Part III/IV: Adaptive first-order methods

Technion-Israel Institute of Technology
First Order Methods

Goal:

\[
\min_{x \in X} f(x)
\]

Update rule:

\[
x_{t+1} = x_t - \eta_t g_t \;
\text{where}\;
\mathbb{E}[g_t|x_t] = \nabla f(x_t)
\]

Output:

\[
\bar{x}_T = \bar{x}_T(x_1, g_1, \ldots, x_t, g_t)
\]
First Order Methods

Goal:
\[
\min_{x \in \mathcal{X}} f(x)
\]

Update rule:
\[
x_{t+1} = x_t - \eta_t g_t \quad \text{where} \quad \mathbb{E}[g_t | x_t] = \nabla f(x_t)
\]

Output:
\[
\bar{x}_T = \bar{x}_T(x_1, g_1, \ldots, x_t, g_t)
\]

Performance Measure:
After \(T\) iterations,
\[
err_T = f(\bar{x}_T) - f(x^*)
\]

Ensure low \(err_T\) in expectation or with high probability.
First Order Methods

Goal:

\[
\min_{x \in X} f(x)
\]

Update rule:

\[
x_{t+1} = x_t - \eta_t g_t \quad \text{where} \quad \mathbb{E}[g_t | x_t] = \nabla f(x_t) \quad ; t = 1 \ldots T
\]

Output:

\[
\bar{x}_T = \bar{x}_T(x_1, g_1, \ldots, x_t, g_t)
\]

Performance Measure in **Non-convex** case:

After \( T \) iterations,

\[
\text{err}_T = \| \nabla f(\bar{x}_T) \|^2
\]

Ensure low \( \text{err}_T \) in *expectation* or *with high probability*
Geometric & Statistical Properties

- \( G \): scale of (stochastic) gradients, \( G := \max_t \| g_t \| \)

- \( L \): smoothness-Lipschitz continuity of gradients, \( \| \nabla f(x) - \nabla f(y) \| \leq L \| x - y \| \)

- \( \sigma^2 \): variance of gradient noise, \( \mathbb{E}[\| g_t - \nabla f(x_t) \|^2 | x_t ] \leq \sigma^2 \)
  
  Noiseless (a.k.a deterministic) case: \( \sigma = 0 \)

- \( D \): distance of initial point to optimum, \( \| x_1 - x^* \| \)
Geometric & Statistical Properties

- $G$: scale of (stochastic) gradients, \( G := \max_t \|g_t\| \)

- $L$: smoothness-Lipschitz continuity of gradients, \( \|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \)

- $\sigma^2$: variance of gradient noise, \( \mathbb{E}[\|g_t - \nabla f(x_t)\|^2|x_t] \leq \sigma^2 \)
  
  Noiseless (a.k.a determinstic) case: $\sigma = 0$

- $D$: distance of initial point to optimum, \( \|x_1 - x^*\| \)

Prior knowledge:
- Problem parameters, $G, L, \sigma^2, D$ should be known in advance to obtain the optimal rates
- These parameters are also required in order to efficiently parallelize the learning process
Convex objective function

<table>
<thead>
<tr>
<th>gradient oracle</th>
<th>Smoothness</th>
<th>GD/SGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>deterministic/stochastic</td>
<td>non-smooth</td>
<td>$O\left(\frac{GD}{\sqrt{T}}\right)$</td>
</tr>
<tr>
<td>deterministic</td>
<td>smooth</td>
<td>$O\left(\frac{LD^2}{T}\right)$</td>
</tr>
<tr>
<td>stochastic</td>
<td>smooth</td>
<td>$O\left(\frac{LD^2}{T} + \frac{\sigma D}{\sqrt{T}}\right)$</td>
</tr>
</tbody>
</table>
Geometric & Statistical Properties $\Rightarrow$ Convergence

