Nonlinear Optimization in Machine Learning

A Series of Lecture Notes at Missouri S&T

Wenqing Hu [*]

Contents

1 Background on Machine Learning: Why Nonlinear Optimization? 1

1.1 Empirical Risk Minimization 1

1.2 Loss Functions 2

1.3 Learning Models 2

2 Basic Information about Convex Functions 6

2.1 Optimization problem and its solutions 6

2.2 Convexity 6

2.3 Taylor's theorem and convexity in Taylor's expansion 7

2.4 Optimality Conditions 9

3 Gradient Descent and Line Search Methods 11

3.1 Gradient Descent 11

3.2 Back Propagation and GD on Neural Networks 13

3.3 Online Principle Component Analysis (PCA) 14

3.4 Line Search Methods 15

3.5 Convergence to Approximate Second Order Necessary Points 16

4 Accelerated Gradient Methods 18

4.1 Polyak's heavy ball method 18

4.2 Nesterov's Accelerated Gradient Descent (AGD) 19

4.3 Conjugate gradient method 23

[*]Department of Mathematics and Statistics, Missouri University of Science and Technology (formerly University of Missouri, Rolla). Web: http://web.mst.edu/~huwen/ Email: huwen@mst.edu

5 Stochastic Gradient Methods **24**
5.1 SGD 24
5.2 Variance–reduction methods: SVRG, SAG, SAGA, SARAH, SPIDER, STORM 26
5.3 Escape from saddle points and nonconvex problems, SGLD, Perturbed GD 32
5.4 Coordinate Descent and Stochastic Coordinate Descent 33

6 Second Order Methods **35**
6.1 Newton's method 35
6.2 Quasi–Newton: BFGS 36
6.3 Stochastic Quasi–Newton 38

7 Dual Methods **41**
7.1 Dual Coordinate Descent/Ascent 41
7.2 SDCA 42

8 Optimization with Constraints **44**
8.1 Subgradient Projection Method, Frank–Wolfe Method 44
8.2 ADMM 45

9 Adaptive Methods **47**
9.1 SGD with momentum 47
9.2 AdaGrad, RMSProp, AdaDelta 47
9.3 Adam 49

10 Optimization with Discontinuous Objective Function **50**
10.1 Proximal operators and proximal algorithms 50

11 Optimization and Generalization **51**
11.1 Generalization Bounds in Statistical Machine Learning 51
11.2 Training Matters: Connecting Optimization with Generalization 52
11.3 Experiments: Real Data Sets 53

A Big-O and Little-O notations **55**

1 Background on Machine Learning: Why Nonlinear Optimization?

1.1 Empirical Risk Minimization

Supervised Learning: Given training data points $(x_1, y_1), ..., (x_n, y_n)$, construct a learning model $y = g(x, \omega)$ that best fits the training data. Here ω stands for the parameters of the learning model.

Here (x_i, y_i) comes from an independent identically distributed family $(x_i, y_i) \sim p(x, y)$, where $p(x, y)$ is the joint density. The model $x \to y$ is a black-box $p(y|x)$, which is to be fit by $g(x, \omega)$.

"Loss function" $L(g(x, \omega), y)$, for example, can be $L(g(x, \omega), y) = (g(x, \omega) - y)^2$.

Empirical Risk Minimization (ERM):

$$\omega_n^* = \arg\min_\omega \frac{1}{n} \sum_{i=1}^n L(y_i, g(x_i, \omega)) \ . \tag{1.1}$$

Regularized Empirical Risk Minimization (R-ERM):

$$\omega_n^* = \arg\min_\omega \frac{1}{n} \sum_{i=1}^n L(y_i, g(x_i, \omega)) + \lambda R(\omega) \ . \tag{1.2}$$

In general let $f_i(\omega) = L(y_i, g(x_i, \omega))$ or $f_i(\omega) = L(y_i, g(x_i, \omega)) + \lambda R(\omega)$, then the optimization problem is

$$\omega_n^* = \arg\min_\omega \frac{1}{n} \sum_{i=1}^n f_i(\omega) \ . \tag{1.3}$$

Key features of nonlinear optimization problem in machine learning: large–scale, nonconvex, ... etc.

Key problems in machine learning: optimization combined with generalization. "Population Loss": $\mathbf{E}L(g(x, \omega), y)$, minimizer

$$\omega^* = \arg\min_\omega \mathbf{E}L(g(x, \omega), y) \ .$$

Generalization Error: $\mathbf{E}L(y, g(x, \omega_n^*))$. Consistency: Do we have $\omega_n^* \to \omega^*$? At what speed?

Key problems in optimization: convergence, acceleration, variance reduction.

How can optimization be related to generalization? Vapnik–Chervonenkis dimension. Radmacher complexity, geometry of the loss landscape.

1.2 Loss Functions

Classification Problems: Choice of Loss function $L(y, g)$, $y = 1, -1$. 0/1 Loss: $\ell_{0/1}(y, g) = 1$ if $yg < 0$ and $\ell_{0/1}(y, g) = 0$ otherwise.

(1) Hinge Loss.

$$L(y, g) = \max(0, 1 - yg) \ ; \tag{1.4}$$

(2) Exponential Loss.

$$L(y, g) = \exp(-yg) \ ; \tag{1.5}$$

(3) Cross Entropy Loss.

$$L(y, g) = -\left(I_{\{y=1\}} \ln \frac{e^g}{e^g + e^{-g}} + I_{\{y=-1\}} \ln \frac{e^{-g}}{e^g + e^{-g}} \right) \ . \tag{1.6}$$

Regression Problems: Choice of Loss function $L(y, g)$.

(1) L^2–Loss.

$$L(y, g) = |y - g|_2^2 \ . \tag{1.7}$$

(2) L^1–Loss.

$$L(y, g) = |y - g|_1 \ . \tag{1.8}$$

(3) L^0–Loss.

$$L(y, g) = |y - g|_0 \ . \tag{1.9}$$

Regularized (penalize) term $R(\omega)$: L^1–regularized $R(\omega) = |\omega|_1$; L^0–regularized $R(\omega) = |\omega|_0$.

1.3 Learning Models

(1) Linear regression: $g(x, \omega) = \omega^T x$. $g(x, \omega) = \dfrac{1}{1 + \exp(-\omega^T x)}$.

Least squares problem:

$$\min_\omega \frac{1}{2m} \sum_{j=1}^m (x_j^T \omega - y_j)^2 = \frac{1}{2m} |A\omega - y|_2^2 \tag{1.10}$$

Here $A = (x_1, ..., x_m)$.

Tikhonov regularization:

$$\min_\omega \frac{1}{2m} |A\omega - y|_2^2 + \lambda |\omega|_2^2 \ . \tag{1.11}$$

LASSO (Least Absolute Shrinkage and Selection Operator):

$$\min_\omega \frac{1}{2m} |A\omega - y|_2^2 + \lambda |\omega|_1 \ . \tag{1.12}$$

(2) Support Vector Machines (SVM): $x_j \in \mathbb{R}^n$, $y_j \in \{1, -1\}$. Goal is to seek $\omega \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$ such that $\omega^T x_j + \beta \geq 1$ if $y_j = 1$ and $\omega^T x_j + \beta \leq -1$ if $y_j = -1$. Hinge Loss $L(\omega, \beta) = \dfrac{1}{m} \sum\limits_{j=1}^{m} \max(0, 1 - y_j(x_j^T \omega - \beta))$.

Classification Problem: Goal is to find a hyperplane $\omega^T x + \beta = 0$ such that it classifies the two kinds of data points most efficiently. The signed distance of any point $x \in \mathbb{R}^n$ to the hyperplane is given by $r = \dfrac{\omega^T x + \beta}{|\omega|}$. If the classification is good enough, we expect to have $\omega^T x_j + \beta > 0$ when $y = 1$ and $\omega^T x_j + \beta < 0$ when $y = -1$. We can then formulate the problem as looking for optimal ω and β such that $\omega^T x_j + \beta \geq 1$ when $y_j = 1$ and $\omega^T x_j + \beta \leq -1$ when $y_i = -1$. The closest few data points that match these two inequality are called "support vectors". The distance to the separating hyperplane created by two support vectors of opposite type is $\dfrac{2}{|\omega|}$. So we have the constrained optimization problem

$$\max_{\omega \in \mathbb{R}^n, \beta \in \mathbb{R}} \frac{2}{|\omega|} \text{ such that } y_j(\omega^T x_j + \beta) \geq 1 \text{ for } j = 1, 2, ..., m \ .$$

Or in other words we have the constrained optimization problem

$$\min_{\omega \in \mathbb{R}^n, \beta \in \mathbb{R}} \frac{1}{2}|\omega|^2 \text{ such that } y_j(\omega^T x_j + \beta) \geq 1 \text{ for } j = 1, 2, ..., m \ . \tag{1.13}$$

"Soft margin": We allow the SVM to make errors on some training data points but we want to minimize the error. In fact, we allow some training data to violate $y_j(\omega^T x_j + \beta) \geq 1$, so that ideally we minimize

$$\min_{\omega \in \mathbb{R}^n, \beta \in \mathbb{R}} \frac{1}{2}|\omega|^2 + C \sum_{j=1}^{m} \ell_{0/1}(y_j(\omega^T x_j + \beta) - 1) \ .$$

Here the 0/1 loss is $\ell_{0/1}(z) = 1$ if $z < 0$ and $\ell_{0/1}(z) = 0$ otherwise. We can then turn the 0/1 loss to hinge loss, that is why hinge loss comes in:

$$\min_{\omega \in \mathbb{R}^n, \beta \in \mathbb{R}} \frac{1}{2}|\omega|^2 + C \sum_{j=1}^{m} \max(0, 1 - y_j(\omega^T x_j + \beta)) \ . \tag{1.14}$$

"slack variables" $\xi_i \geq 0$, "Soft margin SVM"

$$\min_{\omega \in \mathbb{R}^n, \beta \in \mathbb{R}} \frac{1}{2}|\omega|^2 + C \sum_{j=1}^{m} \xi_j \text{ s.t. } y_j(\omega^T x_j + b) \geq 1 - \xi_j \ , \xi_j \geq 0 \ , j = 1, 2, ..., m \ . \tag{1.15}$$

(3) Neural Network: "activation function" σ.

Sigmoid:

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \ ; \tag{1.16}$$

tanh:

$$\sigma(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \ ; \tag{1.17}$$

ReLU (Rectified Linear Unit):

$$\sigma(z) = \max(0, z) \ . \tag{1.18}$$

Vector–valued activation: if $z \in \mathbb{R}^n$ then $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ is defined by $(\sigma(z))_i = \sigma(z_i)$ where each $\sigma(z_i)$ is the scalar activation function.

Fully connected neural network prediction function

$$g(x, \omega) = a^T \left(\sigma \left(W^{(H)}(\sigma(W^{(H-1)}(...(\sigma(W^{(1)}x + b_1))...) + b_{H-1}) + b_H \right) \right) \ . \tag{1.19}$$

Optimization

$$\min_\omega \frac{1}{2} \sum_{i=1}^n (g(x_i, \omega) - y_i)^2 \ .$$

Many other different network structures: convolutional, recurrent (Gate: GRU, LSTM), ResNet, ...

