Stochastic Gradient Descent (SGD)
(Statistical Classification Lecture 2)

Yiming Yang
(based on the original slides by Hanxiao Liu)

February 22, 2018
Outline

1. Gradient Descent (GD)
2. Stochastic Gradient Descent (SGD)
   ▶ Formulation
   ▶ Comparison with GD
3. Useful large-scale SGD solvers
   ▶ Support Vector Machines
   ▶ Matrix Factorization
Risk Minimization

\[
\{(x_i, y_i)\}_{i=1}^{n}: \text{training data \ i.i.d. } \sim \mathcal{D}.
\]

\[
\min_{f} \mathbb{E}_{(x,y) \sim \mathcal{D}} \ell (f(x), y) \quad \Rightarrow \quad \min_{f} \frac{1}{n} \sum_{i=1}^{n} \ell (f(x_i), y_i)
\]

True risk

\[
\Rightarrow \min_{w} \frac{1}{n} \sum_{i=1}^{n} \ell (f_w(x_i), y_i)
\]

Empirical risk

<table>
<thead>
<tr>
<th>Classifier</th>
<th>(\ell (f_w(x_i), y_i))</th>
<th>Regularization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Regression</td>
<td>(\ln \left(1 + e^{-y_i w^\top x_i}\right))</td>
<td>(\frac{\lambda}{2} |w|^2)</td>
</tr>
<tr>
<td>SVM</td>
<td>(\max (0, 1 - y_i w^\top x_i))</td>
<td>(\frac{\lambda}{2} |w|^2)</td>
</tr>
</tbody>
</table>
Risk Minimization

\[ \{(x_i, y_i)\}_{i=1}^{n}: \text{training data } \overset{i.i.d.}{\sim} D. \]

\[
\min_{f} \mathbb{E}_{(x,y) \sim D} \ell(f(x), y) \quad \Rightarrow \quad \min_{f} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) \quad (1)
\]

\[
\quad \Rightarrow \quad \min_{w} \frac{1}{n} \sum_{i=1}^{n} \ell(f_w(x_i), y_i) \quad (2)
\]

<table>
<thead>
<tr>
<th>Classifer</th>
<th>( \ell(f_w(x_i), y_i) )</th>
<th>Regulurization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Regression</td>
<td>( \ln \left(1 + e^{\lambda y_i w^\top x_i}\right) )</td>
<td>( \frac{\lambda}{2} | w |^2 )</td>
</tr>
<tr>
<td>SVM</td>
<td>( \max (0, 1 - y_i w^\top x_i) )</td>
<td>( \frac{\lambda}{2} | w |^2 )</td>
</tr>
</tbody>
</table>
Gradient Descent (GD)

\[ \ell(f_w(x_i), y_i) \overset{\text{def}}{=} \ell_i(w), \quad \ell(w) \overset{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} \ell_i(w) \]

Training objective (omit regularization term for simplicity):

\[ \min_w \ell(w) \tag{3} \]

Gradient update: \[ w^{(k)} = w^{(k-1)} - \eta_k \nabla \ell(w^{(k-1)}) \]

- \( \eta_k \) is pre-specified or determined via backtracking;

- If the loss function is nonsmooth
  - Gradient \( \implies \) Subgradient
Gradient Descent (GD)

\[ \ell(f_w(x_i), y_i) \overset{\text{def}}{=} \ell_i(w), \quad \ell(w) \overset{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} \ell_i(w) \]

Training objective (omit regularization term for simplicity):

\[ \min_w \ell(w) \] (3)

Gradient update: \[ w^{(k)} = w^{(k-1)} - \eta_k \nabla \ell (w^{(k-1)}) \]

- \( \eta_k \) is pre-specified or determined via backtracking;
- If the loss function is nonsmooth
  - Gradient \( \rightarrow \) Subgradient
Gradient Descent (GD)

\[ \ell( f_w(x_i), y_i ) \overset{def}{=} \ell_i(w), \quad \ell(w) \overset{def}{=} \frac{1}{n} \sum_{i=1}^{n} \ell_i(w) \]

Training objective (omit regularization term for simplicity):

\[ \min_w \ell(w) \quad (3) \]

Gradient update:

\[ w^{(k)} = w^{(k-1)} - \eta_k \nabla \ell(w^{(k-1)}) \]

- \( \eta_k \) is pre-specified or determined via backtracking;

- If the loss function is nonsmooth
  - Gradient \( \implies \) Subgradient
How fast does GD converge?