Convex objective function

<table>
<thead>
<tr>
<th>gradient oracle</th>
<th>Smoothness</th>
<th>GD/SGD</th>
<th>Accelerated GD/SGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>deterministic/stochastic</td>
<td>non-smooth</td>
<td>$O\left(\frac{GD}{\sqrt{T}}\right)$</td>
<td>$O\left(\frac{GD}{\sqrt{T}}\right)$</td>
</tr>
<tr>
<td>deterministic</td>
<td>smooth</td>
<td>$O\left(\frac{LD^2}{T}\right)$</td>
<td>$O\left(\frac{LD^2}{T^2}\right)$</td>
</tr>
<tr>
<td>stochastic</td>
<td>smooth</td>
<td>$O\left(\frac{LD^2}{T} + \frac{\sigma D}{\sqrt{T}}\right)$</td>
<td>$O\left(\frac{LD^2}{T^2} + \frac{\sigma D}{\sqrt{T}}\right)$</td>
</tr>
</tbody>
</table>

Prior knowledge:
- Problem parameters, $G, L, \sigma^2, D$ should be known in advance to obtain the optimal rates
- These parameters are also required in order to efficiently parallelize the learning process
Geometric & Statistical Properties $\Rightarrow$ Convergence

Convex objective function

<table>
<thead>
<tr>
<th>gradient oracle</th>
<th>Smoothness</th>
<th>GD/SGD</th>
<th>Accelerated GD/SGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>deterministic/stochastic</td>
<td>non-smooth</td>
<td>$O\left(\frac{GD}{\sqrt{T}}\right)$</td>
<td>$O\left(\frac{GD}{\sqrt{T}}\right)$</td>
</tr>
<tr>
<td>deterministic</td>
<td>smooth</td>
<td>$O\left(\frac{LD^2}{T}\right)$</td>
<td>$O\left(\frac{LD^2}{T^2}\right)$</td>
</tr>
<tr>
<td>stochastic</td>
<td>smooth</td>
<td>$O\left(\frac{LD^2}{T} + \frac{\sigma D}{\sqrt{T}}\right)$</td>
<td>$O\left(\frac{LD^2}{T^2} + \frac{\sigma D}{\sqrt{T}}\right)$</td>
</tr>
</tbody>
</table>

Today:

Adaptive methods that obtain optimal rates without any prior knowledge
$\Rightarrow$ efficient and practical parallelization
Benefits of Adaptivity

- Does not require prior knowledge
- Saves expensive hyperparameter tuning
- Adapts to local structure
- Enables efficient & practical Parallelization
Large Batch Training & Parallelization

Minibatch SGD Update rule:

\[ x_{t+1} = x_t - \eta_t g_t \]

where \( g_t \) is a gradient estimate based on \( m \) samples.
Large Batch Training & Parallelization

Minibatch SGD Update rule:

\[ x_{t+1} = x_t - \eta_t g_t ; \quad \text{where } g_t \text{ is a gradient estimate based on } m \text{ samples} \]
Large Batch Training & Parallelization

Minibatch SGD Update rule:

\[ x_{t+1} = x_t - \eta_t g_t \; ; \quad \text{where } g_t \text{ is a gradient estimate based on } m \text{ samples} \]

Parallel SGD using Large Batch

- Use large batch size \( b \) & Distribute computation of gradient estimate across machines
Large Batch Training & Parallelization

Minibatch SGD Update rule:

\[ x_{t+1} = x_t - \eta_t g_t; \]  
where \( g_t \) is a gradient estimate based on \( m \) samples

Parallel SGD using Large Batch

- Use large batch size \( m \) & Distribute computation of gradient estimate across machines
- Master node collects estimates, updates & communicates weights
When Do we Benefit From Parallelization? (when do \( m \) machines are better than a single one?)

Using \( m \) machines in parallel:

- Variance decreases: \( \sigma_m = \sigma_1 / \sqrt{m} \)
- At every iteration \( T \) we use minibatchsize \( \propto m \) gradients \( \Rightarrow \) #Samples = \( mT \)
When Do we Benefit From Parallelization? (when do \( m \) machines are better than a single one?)

