Chapter 7

Stochastic gradient descent

Contents (class version)

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.0</td>
<td>Introduction</td>
<td>7.2</td>
</tr>
<tr>
<td>7.1</td>
<td>Stochastic gradient descent (SGD)</td>
<td>7.6</td>
</tr>
<tr>
<td></td>
<td>Stochastic gradient algorithm: analysis</td>
<td>7.9</td>
</tr>
<tr>
<td></td>
<td>Variance reduction: overview</td>
<td>7.16</td>
</tr>
<tr>
<td></td>
<td>Momentum</td>
<td>7.18</td>
</tr>
<tr>
<td></td>
<td>Adaptive step-sizes</td>
<td>7.20</td>
</tr>
<tr>
<td>7.2</td>
<td>Subgradient method</td>
<td>7.23</td>
</tr>
<tr>
<td></td>
<td>Hinge loss with 1-norm regularizer for binary classifier design</td>
<td>7.23</td>
</tr>
<tr>
<td></td>
<td>Example: X-ray CT reconstruction</td>
<td>7.25</td>
</tr>
<tr>
<td>7.3</td>
<td>Summary</td>
<td>7.34</td>
</tr>
</tbody>
</table>

This chapter was based on slides made by Naveen Murthy in April 2019.
Many applications in signal processing and machine learning involve finding the minimizer of a cost function $f : \mathbb{R}^N \rightarrow \mathbb{R}$ that is the sum (or average) of $M$ functions:

$$\hat{x} = \arg\min_x f(x) = \arg\min_x \frac{1}{M} \sum_{m=1}^{M} f_m(x).$$

For much of this chapter we assume:
- Each $f_m$ is convex
- Each $f_m$ is differentiable, with an $L_m$-Lipschitz continuous gradient
  $$\|\nabla f_m(x) - \nabla f_m(z)\|_2 \leq L_m x - z, \quad \forall x, z \in \text{dom}(f).$$
- Thus a Lipschitz constant for $f$ is $L \triangleq \frac{1}{M} \sum_{m=1}^{M} L_m$, though this might not be the best Lipschitz constant.

This setting covers a wide variety of machine learning problems and general inverse problems.
Example: Regularized logistic regression

Consider $M$ training examples with feature vectors $\{v_m\} \in \mathbb{R}^N$ and labels $y_m \in \{-1, +1\}$. The problem of learning weights $x \in \mathbb{R}^N$ for binary classification (using logistic regression) can be written as

$$\arg \min_x f(x) = \arg \min_x \frac{1}{M} \sum_{m=1}^M f_m(x), \quad f_m(x) = h(\langle y_m v_m, x \rangle) + \frac{1}{2} \beta \|x\|_2^2, \quad h(z) = \log(1 + e^{-z}).$$

Other formulations:
- Hinge loss (instead of logistic loss) $\implies$ SVM classifier
- Quadratic loss $\implies$ least-squares regression classifier
- 1-norm instead of 2-norm regularizer (not differentiable)
Detour - Rate of convergence review

- Suppose the sequence \( \{x_k\} \) converges to \( x^*_\). Consider

\[
\mu \triangleq \lim_{k \to \infty} \frac{\|x_{k+1} - \hat{x}\|_2}{\|x_k - x^*_\|_2}.
\]

- Rate of convergence of the sequence:
  - Converges \textit{linearly} with rate \( \mu \) if \( \mu \in (0, 1) \)
  - Converges \textit{sub-linearly} if \( \mu = 1 \)
  - Converges \textit{super-linearly} if \( \mu = 0 \)

Example:

- \( x_k = \rho^k \), with \( |\rho| < 1 \): convergence = linear

- \( x_k = 1/k^c \), with \( c > 0 \): convergence = sublinear
Gradient Descent (GD)

- GD update:
  \[ \mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f(\mathbf{x}_k) = \mathbf{x}_k - \eta \frac{1}{M} \sum_{m=1}^{M} \nabla f_m(\mathbf{x}_k). \]

