Statistical Machine Learning (BE4M33SSU)
Lecture 7a: Stochastic Gradient Descent

Jan Drchal

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science
Gradient Descent (GD)

- **Task**: find parameters which minimize loss over the training dataset:

  $$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta)$$

  where $\theta$ is a set of all parameters defining the ANN

- **Gradient descent**: $\theta_{k+1} = \theta_k - \alpha_k \nabla \mathcal{L}(\theta_k)$

  where $\alpha_k > 0$ is the **learning rate** or **stepsize** at iteration $k$
Stochastic Gradient Descent (SGD): Motivation

- The loss has typically an additive structure, hence:

\[
\nabla L(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla \ell(y_i, h_{\theta}(x_i))
\]

- Evaluation of \( \nabla L(\theta) \) takes \( O(m) \) time
- What if we have duplicate samples in \( T^m \)?
- Online learning, *infinite data* (augmentation)?
- Specialized loss functions (not based on likelihood)?
- Use a single sample or a *mini-batch* instead of the *full-batch* approach
  \( \Rightarrow \) Stochastic Gradient Descent (SGD)
- Recommended reading: *Bottou, Curtis and Nocedal: Optimization Methods for Large-Scale Machine Learning*, 2018
## SGD Algorithm

- **Stochastic Gradient Descent**

1. Choose an initial iterate $\theta_1$
2. **for** $k = 1, 2, \ldots$
3. Draw a batch $B_k^M \subset \mathcal{T}^m$
4. Compute a stochastic gradient estimate vector $g(\theta_k)$ for $B_k^M$
5. Choose a stepsize $\alpha_k > 0$
6. Set the new iterate as $\theta_{k+1} \leftarrow \theta_k - \alpha_k g(\theta_k)$

- The stochastic gradient estimate is defined as:

$$
g(\theta_k) = \frac{1}{M} \sum_{i=1}^{M} \nabla \ell(y_i, h_\theta(x_i)), \ (x_i, y_i) \in B_k^M$$
Drawing Batches

- Random samples with replacement
  ⇒ some training samples may be left unused!

- Shuffle data once and split to batches

- **Shuffle data each epoch before splitting to batches**

The main theorem shows that the *expected optimality gap*

\[ \mathbb{E}[\mathcal{L}(\theta_k) - \mathcal{L}_*] \xrightarrow{k \to \infty} 0 \]

where \( \mathcal{L}_* \) is the optimal (minimal) loss

**Selected assumptions:**
1. Strong convexity of \( \mathcal{L} \)
2. Lipschitz continuous gradient \( \nabla \mathcal{L} \)
3. Bounds on \( \mathcal{L} \) and \( g(\theta_k) \):
   - directions of \( g(\theta_k) \) and \( \nabla \mathcal{L}(\theta_k) \) *similar*,
   - their norms are also *similar*
Convexity

- Convex function definition:

\[
L(t\theta + (1 - t)\bar{\theta}) \leq tL(\theta) + (1 - t)L(\bar{\theta})
\]

for all \((\theta, \bar{\theta}) \in \mathbb{R}^d \times \mathbb{R}^d\)

- Equivalently (first-order condition):

\[
L(\bar{\theta}) \geq L(\theta) + \nabla L(\theta)^T (\bar{\theta} - \theta)
\]

the function lies above all its tangents

- See A4B33OPT
- But we need a stronger assumption...
Assumption 1: Strong Convexity

- The loss function $\mathcal{L} : \mathbb{R}^d \rightarrow \mathbb{R}$ is strongly convex if there exists constant $c > 0$ such that

$$\mathcal{L}(\bar{\theta}) \geq \mathcal{L}(\theta) + \nabla \mathcal{L}(\theta)^T (\bar{\theta} - \theta) + \frac{1}{2} c \| \bar{\theta} - \theta \|^2_2$$

for all $(\theta, \bar{\theta}) \in \mathbb{R}^d \times \mathbb{R}^d$

- **Intuition**: quadratic lower bound on function growth

- Constant $c$ quantifies the level of convexity $\Rightarrow$ higher $c$ indicates "more convex" function
Assumption 2: Lipschitz Continuous Gradient

- The loss function is continuously differentiable and the gradient is *Lipschitz continuous* with *Lipschitz constant* $L > 0$:

$$\|\nabla \mathcal{L}(\theta) - \nabla \mathcal{L}(\bar{\theta})\|_2 \leq L \|\theta - \bar{\theta}\|_2, \text{ for all } (\theta, \bar{\theta}) \in \mathbb{R}^d \times \mathbb{R}^d$$

- *Intuition*: the gradient does not change too quickly w.r.t. $\theta$

- Provides a **quantified** indicator for how far to move to decrease $\mathcal{L}$
## Assumptions Summary

<table>
<thead>
<tr>
<th>constant</th>
<th>description</th>
<th>higher value means</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c &gt; 0$</td>
<td>strong convexity (lower bound)</td>
<td>&quot;more convex&quot;</td>
</tr>
<tr>
<td>$L &gt; 0$</td>
<td>Lipschitz continuous gradient (upper bound)</td>
<td>higher gradient change allowed</td>
</tr>
<tr>
<td>$\mu &gt; 0$</td>
<td>$g(\theta_k)$ direction comparable to $\nabla L(\theta_k)$</td>
<td>smaller angular difference between $\mathbb{E}[g(\theta_k)]$ and $\nabla L(\theta_k)$</td>
</tr>
<tr>
<td>$M \geq 0$</td>
<td>compares norms of $\mathbb{E}[g(\theta_k)]$ and $\nabla L(\theta_k)$</td>
<td>higher variance in norms allowed</td>
</tr>
</tbody>
</table>
Theorem (simplified): if the assumptions hold, the SGD is run with a fixed (and bound) stepsize $\alpha_k = \alpha$ for all $k \in \mathbb{N}$. Then the expected optimality gap satisfies the following for all $k$:

$$
\mathbb{E} [ \mathcal{L}(\theta_k) - \mathcal{L}_* ] \leq \frac{\alpha LM}{2c\mu} + \mathcal{O}(\rho^k) \xrightarrow{k \to \infty} \frac{\alpha LM}{2c\mu},
$$

where $\rho \in [0, 1)$ is a constant.