**Theorem (GD convergence)**

If $\ell$ is both convex and differentiable $^1$

$$
\ell (w^{(k)}) - \ell (w^*) \leq \begin{cases} \\
\frac{\|w^{(0)} - w^*\|_2^2}{2\eta k} = O \left( \frac{1}{k} \right) & \ell \text{ is convex} \\
\frac{c^k L \|w^{(0)} - w^*\|_2^2}{2} = O \left( c^k \right) & \ell \text{ is strongly convex}
\end{cases}
$$

where $k$ is the number of iterations and $c \in (0, 1)$.

In general, to achieve $\ell (w^{(k)}) - \ell (w^*) \leq \rho$, GD needs $O \left( \frac{1}{\rho} \right)$ iterations; with strong convexity, it takes $O \left( \log \left( \frac{1}{\rho} \right) \right)$ iterations $^2$.

$^1$the step size $\eta$ must be no larger than $\frac{1}{L}$, where $L$ is the Lipschitz constant satisfying $\|\nabla \ell (a) - \nabla \ell (b)\|_2 \leq L \|a - b\|_2 \ \forall a, b$

$^2$Convex Optimization, S. Boyd & L. Vandenberghe, Ch 9.3
How fast does GD converge?

**Theorem (GD convergence)**

*If \( \ell \) is both convex and differentiable* \(^1\)

\[
\ell \left( w^{(k)} \right) - \ell \left( w^* \right) \leq \begin{cases} 
\frac{\|w^{(0)} - w^*\|_2^2}{2\eta k} = O \left( \frac{1}{k} \right) & \text{if } \ell \text{ is convex} \\
\frac{c^k L \|w^{(0)} - w^*\|_2^2}{2} = O \left( c^k \right) & \text{if } \ell \text{ is strongly convex} 
\end{cases}
\]

where \( k \) is the number of iterations and \( c \in (0, 1) \).

In general, to achieve \( \ell \left( w^{(k)} \right) - \ell \left( w^* \right) \leq \rho \), GD needs \( O \left( \frac{1}{\rho} \right) \) iterations; with strong convexity, it takes \( O \left( \log \left( \frac{1}{\rho} \right) \right) \) iterations \(^2\).

---

\(^1\)the step size \( \eta \) must be no larger than \( \frac{1}{L} \), where \( L \) is the Lipschitz constant satisfying \( \|\nabla \ell \left( a \right) - \nabla \ell \left( b \right)\|_2 \leq L \|a - b\|_2 \ \forall a, b \)

\(^2\)Convex Optimization, S. Boyd & L. Vandenberghe, Ch 9.3
GD Efficiency

Why not happy with GD?

- Fast convergence \(\neq\) high efficiency.

\[
\begin{align*}
    w^{(k)} &= w^{(k-1)} - \eta_k \nabla \ell(w^{(k-1)}) \\
    &= w^{(k-1)} - \eta_k \nabla \left[ \frac{1}{n} \sum_{i=1}^{n} \ell_i(w^{(k-1)}) \right]
\end{align*}
\]  (5)  (6)

- Per-iteration complexity = \(O(n)\) (extremely large)
  - A single cycle of all the data may take forever.
- How to make it cheaper? GD \(\implies\) SGD
GD Efficiency

Why not happy with GD?

- Fast convergence ≠ high efficiency.

\[
\begin{align*}
    w^{(k)} &= w^{(k-1)} - \eta_k \nabla \ell \left( w^{(k-1)} \right) \\
    &= w^{(k-1)} - \eta_k \nabla \left[ \frac{1}{n} \sum_{i=1}^{n} \ell_i \left( w^{(k-1)} \right) \right]
\end{align*}
\]

- Per-iteration complexity = \(O(n)\) (extremely large)
  - A single cycle of all the data may take forever.
- How to make it cheaper? GD \(\implies\) SGD
Approximate the full gradient via an unbiased estimator

\[ w^{(k)} = w^{(k-1)} - \eta_k \nabla \left( \frac{1}{n} \sum_{i=1}^{n} \ell_i (w^{(k-1)}) \right) \]  \hspace{1cm} (7)

\[ \approx w^{(k-1)} - \eta_k \nabla \left( \frac{1}{|B|} \sum_{i \in B} \ell_i (w^{(k-1)}) \right) \quad B \sim_{\text{unif}} \{1, 2, \ldots n\} \]  \hspace{1cm} (8)

\[ \approx w^{(k-1)} - \eta_k \nabla \ell_i (w^{(k-1)}) \quad i \sim_{\text{unif}} \{1, 2, \ldots n\} \]  \hspace{1cm} (9)