Two layer (one hidden layer) fully connected ReLU neural network has specific loss function structure: our $W^{(1)} = \begin{pmatrix} \omega_1^T \\ ... \\ \omega_m^T \end{pmatrix}$ and

$$g(x, \omega) = \sum_{r=1}^m a_r \max \left(\omega_r^T x, 0 \right) \ . \tag{1.20}$$

Optimization problem is given by

$$\min_\omega \frac{1}{2} \sum_{i=1}^n \left(\sum_{r=1}^m a_r \max \left(\omega_r^T x_i, 0 \right) - y_i \right)^2 \ .$$

Non–convexity issues: see

Alon Brutzkus, Amir Globerson, Eran Malach, Shai Shalev-Shwartz, SGD learns over–parameterized networks that provably generalize on linearly separable data, arXiv:1710.10174, *ICLR*, 2018.

Also see problem 1 in Assignment 1.

(4) Matrix Completion: Each A_j is $n \times p$ matrix, and we seek for another $n \times p$ matrix such that

$$\min_X \frac{1}{2m} \sum_{j=1}^m (\langle A_j, X \rangle - y_j)^2 \tag{1.21}$$

where $\langle A, B \rangle = \text{tr}(A^T B)$. We can think of the A_j as "probing" the unknown matrix X.

(5) Nonnegative Matrix Factorization: If the full matrix $Y \in \mathbb{R}^{n\times p}$ is observed, then we seek for $L \in \mathbb{R}^{n\times r}$ and $R \in \mathbb{R}^{p\times r}$ such that

$$\min_{L,R} \|LR^T - Y\|_F^2 \text{ subject to } L \geq 0 \text{ and } R \geq 0 \ . \tag{1.22}$$

Here $\|A\|_F = \left(\sum\sum |a_{ij}|^2\right)^{1/2}$ is the Frobenius norm of a matrix A.

See the following review paper

Chi, Y., Lu, Y.M., Chen, Y., Nonconvex Optimization Meets Low-Rank Matrix Factorization: An Overview. *IEEE Transactions on Signal Processing*, Vol. **67**, No. 20, October 15, 2019.

(6) Principle Component Analysis (PCA): PCA:

$$\max_{v\in\mathbb{R}^n} v^T S v \text{ such that } \|v\|_2 = 1 \ , \ \|v\|_0 \leq k \ . \tag{1.23}$$

The objective function is convex, but if you take into account the constraint, then this problem becomes non–convex. See dimension 1 example.

Online PCA: see

C.J Li, M Wang, H Liu, T Zhang. Near–Optimal Stochastic Approximation for Online Principal Component Estimation. *Mathematical Programming*, **167**(1):75–97, 2018

(7) Sparse inverse covariance matrix estimation: Sample covariance matrix $S = \dfrac{1}{m-1}\sum\limits_{j=1}^{m} a_j a_j^T$. $S^{-1} = X$. “Graphical LASSO”:

$$\min_{X\in \text{ Symmetric } \mathbb{R}^{n\times n}, X\succeq 0} \langle S, X\rangle - \ln\det X + \lambda\|X\|_1 \tag{1.24}$$

where $\|X\|_1 = \sum |X_{ij}|$.

2 Basic Information about Convex Functions

2.1 Optimization problem and its solutions

Notion of Minimizers: $f : \mathbb{R}^n \supset \mathcal{D} \to \mathbb{R}$: Local minimizer, global minimizer, strict local minimizer, isolated local minimizer.

Constrained optimization problem

$$\min_{x \in \Omega} f(x) \ , \tag{2.1}$$

where $\Omega \subset \mathcal{D} \subset \mathbb{R}^n$ is a closed set.

Local Solution. Global Solution.

2.2 Convexity

Convex Set: $x, y \in \Omega \Longrightarrow (1-\alpha)x + \alpha y \in \Omega$ for all $\alpha \in [0,1]$.

Supporting hyperplane.

Given convex set $\Omega \subset \mathbb{R}^n$, the projection operator $P : \mathbb{R}^n \to \Omega$ is given by

$$P(y) = \arg\min_{z \in \Omega} \|z - y\|_2^2 \ . \tag{2.2}$$

$P(y)$ is the point in Ω that is closest to y in the sense of Euclidean norm.

Convex function: $\phi : \mathbb{R}^n \to \mathbb{R} \cup \{\pm\infty\}$ such that

$$\phi((1-\alpha)x + \alpha y) \leq (1-\alpha)\phi(x) + \alpha\phi(y) \tag{2.3}$$

for all $x, y \in \mathbb{R}^n$ and all $\alpha \in [0,1]$.

Effective domain, epigraph, proper convex function, closed proper convex function.

Strongly convex function with modulus of convexity m: a convex function ϕ and there exists some $m > 0$ such that

$$\phi((1-\alpha)x + \alpha y) \leq (1-\alpha)\phi(x) + \alpha\phi(y) - \frac{1}{2}m\alpha(1-\alpha)\|x-y\|_2^2 \tag{2.4}$$

for all x, y in the domain of ϕ.

Indicator function: If $\Omega \subset \mathbb{R}^n$ is convex we define the indicator function

$$I_\Omega(x) = \begin{cases} 0 & \text{if } x \in \Omega \ , \\ +\infty & \text{otherwise} \ . \end{cases} \tag{2.5}$$

Constrained optimization problem $\min\limits_{x \in \Omega} f(x)$ is the same thing as the unconstrained optimization problem $\min[f(x) + I_\Omega(x)]$.

Theorem 2.1 (Minimizers for convex functions)**.** *If the function f is convex and the set Ω is closed and convex, then*

(a) Any local solution of (2.1) *is also global;*

(b) The set of global solutions of (2.1) *is a convex set.*

2.3 Taylor's theorem and convexity in Taylor's expansion

Theorem 2.2 (Taylor's theorem)**.** *Given a continuously differentiable function* $f : \mathbb{R}^n \to \mathbb{R}$ *and given* $x, p \in \mathbb{R}^n$ *we have*

$$f(x+p) = f(x) + \int_0^1 \nabla f(x+\gamma p)^T p d\gamma \ , \tag{2.6}$$

$$f(x+p) = f(x) + \nabla f(x+\gamma p)^T p \ , \quad \text{for some } \gamma \in (0,1) \ . \tag{2.7}$$

If f *is twice continuously differentiable, then*

$$\nabla f(x+p) = \nabla f(x) + \int_0^1 \nabla^2 f(x+\gamma p)^T p d\gamma \ , \tag{2.8}$$

$$f(x+p) = f(x) + \nabla f(x+\gamma p)^T p + \frac{1}{2} p^T \nabla^2 f(x+\gamma p) p \ , \quad \text{for some } \gamma \in (0,1) \ . \tag{2.9}$$

L–Lipschitz: If $f : \mathbb{R}^n \to \mathbb{R}$ is such that

$$|f(x) - f(y)| \le L\|x-y\| \tag{2.10}$$

for $L > 0$ and any $x, y \in \mathbb{R}^n$, then $f(x)$ is L–Lipschitz.

Optimization literatures often assume that ∇f is L–Lipschitz

$$\|\nabla f(x) - \nabla f(y)\| \le L\|x-y\| \ . \tag{2.11}$$

Theorem 2.3. *(1) If* f *is continuously differentiable and convex then*

$$f(y) \ge f(x) + (\nabla f(x))^T (y-x) \tag{2.12}$$

for any $x, y \in dom(f)$.

(2) If f *is differentiable and* m*–strongly convex then*

$$f(y) \ge f(x) + (\nabla f(x))^T (y-x) + \frac{m}{2}\|y-x\|^2 \tag{2.13}$$

for any $x, y \in dom(f)$.

(3) If ∇f *is uniformly Lipschitz continuous with Lipschitz constant* $L > 0$ *and* f *is convex then*

$$f(y) \le f(x) + (\nabla f(x))^T (y-x) + \frac{L}{2}\|y-x\|^2 \tag{2.14}$$

for any $x, y \in dom(f)$.

Theorem 2.4. *Suppose that the function* f *is twice continuously differentiable on* $\mathbb{R}^n$*. Then*

(1) f *is strongly convex with modulus of convexity* m *if and only if* $\nabla^2 f(x) \succeq mI$ *for all* x*;*

(2) ∇f *is Lipschitz continuous with Lipschitz constant* L *if and only if* $\nabla^2 f(x) \preceq LI$ *for all* x.

Proof. (Part (1)) Suppose that the function f is strongly–m convex with the standard inequality (2.4) above for m–convexity. Set $z_\alpha = (1-\alpha)x + \alpha y$. Then by (2.4) we have

$$\alpha f(y) - \alpha f(x) \geq f(z_\alpha) - f(x) + \frac{1}{2} m\alpha(1-\alpha)\|x-y\|^2 \ .$$

Making use of Taylor's expansion

$$\alpha f(y) - \alpha f(x) \geq (\nabla f(x))^T (z_\alpha - x) + O(\|z_\alpha - x\|^2) + \frac{1}{2} m\alpha(1-\alpha)\|x-y\|^2 \ .$$

Since $z_\alpha \to x$ as $\alpha \to 0$, and $z_\alpha - x = \alpha(y-x)$, we can set $\alpha \to 0$ to get from above that for any $x, y \in \mathbb{R}^n$

$$f(y) \geq f(x) + (\nabla f(x))^T (y-x) + \frac{m}{2}\|y-x\|^2 \ . \tag{2.15}$$

Set $u \in \mathbb{R}^n$ and $\alpha > 0$, then we consider the Taylor's expansion

$$f(x+\alpha u) = f(x) + \alpha \nabla f(x)^T u + \frac{1}{2}\alpha^2 u^T \nabla^2 f(x + t\alpha u) u \tag{2.16}$$

for some $0 \leq t \leq 1$. We apply (2.15) with $y = x + \alpha u$ so that

$$f(x+\alpha u) \geq f(x) + \alpha (\nabla f(x))^T u + \frac{m}{2}\alpha^2 \|u\|^2 \ . \tag{2.17}$$

Comparing (2.16) and (2.17) we see that for arbitrary choice of $u \in \mathbb{R}^n$ we have

$$u^T \nabla^2 f(x + t\alpha u) u \geq m\|u\|^2 \ .$$

This implies that $\nabla^2 f(x) \succeq mI$ as claimed. This shows the "only if" part.