- Converges to a minimizer if \( 0 < \eta < 2/L \); simple to analyze
- Achieves rate of convergence of \( O(1/k) \) if \( \eta \leq 1/L \)
- Can be improved to a linear rate \( O(\rho^k) \), where \( \rho < 1 \), under strong convexity assumptions on \( f \)
- Each gradient computation is linear in \( M \), i.e., takes \( O(M) \) time.
  \( \Rightarrow \) Doubling the number of examples in the training set doubles the gradient computational costs: scales poorly

How do we scale this to work with large-scale datasets?

Example. ImageNet [1] contains \( \sim 14 \) million images with more than 20,000 categories.
7.1 Stochastic gradient descent (SGD)

SGD update:

$$x_{k+1} = x_k - \eta_k \nabla f_{m_k}(x_k),$$

where $m_k$ is drawn randomly from \{1, 2, \ldots, M\}.

- Estimates gradient using only one $f_m$ function
- Computation of each update is very fast: $O(1)$ instead of $O(M)$
- Intuition: In expectation, the stochastic gradient equals the full gradient, i.e., if $j$ is drawn randomly from \{1, 2, \ldots, M\}, then

$$\mathbb{E}[\nabla f_j(x)] = \sum_{m=1}^{M} f_m(x) p(j = m) = \frac{1}{M} \sum_{m=1}^{M} f_m(x) = \nabla f(x).$$

- Used widely in software frameworks:

- The name **stochastic gradient descent** is misleading because it does not always descend in general!
Minibatching

Instead of one data vector at a time, use groups. Update:

\[ \mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \frac{1}{b} \sum_{m \in B_k} \nabla f_m(\mathbf{x}_k), \]

where \( B_k \) is drawn randomly from the set of all subsets of \( \{1, 2, \ldots, M\} \) of size \( b \).

- Intermediate approach between a stochastic gradient (SG) and the full gradient.
- Computational cost is more than SG \( O(b) \), but also reduces variance of the gradient estimate by a factor \( b \).
Example: Regularized logistic regression. $M = 10,000$ and $N = 20$; fixed step-sizes used for all methods. (Slide adapted from Pradeep Ravikumar & Aarti Singh (CMU).)
Stochastic gradient algorithm: analysis

Framework for analysis [2]

Choose an initial iterate \( x_0 \)
for \( k = 0, 1, \ldots \)

- Generate a realization of random variable \( \xi_k \)
- Compute a stochastic vector \( g(x_k, \xi_k) \)
- Choose a step-size \( \eta_k > 0 \)
- Update \( x_{k+1} = x_k - \eta_k g(x_k, \xi_k) \)

Here, \( g(x_k, \xi_k) \) is a stochastic estimate of \( \nabla f(x_k) \).

**Stochastic gradients - bias & variance**

**Bias:**

\[
\text{bias}(g(x_k, \xi_k)) \triangleq E_{\xi_k}[g(x_k, \xi_k)] - \nabla f(x_k).
\]

**Variance:**

\[
\text{var}(g(x_k, \xi_k)) \triangleq E_{\xi_k}[\|g(x_k, \xi_k)\|_2^2] - \|E_{\xi_k}[g(x_k, \xi_k)]\|_2^2.
\]
SG - fixed vs. diminishing step-sizes


Issues with fixed step-sizes:
- For fixed step-size, SGD approaches to a noise ball (proportional to step-size) around the optimum.
- Choosing a lower constant size reduces the size of the noise ball, but leads to slower convergence.
- Intuition: use larger step-sizes initially and gradually decrease
SG: diminishing step-sizes

- Diminishing step-size schemes are often chosen to be $\mathcal{O}(1/k)$.
  For example (https://scikit-learn.org/stable/modules/sgd.html):

  $$\eta_k = \frac{1}{\alpha(k_0 + k)}$$

- The hyperparameters $\alpha$ and $k_0$ could be hand-tuned or chosen through an automatic process.