Using \( m \) machines in parallel:

- Variance decreases: \( \sigma_m = \sigma_1 / \sqrt{m} \)
- At every iteration \( T \) we use minibatchsize \( \propto m \) gradients \( \Rightarrow \) \( \# \)Samples = \( mT \)

Plugging this back to the rates we have seen before:

- Non-accelerated stochastic methods, \( m \) machines,

\[
\text{err}^{(m)}(T) \leq \frac{L}{T} + \frac{\sigma_1}{\sqrt{mT}} \Rightarrow m \leq (\# \text{Samples})^{1/2} \quad \text{(Effective Parallelization)}
\]
When Do we Benefit From Parallelization? (when do \( m \) machines are better than a single one?)

Using \( m \) machines in parallel:

- Variance decreases: \( \sigma_m = \sigma_1 / \sqrt{m} \)
- At every iteration \( T \) we use minibatchsize \( \propto m \) gradients \( \Rightarrow \) #Samples = \( mT \)

Plugging this back to the rates we have seen before:

- Non-accelerated stochastic methods, \( m \) machines,
  \[
  \text{err}^{(m)}(T) \leq \frac{L}{T} + \frac{\sigma_1}{\sqrt{mT}} \Rightarrow m \leq (\text{#Samples})^{1/2} \quad \text{(Effective Parallelization)}
  \]

- Accelerated stochastic methods, \( m \) machines,
  \[
  \text{err}^{(m)}(T) \leq \frac{L}{T^2} + \frac{\sigma_1}{\sqrt{mT}} \Rightarrow m \leq (\text{#Samples})^{3/4} \quad \text{(Effective Parallelization)}
  \]
When Do we Benefit From Parallelization? (when do \( m \) machines are better than a single one?)

Using \( m \) machines in parallel:

- Variance decreases: \( \sigma_m = \sigma_1 / \sqrt{m} \)
- At every iteration \( T \) we use minibatchsize \( \propto m \) gradients \( \Rightarrow \) \( \#\text{Samples} = mT \)

Plugging this back to the rates we have seen before:

- Non-accelerated stochastic methods, \( m \) machines,
  \[
  err^{(m)}(T) \leq T \frac{L}{T} + \frac{\sigma_1}{\sqrt{mT}} \Rightarrow m \leq (\#\text{Samples})^{1/2} \quad \text{(Effective Parallelization)}
  \]

- Accelerated stochastic methods, \( m \) machines,
  \[
  err^{(m)}(T) \leq T \frac{L}{T^2} + \frac{\sigma_1}{\sqrt{mT}} \Rightarrow m \leq (\#\text{Samples})^{3/4} \quad \text{(Effective Parallelization)}
  \]

Issue

- Achieving optimal rates (& parallelization) requires tuning according to \( m, L, \sigma, D \) \( \Rightarrow \) Impractical!
- Next: Adaptive Accelerated methods to the rescue
A classical approach: Line-search

- High level Idea: at every step tune the learning rate until a "good" condition holds (e.g. sufficient decrease)
  - Long history: Backtracking, Armijo, steepest descent...
  - Nesterov has designed an accelerated and adaptive line search method
  - has extensions to primal-dual optimization

Issues
- Line search methods are inappropriate for stochastic case!
- must set accuracy a priori

---

Another Approach: Polyak Stepsize

Update rule:

\[ x_{t+1} = x_t - \eta_t \nabla f(x_t) \; ; \; t = 1 \ldots T \]

Polyak stepsize:

\[ \eta_t = \frac{f(x_t) - f(x^*)}{\|g_t\|^2} \]

- Adaptivity: GD with Polyak stepsize is adaptive\(^1,2\). Nevertheless...

  - it is **inappropriate for stochastic** case!
  - requires prior knowledge of \( f(x^*) \)
  - does not obtain accelerated rates

---


The curious case of AdaGrad

Algorithm: General SGD

1: **Input:** Iterations $T$; $x_1$
2: **for** $t = 1, \ldots, T$ **do**
3: Obtain a gradient estimate $g_t$
4: $x_{t+1} = x_t - \eta_t g_t$
5: **end for**
6: **Output:** $\bar{x}_T = \frac{1}{T} \sum_{t=1}^{T} x_t$

AdaGrad (scalar)

$$\eta_t = \frac{D}{\sqrt{\sum_{\tau=1}^{t} \|g_{\tau}\|^2}}$$

AdaGrad’s Adaptivity

- **Non-smooth stochastic case:**
  $$\text{err}_T \leq GD / \sqrt{T}$$
- **Smooth stochastic case:**
  $$\text{err}_T \leq LD^2 / T + \sigma D / \sqrt{T}$$

