Mathematics Foundations for Computer Science

GRADIENT DESCENT

Lecturer: Dr. Nguyen An Khuong

Students: Le Huu Trong - 2470578
Pham Duc Duy Anh - 2470748
Nguyen Trong Nghia - 2470737
Huynh Bao Long - 2470751
Dinh Tien Manh - 2470569

Contents

Contents

Introduction

- Many optimization problems in computer science and machine learning involve minimizing a loss function.
- Direct minimization of this loss function can be very computationally expensive when the data set is large
- The Gradient Descent method and its variants such as Stochastic Gradient Descent (SGD) and Mini-batch SGD.

Contents

One-Dimensional Gradient Descent

Taylor expansion:

$$f(x+\epsilon) = f(x) + \epsilon f'(x) + O\left(\epsilon^2\right)$$

Choose $\epsilon = -\eta f'(x)$. Taylor expansion become:

$$f\left(x - \eta f'(x)\right) = f(x) - \eta f'^2(x) + O\left(\eta^2 f'^2(x)\right)$$

If the derivative $f'(x) \neq 0$ does not vanish we make progress since $\eta f'^2(x) > 0$.

$$f\left(x - \eta f'(x)\right) \lessapprox f(x)$$

Use $x \leftarrow x - \eta f'(x)$. to iterate x, the value of function $f(x)$ might decline

Examples with function $f(x) = x^2 + 5sin(x)$

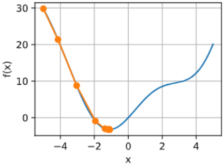


Figure: Gradient descent with step size $\eta = 0.1$ and start point $x = -5$

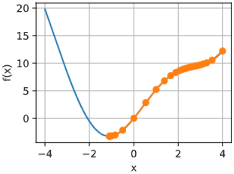


Figure: Gradient descent with step size $\eta = 0.1$ and start point $x = 4$

Examples with function $f(x) = x^2 + 5sin(x)$

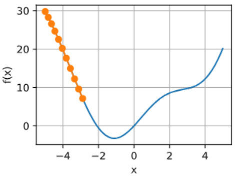


Figure: Gradient descent with step size $\eta = 0.02$ and start point $x = -5$

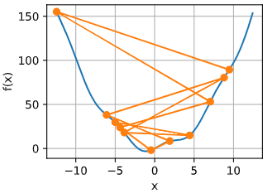


Figure: Gradient descent with step size $\eta = 1.1$ and start point $x = -5$

Local Minima

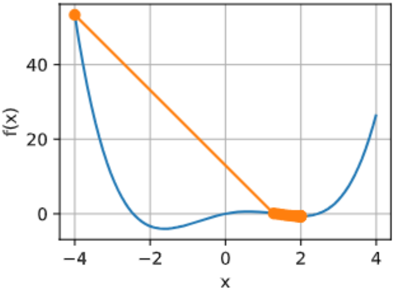


Figure: $f(x) = 0.25x^4 - \frac{1}{3}x^3 - 1.5x^2 + 2x$

Local minimum at $x = 2$, $f(2) \approx -0.67$ and global minimum at $x \approx -1.618$, $f(-1.618) \approx -4.04$. Step size $\eta = 0.08$ start point $x = -4$

Contents

Challenges with Standard Gradient Descent

For a large dataset $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$:

$$\nabla f(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\mathbf{w})$$

Limitations

- High computational cost per update (the computational cost for each independent variable iteration is $O(n)$)
- Inefficient when data has high redundancy

Stochastic Gradient Descent Updates

Main Idea

Select a random sample (or group of samples) to estimate the gradient and update $\mathbf{w}$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla f_{i_t}(\mathbf{w})$$

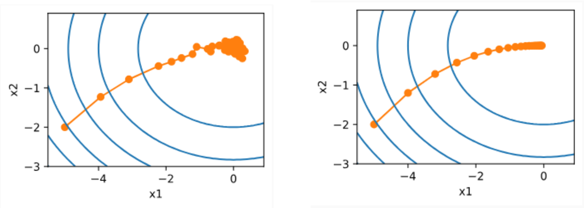
where $i_t \sim$ Uniform$(1, n)$ is the randomly selected sample index at step t and η is learning rate