- Sufficient conditions for convergence:

  $$\sum_{k=1}^{\infty} \eta_k = \infty, \quad \sum_{k=1}^{\infty} \eta_k^2 < \infty$$

So the step sizes should diminish, but not too quickly.
**Convergence rates**

- **Full GD:**
  For convex $f$, with $L$-Lipschitz gradient, under suitable stepsizes:
  \[ f(x_k) - f^* = O(1/k) \]

- **Stochastic GD:**
  For convex $f$, under diminishing stepsizes (along with other conditions):
  \[ E[f(x_k)] - f^* = O(1/\sqrt{k}). \]

**Convergence rates (strong convexity)**

Under strong convexity assumptions on $f$ (with parameter $\mu$)

- **Full GD:**
  For strongly convex $f$, with $L$-Lipschitz gradient, for suitable stepsizes GD has a **linear** rate:
  \[ f(x_k) - f^* = O(\rho^k), \]
  where $\rho < 1$.

- **Stochastic GD:** Under strong convexity (plus other assumptions as before), SGD has a **sub-linear** rate of
  \[ E[f(x_k)] - f^* = O(1/k) \]

Can we do better than sub-linear convergence for SGD?
Improving SGD

- Diminishing stepsizes (learning rates) are needed due to the variability of stochastic gradients.
- Can we reduce the variability of those stochastic gradients and use a constant learning rate?
- Can we use better learning rate schedules to accelerate SGD?

Variance reduction methods:
- SAG, SAGA [3],
- SVRG [4]
- S2GD, SDCA etc.

Adjusting learning rate:
- Momentum, Nesterov Accelerated Gradient [5]
- Fixed/adaptive restart SGDR [6]
SVRG (Stochastic Variance Reduced Gradient)

Algorithm SVRG

1: Choose an initial iterate $w_1 \in \mathbb{R}^d$, stepsize $\alpha > 0$, and positive integer $m$.
2: \textbf{for} $k = 1, 2, \ldots$ \textbf{do}
3: \hspace{1em} Compute the batch gradient $\nabla F(w_k)$.
4: \hspace{1em} Initialize $w_{k,1} \leftarrow w_k$.
5: \hspace{1em} \textbf{for} $j = 1, \ldots, m$ \textbf{do}
6: \hspace{2em} Choose $i$ uniformly from $\{1, \ldots, n\}$.
7: \hspace{2em} Set $g_{k,j} \leftarrow \nabla f_i(w_{k,j}) - (\nabla f_i(w_k) - \nabla F(w_k))$.
8: \hspace{2em} Set $w_{k,j+1} \leftarrow w_{k,j} - \alpha g_{k,j}$.
9: \hspace{1em} \textbf{end for}
10: Option (a): Set $w_{k+1} = \tilde{w}_{m+1}$
11: Option (b): Set $w_{k+1} = \frac{1}{m} \sum_{j=1}^{m} \tilde{w}_{j+1}$
12: Option (c): Choose $j$ uniformly from $\{1, \ldots, m\}$ and set $w_{k+1} = \tilde{w}_{j+1}$.
13: \textbf{end for}

Credit: Leon Bottou, Frank Curtis, Jorge Nocedal
Algorithm SAGA

1: Choose an initial iterate \( w_1 \in \mathbb{R}^d \) and stepsize \( \alpha > 0 \).
2: for \( i = 1, \ldots, n \) do
3: \hspace{1em} Compute \( \nabla f_i(w_1) \).
4: \hspace{1em} Store \( \nabla f_i(w_{[i]}) \leftarrow \nabla f_i(w_1) \).
5: end for
6: for \( k = 1, 2, \ldots \) do
7: \hspace{1em} Choose \( j \) uniformly in \( \{1, \ldots, n\} \).
8: \hspace{1em} Compute \( \nabla f_j(w_k) \).
9: \hspace{1em} Set \( g_k \leftarrow \nabla f_j(w_k) - \nabla f_j(w_{[j]}) + \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w_{[i]}) \).
10: \hspace{1em} Store \( \nabla f_j(w_{[j]}) \leftarrow \nabla f_j(w_k) \).
11: \hspace{1em} Set \( w_{k+1} \leftarrow w_k - \alpha g_k \).
12: end for

