Stochastic Gradient Descent

Weihang Chen, Xingchen Chen, Jinxiu Liang, Cheng Xu,
Zehao Chen and Donglin He

March 26, 2017

Outline

- What is Stochastic Gradient Descent
- Comparison between BGD and SGD
- Analysis on SGD
- Extensions and Variants

What is Stochastic Gradient Descent? I

- Structural Risk Minimization in Machine Learning
  - Given samples $\{(x_i, y_i)\}_{i=1}^{n}$ and a loss function $l(h, y)$
  - Find a prediction function $h(x; w)$ by minimizing a risk measure

$$R(w) = \sum_{i=1}^{n} l(h(x_i; w), y_i) = \sum_{i=1}^{n} f_i(w)$$

  - Update w via BGD

$$w^{(k+1)} = w^{(k)} - t_k \nabla R\left(w^{(k)}\right) = w^{(k)} - t_k \sum_{i=1}^{n} \nabla f_i\left(w^{(k)}\right)$$

What is Stochastic Gradient Descent? II

- Update w via SGD

$$w^{(k+1)} = w^{(k)} - t_k \nabla R\left(w^{(k)}\right) = w^{(k)} - t_k \nabla f_{i_k}\left(w^{(k)}\right)$$

- Suppose we want to minimize the sum of functions

$$min \sum_{i=1}^{m} f_i(x),\ i = 1, 2, \ldots, m$$

- BGD would sum all the gradients

$$x^{(k+1)} = x^{(k)} - t_k \sum_{i=1}^{n} \nabla f_i\left(x^{(k)}\right),\ k = 1, 2, \ldots$$

What is Stochastic Gradient Descent? III

- SGD instead looks at each gradient individually

$$x^{(k+1)} = x^{(k)} - t_k \nabla f_{i_k}\left(x^{(k)}\right), \; k = 1, 2, \ldots$$

Where $i_k \in \{1, \ldots, m\}$ is some chosen index at iteration k

  - Random rule: choose $i_k \in \{1, \ldots, m\}$ uniformly at random (more commom)
  - Circle rule: choose $i_k = 1, 2, \ldots, m, 1, 2, \ldots, m, \ldots$

Comparison between BGD and SGD

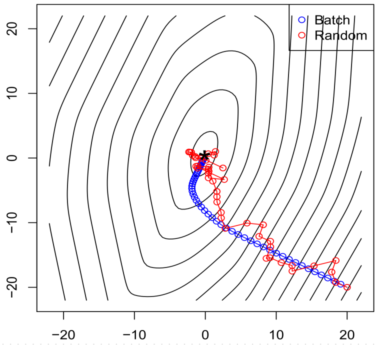


Figure: The "classic picture"

Gradient computation:

- Batch steps: $O(np)$
  - Doable when n is moderate, but not when $n \approx 5 \times 10^8$
- Stochastic steps: $O(p)$
  - So clearly, e.g., 10K stochastic steps are much more affordable
- Rule of thumb: SGD thrive far from optimum and struggle close to optimum

Comparison between BGD and SGD

- Update w via BGD
  - More expensive steps
  - Opportunities for parallelism
- Update w via SGD
  - Very cheap iteration
  - Descent in expectation
- Intuition
  - Using all the sample data in every iteration is inefficient
  - Data involves a good deal of redundancy in many applications
    - Suppose data is 10 copies of a set S.Iteration of BGD 10 times more expensive, while SGD performs same computations
  - Sometimes working with half of the training set is sufficient

Learning Rate Analysis

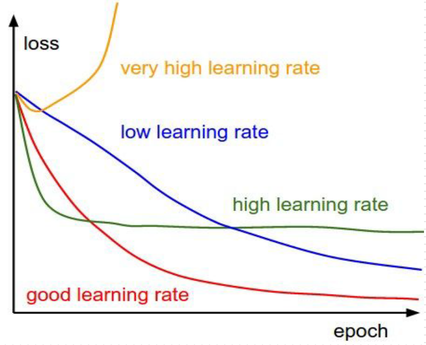


Figure: Effects of learning rate on loss

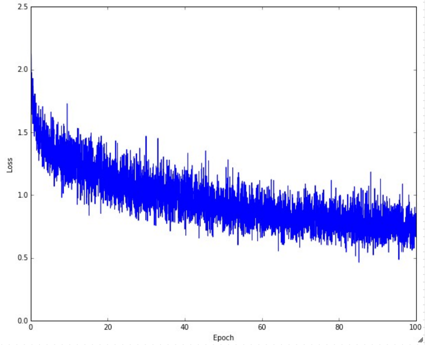


Figure: An example of a typical loss func

Converge Analysis

- Computationally, m stochastic steps$\approx$one batch step
- But what about progress?
  - BGD(one step):

$$x^{(k+1)} = x^{(k)} - t \sum_{i=1}^{m} \nabla f_i \left(x^{(k)} \right)$$

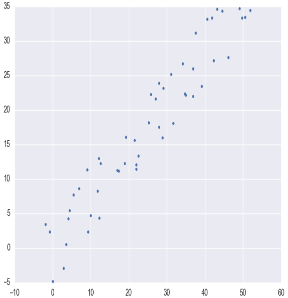
  - SGD(Cyclic rule,$i_k = i$,m steps):

$$x^{(k+m)} = x^{(k)} - t \sum_{i=1}^{m} \nabla f_i \left(x^{(k+i-1)} \right)$$

  - Difference in direction is $\sum_{i=1}^{m} \left[\nabla f_i \left(x^{(k+i-1)} \right) - \nabla f_i \left(x^{(k)} \right) \right]$
- So SGD should converge if each $\nabla f_i(x)$ doesn't vary wildly with x

