1}{\Pr[i]} \right] \]
Improved Estimator

• Previously, Estimate a single term in one estimate

• Recursive Reformulation of the series

\[ M_{S}^{-1} = I + (I - M)(I + \left( \sum_{k=1}^{\infty} (I - M) \right)) \]

Ind. Sample \quad Recursive estimate \quad \lim_{S \to \infty} M_{S-1}

• Truncate after \( S \) steps. Typically \( S \sim \kappa \) (condition # of f)

• \( E \left[ M_{S}^{-1} \right] \to M^{-1} \) as \( S \to \infty \)

• Repeat and average to reduce the variance
LiSSA
Linear-time Second-order Stochastic Algorithm

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

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

**Theorem 1:** For large \( t \), LiSSA returns a point in the parameter space \( w_t \) s.t.

\[ f(w_t) \leq f(w^*) + \epsilon \]

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

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

\( V \) is a bound on the variance of the estimator
- In Practice - a small constant (e.g. 1)
- In Theory - \( V \leq \kappa^2 \)
Hessian Vector Products for Neural Networks

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

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

• $f_i$ - computed via a differentiable circuit of size $O(d)$

• $\nabla f_i$ - computed via a differentiable circuit of size $O(d)$ (Backpropagation)

• Define $g_i(h) = \nabla f_i(h)^T \nu$

\[
\nabla g(h) = \nabla^2 f_i(h) \nu
\]

• There exists a $O(d)$ circuit computing $\nabla^2 f_i(h)\nu$
# LiSSA for non-convex (FastCubic)

<table>
<thead>
<tr>
<th>Method</th>
<th>Time to $|\nabla f(h)| \leq \epsilon$ (Oracle)</th>
<th>Time to $|\nabla f(h)| \leq \epsilon$ (Actual)</th>
<th>Second Order?</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Descent (Folklore)</td>
<td>$O\left(\frac{T_g}{\epsilon^2}\right)$</td>
<td>$O\left(\frac{md}{\epsilon^2}\right)$</td>
<td>N/A</td>
<td>Smoothness</td>
</tr>
<tr>
<td>Stochastic Gradient Descent (Folklore)</td>
<td>$O\left(\frac{T_{sgd}}{\epsilon^4}\right)$</td>
<td>$O\left(\frac{d}{\epsilon^4}\right)$</td>
<td>N/A</td>
<td>Smoothness</td>
</tr>
<tr>
<td>Noisy SGD (Ge et al)</td>
<td>$O\left(\frac{d^{C_1}}{\epsilon^4}\right)$</td>
<td></td>
<td></td>
<td>Smoothness</td>
</tr>
<tr>
<td>Cubic Regularization (Nesterov &amp; Polyak)</td>
<td>$O\left(n\frac{d^{\omega-1} + d^\omega}{\epsilon^{1.5}}\right)$</td>
<td>$\nabla^2 f(h) \succeq -\frac{1}{\epsilon^2} I$</td>
<td></td>
<td>Smooth and Second Order Lipschitz</td>
</tr>
<tr>
<td>Fast Cubic</td>
<td>$O\left(\frac{T_g}{\epsilon^{1.5}} + \frac{T_h}{\epsilon^{1.75}}\right)$</td>
<td>$O\left(\frac{md}{\epsilon^{1.75}}\right)$</td>
<td>$\nabla^2 f(h) \succeq -\frac{1}{\epsilon^2} I$</td>
<td>Smooth and Second Order Lipschitz</td>
</tr>
</tbody>
</table>
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 \( \| \nabla f(h) \| \leq \varepsilon \) and \( \nabla^2 f(h) \preceq -\sqrt{\varepsilon} I \)

---

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
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 $D_t[i]
- D_t[i]$ chosen “adaptively” using $\sqrt[\eta]{f(x_{1:t})[i] - g_{1:t}[i]}

- **AdaGrad:** $D_t[i] := \frac{1}{\sqrt{\sum_{s=1}^{t}(g_s[i])^2}}$
- **RMSprop:** $D_t[i] := \frac{1}{\sqrt{\sum_{s=1}^{t} \beta^{t-s} (g_s[i])^2}}$
- **Adam:** $D_t[i] := \frac{1}{(1-\beta^t)\sqrt{\sum_{s=1}^{t} \beta^{t-s} (g_s[i])^2}}$
What about the *other* AdaGrad?

**diagonal** preconditioning
\[ O(d) \text{ time per iteration} \]

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

**full-matrix** preconditioning
\[ > O(d^2) \text{ time per iteration} \]

\[
x_{t+1} \leftarrow x_t - \left[ \sum_{s=1}^{t} g_s g_s^\top \right]^{-1/2} \cdot g_t
\]
What does adaptive regularization even do?!

- Convex, full-matrix case: [Duchi-Hazan-Singer ‘10]: “best regularization in hindsight”

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

- 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**: \( x_{t+1} \leftarrow x_t - \left[ \text{diag} \left( \sum_{s=1}^{t} g_s^2 \right) \right]^{-1/2} \cdot g_t \)
- **Full-Matrix AdaGrad**: \( x_{t+1} \leftarrow x_t - \left[ \sum_{s=1}^{t} g_s g_s^T \right]^{-1/2} \cdot g_t \)
- **GGT**: \( x_{t+1} \leftarrow x_t - \left[ G_t G_t^T \right]^{-1/2} \cdot g_t \)

![Diagram](image)

\[ \begin{aligned}
  r &\approx 200 \\
  d &\approx 10^7 \\
  G_t &\approx \begin{bmatrix}
  g_t & \beta g_{t-1} & \beta^2 g_{t-2} & \cdots & \beta^{r-1} g_{t-r+1}
\end{bmatrix}
\end{aligned} \]
Why a low-rank preconditioner?

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

\[ (a \times a)^{\frac{1}{2}} = a \times (a \times a)^{\frac{3}{2}} \times a \]
The GGT speedup

Matrix ops: $O(rd^2)$
Huge SVD: $O(d^3)$
Large-Scale Experiments (CIFAR-10, PTB)
Visualizing Gradient Spectra

26-layer ResNet
CIFAR-10

eigs($G_t^T G_t$)
@ $t = 150$

3-layer LSTM
Penn Treebank
(char-level)
Theory: faster convergence vs. non-convex SGD

- **Convex:** \( f(x_T) \leq \arg\min_x f(x) + \varepsilon \) 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* $\mu$:

\[
\mu^2 := \frac{\sum_{t=1}^T \| g_t^{AG} \|^2}{\sum_{t=1}^T \| g_t^{SGD} \|^2} = \frac{\text{AdaGrad regret}}{\text{worst-case OGD regret}}
\]

- [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 $\bar{O} \left( \frac{\mu^2 \sigma^2}{\varepsilon} \right)$ steps

- **Non-convex reduction**: GGT* converges in $\bar{O} \left( \frac{\mu^2 \sigma^2}{\varepsilon^4} \right)$ 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 \( \epsilon \)

- Recent state-of-the-art in SGD++: \( \frac{1}{\epsilon^4} \rightarrow \frac{1}{\epsilon^{3.5}} \)

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

- Our improvement \( \frac{1}{\epsilon^4} \rightarrow \frac{\mu^2}{\epsilon^4} \): 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 \textbf{dimension} rather than \textbf{approximation}

Thank you!