# Chapter 8: Optimization

## 8.1 Introduction

### 8.1.1 Local vs global optimization

### 8.1.2 Constrained vs unconstrained optimization

### 8.1.3 Convex vs nonconvex optimization

### 8.1.4 Smooth vs nonsmooth optimization

## 8.2 First-order methods

### 8.2.1 Descent direction

### 8.2.2 Step size (learning rate)

### 8.2.3 Convergence rates

### 8.2.4 Momentem methods

**Momentum**

$$\boldsymbol{m}_t = \beta \boldsymbol{m}_{t-1} + \boldsymbol{g}_{t-1} \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t \boldsymbol{m}_t$$

- $\boldsymbol{m}_t$ is the momentum (mass times velocity).
- $0 < \beta < 1$. A typical value of $\beta = 0.9$.

## 8.3 Second-order methods

### 8.3.1 Newton's method

### 8.3.2 BFGS and other quasi-Newton methods

### 8.3.3 Trust region methods

## 8.4 Stochastic gradient descent

Goal: minimize

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{z})}[\mathcal{L}(\boldsymbol{\theta}, \mathbf{z})]$$

In each iteration, we assume we observe $\mathcal{L}_t(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}, \mathbf{z}_t)$, where $\mathbf{z}_t \sim q$ (sample one subject $\mathbf{z}_t$ from $q(\mathbf{z})$). We further assume a way to compute the unbiased estimate of $\nabla_\theta \mathcal{L}$. If $q(\mathbf{z}) \perp\!\!\!\perp \boldsymbol{\theta}$, we can use $\boldsymbol{g}_t = \nabla_\theta \mathcal{L}_t(\boldsymbol{\theta}_t)$. Then, the stochastic gradient descent (SGD) can be written as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla \mathcal{L}(\boldsymbol{\theta}_t, \mathbf{z}_t) = \boldsymbol{\theta}_t - \eta_t \boldsymbol{g}_t.$$

As long as $\hat{\nabla} \mathcal{L}$ is unbiased and the step size $\eta_t$ is decaying at a certain rate, this method will converge to a stationary point.

### 8.4.1 Application to finite sum problems

https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a

- **Batch GD**: All the training data is taken into consideration to take a single step: when updating the parameters of interest, we take the average of gradients of all the training examples and use that average to update the parameters.
  - Advantage: unbiased estimation of the gradient.
  - Drawback: computational expensive and time costly for a large training set.
- **Minibatch GD**: In each iteration, we sample a minibatch of $B \ll N$ samples and take the average of gradients of these minibatch examples. Then use that average to update the parameters.

  - Advantage: Unbiased approximation to the full-batch average, and is computationally more sufficient than the batch GD.
- **SGD**: In each iteration, we sample one example from the training set and use the gradient of that example to update the parameters.
  - Advantage: in practice, SGD converges faster than full batch GD.

### 8.4.2 Example: SGD for fitting linear regression

An example of Section 8.4.1.

### 8.4.3 Choosing the step size (learning rate)

*Figure 8.17* shows: overly small learning rate results in underfitting, whereas overly large learning rate results leads to instability of the model.

**Constant learning rate** Start with a small learning rate and gradually increase it, and evaluate the model performance using a small number of minibatches. Then, draw a plot like *Figure 8.17* and pick the learning rate with the lowest loss (p.s. it is better to pick a rate slightly smaller than that to ensure stability).

**Learning rate schedule (adaptively adjust the step size over time)** **Robbins-Monro conditions**:

$$\eta_t \to 0, \ \frac{\sum_{t=1}^{\infty} \eta_t^2}{\sum_{t=1}^{\infty} \eta_t} \to 0.$$

Examples:

- `Piecewise constant`: $\eta_t = \eta_i$ if $t_i \leq t \leq t_{i+1}$
  - Step decay;
  - Reduce-on-plateau: reduce the learning rate when the train or validation loss has plateaued.
- `Exponential decay`: $\eta_t = \eta_0 e^{-\lambda t}$
  - Typically too fast.
- `Polynomial decay`: $\eta_t = \eta_0(\beta t + 1)^{-\alpha}$
  - A common choice;
  - Square-root schedule (with $\alpha = 0.5$ and $\beta = 1$): $\eta_t = \eta_0 \frac{1}{\sqrt{t+1}}$.
- `Learning rate warmup / one-cycle learning rate schedule`:
  - Applied to the deep learning;
  - Quickly increase the learning rate and then gradually decrease it again: a slow learning rate at the beginning helps to find a flatter region of the loss landscape, and, once reaches that region, a fast learning rate accelerates the convergence; to ensure convergence to a stationary point, the learning rate must reduced to 0.
- `Cyclical learning rate`:
  - Increase and decrease the learning rate multiple times in a cyclical fashion.
  - Advantage: escape local optima.
- `Stochastic gradient descent with warm restarts`:
  - Related to `cyclical learning rate`.

**Estimate the learning rate using line search** Caution: need to ensure whether the variance of the gradient noise goes to 0 over time. If not, the parameter estimation may not guarantee to converge.

### 8.4.4 Iterate averaging/ Polyak-Ruppert averaging

The parameter estimates produced by SGD can be unstable over time. To reduce the variance, we may replace $\boldsymbol{\theta}_t$ with

$$\bar{\boldsymbol{\theta}}_t = \frac{1}{t}\sum_{i=1}^{t}\boldsymbol{\theta}_i = \frac{1}{t}\boldsymbol{\theta}_t + \frac{t-1}{t}\bar{\boldsymbol{\theta}}_{t-1}.$$

in the $t$th step, where $\boldsymbol{\theta}_t$ are the usual SGD iterates.

- Statistical benefits: in the case of linear regression, this method is equivalent to $\ell_2$ regularization.
- Related approach: `stochastic weight averaging (SWA)`.

### 8.4.5 Variance reduction*

### 8.4.6 Preconditioned SGD

Consider the following update:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \mathbf{M}_t^{-1}\boldsymbol{g}_t.$$

- $\mathbf{M}_t$: preconditioning matrix (preconditioner), positive-definite;
- $\boldsymbol{g}_t$: gradient of $\mathcal{L}$ at $\boldsymbol{\theta}_t$
- Can be applied when the second-order information (e.g. Hessian matrix) is difficult to obtain or unreliable.
- Most practitioners use a diagonal preconditioner $\mathbf{M}_t$.
- Often result in speedups compared to vanilla SGD.

*Additional notation:* In the following expressions, index $d = 1 : D$ represents the *dimensions of the parameter vector*.

#### AdaGrad (adaptive gradient)

- Originally designed for optimizing convex objectives where many elements of the gradient vector are zero.
- A coordinate-wise update (because of the diagonal $\mathbf{M}_t$).

$$\theta_{t+1,d} = \theta_{t,d} - \eta_t \frac{1}{\sqrt{s_{t,d}+\epsilon}} g_{t,d}$$

where

$$s_{t,d} = \sum_{i=1}^{t} g_{i,d}^2$$

and $\epsilon > 0$ is to avoid dividing by 0.

- Equivalent expression:

$$\Delta\boldsymbol{\theta}_t = -\eta_t \frac{1}{\sqrt{\boldsymbol{s}_t+\epsilon}}\boldsymbol{g}_t$$

- $\mathbf{M}_t = \operatorname{diag}\left(s_{t,1}+\epsilon, \cdots, s_{t,D}+\epsilon\right)^{1/2}$
- The overall stepsize $\eta_t$ needs to be chosen, but the results are less sensitive to it.
  - Usually fix $\eta_t = \eta_0$.
- The term in the denominator gets larger over time.

– Ensure convergence;

– *Drawback*: If the denominator gets large too fast, it may hurt performance.

**RMSPROP & ADADELTA** To address the drawback of `ADAGRAD`.

`RMSProp`: let

$$s_{t,d} = \beta s_{t,d} + (1-\beta) g_{t,d}^2.$$

and plug this term in the previous update expression.
(exponentially weighted moving average, EWMA)

- Usually we use $\beta = 0.9$ and then (RMS (root mean squared)),

$$\sqrt{s_{t,d}} \approx \text{RMS}(\boldsymbol{g}_{1:t,d}) = \sqrt{\frac{1}{t}\sum_{\tau=1}^{t} g_{\tau,d}^2}$$

`AdaDelta`:

- Also keep an EWMA of the updates $\boldsymbol{\delta}_t$

$$\Delta\boldsymbol{\theta}_t = -\eta_t \frac{\sqrt{\boldsymbol{\delta}_{t-1} + \epsilon}}{\sqrt{\boldsymbol{s}_t + \epsilon}} \boldsymbol{g}_t,$$

where

$$\boldsymbol{\delta}_t = \beta \boldsymbol{\delta}_{t-1} + (1-\beta)(\Delta\boldsymbol{\theta}_t)^2$$

and $\boldsymbol{s}_t$ is the same as in `RMSProp`.