---

The curious case of AdaGrad\(^1\)

**Algorithm: General SGD**

1. **Input:** Iterations \( T \); \( x_1 \)
2. **for** \( t = 1, \ldots, T \) **do**
3. Obtain a gradient estimate \( g_t \)
4. \( x_{t+1} = x_t - \eta_t g_t \)
5. **end for**
6. **Output:** \( \bar{x}_T = \frac{1}{T} \sum_{t=1}^{T} x_t \)

**AdaGrad (scalar)** \(^1\)

\[
\eta_t = \frac{D}{\sqrt{\sum_{\tau=1}^{t} \| g_{\tau} \|^2}}
\]

**AdaGrad’s Adaptivity**

- Non-smooth stochastic case:
  \[
  \text{err}_T \leq \frac{GD}{\sqrt{T}}
  \]
- Smooth stochastic case:
  \[
  \text{err}_T \leq \frac{LD^2}{T} + \frac{\sigma D}{\sqrt{T}}
  \]

AdaGrad does not accelerate! ⇒ Not ideal for parallelization

---

Towards an Accelerated Adaptive Method

- AdaGrad adapts to smoothness and noise but does not accelerate

\[
\text{AdaGrad: } \quad \text{err}_T \leq \frac{LD^2}{T} + \frac{\sigma D}{\sqrt{T}}
\]

\[
\text{Optimal Accelerated rate: } \quad \text{err}_T \leq \frac{LD^2}{T^2} + \frac{\sigma D}{\sqrt{T}}
\]

Q: Can we design a method that is both adaptive and accelerates?
AdaGrad with Importance Weights

Update rule:

$$x_{t+1} = x_t - \eta_t \alpha_t g_t \quad \text{(Weighted Gradients)}$$

Output:

$$\bar{x}_T \propto \sum_{t=1}^{T} \alpha_t x_t \quad \text{(Weighted Average)}$$

Weighted Learning rate:

$$\eta_t = \frac{D}{\sqrt{\sum_{\tau=1}^{t} \alpha_{\tau}^2 \|g_{\tau}\|^2}}$$
AcceleGrad \(^1\) Exploiting the linear coupling idea \(^2\)

\textbf{Algorithm:} AcceleGrad for unconstrained optimization

1: \textbf{Input:} Iterations \(T\); \(y_0, z_0 \in \mathbb{R}^p\)
2: \textbf{for} \(t = 0, \ldots, T - 1\) \textbf{do}
3: \hspace{1em} Obtain a gradient estimate \(g_t\) at \(x_t\)
4: \hspace{1em} \(\alpha_t \approx t + 1\)
5: \hspace{1em} \(\eta_t = D / \sqrt{G^2 + \sum_{t=0}^{t} \alpha_t^2 \|g_t\|^2}\)
6: \hspace{1em} \(x_{t+1} = \frac{1}{\alpha_t} z_t + (1 - \frac{1}{\alpha_t}) y_t\),

7: \hspace{1em} \(z_{t+1} = z_t - \alpha_t \eta_t g_t\)
8: \hspace{1em} \(y_{t+1} = x_{t+1} - \eta_t g_t\)
9: \textbf{end for}
10: \textbf{Output:} \(\bar{y}_T \propto \sum_{t=1}^{T} \alpha_t - 1 y_t\)

AcceleGrad\textsuperscript{1} Exploiting the linear coupling idea\textsuperscript{2}

\textbf{Algorithm:} AcceleGrad for unconstrained optimization

1: \textbf{Input:} Iterations \( T; y_0, z_0 \in \mathbb{R}^p \)
2: \textbf{for} \( t = 0, \ldots, T - 1 \) \textbf{do}
3: \hspace{1em} Obtain a gradient estimate \( g_t \) at \( x_t \)
4: \hspace{1em} \( \alpha_t \approx t + 1 \)
5: \hspace{1em} \( \eta_t = D / \sqrt{G^2 + \sum_{t=0}^{t} \alpha_t^2 \|g_t\|^2} \)
6: \hspace{1em} \( x_{t+1} = \frac{1}{\alpha_t} z_t + (1 - \frac{1}{\alpha_t}) y_t, \)
7: \hspace{1em} \( z_{t+1} = z_t - \alpha_t \eta_t g_t \)
8: \textbf{end for}
9: \textbf{Output:} \( \bar{y}_T \propto \sum_{t=1}^{T} \alpha_t - 1 \ y_t \)