- The expectation of the stochastic gradient equals the true gradient:

$$\mathbb{E}[\nabla f_i(\mathbf{w})] = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{w}) = \nabla f(\mathbf{w})$$

- Computes gradient over just 1 sample ($O(1)$ per update step)

Stochastic Gradient Descent Updates



(a) Stochastic Gradient Descent

(b) Gradient descent

Figure: Comparison of the convergence on the objective function $f(x_1, x_2) = x_1^2 + 2x_2^2$

$\Rightarrow$ The uncertainty injected by the instantaneous gradient via $\eta \nabla f_i(\mathbf{w})$ leads to a noisy update trajectory, oscillating around the optimum.

Why Use a Dynamic Learning Rate?

Why Use a Dynamic Learning Rate?

- Fixed learning rates can be suboptimal:
  - Too large $\Rightarrow$ instability
  - Too small $\Rightarrow$ slow convergence

Why Use a Dynamic Learning Rate?

- Fixed learning rates can be suboptimal:
  - Too large $\Rightarrow$ instability
  - Too small $\Rightarrow$ slow convergence
- Dynamic learning rate helps:
  - Speed up early training
  - Fine-tune later epochs

Why Use a Dynamic Learning Rate?

- Fixed learning rates can be suboptimal:
  - Too large $\Rightarrow$ instability
  - Too small $\Rightarrow$ slow convergence
- Dynamic learning rate helps:
  - Speed up early training
  - Fine-tune later epochs

How fast should the learning rate decay?

- Decay too fast $\Rightarrow$ premature convergence
- Decay too slow $\Rightarrow$ wasted computation

Piecewise Constant Learning Rate

$$\eta(t) = \eta_i \quad \text{if } t_i \leq t < t_{i+1}$$

- Decrease η after predefined steps or if progress stalls.
- Simple, practical, widely used in deep learning.

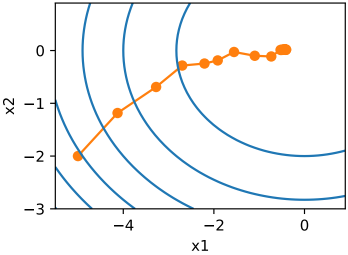


Figure: Optimization progress with piecewise constant LR schedule

Exponential Decay

$$\eta(t) = \eta_0 \cdot e^{-\lambda t}$$

- Rapid decay over time.
- Risk of stopping too early

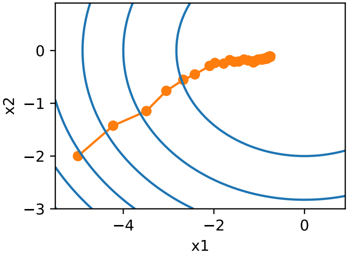


Figure: Optimization progress with exponential LR schedule

Polynomial Decay

$$\eta(t) = \eta_0 \cdot (\beta t + 1)^{-\alpha}$$

- Slower, more flexible decay.
- Often used with $\alpha = 0.5$.

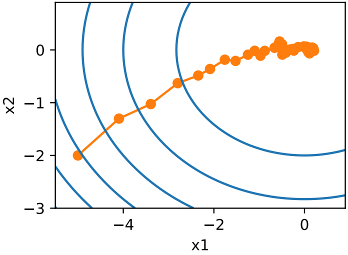


Figure: Optimization progress with polynomial LR schedule

Contents

What is Minibatch ?

- Two main methods:
  - Full-batch Gradient Descent
  - Stochastic Gradient Descent
- Minibatch helps balance between the two methods.

Vectorization and Computational Efficiency

When performing matrix multiplication, there are several approaches:

$$A = BC$$

- Element-wise computation by multiplying corresponding rows and columns.
- Column-wise computation, reducing memory access costs.
- Row-wise computation, optimizing cache usage.
- Block-wise computation, leveraging memory access efficiency.

