Optimization for Machine Learning

November 4, 2022

Program: Advanced stochastic gradient methods

- → Batch variants
- → Variance reduction
- → SG methods in practice

① Batch stochastic gradient methods

Pb $\quad$ minimize $_{w \in \mathbb{R}^d} \; f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w)$, $\quad f_i : \mathbb{R}^d \to \mathbb{R}$, C^1

f_i depends on the i^{th} data point of a dataset

Stochastic Gradient:

$$w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k)$$

i_k drawn at random in $\{1, \dots, n\}$

Gradient Descent: $\quad w_{k+1} = w_k - \alpha_k \nabla f(w_k)$

$$= w_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(w_k)$$

Batch stochastic Gradient:

$$w_{k+1} = w_k - \frac{\alpha_k}{|S_k|} \sum_{i \in S_k} \nabla f_i(w_k)$$

stochastic gradient (approximation of $\nabla f(w_k)$)

where S_k is a set of indices randomly drawn in $\{1, \dots, n\}$

→ Choosing $S_k = \{1, \dots, n\}$ (drawing n indices without replacement) gives GD

→ Choosing $|S_k| = 1$ recovers SG

For $1 < |S_k| < n$, can identify two regimes of batch methods: depending on the batch size $|S_k|$

1) Large batch regime : $1 << |S_k| < n$

⊕ Converges to a tight neighborhood of a "solution" (global minimum / first-order stationary point)

⊖ High per-iteration cost $\Rightarrow$ convergence curves are similar to that of GD

⊕ Still cheaper than GD <u>but</u> significantly more expensive than SG

2) Mini-batch regime : $1 < |S_k| << n$

⊕ Still much cheaper than GD, not so expensive compared to a SG iteration

⊕ Unlike SG, possibility to evaluate $\{\nabla f_i(x_k)\}_{i \in S_k}$ in parallel (with the right computing architecture/hardware)

⊖ Not necessarily better than SG, more sensitive to redundancies

Ex) $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ and $\mathcal{D}' = \{\underbrace{\mathcal{D}, \dots, \mathcal{D}}_{\times 10}\}$

SG with uniform sampling on $\mathcal{D}$ and $\mathcal{D}'$
$\Rightarrow$ Same probability of drawing a point corresponding to (x_1, y_1)

For batch SG with batch size > 1, this is not true

<u>NB</u> : Batch methods also reduce the variability between several runs of the same method (compared to SG)

Connections with CV rates

For SG, under the assumptions that f was $C^{1,1}_L$ and μ-strongly convex, and assuming that the $\{i_k\}$ were independent with $\mathbb{E}_{i_k}[\nabla f_{i_k}(w_k)] = \nabla f(w_k)$ and $\mathbb{E}_{i_k}[\|\nabla f_{i_k}(w_k)\|^2] \leq \|\nabla f(w_k)\|^2 + \sigma^2$, we proved that SG with stepsize $\alpha_k = \alpha$ converges to a neighborhood of the optimal value $\mathbb{E}[f(w_k)] \longrightarrow \left[f^*, f^* + \frac{L\alpha\sigma^2}{2\mu}\right]$

$\left(f^* = \min_{w \in \mathbb{R}^d} f(w)\right)$

↳ For batch SG: Suppose that every batch S_k consists of indices drawn in an iid fashion (e.g. uniformly at random in $\{1, \dots, n\}$ with replacement). Then, $\forall k$

$$\mathbb{E}_{S_k}\left[\frac{1}{|S_k|}\sum_{i \in S_k} \nabla f_i(w_k)\right] = \nabla f(w_k) \quad \text{and}$$

$$\mathbb{E}_{S_k}\left[\left\|\frac{1}{|S_k|}\sum_{i \in S_k} \nabla f_i(w_k)\right\|^2\right] \leq \|\nabla f(w_k)\|^2 + \frac{\sigma^2}{n_b}$$

"variance" lower for batch SG estimates

where $n_b = |S_k|$

$\frac{\sigma^2}{n_b} < \sigma^2$ when $n_b > 1$

$\Rightarrow$ CV to $\left[f^*, f^* + \frac{L\alpha\sigma^2}{2\mu n_b}\right] \subset \left[f^*, f^* + \frac{L\alpha\sigma^2}{2\mu}\right]$

⚠ By using a batch size $n_b > 1$, the number of accesses to data points per iteration increases. For a given epoch budget, a batch method with batch size $n_b > 1$ will perform less iterations than SG.