**AcceleGrad**\(^1\) Exploiting the linear coupling idea\(^2\)

**Algorithm:** AcceleGrad for unconstrained optimization

1. **Input:** Iterations \(T\); \(y_0, z_0 \in \mathbb{R}^p\)
2. **for** \(t = 0, \ldots, T - 1\) **do**
3. Obtain a gradient estimate \(g_t\) at \(x_t\)
4. \(\alpha_t \approx t + 1\)
5. \(\eta_t = \frac{D}{\sqrt{G^2 + \sum_{\tau=0}^{t} \alpha_\tau^2 \|g_\tau\|^2}}\)
6. \(x_{t+1} = \frac{1}{\alpha_t}z_t + \left(1 - \frac{1}{\alpha_t}\right)y_t,\)
7. \(z_{t+1} = z_t - \alpha_t \eta_t g_t\)
8. \(y_{t+1} = x_{t+1} - \eta_t g_t\)
9. **end for**
10. **Output:** \(\bar{y}_T \propto \sum_{t=1}^{T} \alpha_t - 1 y_t\)

---

AcceleGrad\(^1\) Exploiting the linear coupling idea\(^2\)

Algorithm: AcceleGrad for unconstrained optimization

1. **Input:** Iterations \(T\); \(y_0, z_0 \in \mathbb{R}^p\)
2. **for** \(t = 0, \ldots, T - 1\) **do**
   3. Obtain a gradient estimate \(g_t\) at \(x_t\)
   4. \(\alpha_t \approx t + 1\)
   5. \(\eta_t = D/\sqrt{G^2 + \sum_{\tau=0}^{t} \alpha_\tau^2 \|g_\tau\|^2}\)
   6. \(x_{t+1} = \frac{1}{\alpha_t} z_t + (1 - \frac{1}{\alpha_t}) y_t\)
   7. \(z_{t+1} = z_t - \alpha_t \eta_t g_t\)
   8. \(y_{t+1} = x_{t+1} - \eta_t g_t\)
3. **end for**
4. **Output:** \(\bar{y}_T \propto \sum_{t=1}^{T} \alpha_t - 1 y_t\)

Comments

1. AcceleGrad does not require \(L, \sigma\), but requires \(G, D\)
2. Cannot handle constraints!
3. Optimal Accelerated guarantees up to \(\log T\) factors!
   \(\Rightarrow\) practical and effective parallelization

---


Logistic regression
- Data: RCV1
- Oracle: stochastic updates, different mini-batch size

Figure 5: Comparison of AdaGrad and AcceleGrad for logistic regression task using different minibatch sizes. We display the averaged iterates, $\bar{y}_T$ (top), as well as the non-averaged iterates, $y_t$ (bottom). Both methods use the same parameter $D = 104$.

Figure 6: Comparison of AdaGrad and AcceleGrad for training SVM using different minibatch sizes. We display the averaged iterates, $\bar{y}_T$ (top), as well as the non-averaged iterates, $y_t$ (bottom). Both methods use the same parameter $D = 104$. 
Towards an Accelerated Adaptive Method

Next, new techniques for achieving adaptive acceleration for constrained problems,

- Adaptive Learning rate ✓
- Importance weighting ✓
- Mirror Prox updates ←
- Querying gradients at **averages**
Nemirovski’s Mirror Prox

Mirror Prox update:

\[ x_{t+1} = x_t - \eta_t h_t \]

“Good” hints:

when \( h_t \approx \nabla f(x_{t+1}) \) \( \Rightarrow \) better performance
Nemirovski’s Mirror Prox\(^1\)

