Gradient Based Optimization III

Asst. Prof. Dr.-Ing. Sudchai Boonto

Department of Control System and Instrumentation Engineering
King Mongkut's University of Technology Thonburi
Thailand

September 3, 2025

Objective

At the end of this chapter you should be able to:

- Mathematically define the optimality conditions for an unconstrained problem.
- Describe, implement, and use line-search-based methods.
- Gradient Descent based method

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a variant of gradient descent that scales to very high dimensional optimization problems, making it suitable for large-scale neural network training.

- SGD makes the most sense in the context of machine learning, where we are optimizing the parameters $\boldsymbol{\theta}$ of a function $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})$, such as a neural network, to best fit observed data.
- For a given pairs of input-output data $\{\mathbf{x}_i, \mathbf{y}_j\}_{i=1}^N$, we seek to find the parameters $\boldsymbol{\theta}$ so that $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_j)$ best approximates $\mathbf{y}_j$, averaged over the data:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \sum_{j=1}^{N} \|\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_j) - \mathbf{y}_j\|^2$$

- This is a nonlinear least squares problem, which may be approaches via iterative descent methods. It is possible to compute the gradient of the loss function $\mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) = \|\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) - \mathbf{y}\|^2$ with respect to $\boldsymbol{\theta}$.

Stochastic Gradient Descent

From the objective function

$$V(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{\theta}^T\mathbf{x}_{(i)} - \mathbf{y}_{(i)}\right)^2$$

$\mathbf{x}_{(i)}$ means $\mathbf{x}$ of batch i. The gradient descent step (negative gradient)

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha\nabla_{\boldsymbol{\theta}}V(\boldsymbol{\theta})$$

We need to calculate the gradients for the whole dataset to perform just one update, batch gradient descent can be very slow and is intractable for datasets that do not fit in memory. Batch gradient descent also does not allow us to updata our model online.

$$\nabla_{\boldsymbol{\theta}}V(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial}{\partial\boldsymbol{\theta}_1}V(\theta) \\ \frac{\partial}{\partial\boldsymbol{\theta}_2}V(\theta) \\ \vdots \\ \frac{\partial}{\partial\boldsymbol{\theta}_n}V(\theta) \end{bmatrix} = \frac{2}{N}\mathbf{X}^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

Stochastic Gradient Descent

The SGD can improve the speed of the batch gradient descent

$$\boldsymbol{\theta}_{(k+1)} = \boldsymbol{\theta}_{(k)} - \alpha \nabla_{\boldsymbol{\theta}} V(\boldsymbol{\theta}, \mathbf{x}_{(i)}, \mathbf{y}_{(i)}),$$

where $\mathbf{x}_{(i)}$, and $\mathbf{y}_{(i)}$ are each random training sample instant.

- It is must fast than the batch GD. It is possible to train on huge training sets, sinch only oine instance needs to be in memory at iteration.
- The algorithm is much less regular than batch GD: instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average.
- Over time it will end up very close to the minimum, but one it gets there it will continue to bounce around, never settling down.
- The SGD can jump out of local minima from the random manner, so SGD has a better chance of finding the global minimum than batch GD.

Mini-Batch Gradient Descent

We want to minimize an objective function (loss): $J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i;\theta), y_i)$ where θ is parameters (weights, bias, etc.), $f(x_i;\theta)$ is a prediction for input x_i, $\ell(\cdot,\cdot)$ is a loss function, and Data: $\{(x_i, y_i)\}_{i=1}^{N}$

```
Mini-Batch Gradient Descent
Require: Training data $\{(x_i, y_i)\}_{i=1}^{N}$, learning rate $\eta$, batch size $m$, number of
    epochs $T$, initial parameters $\theta$
Ensure: Optimized parameters $\theta$
 1: for epoch $\leftarrow 1$ to $T$ do
 2:     Shuffle training data
 3:     Partition data into batches of size $m$
 4:     for all batch $B$ do
 5:         Compute predictions $f(x;\theta)$ for $x \in B$
 6:         Compute loss $J_B(\theta) \leftarrow \frac{1}{m}\sum_{(x,y)\in B} \ell(f(x;\theta), y)$
 7:         Compute gradient $\nabla_\theta J_B(\theta)$
 8:         Update parameters: $\theta \leftarrow \theta - \eta \cdot \nabla_\theta J_B(\theta)$
 9:     end for
10: end for
11: return $\theta$
```