**Trade-off**: lower computation cost v.s. larger variance

---

3When using GPU, \(|B|\) usually depends on the memory budget.
For strongly convex $\ell(w)$, according to [Bottou, 2012]

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>GD</th>
<th>SGD</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time per-iteration</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>SGD</td>
</tr>
<tr>
<td>Iterations for accuracy $\rho$</td>
<td>$O\left(\log\left(\frac{1}{\rho}\right)\right)$</td>
<td>$\tilde{O}\left(\frac{1}{\rho}\right)$</td>
<td>GD</td>
</tr>
<tr>
<td>Time for accuracy $\rho$</td>
<td>$O\left(n \log \frac{1}{\rho}\right)$</td>
<td>$\tilde{O}\left(\frac{1}{\rho}\right)$</td>
<td>Depends</td>
</tr>
<tr>
<td>Time for test-set error $\epsilon$</td>
<td>$O\left(\frac{1}{\epsilon^{1/\alpha}} \log \frac{1}{\epsilon}\right)$</td>
<td>$\tilde{O}\left(\frac{1}{\epsilon}\right)$</td>
<td>SGD</td>
</tr>
</tbody>
</table>

where $\frac{1}{2} \leq \alpha \leq 1$
SVMs Solver: Pegasos

[Shalev-Shwartz et al., 2011]

Recall

\[ l_i(w) = \max (0, 1 - y_i w^\top x_i) + \frac{\lambda}{2} \|w\|^2 \]  \hspace{1cm} (10)

\[ = \begin{cases} 
\frac{\lambda}{2} \|w\|^2 & y_i w^\top x_i \geq 1 \\
1 - y_i w^\top x_i + \frac{\lambda}{2} \|w\|^2 & y_i w^\top x_i < 1 
\end{cases} \]  \hspace{1cm} (11)

Therefore

\[ \nabla l_i(w) = \begin{cases} 
\lambda w & y_i w^\top x_i \geq 1 \\
\lambda w - y_i x_i & y_i w^\top x_i < 1 
\end{cases} \]  \hspace{1cm} (12)
Recall

\[ l_i(w) = \max (0, 1 - y_i w^\top x_i) + \frac{\lambda}{2} \|w\|^2 \]  

(10)

\[
\begin{aligned}
\frac{\lambda}{2} \|w\|^2 & \quad y_i w^\top x_i \geq 1 \\
1 - y_i w^\top x_i + \frac{\lambda}{2} \|w\|^2 & \quad y_i w^\top x_i < 1
\end{aligned}
\]  

(11)

Therefore

\[
\nabla l_i(w) = \begin{cases} 
\lambda w & \quad y_i w^\top x_i \geq 1 \\
\lambda w - y_ix_i & \quad y_i w^\top x_i < 1
\end{cases}
\]  

(12)
Algorithm 1: Pegasos: SGD solver for SVMs

**Input:** $n, \lambda, T$

**Initialization:** $w \leftarrow 0$

for $k = 1, 2, \ldots, T$ do

$i \sim \{1, 2, \ldots n\}$;

$\eta_k \leftarrow \frac{1}{\lambda k}$;

if $y_i w^{(k)} x_i < 1$ then

$w^{(k+1)} \leftarrow w^{(k)} - \eta_k (\lambda w^{(k)} - y_i x_i)$

else

$w^{(k+1)} \leftarrow w^{(k)} - \eta_k \lambda w^{(k)}$

end

end

**Output:** $w^{(T+1)}$
Empirical Comparisons

SGD v.s. batch solvers\(^4\) on RCV1

<table>
<thead>
<tr>
<th>#Features</th>
<th>#Training examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>47,152</td>
<td>781,265</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time (secs)</th>
<th>Test Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMO (SVM(^{light}))</td>
<td>(\approx 16,000)</td>
<td>6.02%</td>
</tr>
<tr>
<td>Cutting Plane (SVM(^{perf}))</td>
<td>(\approx 45)</td>
<td>6.02%</td>
</tr>
<tr>
<td>SGD</td>
<td>&lt; 1</td>
<td>6.02%</td>
</tr>
</tbody>
</table>

What is the magic?

\(^4\)http://leon.bottou.org/projects/sgd
SGD takes a long time to reach an highly accurate solution on the training data.

However, for low generalization error (on test data) we may not need the training-set accuracy to be too high.