Mirror Prox method

1. Standard GD update: \( x_{t+\frac{1}{2}} = x_t - \eta_t \nabla f(x_t) \)
2. Taking a hint: \( h_t = \nabla f(x_{t+\frac{1}{2}}) \)
3. Optimistic update: \( x_{t+1} = x_t - \eta_t h_t \)

Intuition: when \( f \) is smooth \( h_t \approx \nabla f(x_{t+1}) \Rightarrow \) better performance when \( f \) is non-smooth \( h_t \) might not help, but is does not hurt to use it

---

\(^1\) A. Nemirovski, “Prox-method with rate of convergence \( O(1/t) \) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems.”, SIAM Journal on Optimization, 2004.
Nemirovski’s Mirror Prox

Mirror Prox method

1. Standard GD update: \[ x_{t+\frac{1}{2}} = x_t - \eta_t \nabla f(x_t) \]
2. Taking a hint: \[ h_t = \nabla f(x_{t+\frac{1}{2}}) \]
3. Optimistic update: \[ x_{t+1} = x_t - \eta_t h_t \]

Intuition:

when \( f \) is smooth \( h_t \approx \nabla f(x_{t+1}) \) \( \Rightarrow \) better performance
when \( f \) is non-smooth \( h_t \) might not help, but is does not hurt to use it

---

\(^1\) A. Nemirovski, “Prox-method with rate of convergence \( O(1/t) \) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems.”, SIAM Journal on Optimization, 2004.
Nemirovski’s Mirror Prox<sup>1</sup>

- Mirror Prox also works in stochastic case

**Mirror Prox method**

1. Standard GD update: $x_{t+\frac{1}{2}} = x_t - \eta_t g_t$
2. Taking a hint: $h_t$
3. Optimistic update: $x_{t+1} = x_t - \eta_t h_t$

where,

$$
\mathbb{E}[g_t|x_t] = \nabla f(x_t) \quad \& \quad \mathbb{E}[h_t|x_t] = \nabla f(x_{t+\frac{1}{2}})
$$

---

Towards an Accelerated Adaptive Method

- Combining the techniques below ⇒ adaptive acceleration
  - Adaptive Learning rate ✓
  - Importance weighting ✓
  - Optimistic updates ✓
  - Querying gradients at **averages** ✓
Towards an Accelerated Adaptive Method

Algorithm: AdaGrad

1: **Input:** Iterations $T$; $x_1 \in X \subset \mathbb{R}^d$
2: **for** $t = 1, \ldots, T$ **do**
3: Obtain a gradient estimate $g_t$ at $x_t$
4: Set:
   $$\eta_t = D / \sqrt{\sum_{\tau=1}^{t} \|g_{\tau}\|^2}$$
5: **Update:**
   $$x_{t+1} = x_t - \eta_t g_t$$
6: **end for**
7: **Output:** $\bar{x}_T \propto \sum_{t=1}^{T} x_t$

- Adaptive Learning rate ✓
- Importance weighting
- Optimistic updates
- Querying gradients at averages
Towards an Accelerated Adaptive Method

Algorithm: **Weighted AdaGrad**

1. **Input:** Iterations $T$; $x_1 \in \mathcal{X} \subset \mathbb{R}^d$
2. **for** $t = 1, \ldots, T$ **do**
   3. Set weight $\alpha_t = t$
   4. Obtain a gradient estimate $g_t$ at $x_t$
   5. Set:
      $$\eta_t = D / \sqrt{\sum_{\tau=1}^{t} \alpha_{\tau}^2 \|g_{\tau}\|^2}$$
   6. Update:
      $$x_{t+1} = x_t - \eta_t \alpha_t g_t$$
3. **end for**
4. **Output:** $\bar{x}_T \propto \sum_{t=1}^{T} \alpha_t x_t$

- Adaptive Learning rate ✓
- Importance weighting ✓
- Optimistic updates
- Querying gradients at averages
Towards an Accelerated Adaptive Method