Stochastic Gradient Descent

```
Stochastic Gradient Descent
Require: Initial learning rate $\eta_0$, Decay rate $k$, Training data $D = \{(x_i, y_i)\}_{i=1}^{N}$,
    Epochs $T$, Initial parameters $\theta$
Ensure: Optimized parameters $\theta$
 1: for epoch $t \leftarrow 1$ to $T$ do
 2:                                   ▷ Update the learning rate for the current epoch
 3:     $\eta_t \leftarrow \eta_0 / (1 + k \cdot t)$
 4:     Shuffle the training data $D$ randomly.
 5:     for $i \leftarrow 1$ to $N$ do
 6:         Compute gradient for a single example: $g \leftarrow \nabla_\theta J(\theta; x_i, y_i)$
 7:         Update parameters: $\theta \leftarrow \theta - \eta_t \cdot g$ ▷ Use the newly calculated learning rate
 8:     end for
 9: end for
10: return $\theta$
```

Adaptive Moment Estimation (Adam)

Adam is a highly effective and widely used optimization algorithm that combines the best features of two other popular methods: **Momentum** and **RMSprop**.
The core of the Adam optimizer is its parameter update rule, where the parameters θ at time-step k are updated from the previous step $k-1$ according to the following equation

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \alpha \frac{\hat{\mathbf{m}}_k}{\sqrt{\hat{\mathbf{v}}_k} + \epsilon} \tag{1}$$

- The update is driven by the **learning rate,** α, which scales the overall step size.
- The numerator, $\hat{m}_t$ is the **bias-corrected first moment estimate** of the gradient. This term acts like **momentum**, accumulating an exponentially decaying average of past gradients to help accelerate in a consistent direction. It is calculated from the gradient $\nabla f(\mathbf{x}_k)$ and the decay rate β_1:

$$\mathbf{m}_{k+1} = \beta_1 \mathbf{m}_k + (1-\beta_1)\nabla f(\mathbf{x}_k), \qquad \hat{\mathbf{m}}_k = \frac{\beta_1 \mathbf{m}_{k-1} + (1-\beta_1)\nabla f(x_k)}{1-\beta_1^{k-1}}$$

Adaptive Moment Estimation (Adam)

- The denominator, $\sqrt{\hat{v}_k}$ provides **adaptive, per-parameter scaling**. It is the square root of the **bias-corrected second moment estimate**, which accumulates an exponentially decaying average of past squared gradient, effectively creating an individual learning rate for each parameter. It is calculated using the decay rate β_2:

$$\mathbf{v}_{k+1} = \beta_2 \mathbf{v}_k + (1-\beta_2)(\nabla f(\mathbf{x}_k))^2, \qquad \hat{v}_k = \frac{\beta_2 v_{k-1} + (1-\beta_2)(\nabla f(\mathbf{x}_k))^2}{1-\beta_2^{k-1}}$$

- Finally, ϵ is a very small constant added to the denominator to ensure **numerical stability** by preventing division by zero.

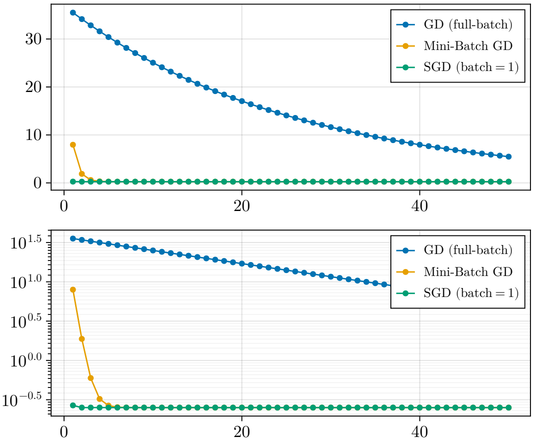
Note: $(\nabla f(\mathbf{x}_k))^2)$ is an element-wise square.