Credit: Leon Bottou, Frank Curtis, Jorge Nocedal
Variance reduction: overview

Consider 2 random variables $X, Y$. Define a new random variable as follows:

$$ Z_\alpha \triangleq \alpha (X - Y) + \mathbb{E}[Y], \quad \alpha \in [0, 1]. $$

Can we design $Z_\alpha$ to obtain a “good” estimator of $\mathbb{E}[X]$?

- $\alpha = 1$ results in an unbiased estimator: $\mathbb{E}[Z_1] = \mathbb{E}[X]$.
- $\text{Var}\{Z_\alpha\} = \alpha^2 \left( \text{Var}\{X\} + \text{Var}\{Y\} - 2 \ \text{Cov}\{X, Y\} \right)$.
- If $X$ and $Y$ are highly correlated, $Z_\alpha$ has reduced variance.
SGD variants with reduced variance

Let $X$ be the current SGD direction $\nabla f_j (x_k)$ and $Y$ be a past stored gradient $\nabla f_j (\phi_j^{(k)})$.

(SAG)  \[ x_{k+1} = x_k - \eta \left( \frac{1}{M} \left( \nabla f_j (x_k) - \nabla f_j (\phi_j^{(k)}) \right) + \frac{1}{M} \sum_{m=1}^{M} \nabla f_m (\phi_m^{(k)}) \right) \]

(SAGA)  \[ x_{k+1} = x_k - \eta \left( \nabla f_j (x_k) - \nabla f_j (\phi_j^{(k)}) + \frac{1}{M} \sum_{m=1}^{M} \nabla f_m (\phi_m^{(k)}) \right) \]

(SVRG)  \[ x_{k+1} = x_k - \eta \left( \nabla f_j (x_k) - \nabla f_j (\tilde{x}) + \frac{1}{M} \sum_{m=1}^{M} \nabla f_m (\tilde{x}) \right) \]

- SAG - Stochastic Average Gradient
- SVRG - Stochastic Variance Reduced Gradient
**Momentum**

\[ \mathbf{v}_{k+1} = \mu \mathbf{v}_k - \eta_k \nabla f(\mathbf{x}_k) \]

\[ \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{v}_{k+1} \]

- \( \mu \in [0, 1] \) is the momentum coefficient; typically chosen to be high (\( \sim 0.9 \))
- \( \mu = 0 \) same as ordinary GD

Image taken from [5]
Nesterov momentum

\[ \mathbf{v}_{k+1} = \mu \mathbf{v}_k - \eta_k \nabla f(\mathbf{x}_k + \mu \mathbf{v}_k) \]

\[ \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{v}_{k+1} \]

- Gradient is computed at lookahead point \( \mathbf{x}_k + \mu \mathbf{v}_k \) instead of at \( \mathbf{x}_k \).
- \( \mu = 0 \) reverts to ordinary GD

Adaptive step-sizes

- Idea: Instead of assigning the same learning rate for each feature, why not vary the rate per feature (depending on importance)?

- Choose step-sizes adaptively based on measurements and heuristics (instead of an \textit{a priori} schedule).

- Examples:
  
  - **Adagrad**:
    - Tracks sum of squared gradients.
    - Learning rate is different for different directions (depending on past gradients).
    - Caveat: The learning rate decays too aggressively (due to monotonicity).

  - **RMSprop**:
    - Similar to Adagrad, but looks at past gradients only over a moving window.
    - Due to this short-term memory, learning rates do not get monotonically smaller; better than Adagrad.