**Algorithm:** Mirror Prox Weighted AdaGrad

1. **Input:** Iterations $T$; $x_1 \in \mathcal{X} \subset \mathbb{R}^d$
2. **for** $t = 1, ..., T$ **do**
3. Set weight $\alpha_t = t$
4. Obtain a gradient estimate $g_t$ at $x_t$
5. Set:
   \[ \eta_t = \frac{D}{\sqrt{1 + \sum_{\tau=1}^{t-1} \alpha_\tau^2 \|g_\tau - h_\tau\|^2}} \]
6. **Update:**
   \[ x_{t+\frac{1}{2}} = x_t - \eta_t \alpha_t g_t \]
   Compute $h_t$ an unbiased gradient estimate at $x_{t+\frac{1}{2}}$
   \[ x_{t+1} = x_t - \eta_t \alpha_t h_t \]
7. **end for**
8. **Output:** $\bar{x}_T \propto \sum_{t=1}^{T} \alpha_t x_t$
UnixGrad - Universal eXtra Gradient method

Algorithm: UnixGrad$^1$ = Anytime Optimistic Weighted AdaGrad

1. **Input:** Iterations $T$; $x_1 \in \mathcal{X} \subset \mathbb{R}^d$, weights $\alpha_t = t$
2. **for** $t = 1, \ldots, T$ **do**
3. Obtain a gradient estimate $g_t$ at $\bar{x}_t \propto \alpha_t x_t + \sum_{\tau=1}^{t-1} \alpha_\tau x_\tau + \frac{1}{2}$
4. Set:
   \[ \eta_t = \frac{D}{\sqrt{1 + \sum_{\tau=1}^{t-1} \alpha_\tau^2 \|g_\tau - h_\tau\|^2}} \]
5. **Update:**
   \[ x_{t+\frac{1}{2}} = x_t - \eta_t \alpha_t g_t \]
   Compute $h_t$ an unbiased gradient estimate at $\bar{x}_{t+\frac{1}{2}} \propto \alpha_t x_{t+\frac{1}{2}} + \sum_{\tau=1}^{t-1} \alpha_\tau x_{\tau+\frac{1}{2}}$
   \[ x_{t+1} = x_t - \eta_t \alpha_t h_t \]
6. **end for**
7. **Output:** $\bar{x}_{T+\frac{1}{2}} \propto \sum_{t=1}^{T} \alpha_t x_{t+\frac{1}{2}}$

---

UnixGrad - an Adaptive Accelerated Optimal Method

- UnixGrad ensures optimal guarantees,
  1. $O\left(\frac{GD}{\sqrt{T}}\right)$ - non-smooth deterministic/stochastic
  2. $O\left(\frac{LD^2}{T^2}\right)$ - smooth deterministic case
  3. $O\left(\frac{LD^2}{T^2} + \frac{\sigma D}{\sqrt{T}}\right)$ - smooth stochastic case

- Comments:
  - UnixGrad adapts to $G, L, \sigma^2$, but requires a bound on $D$
  - UnixGrad can be applied to constrained problems
  - No guarantees for non-convex problems!

---

Neural network training: ADAM vs. AcceleGrad

**Figure:** Resnet classifier optimization (train loss)

**Figure:** Resnet classifier optimization (test loss)
Conclusions

- Adaptive accelerated methods ⇒ **practical and efficient acceleration.** Still...lots of interesting questions.

**Adaptive methods for non-convex problems,**
- AdaGrad adapts to smoothness and noise in non-convex problems\(^1\).
  - Can we design an accelerated adaptive method?
- Can we design adaptive methods that provide stronger guarantees rather than stationarity?
- Is there a prevalent non-convex structure that we can adaptively exploit? (other than smoothness and noise)

**Strong-convexity** is a property that often arises in regularized problems, Simple algorithms automatically adapt to strong convexity under broad assumptions
- GD achieves linear rate with \(\eta = 1/L\), & SGD achieves \(O(1/T)\)-rate with \(\eta_t = O(1/t)\)
- PDHG achieves linear rate under metric subregularity\(^2\)

- Adaptive methods are promising but are not yet truly universal...
  - AdaGrad/Accelegrad/UniXgrad does not adapt to strong convexity
  - Adam-type does not adapt to strong convexity
  - MetaGrad comes close but is not universal yet\(^5\)

- Still seeking one algorithm to rule them all!

---