Indeed one can also show the "if" part. To this end assume that $\nabla^2 f(x) \succeq mI$. Then for any $z \in \mathbb{R}^n$ we have that $(x-z)^T \nabla^2 f(z + t(x-z))(x-z) \geq m\|x-z\|^2$. Thus

$$\begin{aligned} f(x) &= f(z) + (\nabla f(z))^T (x-z) + \frac{1}{2}(x-z)^T \nabla^2 f(z + t(x-z))(x-z) \\ &\geq f(z) + (\nabla f(z))^T (x-z) + \frac{m}{2}\|x-z\|^2 \ . \end{aligned} \tag{2.18}$$

Similarly

$$\begin{aligned} f(y) &= f(z) + (\nabla f(z))^T (y-z) + \frac{1}{2}(y-z)^T \nabla^2 f(z + t(y-z))(y-z) \\ &\geq f(z) + (\nabla f(z))^T (y-z) + \frac{m}{2}\|y-z\|^2 \ . \end{aligned} \tag{2.19}$$

We consider $(1-\alpha)$(2.18)$+\alpha$(2.19) and we set $z = (1-\alpha)x + \alpha y$. This gives

$$\begin{aligned} &(1-\alpha)f(x) + \alpha f(y) \\ &\geq (\alpha + (1-\alpha))f(z) + (\nabla f(z))^T ((1-\alpha)(x-z) + \alpha(y-z)) + \frac{m}{2}\left((1-\alpha)\|x-z\|^2 + \alpha\|y-z\|^2\right) \\ &= f(z) + (\nabla f(z))^T ((1-\alpha)(x-z) + \alpha(y-z)) + \frac{m}{2}\left((1-\alpha)\|x-z\|^2 + \alpha\|y-z\|^2\right) \ . \end{aligned} \tag{2.20}$$

Since $x-z = \alpha(x-y)$ and $y-z = (1-\alpha)(y-x)$, we see that $((1-\alpha)(x-z)+\alpha(y-z)) = 0$. Moreover, this means that

$$(1-\alpha)\|x-z\|^2 + \alpha\|y-z\|^2 = \left[(1-\alpha)\alpha^2 + \alpha(1-\alpha)^2\right]\|x-y\|^2 = \alpha(1-\alpha)\|x-y\|^2 \ .$$

From these we see that (2.20) is the same as saying

$$(1-\alpha)f(x) + \alpha f(y) \geq f((1-\alpha)x + \alpha y) + \frac{m}{2}\alpha(1-\alpha)\|x-y\|^2 \ ,$$

which is (2.4). □

Theorem 2.5 (Strongly convex functions have unique minimizers)**.** *Let f be differentiable and strongly convex with modulus $m > 0$. Then the minimizer x^* of f exists and is unique.*

More technical issues...

Theorem 2.6. *Let f be convex and continuously differentiable and ∇f be L–Lipschitz. Then for any $x, y \in dom(f)$ the following bounds hold:*

$$f(x) + \nabla f(x)^T(y-x) + \frac{1}{2L}\|\nabla f(x) - \nabla f(y)\|^2 \leq f(y) \ , \tag{2.21}$$

$$\frac{1}{L}\|\nabla f(x) - \nabla f(y)\|^2 \leq (\nabla f(x) - \nabla f(y))^T(x-y) \leq L\|x-y\|^2 \ . \tag{2.22}$$

In addition, let f be strongly convex with modulus m and unique minimizer x^*. Then for any $x, y \in dom(f)$ we have that

$$f(y) - f(x) \geq -\frac{1}{2m}\|\nabla f(x)\|^2 \ . \tag{2.23}$$

Generalized Strong Convexity: $(f^* = f(x^*))$

$$\|\nabla f(x)\|^2 \geq 2m[f(x) - f^*] \text{ for some } m > 0 \ . \tag{2.24}$$

It holds in situations other than when f is strongly convex.

2.4 Optimality Conditions

Smooth unconstrained problem

$$\min_{x\in\mathbb{R}^n} f(x) \ , \tag{2.25}$$

where $f(x)$ is a smooth function.

Theorem 2.7 (Necessary conditions for smooth unconstrained optimization)**.** *We have*

(a) *(first–order necessary condition) Suppose that f is continuously differentiable. Then if x^* is a local minimizer of* (2.25)*, then $\nabla f(x^*) = 0$;*

(b) *(second–order necessary condition) Suppose that f is twice continuously differentiable. Then if x^* is a local minimizer of* (2.25)*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite.*

Theorem 2.8 (Sufficient conditions for convex functions in smooth unconstrained optimization)**.** *If f is continuously differentiable and convex, then if $\nabla f(x^*) = 0$, then x^* is a global minimizer of* (2.25)*. In addition, if f is strongly convex, then x^* is the unique global minimizer.*

Theorem 2.9 (Sufficient conditions for nonconvex functions in smooth unconstrained optimization)**.** *Suppose that f is twice continuously differentiable and that for some x^* we have $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite, then x^* is a strict local minimizer of* (2.25)*.*

3 Gradient Descent and Line Search Methods

3.1 Gradient Descent

Want to solve $\min\limits_{x\in\mathbb{R}^n} f(x)$.

Want to construct an approximating sequence $\{x^k\}$ such that $f(x^{k+1}) < f(x^k)$, $k = 0, 1, 2, ...$

"descent direction" $d \in \mathbb{R}^n$ such that

$$f(x+td) < f(x) \text{ for all } t > 0 \text{ sufficiently small.}$$

Say f is continuously differentiable

$$f(x+td) = f(x) + t\nabla f(x+\gamma td)^T d \text{ for some } \gamma \in (0,1) \ .$$

If $d^T\nabla f(x) < 0$, then d is a descent direction.

"Steepest Descent": $\inf\limits_{\|d\|=1} d^T\nabla f(x) = -\|\nabla f(x)\|$.

The inf is achieved when $d = -\dfrac{\nabla f(x)}{\|\nabla f(x)\|}$. This is called the Gradient Descent Method.

Algorithm 3.1 Gradient Descent

```
1: Input: Input initialization $x^0$, stepsize $\alpha_k > 0$, $k = 0, 1, 2, ...$
2: for $k = 0, 1, 2, ..., K-1$ do
3:   Calculate $\nabla f(x^k)$
4:   $x^{k+1} = x^k - \alpha_k \nabla f(x^k)$
5: end for
6: Output: Output $x^K$
```

Convergence of optimization algorithms:

$$\|x_T - x^*\| \le \varepsilon(T) \text{ or } \mathbf{E}\|x_T - x^*\|^2 \le \varepsilon(T) \text{ or } \mathbf{E}f(x_T) - f(x^*) \le \varepsilon(T) \ .$$

$\varepsilon(T) \to 0$ as $T \to \infty$: convergence.

Convergence Rates are measured by $\ln \varepsilon(T)$

(1) If $\ln \varepsilon(T) = O(-T)$, then we say the algorithm has linear convergence rate;

(2) If $\ln \varepsilon(T)$ decays slower than $O(-T)$, then we say the algorithm has sub–linear convergence rate;

(3) If $\ln \varepsilon(T)$ decays faster than $O(-T)$, then we say the algorithm has super–linear convergence rate. In particular, if $\ln \ln \varepsilon(T) = O(-T)$, then we say the algorithm has second–order convergence rate.

Theorem 3.1. *Assume that f is twice continuously differentiable, m–strongly convex and ∇f is L–Lipschitz, so that $mI \preceq \nabla^2 f(x) \preceq LI$ for all $x \in \mathbb{R}^n$. Then the GD Algorithm 3.1 with constant stepsize $\alpha_k = \alpha$ and $\alpha \in \left[\frac{1-\beta}{m}, \frac{1+\beta}{L}\right]$ for some $0 \le \beta < 1$ has linear convergence rate.*

Proof. Use fixed point iteration. Let $\Phi(x) = x - \alpha \nabla f(x)$, $\alpha > 0$. Then $x^{k+1} = \Phi(x^k)$. Calculate $\|\Phi(x) - \Phi(z)\| \le \beta \|x - z\|$.

In order $\|x^T - x^*\| \le \varepsilon$ we want $T \ge \dfrac{\ln\left(\dfrac{\|x^0 - x^*\|}{\varepsilon}\right)}{|\ln \beta|}$.

Want $-\beta \le 1 - \alpha L \le 1 - \alpha m \le \beta$, and when $1 - \alpha L = 1 - \alpha m$, we have $\beta = \dfrac{L-m}{L+m}$ and $\alpha = \dfrac{2}{L+m}$. □

Standard procedure in optimization theory: make use of Taylor's expansion. Consider

$$\begin{aligned} f(x + \alpha d) &= f(x) + \alpha \nabla f(x)^T d + \alpha \int_0^1 [\nabla f(x + \gamma \alpha d) - \nabla f(x)]^T d \, d\gamma \\ &\le f(x) + \alpha \nabla f(x)^T d + \alpha \int_0^1 \|\nabla f(x + \gamma \alpha d) - \nabla f(x)\| \|d\| d\gamma \\ &\le f(x) + \alpha \nabla f(x)^T d + \alpha^2 \frac{L}{2} \|d\|^2 \ . \end{aligned} \tag{3.1}$$

From (3.1) and the line search $d = -\nabla f(x^k)$, $x = x^k$ on RHS of (3.1), then the $\alpha = \dfrac{1}{L}$ minimizes RHS of (3.1), and thus for GD in particular

$$f(x^{k+1}) \le f(x^k) + \left(-\alpha + \alpha^2 \frac{L}{2}\right) \|\nabla f(x^k)\|^2 = f(x^k) - \frac{1}{2L} \|\nabla f(x^k)\|^2 \ . \tag{3.2}$$

Estimates of type (3.2) is a standard procedure that leads to convergence analysis of iterative optimization algorithms. Using it, for GD Algorithm 3.1 with $\alpha = \dfrac{1}{L}$ we have

- General nonconvex case: $\min\limits_{0 \le k \le T-1} \|\nabla f(x^k)\| \le \sqrt{\dfrac{2L(f(x^0) - f(x^*))}{T}}$.

  Number of iterations $0 \le k \le T$ until $\|\nabla f(x^k)\| \le \varepsilon$ is $T \ge \dfrac{2L(f(x^0) - f(x^*))}{\varepsilon^2}$, sublinear;

- Convex case: $f(x^k) - f(x^*) \le \dfrac{L}{2k} \|x^0 - x^*\|^2$.

  Number of iterations $0 \le k \le T$ until $f(x^k) - f(x^*) \le \varepsilon$ is $T \ge \dfrac{f(x^0) - f(x^*)}{\varepsilon}$, sublinear;

- Strongly convex case: $f(x^k)-f(x^*) \leq \left(1-\dfrac{m}{L}\right)^k (f(x^0)-f(x^*))$.

  Number of iterations $0 \leq k \leq T$ until $f(x^k)-f(x^*) \leq \varepsilon$ is $T \geq \dfrac{L}{m}\ln\left(\dfrac{f(x^0)-f(x^*)}{\varepsilon}\right)$, linear.

3.2 Back Propagation and GD on Neural Networks

Classical algorithm of training multi–layer neural networks: Back–propagation.