In practice, the fourth approach is the most efficient, reducing latency and optimizing bandwidth. This is also the idea applied in Minibatch Gradient Descent.

Minibatch

$$\mathbf{g}_t = \partial \mathbf{w} f(\mathbf{x}_t, \mathbf{w})$$

$$\mathbf{g}_t = \partial_{\mathbf{w}} \frac{1}{|\mathcal{B}_t|} \sum_{i \in \mathcal{B}_t} f(\mathbf{x}_i, \mathbf{w})$$

- Group data into minibatches instead of processing each point individually.
- Reduce gradient variance, leading to more stable convergence.
- Increase computational efficiency, leveraging parallel processing capabilities.
- Balance between accuracy and speed, optimizing hardware resources.

Contents

Overview

- Using the Linear Regression model with respective optimization algorithm:
  - Full-batch Gradient Descent
  - Stochastic Gradient Descent
  - Minibatch Stochastic Gradient Descent
- Dataset: Airfoil Self-Noise
  - N $\approx$ 1500 samples, 5 input features and 1 output target.
  - Predict noise level generated by an airfoil under specific aerodynamic conditions.

Implementation with GD

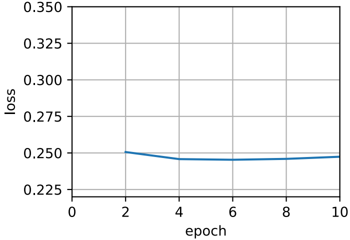


Figure: Training result with Gradient Descent

Result

- Loss: 0.247, 0.024 sec/epoch.
- Model parameters are only updated once per epoch.
- Little improvement between epochs.

Implementation with SGD

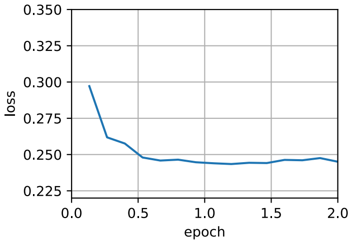


Figure: Training result with Stochastic Gradient Descent

Result

- Loss: 0.245, 0.382 sec/epoch.
- Model parameters are updated N times per epoch.
- Sharp decline in value of loss function but slows down after.
- Consume more time than GD to train.

Implementation with Minibatch SGD

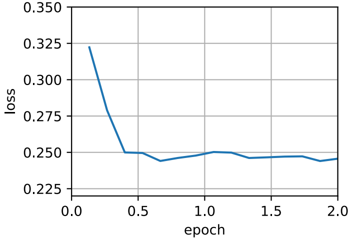


Figure: Training result with Minibatch Stochastic Gradient Descent (1)

Result with batch size = 100

- Loss: 0.247, 0.019 sec/epoch.
- Model parameters are updated 15 times per epoch.
- Sharp decline in value of loss function but slows down after.
- Consume less time to train than GD and SGD.

Implementation with Minibatch SGD (cont.)

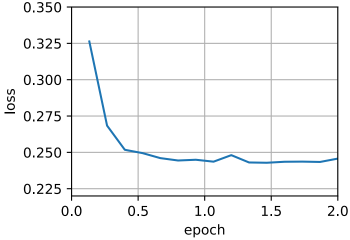


Figure: Training result with Minibatch Stochastic Gradient Descent (2)

Result with batch size = 10

- Loss: 0.247, 0.050 sec/epoch.
- Model parameters are updated 150 times per epoch.
- Sharp decline in value of loss function but slows down after.
- Consume more time to train than GD but less than SGD.

Summary

Factor	GD	SGD	Minibatch SGD
Data size used each step	Data set	A sample	A small set
Parameters updating	1 time/epoch	N times/epoch	N/B times/epoch
Calculation Speed	Slow	Fast	Moderate
Gradient Stability	Stable, less fluctuating	Strong oscillation	Moderate oscillation
Convergence	Slow but smooth	Fast initially, may fluctuate	Balance
Local Minima Escape	Low	High	Pretty High
Required RAM	High	Low	Moderate
Practical used	Rarely	Mostly for Online learning	Widely used

Q & A