Example

Problem:



Solution:

- The linear regression loss:

$$\min_{w} \sum_{i=1}^{m} \frac{1}{2}(y_i - w_i x_i)^2$$

- Update w via BGD:

$$w^{(k+1)} = w^{(k)} - t_k \sum_{i=1}^{m} \left(w_i^{(k)} x_i^2 - y_i x_i \right)$$

- Update w via SGD:

$$w^{(k+1)} = w^{(k)} + t_k \left(w_{i_k}^{(k)} x_{i_k}^2 - x_{i_k} y_{i_k} \right)$$

Example

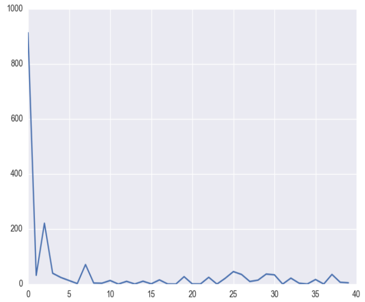


Figure: SGD loss-iteration times

Figure: Result of linear regression via SGD

Mini-Batch Gradient Descent

- Batch Gradient Descent

$$x^{(k+1)} = x^{(k)} - t_k \sum_{i=1}^{m} \nabla f_i \left(x^{(k)} \right)$$

- Stochastic Gradient Descent

$$x^{(k+1)} = x^{(k)} - t_k \nabla f_{i_k} \left(x^{(k)} \right)$$

- mini-Batch Gradient Descent

$$x^{(k+1)} = x^{(k)} - t_k \sum_{i=1}^{m'} \nabla f_{i_k} \left(x^{(k)} \right)$$

Challenges

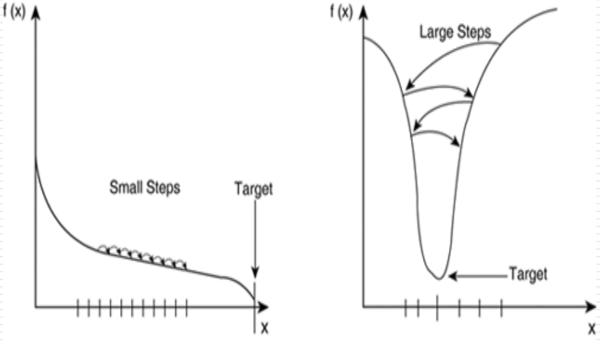


Figure: Problems with the learning rate

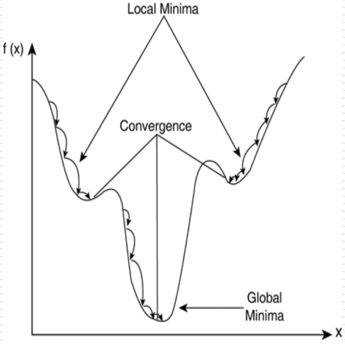


Figure: Local Minima

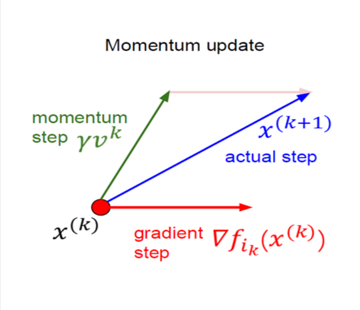
SGD with momentum

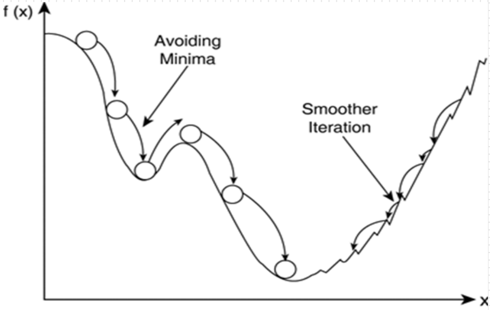
- Accelerate SGD in the relevant direction and dampens oscillations
  - Take a big jump in direction of updated accumulated gradient
  - Compute the gradient at the current location

$$\nu^{(k+1)} = \gamma \nu^{(k)} + t_k \nabla f_{i_k}\left(x^{(k)}\right)$$

$$x^{(k+1)} = x^{(k)} - \nu^{(k+1)}$$

SGD: $x^{(k+1)} = x^{(k)} - t_k \nabla f_{i_k}\left(x^{(k)}\right)$





Nesterov Accelerated Gradient

- Accelerate SGD in the relevant direction and dampens oscillations
  - Take a big jump in direction of previous accumulated gradient
  - Measure gradient where you end up and make a correction

