Stochastic Gradient Descent. Finite-sum problems

Seminar

Optimization for ML. Faculty of Computer Science. HSE University

Finite-sum problem

We consider classic finite-sample average minimization:

$$\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

The gradient descent acts like follows:

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(x) \tag{GD}$$

- Convergence with constant α or line search.
- Iteration cost is linear in n. For ImageNet $n \approx 1.4 \cdot 10^7$, for WikiText $n \approx 10^8$.

Finite-sum problem

We consider classic finite-sample average minimization:

$$\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

The gradient descent acts like follows:

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(x) \tag{GD}$$

- Convergence with constant α or line search.
- Iteration cost is linear in n. For ImageNet $n \approx 1.4 \cdot 10^7$, for WikiText $n \approx 10^8$.

Let's/ switch from the full gradient calculation to its unbiased estimator, when we randomly choose i_k index of point at each iteration uniformly:

$$x_{k+1} = x_k - \alpha_k \nabla f_{i_k}(x_k) \tag{SGD}$$

With $p(i_k = i) = \frac{1}{n}$, the stochastic gradient is an unbiased estimate of the gradient, given by:

$$\mathbb{E}[\nabla f_{i_k}(x)] = \sum_{i=1}^{n} p(i_k = i) \nabla f_i(x) = \sum_{i=1}^{n} \frac{1}{n} \nabla f_i(x) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x) = \nabla f(x)$$

This indicates that the expected value of the stochastic gradient is equal to the actual gradient of $f(x)$.

Results for Gradient Descent

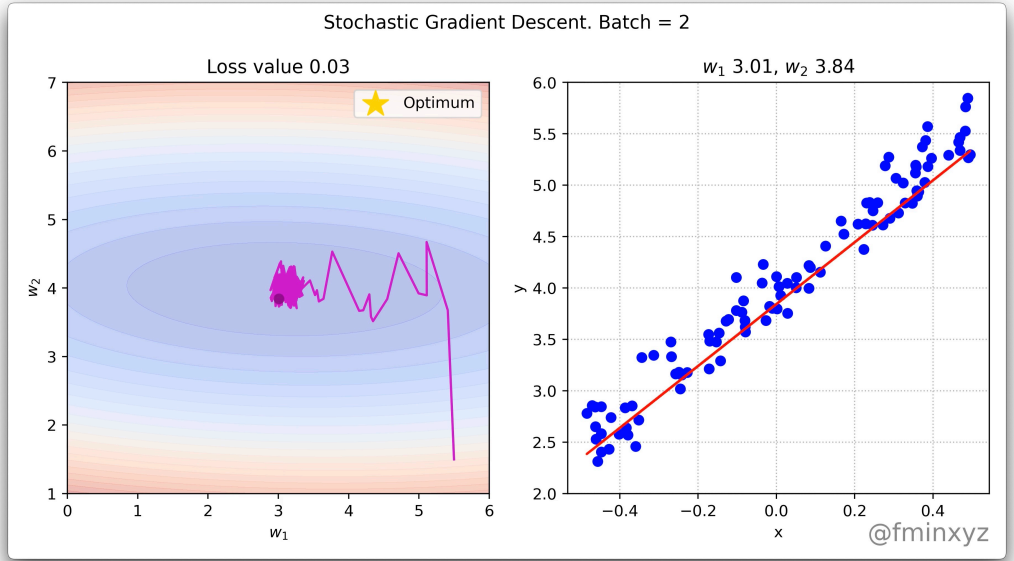
Stochastic iterations are n times faster, but how many iterations are needed?

If ∇f is Lipschitz continuous then we have:

Assumption	Deterministic Gradient Descent	Stochastic Gradient Descent
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Convex	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Non-Convex	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- Stochastic has low iteration cost but slow convergence rate.
  - Sublinear rate even in strongly-convex case.
  - Bounds are unimprovable under standard assumptions.
  - Oracle returns an unbiased gradient approximation with bounded variance.
- Momentum and Quasi-Newton-like methods do not improve rates in stochastic case. Can only improve constant factors (bottleneck is variance, not condition number).