  - **Adam**:
    - Can be thought of as RMSprop with momentum.
    - Includes bias correction terms for first and second moments.
    - Very widely used for training deep networks.
Optimizer choice

- No one-size-fits-all choice
- Use momentum approaches for higher training speed
- If dataset is sparse, adaptive learning rates are a reasonable choice
- Default algorithms to try:
  - Adam is widely used for training networks
  - SGD + momentum is also surprisingly effective for many applications

Visualizations of optimizer trajectories:
- https://github.com/Jaewan-Yun/optimizer-visualization
**Restart**

Blue - Optimal momentum (with strong convexity parameter known)
Red - Nesterov Momentum
Green - Adaptive restart
Image from [7]

- Function scheme: Restart whenever objective function value increases
- Gradient scheme: Restart when negative gradient and momentum form an obtuse angle
- Stochastic Gradient Descent with Warm Restarts (SGDR)
7.2 Subgradient method

The term **stochastic gradient descent** (SGD) is used today even for non-differentiable functions (that have no gradient).

A more accurate term would be the **stochastic subgradient method**, because for non-differentiable functions we generally use the **subgradient** instead of a gradient, as discussed in Ch. 5, and also “descent” is not ensured in general.

**Hinge loss with 1-norm regularizer for binary classifier design**

One motivating application is binary classifier design using the **hinge loss** function and a 1-norm to encourage parsimony:

$$
\Psi(x) = \frac{1}{M} \mathbf{1}' h(Ax) + \beta \|x\|_1, \quad h(t) = \max(1 - t, 0).
$$

Both terms in this cost function are non-smooth, but both are convex and have well defined subgradients.

For $g(x) = |x|$ the **subdifferential** is

$$
\partial g(x) = \begin{cases} 
-1, & x < 0 \\
[-1, 1], & x = 0, \\
1, & x > 0.
\end{cases}
$$
Thus for $g(x) = \|x\|_1$, an appropriate subgradient is

$$\partial g(x) = \text{sign} \cdot (x).$$

For $h(t) = \max(1 - t, 0)$ the subdifferential is

$$\partial h(t) = \begin{cases} 
-1, & t < 1 \\
0, & t > 0, \\
[-1, 0], & t = 1.
\end{cases}$$

Thus for $f(x) = 1'h(Ax)$ an appropriate subgradient is

$$\partial f(x) = A'h(Ax).$$

To apply a stochastic subgradient method to this problem, we write the cost function as:

$$\Psi(x) = \frac{1}{M} \sum_{m=1}^{M} f_m(x), \quad f_m(x) \triangleq h([Ax]_m) + M\beta \|x\|_1.$$ 

For each update we pick $m$ at random, compute a subgradient $g_m(x_k)$ of $f_m$, select a step size $\eta_k$, and update

$$x_{k+1} = x_k - \eta_k g_m(x_k).$$

Then repeat until some convergence criterion is reached.

A HW problem will compare the subgradient method to ADMM for this application.
Example: X-ray CT reconstruction

$$\hat{x} = \arg \min_{x \in X} \Psi(x) \triangleq \frac{1}{2} \| y - Ax \|_W^2 + \beta R(x)$$

<p>| TABLE I |</p>
<table>
<thead>
<tr>
<th>SQS METHODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Initialize $x^{(0)}$ and compute $D$ such that (5) and (6) hold.</td>
</tr>
<tr>
<td>2: for $n = 0, 1, \cdots$</td>
</tr>
<tr>
<td>3: $x^{(n+1)} = \mathcal{P}_X [x^{(n)} - D^{-1} \nabla \Psi(x^{(n)})]$</td>
</tr>
</tbody>
</table>

Can use an MM algorithm with $D = \text{diag}\{A'WA1\}$, called separable quadratic surrogates (SQS [8])