$$\hat{x}^{(k)} = x^{(k)} + \gamma\nu^{(k)}$$

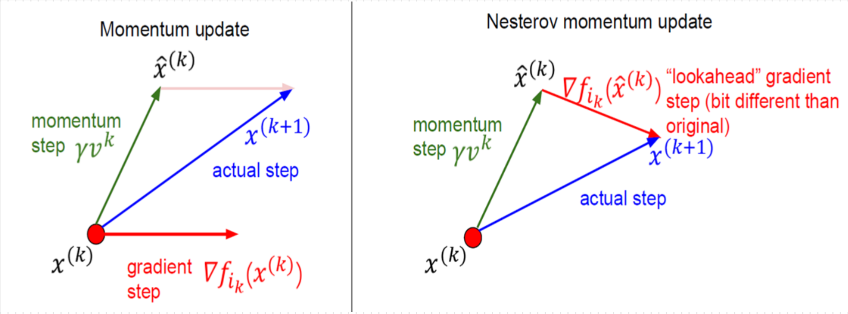
$$\nu^{(k+1)} = \gamma\nu^{(k)} + t_k \nabla f_{i_k}\left(\hat{x}^{(k)}\right)$$

$$x^{(k+1)} = x^{(k)} - \nu^{(k+1)}$$

SGD with momentum

$$\nu^{(k+1)} = \gamma\nu^{(k)} + t_k \nabla f_{i_k}\left(x^{(k)}\right)$$

$$x^{(k+1)} = x^{(k)} - \nu^{(k+1)}$$



Adaptive Gradient Algorithm

- Adapts the learning rate to the parameters
  - Performs larger updates for infrequent
  - Performs smaller updates for frequent parameters
- It is well-suited for dealing with sparse data

$$\text{SGD: } \boxed{x^{(k+1)} = x^{(k)} - t_k \nabla f_{i_k}\left(x^{(k)}\right)}$$

$$\nu^{(k+1)} = \nu^{(k)} + \nabla f_{i_k}\left(x^{(k)}\right)^2$$

$$x^{(k+1)} = x^{(k)} - \frac{\alpha}{\sqrt{\nu^{(k+1)} + \epsilon}} \nabla f_{i_k}\left(x^{(k)}\right)$$

ϵ is a smoothing term that avoids division by zero(usually on the order of $1e^{-8}$)

Adadelta

- Restricts window of accumulated past gradients to fixed size
  - Reduce AdaGrad's aggressive, monotonically decreasing learning rate

As a fraction γ similarly to the Momentum term

$$v^{(k+1)} = \gamma v^{(k)} + (1-\gamma)\nabla f_{i_k}\left(x^{(k)}\right)^2$$

$$x^{(k+1)} = x^{(k)} - \frac{\alpha}{\sqrt{v^{(k+1)} + \epsilon}}\nabla f_{i_k}\left(x^{(k)}\right)$$

Runing Average v^k at step k depends only on the previous average and the current gradient (as a fraction γ similarly to the Momentum term)

Adaptive Moment Estimation

- Keeps an average of past gradients additionally
  - Similar to momentum
- m_t and v_t are biased towards zero
  - In the initial time steps as they are initialized as vectors of 0's
  - When the decay rates are small (i.e. β_1 and β_2 are close to 1)

$$m^{(k+1)} = \beta_1 m^{(k)} + (1 - \beta_1) \nabla f_{i_k} \left(x^{(k)} \right)$$

$$v^{(k+1)} = \beta_2 v^{(k)} + (1 - \beta_2) \nabla f_{i_k} \left(x^{(k)} \right)^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$x^{(k+1)} = x^{(k)} - \frac{t_k}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

bias-corrected first and second moment estimates

Which optimizer to use?

- You should use one of the adaptive learning-rate methods:
  - If input data is sparse.
  - For faster convergence and deep or complex neural network training.
- Insofar, adadelta and adam are very similar algorithms that do well in similar circumstances.
- Adam slightly outperform adadelta towards the end of optimization as gradients become sparser.
- Insofar, adam might be the best overall choice.

Reference

- Hongmin Cai(2016): Sub-gradient Method, Lecture 7
- Cnblogs Murongxixi(2013): Stochastic Gradient Descent
- Leon Bottou(2016): Optimization Methods for Large-Scale Machine Learning
- Abdelkrim Bennar(2007): Almost sure convergence of a stochastic approximation process in a convex set
- A. Shapiro, Y. Wardi(1996): Convergence Analysis of Gradient Descent Stochastic Algorithms
- Wikipedia: Stochastic gradient descent
- Sebastian Ruder (2016): An overview of gradient descent optimization algorithms
- Zhihua Zhou(2016): Machine Learning, Chapter 6

Thank you for your time!