Adaptive Moment Estimation (Adam)

```
Adaptive Moment Estimation (Adam)
Require: Learning rate $\alpha$, decay rates $\beta_1, \beta_2$, stability constant $\epsilon$
Require: Initial parameters $\theta$, Objective function $J(\theta)$
 1: Initialize 1st moment vector: $m \leftarrow 0$, Initialize 2nd moment vector: $v \leftarrow 0$
 2: Initialize time-step: $k \leftarrow 0$
 3: while $\theta$ has not converged do
 4:     $k \leftarrow k + 1$
 5:     Compute gradient: $g \leftarrow \nabla_\theta J_k(\theta)$
 6:     $m \leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot g$
 7:     $v \leftarrow \beta_2 \cdot v + (1 - \beta_2) \cdot g^2$                ▷ $g^2$ is element-wise
 8:                                   ▷ Compute bias-corrected moment estimates
 9:     $\hat{m} \leftarrow m/(1 - \beta_1^t)$
10:     $\hat{v} \leftarrow v/(1 - \beta_2^t)$
11:                                                      ▷ Update parameters
12:     $\theta \leftarrow \theta - \alpha \cdot \hat{m}/(\sqrt{\hat{v}} + \epsilon)$       ▷ $\sqrt{\hat{v}}$ is element-wise
13: end while
14: return $\theta$
```

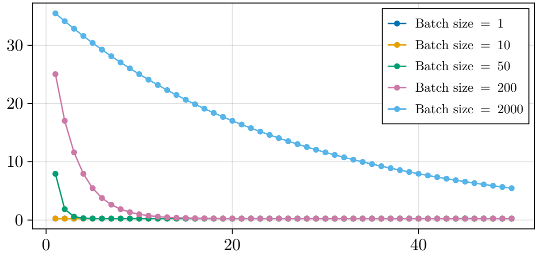
Example: Linear Regression

To see the performance of SGD, consider a linear model $y = \mathbf{A}\mathbf{x} + b$ with a two-dimensional state $\mathbf{x}$. The three methods under comparison; standard (full-batch) gradient descent, mini-batch stochastic gradient descent, and stochastic gradient descent with $n = 1$.



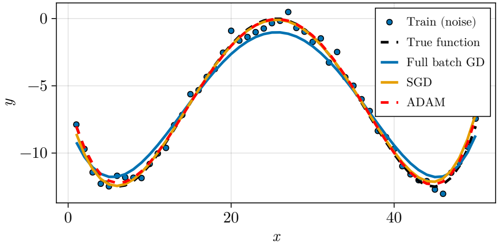
Example: Linear regression

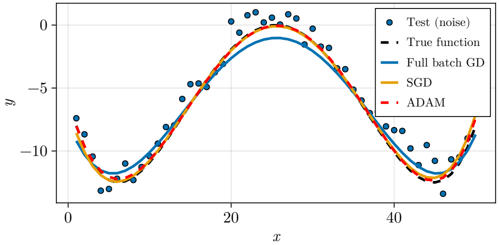
- The mini-batch and SGD algorithms converge in many fewer epochs than full-batch gradient descent.
- The epoch in each method implies a single pass through all data, so an epoch of SGD with batch size 1 involves 2000 single steps with a single data point (consider faster)
- SGD and full-batch can be considered as the extremes of mini-batch stochastic gradient descent. It appears that a mini-batch of 10 gives a balance of convergence and computation time.



Example: NonLinear regression

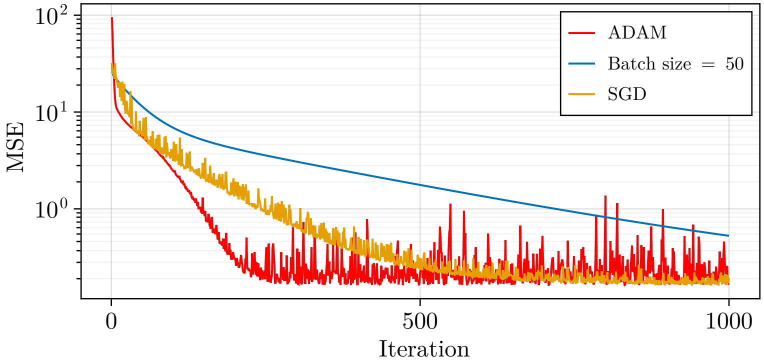
Now we apply SGD and Adam to a nonlinear regression problem of fitting a polynomial of high-order to noisy data. $y(x) = -10x^2 + 2x^4$. We will fit this polynomial in a space of 5 degree polynomial. (After tuning)





Example: NonLinear regression

The convergence of the methods vs. epoch, again with Adam achieving the lowest MSE. Adam outperforms full-batch gradient descent and SGD, as expected.



Reference

1. Joaquim R. R. A. Martins, and Andrew Ning, ”*Engineering Design Optimization*,” Cambridge University Press, 2021
2. Mykel J. Kochenderfer, and Tim A. Wheeler, ”*Algorithms for Optimization*,” The MIT Press, 2019