**Background: X-ray CT model**

- $x$: Unknown attenuation image to be reconstructed
- $y$: Noisy sinogram data
- $A$: CT system matrix
\[
\begin{bmatrix}
\text{Source} \\
\text{Detectors}
\end{bmatrix}
\begin{bmatrix}
\begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\begin{bmatrix}
\begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
\begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
\begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
\text{x} \\
\text{e}
\end{bmatrix}
\]
\[
y = A x + e
\]
\[ y = A x + \varepsilon \]
Ordered subsets

- We can rewrite the cost function as:

\[
\Psi(x) = \sum_{m=1}^{M} \Psi_m(x), \quad \Psi_m(x) \triangleq \frac{1}{2} \| y_m - A_m x \|_W^2 + \frac{1}{M} R(x), \quad m = 1, \ldots, M.
\]

- Intuition:

Divide the total projections into $M$ “subsets” (akin to minibatches in a machine learning problem). The idea is to save computation by replacing a full gradient with a subset-specific gradient.

Ordered subsets + momentum

- Incorporate momentum into the algorithm with ordered subsets [8] (somewhat analogous to minibatch-SGD plus momentum).
- Choose subsets to maintain the subset balance approximation (to reduce variance):

\[
\nabla \Psi(x) \approx M \nabla \Psi_1(x) \approx \ldots \approx M \nabla \Psi_M(x)
\]
TABLE III
PROPOSED OS-SQS METHODS WITH MOMENTUM IN [17] (OS-mom1)

1: Initialize $x^{(0)} = z^{(0)}$, $t_0 = 1$ and compute $D$.
2: for $n = 0, 1, \ldots$
3: for $m = 0, 1, \ldots, M - 1$
4: \[ k = nM + m \]
5: \[ t_{k+1} = \frac{1}{2} \left( 1 + \sqrt{1 + 4t_k^2} \right) \]
6: \[ x^{\left( \frac{k+1}{M} \right)} = \mathcal{P}_\chi \left[ z^{\left( \frac{k}{M} \right)} - D^{-1} M \nabla \Psi_m \left( z^{\left( \frac{k}{M} \right)} \right) \right] \]
7: \[ z^{\left( \frac{k+1}{M} \right)} = x^{\left( \frac{k+1}{M} \right)} + \frac{t_k}{t_{k+1}} \left( x^{\left( \frac{k+1}{M} \right)} - x^{\left( \frac{k}{M} \right)} \right) \]
Experimental setup - CT fan beam reconstruction

- Image size $128 \times 128$
- Sinogram size $222 \times 50$
- Hyperbola function used as the regularizer potential function
- Reference image was generated by running 5000 iterations of the 1 subset version of OS-SQS algorithm.
- Regularization parameter = 16

- At iteration $k$, root mean square difference (RMSD) = $\sqrt{\frac{1}{|\Omega|} \sum_{j \in \Omega} |x_j^{(k)} - \hat{x}_j|^2}$

$\Omega = $ cylindrical region-of-interest
RMSD plots

- All algorithms were run with 8 subsets. Restart scheme was function-based.
- Note: Using momentum causes plot to diverge; restart helps stabilize it.

Errata: RMSD is in units of inverse length (1/cm), *not* in HU.
Reconstructed images

- $x_{\text{ref}}$ = reference image; initial image $x_0$ = FBP ramp image
7.3 Summary

- For large-scale machine learning problems, SGD (or one of its variants) is a suitable optimization algorithm.
- Can be used for a wide variety of smooth/non-smooth, and convex/non-convex problems.
- Some broad approaches to improve convergence of SGD:
  - Reduce variance of the stochastic estimate (keeping the learning rate fixed).
  - Design better learning rate schedules, using momentum methods or adaptive methods like Adagrad, RMSprop and Adam.
- Caveat: Hyperparameter tuning can prove to be difficult for many of these algorithms (in contrast with GD, POGM, NCG etc.)

Bibliography