- *Advantage*: the "units" of the numerator and denominator cancel, so we are just elementwise-multiplying the gradient by a scalar.

- *Drawback*: Common to fix $\eta_t = 1$. But, if we don't explicitly force the adaptive learning rate $\eta_t$ to decrease with time, these two methods are not guaranteed to converge to a solution.

**ADAM (Adaptive moment estimation)**

- Combine `RMSProp` with momentum: let

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t \boldsymbol{s}_t = \beta_2 \boldsymbol{s}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$$

  and the perform

$$\Delta\boldsymbol{\theta}_t = -\eta_t \frac{1}{\sqrt{\boldsymbol{s}_t} + \epsilon} \boldsymbol{m}_t$$

- Standard values: $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-6}$.
- Common to fix $\eta_t = 0.001$.
  – The adaptive learning rate may not decrease over time, convergence is not guaranteed.
- If initialize $\boldsymbol{m}_0 = \boldsymbol{s}_0 = 0$, then initial estimates will be biased towards small values.
  – The authors recommend using the bias-corrected moments:

$$\hat{\boldsymbol{m}}_t = \boldsymbol{m}_t / (1-\beta_1^t) \hat{\boldsymbol{s}}_t = \boldsymbol{s}_t / (1-\beta_2^t)$$

**Issues with adaptive learning rates**

**Non-diagonal preconditioning matrices** `full-matrix Adagrad`

$$\mathbf{M}_t = \left[(\mathbf{G}_t\mathbf{G}_t^\top)^{\frac{1}{2}} + \epsilon\mathbf{I}_D\right]^{-1}$$

where

$$\mathbf{G}_t = [\boldsymbol{g}_t, \cdots, \boldsymbol{g}_1], \boldsymbol{g}_i = \nabla_\psi c(\psi_i).$$

## 8.5 Constrained optimization

Consider the following problem

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\mathcal{C}} \mathcal{L}(\boldsymbol{\theta})$$

where $\mathcal{C}$ is the feasible (constraint) set

$$\mathcal{C} = \left\{\boldsymbol{\theta} \in \mathbb{R}^D : h_i(\boldsymbol{\theta}) = 0, i \in \mathcal{E};\ g_j(\boldsymbol{\theta}) \leq 0, j \in \mathcal{I}\right\}$$

where $\mathcal{E}$ is the set of equality constraints, and $\mathcal{I}$ is the set of inequality constraints.

### 8.5.1 Lagrange multipliers.

- Deal with equality constraints $h_i(\boldsymbol{\theta}) = 0, i \in \mathcal{E}$.
- In other words, we seek a point $\boldsymbol{\theta}^*$ on the constraint surface $h(\boldsymbol{\theta}) = 0$ that minimizes $\mathcal{L}(\boldsymbol{\theta})$.
- ....

$$L(\theta, \lambda) = \mathcal{L}(\theta) + \sum_{j=1}^{m} \lambda_j h_j(\theta)$$

Then, let

$$\nabla_{\theta,\lambda} L(\theta, \lambda) = 0$$
$$\Leftrightarrow \lambda\nabla_\theta h(\theta) = \nabla\mathcal{L}(\theta), h(\theta) = 0.$$

and solve these equations.

- $\lambda$ is called a Lagrange multiplier.
- Now we transfer the $D$-dim constrained optimization problem to a $D+m$-dim unconstrained optimization problem.

### 8.5.2 The KKT conditions

- Deal with inequality constraint $g(\theta) \leq 0$.

### 8.5.3 Linear programming

### 8.5.4 Quadratic programming

### 8.5.5 Mixed integer linear programming*

## 8.6 Proximal gradient method*

## 8.7 Bound optimization*