Reference: Catherine F. Higham, Desmond J. Higham, Deep Learning: An Introduction for Applied Mathematicians, arXiv:1801.05894[math.HO]

Back–propagation method goes as follows. Let us set

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]} \text{ for } l = 2, 3, ..., L \ ,$$

in which l stands for the number of layers in the neural network. The neural network iteration can be viewed as

$$a^{[l]} = \sigma(z^{[l]}) \text{ for } l = 2, 3, ..., L \ .$$

We can view the neural network function as a chain with $a^{[1]} = x$ and $a^{[L]} = g(x;\omega)$. Let the training data be one point (x, y). Then the loss function is given by

$$C = C(\omega, b) = \frac{1}{2}\|y - a^{[L]}\|^2 \ .$$

To perform the GD algorithm as in Algorithm 3.1, we shall need the derivatives of the loss function C with respect to the neural–network parameters ω and b, that is $\dfrac{\partial C}{\partial w_{jk}^{[l]}}$ and $\dfrac{\partial C}{\partial b_j^{[l]}}$. Set the *error* in the j–th neuron at layer l to be $\delta^{[l]} = (\delta_j^{[l]})_j$ (viewed as a column vector) with $\delta_j^{[l]} = \dfrac{\partial C}{\partial z_j^{[l]}}$. Then one can calculate directly to obtain the following lemma

Lemma 3.2. *If $x, y \in \mathbb{R}^n$, let $x \circ y \in \mathbb{R}^n$ to be the Hadamard product defined by $(x \circ y)_i = x_i y_i$. We can directly calculate*

$$\delta^{[L]} = \sigma'(z^{[L]}) \circ (a^{[L]} - y) \ , \tag{3.3}$$

$$\delta^{[l]} = \sigma'(z^{[l]}) \circ (W^{[l+1]})^T \delta^{[l+1]} \ , \quad \textit{for } 2 \leq l \leq L-1 \ , \tag{3.4}$$

$$\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]} \ , \ \textit{for } 2 \leq l \leq L \ , \tag{3.5}$$

$$\frac{\partial C}{\partial \omega_{jk}^{[l]}} = \delta_j^{[l]} a_k^{[l-1]} \ , \ \textit{for } 2 \leq l \leq L \ . \tag{3.6}$$

Algorithm 3.2 Gradient Descent on Neural Networks: Backpropagation

```
1: **Input**: Training data $(x, y)$
2: **Forward pass**: $x = a^{[1]} \to a^{[2]} \to ... \to a^{[L-1]} \to a^{[L]}$
3: Obtain $\delta^{[L]}$ from (3.3)
4: **Backward pass**: Form (3.4) obtain $\delta^{[L]} \to \delta^{[L-1]} \to \delta^{[L-2]} \to ... \to \delta^{[2]}$
5: **Calculation of the gradient of the loss $C$**: Use (3.5) and (3.6)
```

Training GD on Over–parametrized Neural Networks has been understood only recently. See the following references:

Zhang, X., Yu, Y., Wang, L., Gu, Q., Learning One–hidden–layer ReLU networks via Gradient Descent, arXiv:1806.07808v1[stat.ML]

Du, S.S., Zhai, X., Póczos, B., Singh, A., Gradient Descent Provably Optimizes Over–parameterized Neural Networks, arXiv:1810.02054v1[cs.LG]

Allen–Zhu, Z., Li, Y., Song, Z., A Convergence Theory for Deep Learning via Over–Parametrization, arXiv:1811.03962v3[cs.LG]

Du, S.S., Lee, J.D., Li, H., Wang, L., Zhai, X., Gradient Descent Finds Global Minima of Deep Neural Networks, arXiv:1811.03804v1[cs.LG]

Neural Tangent Kernel. See the following paper:

Jacot, A., Gabriel, F., Hongler, C., Neural tangent kernel: Convergence and generalization in neural networks, NIPS, 2018.

Computational Graph and General Backpropagation for taking derivatives: basic idea in tensorflow. `https://www.tensorflow.org/tutorials/customization/autodiff`

3.3 Online Principle Component Analysis (PCA)

Reference: Li, C.J., Wang, M., Liu, H. and Zhang, T., Near-optimal stochastic approximation for online principal component estimation, *Mathematical Programming, Ser. B* (2018) 167:75-97.

Let $\boldsymbol{X}$ be a random vector in $\mathbb{R}^d$ with mean 0 and unknown covariance matrix

$$\Sigma = \mathbf{E}\left[\boldsymbol{X}\boldsymbol{X}^T\right] \in \mathbb{R}^{d\times d} .$$

Let the eigenvalues of Σ be ordered as $\lambda_1 > \lambda_2 \geq ... \geq \lambda_d \geq 0$. Principal Component Analysis (PCA) aims to find the principal eigenvector of Σ that corresponds to the largest eigenvalue λ_1, based on independent and identically distributed sample realizations $\boldsymbol{X}^{(1)}, ..., \boldsymbol{X}^{(n)}$. This can be casted into a *nonconvex stochastic optimization problem* given by

$$\begin{aligned} &\text{maximize } \boldsymbol{u}^T \mathbf{E}\left[\boldsymbol{X}\boldsymbol{X}^T\right]\boldsymbol{u} \ , \\ &\text{subject to } \|\boldsymbol{u}\| = 1, \boldsymbol{u} \in \mathbb{R}^d \ . \end{aligned}$$

Here $\| \bullet \|$ denotes the Euclidian norm. Assume the principle component $\boldsymbol{u}^*$ is unique.

Classical PCA:

$$\widehat{\boldsymbol{u}}^{(n)} = \arg\max_{\|\boldsymbol{u}\|=1} \boldsymbol{u}^T \widehat{\Sigma}^{(n)} \boldsymbol{u} \ .$$

Here $\widehat{\Sigma}^{(n)}$ is the empirical covariance matrix based on n samples:

$$\widehat{\Sigma}^{(n)} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{X}^{(i)}(\boldsymbol{X}^{(i)})^T \ .$$

Online PCA: $\Pi \boldsymbol{u} = \dfrac{\boldsymbol{u}}{\|\boldsymbol{u}\|}$ retraction.

Algorithm 3.3 Online PCA Algorithm

```
1: Input: Initialize $\boldsymbol{u}^{(0)}$ and choose the stepsize $\beta > 0$
2: for $n = 1, 2, ..., N$ do
3:     Draw one sample $\boldsymbol{X}^{(n)}$ from the (streaming) data source
4:     Update the iterate $\boldsymbol{u}^{(n)}$ by
           $\boldsymbol{u}^{(n)} = \Pi\left\{\boldsymbol{u}^{(n-1)} + \beta \boldsymbol{X}^{(n)}(\boldsymbol{X}^{(n)})^T \boldsymbol{u}^{(n-1)}\right\}$ .
5: end for
6: Output: $\boldsymbol{u}^{(N)}$
```

Oja's algorithm. See

Oja, E. (1982). Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, **15**(3), 267-273.

3.4 Line Search Methods

Line search methods in general

$$x^{k+1} = x^k + \alpha_k d^k \ , \ k = 0, 1, 2, ... \tag{3.7}$$

Here $\alpha_k > 0$ is the stepsize and $d^k \in \mathbb{R}^n$ is the descent direction.

Preliminary conditions:

$$\begin{aligned} &(1) \quad -(d^k)^T \nabla f(x^k) \geq \bar{\varepsilon} \|\nabla f(x^k)\| \cdot \|d^k\|; \\ &(2) \quad \gamma_1 \|\nabla f(x^k)\| \leq \|d^k\| \leq \gamma_2 \|\nabla f(x^k)\| \text{ where } \bar{\varepsilon}, \gamma_1, \gamma_2 > 0. \end{aligned} \tag{3.8}$$

Notice that if $d^k = -\nabla f(x^k)$, then $\bar{\varepsilon} = \gamma_1 = \gamma_2 = 1$.

In fact one can expand

$$
\begin{aligned}
f(x^{k+1}) &= f(x^k + \alpha d^k) \\
&= f(x^k) + \alpha \nabla f(x^k)^T d^k + \alpha \int_0^1 [\nabla f(x^k + \gamma \alpha d^k) - \nabla f(x^k)] d^k d\gamma \\
&\leq f(x^k) - \alpha \left(\bar{\varepsilon} - \alpha \frac{L}{2} \gamma_2 \right) \|\nabla f(x^k)\| \cdot \|d^k\| \ .
\end{aligned}
\tag{3.9}
$$

So if $\alpha \in \left(0, \dfrac{2\bar{\varepsilon}}{L\gamma_2}\right)$, then $f(x^{k+1}) < f(x^k)$.

Schemes of descent type produce iterates that satisfy a bound of the form

$$
f(x^{k+1}) \leq f(x^k) - C\|\nabla f(x^k)\|^2 \text{ for some } C > 0 \ . \tag{3.10}
$$

Various Line Search Methods: (1) $d^k = -S^k \nabla f(x^k)$, S^k symmetric positively definite, $\lambda(S^k) \in [\gamma_1, \gamma_2]$; (2) Gauss-Southwell; (3) Stochastic coordinate descent; (4) Stochastic Gradient Methods; (5) Exact line search $\min\limits_{\alpha > 0} f(x^k + \alpha d^k)$; (6) Approximate Line Search, "weak Wolfe conditions"; (7) Backtracking Line Search.

3.5 Convergence to Approximate Second Order Necessary Points

Second–order necessary point: $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.

Definition 3.1. *We call point $x \in \mathbb{R}^n$ an $(\varepsilon_g, \varepsilon_H)$–approximate second order stationary point if*

$$
\|\nabla f(x)\| \leq \varepsilon_g \ \textit{and} \ \lambda_{min}(\nabla^2 f(x)) \geq -\varepsilon_H \tag{3.11}
$$

for some choices of small constants $\varepsilon_g > 0$ and $\varepsilon_H > 0$.

How to search for such a second–order stationary point?

Set $M > 0$ to be the Lipschitz bound for the Hessian $\nabla^2 f(x)$, so that

$$
\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq M\|x - y\| \text{ for all } x, y \in \text{dom}(f) \ . \tag{3.12}
$$

Cubic upper bound

$$
f(x + p) \leq f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x) p + \frac{1}{6} M \|p\|^3 \ . \tag{3.13}
$$

For steepest descent step we have from (3.2) that

$$
f(x^{k+1}) \leq f(x^k) - \frac{1}{2L}\|\nabla f(x^k)\|^2 \leq f(x^k) - \frac{\varepsilon_g^2}{2L} \ .
$$

Otherwise, the negative direction Hessian eigenvector search will output, by using third–order Taylor expansion and (3.12), that

Algorithm 3.4 Search for second–order approximate stationary point based on Gradient Descent

```
1: Input: Input initialization $x^0$, $k = 0, 1, 2, ...$
2: while $\|\nabla f(x^k)\| > \varepsilon_g$ or $\lambda_k < -\varepsilon_H$ do
3:   if $\|\nabla f(x^k)\| > \varepsilon_g$ then
4:     $x^{k+1} = x^k - \frac{1}{L}\nabla f(x^k)$
5:   else
6:     $\lambda^k := \lambda_{\min}(\nabla^2 f(x^k))$
7:     Choose $p^k$ to be the eigenvector corresponding to the most negative eigenvalue
       of $\nabla^2 f(x^k)$
8:     Choose the size and sign of $p^k$ such that $\|p^k\| = 1$ and $(p^k)^T\nabla f(x^k) \le 0$
9:     $x^{k+1} = x^k + \alpha_k p^k$, $\alpha_k = \frac{2\lambda_k}{M}$
10:  end if
11:  $k \leftarrow k + 1$
12: end while
13: Output: An estimate of an $(\varepsilon_g, \varepsilon_H)$–approximate second order stationary point $x^k$
```

$$\begin{aligned}
f(x^{k+1}) &\le f(x^k) + \alpha_k \nabla f(x^k)^T p^k + \frac{1}{2}\alpha_k^2 (p^k)^T \nabla^2 f(x^k) p^k + \frac{1}{6} M \alpha_k^3 \|p^k\|^3 \\
&\le f(x^k) - \frac{1}{2}\left(\frac{2|\lambda_k|}{M}\right)^2 |\lambda_k| + \frac{1}{6} M \left(\frac{2|\lambda_k|}{M}\right)^3 \\
&= f(x^k) - \frac{2}{3}\frac{|\lambda_k|^3}{M^2} \\
&= f(x^k) - \frac{2}{3}\frac{\varepsilon_H^3}{M^2}\ .
\end{aligned}$$

Number of iterations

$$K \le \max\left(2L\varepsilon_g^{-2}, \frac{3}{2}M^2\varepsilon_H^{-3}\right)\left(f(x^0) - f(x^*)\right)\ .$$

4 Accelerated Gradient Methods

4.1 Polyak's heavy ball method

Why acceleration in GD?