In general, for the fixed stepsize, the optimality gap tends to zero, but converges to $\frac{\alpha LM}{2c\mu} \geq 0$.

Note on $\alpha$: lower $L$ and higher $\mu$ allow longer stepsize.

Note on $c$: having $c > 0$ is critical to keep $\rho < 1$. 
Full-Batch Gradient Descent

- How does the theorem apply to the full-batch setting (GD)?

- The $g(\theta_k)$ is an unbiased estimate of $\nabla L(\theta_k)$:

  $$\mathbb{E}[g(\theta_k)] = \nabla L(\theta_k)$$

- Zero variance implies $M = 0$, hence, $\frac{\alpha LM}{2c\mu} = 0$

- The optimality gap simplifies to:

  $$\epsilon_k = \mathbb{E}[L(\theta_k) - L_*] \leq O(\rho^k) \xrightarrow{k \to \infty} 0$$

- For any given gap $\epsilon$, the number of iterations $k$ is proportional to $\log(1/\epsilon)$ in the worst case.
SGD Convergence: Strongly Convex $\mathcal{L}$, Diminishing Stepsize

- **Theorem (simplified):** if the assumptions hold and the SGD is run with a diminishing stepsize ($\alpha_k \approx$ inversely proportional to $k$). Then the expected optimality gap satisfies the following for all $k$:

$$
\mathbb{E}[\mathcal{L}(\theta_k) - \mathcal{L}^*] \leq O\left(\frac{1}{k}\right) \xrightarrow{k \to \infty} 0
$$

- The number of iterations $k$ is proportional to $1/\epsilon$ in the worst case
- The diminishing stepsize is needed to compensate for the noise & to prevent overshooting the minimum
- Note: in practise $\alpha_k$ is often reduced in steps, e.g., halved after each $N$ epochs
SGD for Nonconvex Objectives

- Corresponding theorems can be proven for nonconvex objectives
- For assumptions similar to the theorem for the diminishing stepsizes (and excluding the strong convexity) we get:

\[
\lim_{k \to \infty} \mathbb{E} \left[ \left\| \nabla \mathcal{L}(\theta_k) \right\|_2^2 \right] = 0
\]
## GD vs SGD

<table>
<thead>
<tr>
<th></th>
<th>GD</th>
<th>SGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>time per iteration</td>
<td>$m$</td>
<td>1</td>
</tr>
<tr>
<td>iterations for accuracy $\epsilon$</td>
<td>$\log(1/\epsilon)$</td>
<td>$1/\epsilon$</td>
</tr>
<tr>
<td>time for accuracy $\epsilon$</td>
<td>$m \log(1/\epsilon)$</td>
<td>$1/\epsilon$</td>
</tr>
<tr>
<td>error for limited time budget $T_{max}$</td>
<td>$\frac{\log(T_{max})}{T_{max}} + \frac{1}{T_{max}}$</td>
<td>$\frac{1}{T_{max}}$</td>
</tr>
</tbody>
</table>

- SGD time does not depend on dataset size (if not exhausted)
- For large-scale problems (large $m$) SGD is faster
- For limited time budget $T_{max}$ SGD achieves lower error
- It is harder to tune stepsize schedule for SGD, but you can experiment on a small representative subset of the dataset
- In practice, *mini-batches* are used to leverage optimization/parallelization on CPU/GPU
Momentum

- Simulate inertia to overcome plateaus in the error landscape:

\[
\begin{align*}
    \mathbf{v}_{k+1} & \leftarrow \mu \mathbf{v}_k - \alpha_k \; g(\theta_k) \\
    \theta_{k+1} & \leftarrow \theta_k + \mathbf{v}_{k+1}
\end{align*}
\]

where \( \mu \in [0, 1] \) is the momentum parameter

- Momentum damps oscillations in directions of high curvature

- It builds velocity in directions with consistent (possibly small) gradient
Adagrad

- Adaptive Gradient method (Duchi, Hazan and Singer, 2011)
- **Motivation:** a magnitude of gradient differs a lot for different parameters
- **Idea:** reduce learning rates for parameters having high values of gradient

\[
G_{k+1,i} \leftarrow G_{k,i} + [g(\theta_k)]_i^2
\]

\[
\theta_{k+1,i} \leftarrow \theta_{k,i} - \frac{\alpha}{\sqrt{G_{k+1,i} + \epsilon}} \cdot [g(\theta_k)]_i
\]

- \(G_{k,i}\) accumulates squared partial derivative approximations w.r.t. to the parameter \(\theta_{k,i}\)
- \(\epsilon\) is a small positive number to prevent division by zero
- **Weakness:** ever increasing \(G_i\) leads to slow convergence eventually
- Better methods: RMSProp, Adam, . . .
$N = 15, \ a = 0.1$

$N = 15, \ a = 0.4$

$N = 15, \ a = 0.55$
\[ \mathcal{L}(\theta) + \nabla \mathcal{L}(\theta)^T (\bar{\theta} - \theta) \]