1 epoch $\equiv$ 1 iteration of GD

$\equiv n$ iterations of SG

$\equiv \frac{n}{n_b}$ iterations of batch SG with batch size n_b

$\Rightarrow$ Tradeoff: → Increasing the batch size reduces the variance of the stochastic gradient estimates

→ But increasing the batch size increases the number of data points accessed during 1 iteration

(2) Variance reduction techniques

Goal: Reduce the variance of stochastic gradients to avoid steps that are too different from gradient steps.

$\Rightarrow$ Reduce the overall variance within the algorithm

2.1. Increase the batch size

→ More expensive stochastic gradient estimates

→ Reduces the variance of those estimates and the variability between algorithmic runs

2.2 Averaging

SG iteration: $w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k)$

Averaging = maintaining an averaged iterate $\overline{w}_k = \frac{1}{k+1} \sum_{i=0}^{k} w_i$

$\{\overline{w}_k\}$ has better guarantees than $\{w_k\}$ (statistical theory) in the convex setting

Computing $\{\bar{w}_k\}$ requires an additional vector

$$\bar{w}_{k+1} = \frac{1}{k+2}\left((k+1)\bar{w}_k + w_{k+1}\right)$$

Many iterations of SG $\Rightarrow \frac{1}{k+1}$ gets small very quickly (numerical issues with computing $\bar{w}_k$)

2.3. Gradient aggregation methods

↳ Idea: Take full gradient steps every once in a while to reduce the variance of the method

SVRG (Stochastic Variance Reduced Gradient) 2013

↳ Two loops within the algorithm

* outer loop (k): compute a full gradient $\nabla f(\tilde{w}_k)$
* Inner loop (j): Perform m iterations

$$\tilde{w}_0 = \tilde{w}_k$$

$$\tilde{w}_{j+1} = \tilde{w}_j - \alpha_j \tilde{g}_j$$

$$\tilde{g}_j = \nabla f_{i_j}(\tilde{w}_j) - \nabla f_{i_j}(\tilde{w}_k) + \nabla f(\tilde{w}_k)$$

Full gradient computed once per outer loop

stochastic gradient from iteration j of the inner loop

where i_j drawn at random in $\{1, \dots, n\}$

already computed upon calculating $\nabla f(\tilde{w}_k) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\tilde{w}_k)$

$\mathbb{E}_{i_j}[\tilde{g}_j] = \nabla f(\tilde{w}_j)$ and $\tilde{g}_j$ has a lower variance than $\nabla f_{i_j}(\tilde{w}_j)$

Cost of an outer iteration = 1 iteration of GD + m iterations of SG

⊕ 1 iteration of SVRG ⇒ m updates of the iterate

⊕ Reduced variance

⊖ Need to store $\nabla f(w_k)$ and $\{\nabla f_i(w_k)\}_i$

⊕ Better CV theory than SG
(in particular, $\mathbb{E}[f(w_k)] \to f^*$ at a linear rate for f strongly convex $\approx$ GD rate)

Beneficial with a (very) large budget and efficient storage of $\nabla f_i(w_k)$ (not always possible)

SAGA (Stochastic Average Gradient ++)

Approach: Improve over SVRG by putting the emphasis on storing information rather than computing full gradients.

Algorithm:

Iteration 0: Compute $\{\nabla f_i(w_0)\}_{i=1}^n$ and define

$$\nabla f_i(w_{[i]}) = \nabla f_i(w_0) \quad \forall i$$

Iteration k: Draw i_k at random in $\{1, \dots, n\}$

Compute
$$g_k = \nabla f_{i_k}(w_k) - \nabla f_{i_k}(w_{[i_k]}) + \frac{1}{n}\sum_{i=1}^n \nabla f_i(w_{[i]})$$

and $w_{k+1} = w_k - \alpha_k g_k$

Set $\nabla f_{i_k}(w_{[i_k]}) = \nabla f_{i_k}(w_k)$

⊕ Same theoretical guarantees than SVRG

⊕ Same iteration cost (in terms of accesses to data points) than SG except for iteration 0

⊖ Need to store $\{\nabla f_i(w_{[i]})\}_{i=1}^n$ (and need to compute $\frac{1}{n}\sum_{i=1}^n \nabla f_i(w_{[i]})$)

Takeaways:

→ Gradient aggregation methods have nice theoretical guarantees and converge faster than SG in terms of epochs

→ But their storage cost and the need for full gradient evaluations make them prohibitive in certain applications like deep learning

→ SAGA implemented in Scikit-learn, and has had some recent success in reinforcement learning.