The GD method id "greedy" in the sense that it steps in the direction that is most productive at current iterate – no explicit use of knowledge of f at earlier iterations.

Adding "momentum" = search direction tends to be similar to that one used in the previous step.

Say $f(x) = \frac{1}{2}x^T Qx - b^T x + c$ is a quadratic function. It may well happen in deep neural network models that Q has a large condition number $\kappa = \frac{L}{m}$, see this paper

Chaudhari, P., Soatto, S., Stochastic Gradient Descent performes variational inference, converges to limit cycles for deep networks, *ICLR*, 2018.

Slow convergence for κ large: for the strongly convex case we showed that $\beta = 1-\frac{1}{\kappa}$ which $\to 1$ when $\kappa \to \infty$. Demonstrate a picture of oscillations of GD near local min when the Hessian has a large condition number.

Motivated from classical mechanics, second order equation that models a nonlinear oscillator:

GD: $\frac{dx}{dt} = -\nabla f(x)$; AGD: $\mu\frac{d^2x}{dt^2} = -\nabla f(x) - \mu b\frac{dx}{dt}$.

"Small mass limit" $\mu \to 0$. Term $\mu\frac{d^2x}{dt^2}$ serves as "regularization".

Finite difference approximation:

$$\mu\frac{x(t+\Delta t) - 2x(t) + x(t-\Delta t)}{(\Delta t)^2} \approx -\nabla f(x(t)) - \frac{\mu b(x(t+\Delta t) - x(t))}{\Delta t} .$$

i.e.

$$x(t+\Delta t) = x(t) - \alpha\nabla f(x(t)) + \beta(x(t) - x(t-\Delta t)) .$$

"heavy ball method" (see Polyak, B., Some methods of speeding up the convergence of iteration methods, *USSR Computational Mathematics and Mathematical Physics*, Volume 4, Issue 5, 1964, pp. 1–17.)

$$x^{k+1} = x^k - \alpha\nabla f(x^k) + \beta(x^k - x^{k-1}) \ , \ x^{-1} = x^0 \ .$$

Algorithm 4.1 Polyak's heavy ball method (GD with momentum)

```
1: Input: Input initialization $x^{-1} = x^0$, $p^{-1} = 0$, stepsizes $\alpha, \beta > 0$, $k = 0, 1, 2, ...$
2: for $k = 0, 1, 2, ..., K-1$ do
3:     $p^k = -\alpha\nabla f(x^k) + \beta p^{k-1}$
4:     $x^{k+1} = x^k + p^k$
5: end for
6: Output: Output $x^K$
```

"Momentum": $p_k = x^{k+1} - x^k$. The momentum iteration

$$p^k = \beta p^{k-1} - \alpha \nabla f(x^k)$$

can be understood as doing an exponential moving average over the gradients. This is where the "memory" of "inertia" comes in, that inspires the adaptive methods.

4.2 Nesterov's Accelerated Gradient Descent (AGD)

"first order optimal method": Instead of evaluating $\nabla f(x^k)$, we evaluate $\nabla f(x^k + \beta(x^k - x^{k-1}))$.

Nesterov's accelerated gradient descent:

$$x^{k+1} = x^k - \alpha \nabla f(x^k + \beta(x^k - x^{k-1})) + \beta(x^k - x^{k-1}) \ , \ x^{-1} = x^0 \ .$$

Algorithm 4.2 Nesterov's accelerated gradient descent

```
1: Input: Input initialization $x^{-1} = x^0$, stepsizes $\alpha_k, \beta_k > 0$, $k = 0, 1, 2, ...$
2: for $k = 0, 1, 2, ..., K-1$ do
3:     $y^k = x^k + \beta_k(x^k - x^{k-1})$
4:     $x^{k+1} = y^k - \alpha_k \nabla f(y^k)$
5: end for
6: Output: Output $x^K$
```

By introducing the momentum variable $p^k = x^{k+1} - x^k$ we can also rewrite the Nesterov's acclerated gradient descent in terms of momentum, this is the Netsterov's momentum algorithm:

Algorithm 4.3 Nesterov's momentum (equivalent to Nesterov's accelerated gradient descent)

```
1: Input: Input initialization $x^{-1} = x^0$, $p^{-1} = 0$, stepsizes $\alpha_k, \beta_k > 0$, $k = 0, 1, 2, ...$
2: for $k = 0, 1, 2, ..., K-1$ do
3:     $p^k = \beta_k p^{k-1} - \alpha_k \nabla f(x^k + \beta_k p^{k-1})$
4:     $x^{k+1} = x^k + p^k$
5: end for
6: Output: Output $x^K$
```

Convergence proof of Nesterov's method:

(1) Convex quadratic objective function: Spectral method.

$$f(x) = \frac{1}{2} x^T Q x - b^T x + c \ , \ Q \succ 0 \ , \ 0 < m = \lambda_n \leq \lambda_{n-1} \leq ... \leq \lambda_2 \leq \lambda_1 = L \ .$$

Condition number $\kappa = \dfrac{L}{m}$.

Minimizer of f is $x^* = Q^{-1}b$ and $\nabla f(x) = Qx - b = Q(x - x^*)$.

"Spectral method"

$$x^{k+1} - x^* = (x^k - x^*) - \alpha Q\left(x^k + \beta(x^k - x^{k-1}) - x^*\right) + \beta\left((x^k - x^*) - (x^{k-1} - x^*)\right) .$$

So we have

$$\begin{bmatrix} x^{k+1} - x^* \\ x^k - x^* \end{bmatrix} = \begin{bmatrix} (1+\beta)(I - \alpha Q) & -\beta(I - \alpha Q) \\ I & 0 \end{bmatrix} \begin{bmatrix} x^k - x^* \\ x^{k-1} - x^* \end{bmatrix} \ , \ k = 1, 2, ...$$

Set $\omega^k = \begin{pmatrix} x^{k+1} - x^* \\ x^k - x^* \end{pmatrix}$ with $x^{-1} = x^0$, and set $T = \begin{bmatrix} (1+\beta)(I - \alpha Q) & -\beta(I - \alpha Q) \\ I & 0 \end{bmatrix}$, then $\omega^k = T\omega^{k-1}$. So $\omega^k = T^k\omega^0$.

Theorem 4.1. *Pick* $\alpha = \dfrac{1}{L}$, $\beta = \dfrac{\sqrt{L} - \sqrt{m}}{\sqrt{L} + \sqrt{m}} = \dfrac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}$, *then the eigenvalues of* T *are*

$$\nu_{i,1} = \frac{1}{2}\left[(1+\beta)(1 - \alpha\lambda_i) + i\sqrt{4\beta(1 - \alpha\lambda_i) - (1+\beta)^2(1 - \alpha\lambda_i)^2}\right] \ ,$$

$$\nu_{i,2} = \frac{1}{2}\left[(1+\beta)(1 - \alpha\lambda_i) - i\sqrt{4\beta(1 - \alpha\lambda_i) - (1+\beta)^2(1 - \alpha\lambda_i)^2}\right] \ .$$

In particular, the spectral radius $\rho(T) \equiv \lim\limits_{k\to\infty} \left(\|T^k\|\right)^{1/k} \leq 1 - \dfrac{1}{\sqrt{\kappa}}$.

By Theorem 4.1, we conclude for any $\varepsilon > 0$ that

$$\|\omega^k\| \leq C\|\omega^0\|(\rho(T) + \varepsilon)^k \ .$$

Comparison of the rates with GD: when $\alpha = \dfrac{1}{L}$, GD needs $O\left(\dfrac{L}{m}|\ln\varepsilon|\right)$ to reach $f(x^k) - f^* \leq \varepsilon$ while Nesterov needs $O\left(\sqrt{\dfrac{L}{m}}|\ln\varepsilon|\right)$ to reach a factor ε in $\|\omega^k\|$. Key parameter is $\kappa = \dfrac{L}{m}$.

(2) Strongly convex objective functions: Lyapunov's method.

Lyapunov function $V : \mathbb{R}^d \to \mathbb{R}$, $V(\omega) > 0$ for all $\omega \neq \omega^* \in \mathbb{R}^d$, $V(\omega^*) = 0$, $V(\omega^{k+1}) < \rho^2 V(\omega^k)$ for $0 < \rho < 1$.

Indeed, $f(x) - f^*$ is a Lyapunov function for GD.

What is the Lyapunov function for Nesterov?

$\widetilde{x}^k = x^k - x^*$, $\widetilde{y}^k = y^k - y^*$, set

$$V_k = f(x^k) - f(x^*) + \frac{L}{2}\|\widetilde{x}^k - \rho^2\widetilde{x}^{k-1}\|^2 \ .$$

Calculations... Possible to show $\alpha_k = \dfrac{1}{L}$, $\beta_k = \dfrac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$, $\rho^2 = 1 - \dfrac{1}{\sqrt{\kappa}}$ such that $V_{k+1} \le \rho^2 V_k$.

This is to say we have

$$f(x^k) - f^* \le \left(1 - \frac{1}{\sqrt{\kappa}}\right)^k \left\{ f(x_0) - f^* + \frac{m}{2}\|x_0 - x^*\|^2 \right\} \ ,$$

where one can verify that $V_0 = f(x_0) - f^* + \dfrac{m}{2}\|x_0 - x^*\|^2$.

(3) Convex objective functions: Lyapunov's method.

Set $\alpha_k = \dfrac{1}{L}$ but β_k is variable, and thus ρ_k is variable.

$$V_k = f(x^k) - f^* + \frac{L}{2}\|\widetilde{x}^k - \rho_{k-1}^2 \widetilde{x}^{k-1}\|^2 \ .$$

(Weakly) Convex objective: $m = 0$.

Obtain

$$V_{k+1} \le \rho_k^2 V_k + R_k^{(\text{weak})} \ ,$$

where

$$R_k^{(\text{weak})} = \frac{L}{2}\|\widetilde{y}^k - \rho_k^2 \widetilde{x}^k\|^2 - \frac{\rho_k^2 L}{2}\|\widetilde{x}^k - \rho_{k-1}^2 \widetilde{x}^{k-1}\|^2 \ .$$

Recursive relation: $\rho_k^2 = \dfrac{(1-\rho_k^2)^2}{(1-\rho_{k-1}^2)^2}$, $k = 1, 2, \ldots.$ We have $1 + \beta_k - \rho_k^2 = \rho_k$, $\beta_k = \rho_k \rho_{k-1}^2$. This makes $R_k^{(\text{weak})} = 0$.