SGD for Matrix Factorization

The idea of SGD can be trivially extended to MF

\[
\ell(U, V) = \frac{1}{|\mathcal{O}|} \sum_{(a, b) \in \mathcal{O}} \ell_{a,b} (u_a, v_b)
\]

(13)

e.g. \((r_{ab} - u_a^t v_b)^2\)

SGD updating rule: for each randomly picked user-item pair \((a, b) \sim \mathcal{O}\) (the training set)

\[
u_a^{(k)} := u_a^{(k-1)} - \eta_k \nabla_{u_a} \ell_{a,b} (u_a^{(k-1)})
\]

(14)

\[
u_b^{(k)} := u_b^{(k-1)} - \eta_k \nabla_{v_b} \ell_{a,b} (v_b^{(k-1)})
\]

(15)

Buildingblock for distributed SGD for MF

\footnote{We omit the regularization term for simplicity.}
The idea of SGD can be trivially extended to MF

\[ \ell(U, V) = \frac{1}{|\mathcal{O}|} \sum_{(a,b) \in \mathcal{O}} \ell_{a,b}(u_a, v_b) \]  

(13)

e.g. \((r_{ab} - u_a^\top v_b)^2\)

SGD updating rule: for each randomly picked user-item pair \((a, b) \sim \mathcal{O}\) (the training set)

\[ u_a^{(k)} := u_a^{(k-1)} - \eta_k \nabla_{u_a} \ell_{a,b} \left(u_a^{(k-1)}\right) \]  

(14)

\[ v_b^{(k)} := v_b^{(k-1)} - \eta_k \nabla_{v_b} \ell_{a,b} \left(v_b^{(k-1)}\right) \]  

(15)

Building block for distributed SGD for MF

\(^5\)We omit the regularization term for simplicity.
Empirical Comparisons

On Netflix [Gemulla et al., 2011]

**DSGD, PSGD:** Distributed SGD

**ALS:** Alternating least squares, one of the state-of-the-art batch solvers

**DGD:** Distributed GD
Popular SGD Variants

A non-exhaustive list

1. AdaGrad [Duchi et al., 2011]
2. Momentum [Rumelhart et al., 1988]
3. Nesterov’s method [Nesterov et al., 1994]
5. Rprop & Rmsprop [Tieleman and Hinton, 2012]
6. Stochastic Variance Reduced Gradient [Johnson and Zhang, 2013]
7. ADAM [Kingma and Ba, 2014]

All are empirically found effective in solving nonconvex problems (e.g., deep neural nets).

Demos 6: Animation 0, 1, 2, 3

6https://www.reddit.com/r/MachineLearning/comments/2gopfa/visualizing_gradient_optimization_techniques/cklhott
Popular SGD Variants

A non-exhaustive list

1. AdaGrad [Duchi et al., 2011]
2. Momentum [Rumelhart et al., 1988]
3. Nesterov’s method [Nesterov et al., 1994]
5. Rprop & Rmsprop [Tieleman and Hinton, 2012]
6. Stochastic Variance Reduced Gradient [Johnson and Zhang, 2013]
7. ADAM [Kingma and Ba, 2014]

All are empirically found effective in solving nonconvex problems (e.g., deep neural nets).

Demos⁶: Animation 0, 1, 2, 3

⁶https://www.reddit.com/r/MachineLearning/comments/2gopfa/visualizing_gradient_optimization_techniques/cklhott
Summary

Today’s talk

1. GD - expensive, accurate gradient evaluation
2. SGD - cheap, noisy gradient evaluation
3. SGD-based solvers (SVMs, MF)

Remarks about SGD

- extremely handy for large problems
- only one of many handy tools
  - alternatives: quasi-Newton (BFGS), Coordinate descent, ADMM, CG, etc.
  - depending on the problem structure
Summary

Today’s talk

1. GD - expensive, accurate gradient evaluation
2. SGD - cheap, noisy gradient evaluation
3. SGD-based solvers (SVMs, MF)

Remarks about SGD

- extremely handy for large problems
- only one of many handy tools
  - alternatives: quasi-Newton (BFGS), Coordinate descent, ADMM, CG, etc.
  - depending on the problem structure
Reference I


In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM.

Adam: A method for stochastic optimization.

*Interior-point polynomial algorithms in convex programming*, volume 13. SIAM.

Learning representations by back-propagating errors.
*Cognitive modeling*, 5:3.

Pegasos: Primal estimated sub-gradient solver for svm.

Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.
*COURSERA: Neural Networks for Machine Learning*, 4.
Adadelta: An adaptive learning rate method.

Zhang, T.
Modern optimization techniques for big data machine learning.