(3) Stochastic gradient methods for deep learning

General framework

$$\underset{w\in\mathbb{R}^d}{\text{minimize}} \quad f(w) = \frac{1}{n}\sum_{i=1}^n f_i(w)$$

At every iteration k, i_k is drawn at random in $\{1,\dots,n\}$ and $\nabla f_{i_k}(w_k)$ is used to compute w_{k+1}

Iteration k:

$$w_{k+1} = w_k - \alpha \; m_k \,/\, v_k$$

(componentwise division)

$\alpha > 0$ (constant for simplicity)

m_k: direction of the step

v_k: vector of scalings for the learning rate

$\forall i = 1..d$

$$[w_{k+1}]_i = [w_k]_i - \frac{\alpha}{[v_k]_i}[m_k]_i$$

↳ We recover SG by setting $m_k = \nabla f_{i_k}(w_k)$

and $v_k = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^d$

↳ The variants used in practice use more elaborate formulas for m_k and/or v_k that depend on the previous iterations

3.1. Stochastic gradient with momentum ("SGD with momentum")

↳ $v_k = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^d$, $m_k = \beta_1 m_{k-1} + (1-\beta_1)\nabla f_{i_k}(w_k)$

$(m_{-1} = 0)$ $\quad \beta_1 \in (0,1)$

$(\beta_1 = 0 \Rightarrow SG)$

↳ Related to accelerated gradient techniques, and in particular the heavy-ball method

$(w_{k+1} = w_k - \alpha \nabla f(w_k) + \beta_k (w_k - w_{k-1}))$

⇒ This method was used in the breakthrough results on training NN for image processing (~ 2012)

⇒ Very difficult to get theoretical guarantees for this method, especially in a (nonconvex) deep learning

3.2. AdaGrad (~ 2011)

↳ Issue: Scaling of learning rate per coordinate

↳ $m_k = \nabla f_{i_k}(w_k)$

$\forall i = 1..d$, $[R_k]_i = [R_{k-1}]_i + [\nabla f_{i_k}(w_k)]_i^2$ → sum of squares of all previous values of $[\nabla f_{i_j}(w_j)]_i$

$$[v_k]_i = \sqrt{[R_k]_i + \varepsilon}$$

$\Rightarrow$ estimate of the magnitude of $[\nabla f(w)]_i$

ε: For numerical stability (e.g. 10^{-8})

$\Rightarrow \forall k, \forall i$, the learning rate at iteration k for the i^{th} coordinate is

$$\frac{\alpha}{\sqrt{\sum_{j=0}^{k} [\nabla f_{i_j}(w_j)]_i^2 + \varepsilon}}$$

⊕ Theory, very efficient for problems with sparse gradients (a lot of zero coordinates) $\Rightarrow$ AdaGrad is efficient for recommender systems

⊖ If the problem is not sparse, AdaGrad might drive the stepsize too quickly towards 0

3.3. RMSProp

↳ For training very deep networks

↳ $m_k = \nabla f_{i_k}(w_k)$,

$$[R_k]_i = \beta_2 [R_{k-1}]_i + (1-\beta_2) [\nabla f_{i_k}(w_k)]_i^2$$

($\beta_2 \in (0,1)$; weighted average of previous and current information)

$$[v_k]_i = \sqrt{[R_k]_i + \varepsilon}$$

3.4. Adam (2015)

↳ Idea: Combine momentum and learning rate scaling through geometric averages

$$m_k = \frac{(1-\beta_1)\sum_{j=0}^{k} \beta_1^{k-j} \nabla f_{ij}(w_j)}{1-\beta_1^{k+1}}$$

$\forall i = 1 \dots d$

$$[v_k]_i = \sqrt{\frac{(1-\beta_2)\sum_{j=0}^{k} \beta_2^{k-j} [\nabla f_{ij}(w_j)]_i^2}{1-\beta_2^{k+1}}} + \varepsilon$$

→ The method of choice today for training deep models, especially in NLP (Adam & AdamW)

→ CV proof in 2015 was found incorrect in 2018