So $V_k \le \rho_{k-1}^2 V_{k-1}$ and thus

$$V_k \le \rho_{k-1}^2 \rho_{k-2}^2 \ldots \rho_1^2 V_1 = \frac{(1-\rho_{k-1}^2)^2}{(1-\rho_0^2)^2} V_1 \ .$$

Bound $V_1 \le (1-\rho_{k-1}^2)^2 \dfrac{L}{2}\|x^0 - x^*\|^2$ and inductive shows that $1 - \rho_k^2 \le \dfrac{2}{k+2}$.

Finally we obtain the rate

$$f(x^k) - f^* \le V_k \le \frac{2L}{(k+1)^2}\|x^0 - x^*\|^2 \ .$$

See Nesterov, Y., A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Math Dokl*, **27**(2), 1983.

Algorithm 4.4 Nesterov's optimal algorithm for general convex function f

```
1: Input: Input initialization $x^{-1} = x^0$, $\beta_0 = \rho_0 = 0$ and Lipschitz constant $L$ for $f$
2: for $k = 0, 1, 2, ..., K-1$ do
3:     $y^k = x^k + \beta_k(x^k - x^{k-1})$
4:     $x^{k+1} = y^k - \frac{1}{L}\nabla f(y^k)$
5:     Define $\rho_{k+1} \in [0,1]$ to be the root of the quadratic equation
           $$1 + \rho_{k+1}(\rho_k^2 - 1) - \rho_{k+1}^2 = 0$$
6:     Set $\beta_{k+1} = \rho_{k+1}\rho_k^2$
7: end for
8: Output: Output $x^K$
```

Connection with differential equations: Su, W., Boyd, S., Candés, E.J., A differential equation for modeling Nesterov's accelerated gradient method: theory and insights. *Journal of Machine Learning Research*, **17**:143, 2016. Suggest choosing $\beta_k = \dfrac{k-1}{k+2}$?

Su-Boyd-Candés differential equation:

$$\ddot{X}_t + \frac{3}{t}\dot{X}_t + \nabla f(X_t) = 0 \ .$$

Nesterov's method achieves first order optimal convergence rate: Example. See Nesterov, Y., *Introductory Lectures on Convex Optimization*, Springer, 2004, §2.1.2.

Indeed, one can construct a carefully designed function for which *no* method that makes use of all gradient observed up to and including iteration k (namely $\nabla f(x^i)$, $i = 0, 1, 2, ..., k$) can produce a sequence $\{x^k\}$ that achieves a rate better than the order $O(1/k^2)$.

Set $f(x) = \dfrac{1}{2}x^T Ax - e_1^T x$, where

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 & ... & ... & 0 \\ -1 & 2 & -1 & 0 & ... & ... & 0 \\ ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix} \ , \ e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ ... \\ 0 \end{pmatrix} \ .$$

Optimization $\min\limits_{x\in\mathbb{R}^n} f(x)$ has a solution x^* such that $Ax^* = e_1$, and its components are $x_i^* = 1 - \dfrac{i}{n+1}$ for $i = 1, 2, ..., n$. Let the starting point $x^0 = 0$, and the iterate

$$x^{k+1} = x^k + \sum_{j=0}^{k} \gamma_j \nabla f(x^j)$$

for some coefficients γ_j, $j = 0, 1, ..., k$. Then each iterate x^k can have nonzero entries only in its first k components. Thus

$$\|x^k - x^*\|^2 \geq \sum_{j=k+1}^{n} (x_j^*)^2 = \sum_{j=k+1}^{n} \left(1 - \frac{j}{n+1}\right)^2 .$$

One can show that for $k = 1, 2, ..., \frac{n}{2} - 1$ we have

$$\|x^k - x^*\|^2 \geq \frac{1}{8}\|x^0 - x^*\|^2 .$$

This will lead to the bound that

$$f(x^k) - f(x^*) \geq \frac{3L}{32(k+1)^2}\|x^0 - x^*\|^2 , \ k = 1, 2, ..., \frac{n}{2} - 1$$

where $L = \|A\|_2$.

4.3 Conjugate gradient method

Conjugate gradient method is a more classical method in computational mathematics. Unlike the Nesterov's accelerated gradient method, the conjugate gradient method does not require knowledge of the convexity parameters L and m to compute the appropriate step–sizes. Unfortunately, it does not admit particularly rigorous analysis when the objective function f is not quadratic.

Originally this method comes from numerical linear algebra: goal is to solve $Qx = p$. But this can be formulated as optimization problem $\min_x f(x)$ with $f(x) = \frac{1}{2}x^TQx - p^Tx + c$.

Algorithm 4.5 Conjugate Gradient Method

```
1: Input: Input initialization $x^0$, $y^0 = -\nabla f(x^0)$, $f(x) = \frac{1}{2}x^TQx - p^Tx + c$, $k = 0, 1, 2, ...$
2: for $k = 0, 1, 2, ..., K-1$ do
3:     $r^k = Qx^k - p$
4:     $\alpha_k = \frac{\langle y^k, r^k \rangle}{\langle y^k, Qy^k \rangle}$
5:     $x^{k+1} = x^k - \alpha_k y^k$
6:     $y^{k+1} = -\nabla f(x^{k+1}) + \beta_k y^k$
7:     $\beta_{k+1} = \frac{\langle r^k, Qy^k \rangle}{\langle y^k, Qy^k \rangle}$
8: end for
9: Output: Output $x^K$
```

Exact line search in the direction y^k: $\alpha_k = \arg\min_{\alpha > 0} f(x^k + \alpha y^k)$.

Conjugate gradient direction: β_{k+1} is so chosen that $\langle y^{k+1}, Qy^k \rangle = 0$.

5 Stochastic Gradient Methods

5.1 SGD

Stochastic Gradient Descent (SGD) is a stochastic analogue of the Gradient Descent algorithm, aiming at finding the local or global minimizers of the function expectation parameterized by some random variable. Schematically, the algorithm can be interpreted as targeting at finding a local minimum point of the expectation of function

$$f(x) = \mathbf{E} f(x;\zeta) \ , \tag{5.1}$$

where the index random variable ζ follows some prescribed distribution $\mathcal{D}$, and the weight vector $x \in \mathbb{R}^d$. SGD updates via the iteration

$$x_{k+1} = x_k - \alpha_k g(x_k;\zeta_k) \ , \tag{5.2}$$

where the noisy gradient $g(x;\zeta) = \nabla f(x;\zeta)$, $\alpha_k > 0$ is the learning rate, and ζ_k are i.i.d. random variables that have the same distribution as $\zeta \sim \mathcal{D}$.

In particular, in the case of training a deep neural network, the random variable ζ samples size m ($m \le n$) mini–batches $\mathcal{B}$ uniformly from an index set $\{1,2,...,n\}$: $\mathcal{B} \subset \{1,2,...,n\}$ and $|\mathcal{B}| = m$. In this case, given loss functions $f_1(x),...,f_n(x)$ on training data, we have

$$f(x;\zeta) = \frac{1}{m}\sum_{i\in\mathcal{B}} f_i(x) \text{ and } f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) \ .$$

Algorithm 5.1 Stochastic Gradient Descent

```
1: Input: Input initialization $x^0$, index random variable $\zeta \sim \mathcal{D}$, noisy gradient
   $g(x;\zeta) = \nabla f(x;\zeta)$ such that $\mathbf{E} g(x;\zeta) = \nabla f(x)$, learning rate $\alpha_k > 0$, $k = 0,1,2,...$
2: for $k = 0,1,2,...,K-1$ do
3:     Sample the i.i.d. random variable $\zeta_k \sim \mathcal{D}$
4:     Calculate the noisy gradient $g(x_k;\zeta_k) = \nabla f(x_k;\zeta_k)$
5:     $x_{k+1} = x_k - \alpha_k g(x_k;\zeta_k)$
6: end for
7: Output: Output $x_K$
```

Convergence analysis of SGD.

f is strongly convex: measure $\mathbf{E}\left[\|x_k - x^*\|^2\right]$; f is convex but not necessarily strongly convex: measure $f(x_k) - f(x^*)$.

Assumption 5.1. *We assume that for some $L_g > 0$ and $B > 0$ we have*

$$\mathbf{E}_\zeta \left[\|g(x;\zeta)\|_2^2\right] \le L_g^2\|x-x^*\|^2 + B^2 \tag{5.3}$$

for all x.

Remark 5.1. *In the mini-batch stochastic gradient case, $g(x;\zeta) = \dfrac{1}{m}\sum\limits_{i\in\mathcal{B}} \nabla f_i(x)$. Then we have*

$$\begin{array}{rl} & \mathbf{E}_\zeta\|g(x;\zeta)\|_2^2 \\ = & \mathbf{E}\left\|\dfrac{1}{m}\sum\limits_{i\in\mathcal{B}}\nabla f_i(x)\right\|^2 \\ = & \mathbf{E}\left\|\dfrac{1}{m}\sum\limits_{i\in\mathcal{B}}(\nabla f_i(x)-\nabla f_i(x^*)) + \dfrac{1}{m}\sum\limits_{i\in\mathcal{B}}\nabla f_i(x^*)\right\|^2 \\ \le & 2\mathbf{E}\left\|\dfrac{1}{m}\sum\limits_{i\in\mathcal{B}}(\nabla f_i(x)-\nabla f_i(x^*))\right\|^2 + 2\mathbf{E}\left\|\dfrac{1}{m}\sum\limits_{i\in\mathcal{B}}\nabla f_i(x^*)\right\|^2 \\ \le & 2\mathbf{E}\sum\limits_{i\in\mathcal{B}}\dfrac{1}{m^2}\sum\limits_{i\in\mathcal{B}}\|\nabla f_i(x)-\nabla f_i(x^*)\|^2 + B^2 \\ \le & 2\mathbf{E}\dfrac{1}{m}\sum\limits_{i\in\mathcal{B}}\|\nabla f_i(x)-\nabla f_i(x^*)\|^2 + B^2 \\ \le & 2\mathbf{E}\|\nabla f_i(x)-\nabla f_i(x^*)\|^2 + B^2\ . \end{array}$$

So we can pick $L_g = \sqrt{2}L_{\nabla f}$. If $L_g = 0$ in (5.3), then f cannot be m-strongly convex over an unbounded domain, since by Jensen's inequality $\|\nabla f(x)\|^2 = \|\mathbf{E}g(x;\xi)\|^2 \le \mathbf{E}\left[\|g(x;\xi)\|^2\right]$ but $\langle x-x^*, \nabla f(x)\rangle = \langle x-x^*, \nabla f(x)-\nabla f(x^*)\rangle = \langle x-x^*, \int_0^1 \nabla^2 f(x^*+\gamma(x-x^*))(x-x^*)d\gamma\rangle \ge m\|x-x^*\|^2$ for all x.