Typical behaviour



Computational experiments

Visualization of SGD.

Let's look at computational experiments for SGD .

Adagrad (Duchi, Hazan, and Singer 2010)

Very popular adaptive method. Let $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, and update for $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Notes:

- AdaGrad does not require tuning the learning rate: $\alpha > 0$ is a fixed constant, and the learning rate decreases naturally over iterations.
- The learning rate of rare informative features diminishes slowly.
- Can drastically improve over SGD in sparse problems.
- Main weakness is the monotonic accumulation of gradients in the denominator. AdaDelta, Adam, AMSGrad, etc. improve on this, popular in training deep neural networks.
- The constant ϵ is typically set to 10^{-6} to ensure that we do not suffer from division by zero or overly large step sizes.

RMSProp (Tieleman and Hinton, 2012)

An enhancement of AdaGrad that addresses its aggressive, monotonically decreasing learning rate. Uses a moving average of squared gradients to adjust the learning rate for each weight. Let $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ and update rule for $j = 1, \dots, p$:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)(g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Notes:

- RMSProp divides the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight.
- Allows for a more nuanced adjustment of learning rates than AdaGrad, making it suitable for non-stationary problems.
- Commonly used in training neural networks, particularly in recurrent neural networks.

Adadelta (Zeiler, 2012)

An extension of RMSProp that seeks to reduce its dependence on a manually set global learning rate. Instead of accumulating all past squared gradients, Adadelta limits the window of accumulated past gradients to some fixed size w. Update mechanism does not require learning rate α:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)(g_j^{(k)})^2$$

$$\tilde{g}_j^{(k)} = \frac{\sqrt{\Delta x_j^{(k-1)} + \epsilon}}{\sqrt{v_j^{(k)} + \epsilon}} g_j^{(k)}$$

$$x_j^{(k)} = x_j^{(k-1)} - \tilde{g}_j^{(k)}$$

$$\Delta x_j^{(k)} = \rho \Delta x_j^{(k-1)} + (1-\rho)(\tilde{g}_j^{(k)})^2$$

Notes:

- Adadelta adapts learning rates based on a moving window of gradient updates, rather than accumulating all past gradients. This way, learning rates adjusted are more robust to changes in model's dynamics.
- The method does not require an initial learning rate setting, making it easier to configure.
- Often used in deep learning where parameter scales differ significantly across layers.

Adam (Kingma and Ba, 2014) [1] [2]

Combines elements from both AdaGrad and RMSProp. It considers an exponentially decaying average of past gradients and squared gradients.

EMA:
$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$
$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2) \left(g_j^{(k)}\right)^2$$

Bias correction:
$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}$$
$$\hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

Update:
$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Notes: * It corrects the bias towards zero in the initial moments seen in other methods like RMSProp, making the estimates more accurate. * One of the most cited scientific papers in the world * In 2018-2019, papers were published pointing out errors in the original article * Fails to converge on some simple problems (even convex ones) * Somehow works exceptionally well for certain complex problems * Performs much better for language models than for computer vision tasks — why?

[1] Adam: A Method for Stochastic Optimization
[2] On the Convergence of Adam and Beyond

AdamW (Loshchilov & Hutter, 2017)

Addresses a common issue with ℓ_2 regularization in adaptive optimizers like Adam. Standard ℓ_2 regularization adds $\lambda\|x\|^2$ to the loss, resulting in a gradient term λx. In Adam, this term gets scaled by the adaptive learning rate $\left(\sqrt{\hat{v}_j} + \epsilon\right)$, coupling the weight decay to the gradient magnitudes.

AdamW decouples weight decay from the gradient adaptation step.

Update rule:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2)(g_j^{(k)})^2$$

$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}, \quad \hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \left(\frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon} + \lambda x_j^{(k-1)} \right)$$

Notes:

- The weight decay term $\lambda x_j^{(k-1)}$ is added *after* the adaptive gradient step.
- Widely adopted in training transformers and other large models. Default choice for huggingface trainer.