Square expansion for a stochastic iteration:

$$\begin{aligned} \|x_{k+1}-x^*\|^2 &= \|x_k - \alpha_k g(x_k;\zeta_k) - x^*\|^2 \\ &= \|x_k-x^*\|^2 - 2\alpha_k\langle g(x_k;\zeta_k), x_k-x^*\rangle + \alpha_k^2\|g(x_k;\zeta_k)\|^2\ . \end{aligned} \tag{5.4}$$

Set $a_k = \mathbf{E}\left[\|x_k-x^*\|^2\right]$, then

$$a_{k+1} \le \left(1+\alpha_k^2L_g^2\right)a_k - 2\alpha_k\mathbf{E}\left[\langle\nabla f(x_k), x_k-x^*\rangle\right] + \alpha_k^2B^2\ . \tag{5.5}$$

Case 1: $L_g = 0$. Average output: $\bar{x}_K = \dfrac{\sum\limits_{j=0}^K \alpha_j x_j}{\sum\limits_{j=0}^K \alpha_j}$.

Set $\alpha_k = \alpha > 0$ and $\alpha_{\text{opt}} = \dfrac{D_0}{B\sqrt{T+1}}$, $\theta = \dfrac{\alpha}{\alpha_{\text{opt}}}$. Set $L_g = 0$. Then

$$\mathbf{E}\left[f(\bar{x}_T) - f(x^*)\right] \le \left(\frac{1}{2}\theta + \frac{1}{2}\theta^{-1}\right)\frac{BD_0}{\sqrt{T+1}}\ .$$

Case 2: $B = 0$. Assume that $\langle \nabla f(x), x - x^* \rangle \geq m\|x - x^*\|^2$ for some $m > 0$. Then if $\alpha_k = \alpha \in \left(0, \frac{2m}{L_g^2}\right)$, then we obtain a linear rate of convergence. Optimal $\alpha = \frac{2}{L_g^2}$.

$$a_k \leq \left(1 - \frac{m^2}{L_g^2}\right)^k D_0^2 < \varepsilon \Longleftrightarrow T \geq \left\lceil \frac{L_g^2}{m^2} \ln\left(\frac{D_0^2}{\varepsilon}\right) \right\rceil \ .$$

Case 3: $B > 0$ and $L_g > 0$, and f is strongly convex. Choose $\alpha \in \left(0, \frac{2m}{L_g^2}\right)$ and $\alpha_k = \alpha$.

$$a_k \leq \left(1 - 2m\alpha + \alpha^2 L_g^2\right)^k D_0^2 + \frac{\alpha B^2}{2m - \alpha L_g^2} \ .$$

"threshold value" $\frac{\alpha B^2}{2m - \alpha L_g^2}$. "epoch doubling", "hyper–parameter tuning"

5.2 Variance–reduction methods: SVRG, SAG, SAGA, SARAH, SPIDER, STORM

Variance Reduction Methods: Goal is to reduce the variance produced by the stochastic terms in the iteration scheme.

SVRG: Johnson, R., Zhang, T., Accelerating Stochastic Gradient using Predictive Variance Reduction, *NIPS*, 2013.

Goal is to minimize $\min P(\omega)$ where $P(\omega) = \frac{1}{n}\sum_{i=1}^{n} \psi_i(\omega)$.

SGD: draw i_t randomly from $\{1, ..., n\}$ and set

$$\omega^{(t)} = \omega^{(t-1)} - \eta_t \nabla \psi_{i_t}(\omega^{(t-1)}) \ .$$

How to reduce variance? We can keep a snapshot $\widetilde{\omega}$ after every m SGD iterations, and we maintain the average gradient

$$\widetilde{\mu} = \nabla P(\widetilde{\omega}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \psi_i(\widetilde{\omega}) \ .$$

Variance–reduction:

$$\omega^{(t)} = \omega^{(t-1)} - \eta_t \left(\nabla \psi_{i_t}(\omega^{(t-1)}) - \nabla \psi_{i_t}(\widetilde{\omega}) + \widetilde{\mu}\right) \ .$$

Algorithm 5.2 Stochastic Variance Reduced Gradient

```
1: Input: Input initialization $\widetilde{\omega}_0$, update frequency $m$ and learning rate $\eta$
2: for $s = 1, 2, ...$ do
3:     $\widetilde{\omega} = \widetilde{\omega}_{s-1}$
4:     $\widetilde{\mu} = \frac{1}{n}\sum_{i=1}^{n} \nabla\psi_i(\widetilde{\omega})$
5:     $\omega_0 = \widetilde{\omega}$
6:     for $t = 1, 2, ..., m$ do
7:         Sample $i_t$ independently and uniformly from $\{1, ..., n\}$
8:         Update the weight $\omega_t = \omega_{t-1} - \eta\left(\nabla\psi_{i_t}(\omega_{t-1}) - \nabla\psi_{i_t}(\widetilde{\omega}) + \widetilde{\mu}\right)$
9:     end for
10:    $\widetilde{\omega}_s \leftarrow \omega_t$ for randomly chosen $t \in \{0, ..., m-1\}$
11: end for
12: Output: $\widetilde{\omega}_s$
```

Convergence analysis of SVRG. Follow the original SVRG paper.

Theorem 5.2. *Consider the SVRG algorithm 5.2 and assume that each $\nabla\psi_i$ is $L-$Lipschitz and convex, and the function $P(\omega) = \frac{1}{n}\sum_{i=1}^{n}\psi_i(\omega)$ is $\gamma-$strongly convex. Set $\omega_* = \arg\min_{\omega} P(\omega)$ and assume that the epoch length m is so large that*

$$\alpha = \frac{1}{\gamma\eta(1-2L\eta)m} + \frac{2L\eta}{1-2L\eta} < 1 \ ,$$

then we have the linear convergence in expectation of SVRG algorithm 5.2, i.e.

$$\mathbf{E}[P(\widetilde{\omega}_s) - P(\omega_*)] \leq \alpha^s[P(\widetilde{\omega}_0) - P(\omega_*)] \ .$$

SAG: Roux, Nicolas L., Mark Schmidt, Francis R. Bach, A stochastic gradient method with an exponential convergence rate for finite training sets. *NIPS*, 2012.

Guaranteed linear convergence rate for strongly convex objective $P(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$.

Algorithm 5.3 Stochastic Average Gradient: minimizing $P(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$

```
1: Input: Input initialization $x_0$, stepsize $\alpha > 0$, $d = 0$, $y_i = 0$, $i = 1, 2, ..., n$
2: for $k = 0, 1, ..., K-1$ do
3:     Sample $i$ uniformly randomly from $\{1, 2, ..., n\}$
4:     $d \leftarrow d - y_i + \nabla f_i(x)$
5:     $y_i = \nabla f_i(x)$
6:     $x \leftarrow x - \frac{\alpha}{n} d$
7: end for
8: Output: $x_K$
```

SAGA: Defazio, A., Bach, F., Lacoste-Julien, S., SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives, *NIPS*, 2014.

Guaranteed linear convergence rate for strongly convex objective $P(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$.

Algorithm 5.4 SAGA: minimizing $P(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$

```
1: Input: Input initial vector $x^0 \in \mathbb{R}^d$ and known gradients $\nabla f_i(\phi_i^0) \in \mathbb{R}^d$ with $\phi_i^0 = x^0$
   for each $i$. These derivatives are stored in an $n \times d$ matrix.
2: for $k = 0, 1, ..., K-1$ do
3:     Sample $j$ uniformly randomly from $\{1, 2, ..., n\}$
4:     Take $\phi_j^{k+1} = x^k$ and store $\nabla f_j(\phi_j^{k+1})$ in the table. All other entries in the table
       remain unchanged. The quantity $\phi_j^{k+1}$ is not explicitly stored.
5:     Update $x$ using $\nabla f_j(\phi_j^{k+1})$, $\nabla f_j(\phi_j^k)$ and the table average:
       $$w^{k+1} = x^k - \gamma \left[ \nabla f_j(\phi_j^{k+1}) - \nabla f_j(\phi_j^k) + \frac{1}{n}\sum_{i=1}^{n} \nabla f_j(\phi_i^k) \right] ,$$
       $$x^{k+1} = \text{prox}_\gamma^h(w^{k+1}) = \arg\min_{x \in \mathbb{R}^d} \left\{ h(x) + \frac{1}{2\gamma}\|x - w^{k+1}\|^2 \right\} .$$
6: end for
7: Output: $x^K$
```

SARAH: Lam M. Nguyen, Jie Liu, Katya Scheinberg, Martin Takáč, SARAH: A Novel Method for Machine Learning Problems Using Stochastic Recursive Gradient, *Proceedings of the 34th International Conference on Machine Learning, PMLR* **70**:2613-2621, 2017.

Guaranteed linear convergence rate for strongly convex objective $P(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$.

Algorithm 5.5 StochAstic Recursive grAdient algoritHm: minimizing $P(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$

```
1: Input: Initialization $\widetilde{w}_0$, the learning rate $\eta > 0$ and the inner loop size $m$
2: for $s = 1, 2, ...$ do
3:     $w_0 = \widetilde{w}_{s-1}$
4:     $v_0 = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w_0)$
5:     $w_1 = w_0 - \eta v_0$
6:     for $t = 1, ..., m-1$ do
7:         Sample $i_t$ uniformly randomly in $\{1, 2, ..., n\}$
8:         $v_t = \nabla f_{i_t}(w_t) - \nabla f_{i_t}(w_{t-1}) + v_{t-1}$
9:         $w_{t+1} = w_t - \eta v_t$
10:    end for
11:    Set $\widetilde{w}_s = w_t$ with $t$ chosen uniformly randomly from $\{0, 1, ..., m-1\}$
12: end for
13: Output: $\widetilde{w}_s$
```

SPIDER: Cong Fang, Chris Junchi Li, Zhouchen Lin, Tong Zhang, SPIDER: Near-Optimal Non-Convex Optimization via Stochastic Path-Integrated Differential Estimator, *NIPS*, 2018.

No assumption on convexity. Optimal Rate $O\left(\min(n^{1/2}\varepsilon^{-2}, \varepsilon^{-3})\right)$ for finding an ε–approximate first order stationary point.

Algorithm 5.6 Stochastic Path–Integrated Differential EstiamtoR: Searching First Order stationary point

```
1: Input: Input $\boldsymbol{x}^0$, $q$, $S_1$, $S_2$, $n_0$, $\varepsilon$ and $\widetilde{\varepsilon}$
2: for $k = 0, 1, ..., K-1$ do
3:   if $\mathrm{mod}(k, q) = 0$ then
4:     Draw $S_1$ samples (or compute the full gradient in the finite sum case), let $\boldsymbol{v}^k = \nabla f_{\mathcal{S}_1}(\boldsymbol{x}^k)$
5:   else
6:     Draw $S_2$ samples, and let $\boldsymbol{v}^k = \nabla f_{\mathcal{S}_2}(\boldsymbol{x}^k) - \nabla f_{\mathcal{S}_2}(\boldsymbol{x}^{k-1}) + \boldsymbol{v}^{k-1}$
7:   end if
8:   —Option I—
9:   if $\|\boldsymbol{v}^k\| \leq 2\widetilde{\varepsilon}$ then
10:    Return $\boldsymbol{x}^k$
11:  else
12:    $\eta = \dfrac{\varepsilon}{Ln_0}$
13:    $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \eta \dfrac{\boldsymbol{v}^k}{\|\boldsymbol{v}^k\|}$
14:  end if
15:  —Option II—
16:  $\eta^k = \min\left(\dfrac{\varepsilon}{Ln_0\|\boldsymbol{v}^k\|}, \dfrac{1}{2Ln_0}\right)$
17:  $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \eta^k \boldsymbol{v}^k$
18: end for
19: Output: Output $\widetilde{\boldsymbol{x}}$ uniformly at random from $\{\boldsymbol{x}^k\}_{k=0}^{K-1}$
```

Remark 5.3. *In the classical SGD algorithm, the minibatches $\mathcal{B}$ are sampled uniformly randomly from the index set $\{1, 2, ..., n\}$ with fixed minibatch size $m \leq n$. This kind of minibatches can be regarded as* "sample without replacement" *minibatches. This is the commonly understood minibatches in practice.*

In the SPIDER algorithm, the samples $\mathcal{S}_1$ and $\mathcal{S}_2$ are drawn with replacement, so that they form minibatches $\mathcal{S}$ that are sampled uniformly randomly from the index set $\{1, 2, ..., n\}$ but with replacement, and the minibatch has a given size, say m but counting multiplicities. This kind of minibacthes can be regarded as "sample with replacement" *minibacthes. This is less seen in the literature.*

The reason why "sample with replacement" minibatches are used in the SPIDER algorithm is because of the following fact: consider estimating the variance of the stochastic estimator $\dfrac{1}{m}\sum\limits_{i\in\mathcal{S}} \nabla f_i(x)$ where $\mathcal{S}$ is the "sample with replacement" minibatch with size

m, then

$$\mathbf{E}\left\|\frac{1}{m}\sum_{i\in\mathcal{S}}\nabla f_i(x)-\nabla f(x)\right\|^2=\mathbf{E}\left\|\frac{1}{m}\sum_{i\in\mathcal{S}}(\nabla f_i(x)-\nabla f(x))\right\|^2=\frac{1}{m}\mathbf{E}\|\nabla f_i(x)-\nabla f(x)\|^2\ ,$$

where in the last expectation the variable i is taken uniformly randomly from $\{1,2,...,n\}$.

On the contrary, if $\mathcal{B}$ is the "sample without replacement" minibatch with the same size m, then instead we have

$$\begin{aligned}\mathbf{E}\left\|\frac{1}{m}\sum_{i\in\mathcal{B}}\nabla f_i(x)-\nabla f(x)\right\|^2 &= \mathbf{E}\left\|\frac{1}{m}\sum_{i\in\mathcal{B}}(\nabla f_i(x)-\nabla f(x))\right\|^2\\ &\leq \mathbf{E}\sum_{i\in\mathcal{B}}\frac{1}{m^2}\sum_{i\in\mathcal{B}}\|\nabla f_i(x)-\nabla f(x)\|^2=\mathbf{E}\|\nabla f_i(x)-\nabla f(x)\|^2\ .\end{aligned}$$

So we see a factor of $1/m$ is lost.

Previous mentioned methods typically use *non-adaptive learning rates* and *reliance on giant batch sizes* to construct variance reduced gradients through the use of low-noise gradients calculated at a "checkpoint".

STOchastic Recursive Momentum (STORM, Cutkosky, A., Orabona, F., Momentum-Based Variance Reduction in Non-Convex SGD, arXiv:1905.10018v2, 2020.) is a recently proposed novel variance-reduction method that achieves variance reduction through the use of a variant of the momentum term.

This method never needs to compute a checkpoint gradient and achieves the optimal convergence rate of $O(1/T^{1/3})$. In fact, the key ideas in momentum-based methods is to replace the gradient estimator with an exponential moving avearge of the gradient estimators in all previous iteration steps, i.e.

$$\begin{aligned}\boldsymbol{d}_t &= (1-a)\boldsymbol{d}_{t-1}+a\nabla f(\boldsymbol{x}_t,\xi_t)\ ,\\ \boldsymbol{x}_t &= \boldsymbol{x}_{t-1}-\eta\boldsymbol{d}_t\ ,\end{aligned}$$

where $\nabla f(x,\xi)$ is the usual stochastic gradient estimator. The STORM estimator incorporates ideas in SARAH estimator to modify the momentum updates as

$$\begin{aligned}\boldsymbol{d}_t &= (1-a)\boldsymbol{d}_{t-1}+a\nabla f(\boldsymbol{x}_t,\xi_t)+(1-a)(\nabla f(\boldsymbol{x}_t,\xi_t)-\nabla f(\boldsymbol{x}_{t-1},\xi_t))\ ,\\ \boldsymbol{x}_t &= \boldsymbol{x}_{t-1}-\eta\boldsymbol{d}_t\ .\end{aligned}$$

We can think of the momentum iteration as an "exponential moving average" version of SARAH, that is

$$\boldsymbol{d}_t=(1-a)\left[\boldsymbol{d}_{t-1}+(\nabla f(\boldsymbol{x}_t,\xi_t)-\nabla f(\boldsymbol{x}_{t-1},\xi_t))\right]+a\nabla f(\boldsymbol{x}_t,\xi_t)\ .$$

Here the first part is like SARAH update, but it is weighted with the stochastic gradient update. Moreover, different from the SARAH update, the STORM update does not have to compute checkpoint gradients. Rather, variance-reduction is achieved by tuning the weight parameter a appropriately. The exact algorithm goes as follows:

Algorithm 5.7 STORM: STOchastic Recursive Momentum

```
1: **Input**: Parameters $k, w, c$, initial point $\boldsymbol{x}_1$
2: Sample $\xi_1$
3: $G_1 \leftarrow \|\nabla f(\boldsymbol{x}_1, \xi_1)\|$
4: $\boldsymbol{d}_1 \leftarrow \nabla f(\boldsymbol{x}_1, \xi_1)$
5: $\eta_0 \leftarrow k/w^{1/3}$
6: **for** $t = 1, 2, ..., T$ **do**
7:     $\eta_t \leftarrow \frac{k}{(w + \sum_{i=1}^{t} G_i^2)^{1/3}}$
8:     $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta_t \boldsymbol{d}_t$
9:     $a_{t+1} \leftarrow c\eta_t^2$
10:    Sample $\xi_{t+1}$
11:    $G_{t+1} \leftarrow \|\nabla f(\boldsymbol{x}_{t+1}, \xi_{t+1})\|$
12:    $\boldsymbol{d}_{t+1} \leftarrow (1-a_{t+1})\boldsymbol{d}_t + a_{t+1}\nabla f(\boldsymbol{x}_{t+1}, \xi_{t+1}) + (1-a_{t+1})[\nabla f(\boldsymbol{x}_{t+1}, \xi_{t+1}) - \nabla f(\boldsymbol{x}_t, \xi_{t+1})]$
13: **end for**
14: **Output**: $\widehat{\boldsymbol{x}}$ sampled uniformly randomly from $\boldsymbol{x}_1, ..., \boldsymbol{x}_T$ (In practice, set $\widehat{\boldsymbol{x}} = \boldsymbol{x}_T$)
```

5.3 Escape from saddle points and nonconvex problems, SGLD, Perturbed GD

Continuous point of view:

Hu, W., Li, C.J., Li, L., Liu, J., On the diffusion approximation of nonconvex stochastic gradient descent. *Annals of Mathematical Science and Applications*, Vol. **4**, No. 1 (2019), pp. 3-32.

SGLD: Maxim Raginsky, Alexander Rakhlin, Matus Telgarsky, Non-convex learning via Stochastic Gradient Langevin Dynamics: a nonasymptotic analysis, *Proceedings of the 2017 Conference on Learning Theory, PMLR* **65**:1674-1703, 2017.

SGLD is a continuous version of SGD with additional moderate and isotropic noise.

Convergence Guarantee: Xu, P., Chen, J., Zou, D., Gu, Q., Global convergence of Langevin Dynamics Based Algorithms for Nonconvex Optimization, *NIPS*, 2018.

Algorithm 5.8 Stochastic Gradient Langevin Dynamics: minimizing $P(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$

```
1: Input: Input initialization $w_0 \in \mathbb{R}^d$, stepsize $\alpha > 0$, inverse temperature $\beta > 0$
2: for $k = 0, 1, ..., K-1$ do
3:     Sample a random minibatch $B \subset \{1, 2, ..., n\}$
4:     Calculate $\nabla f_i(w_k)$ for $i \in B$
5:     Sample i.i.d. $\varepsilon_k \sim \mathcal{N}(0, I_d)$
6:     $w_{k+1} = w_k - \frac{\eta}{|B|}\sum_{i\in B} \nabla f_i(w_k) + \sqrt{2\eta\beta^{-1}}\varepsilon_k$
7: end for
8: Output: $w_K$
```

Perturbed GD: Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M. Kakade, Michael I. Jordan, How to Escape Saddle Points Efficiently, *ICML* 2017.

Perturbed GD has guaranteed escape rate from saddle points.

Algorithm 5.9 Perturbed Gradient Descent: PGD$(\boldsymbol{x}_0, \ell, \rho, \varepsilon, c, \delta, \Delta_f)$

```
1: Input: Input Initialization $\boldsymbol{x}_0$.
2: Parameters $\chi \leftarrow 3\max\left\{\ln\left(\frac{d\ell\Delta_f}{c\varepsilon^2\delta}\right), 4\right\}$, $\eta \leftarrow \frac{c}{\ell}$, $r \leftarrow \frac{\sqrt{c}}{\chi^2}\cdot\frac{\varepsilon}{\ell}$, $g_{\text{thres}} \leftarrow \frac{\sqrt{c}}{\chi^2}\cdot\varepsilon$, $f_{\text{thres}} \leftarrow$
   $\frac{c}{\chi^3}\cdot\sqrt{\frac{c^3}{\rho}}$, $t_{\text{thres}} \leftarrow \frac{\chi}{c^2}\cdot\frac{\ell}{\sqrt{\rho\varepsilon}}$, $t_{\text{noise}} \leftarrow -t_{\text{thres}} - 1$.
3: for $t = 0, 1, ..., K-1$ do
4:     if $\|\nabla f(\boldsymbol{x}_t)\| \le g_{\text{thres}}$ and $t - t_{\text{noise}} > t_{\text{thres}}$ then
5:         $\widetilde{\boldsymbol{x}}_t \leftarrow \boldsymbol{x}_t$, $t_{\text{noise}} \leftarrow t$
6:         $\boldsymbol{x}_t \leftarrow \widetilde{\boldsymbol{x}}_t + \xi_t$, $\xi_t$ uniformly $\sim \mathbb{B}_0(r)$
7:     end if
8:     if $t - t_{\text{noise}} = t_{\text{thres}}$ and $f(\boldsymbol{x}_t) - f(\widetilde{\boldsymbol{x}}_{t_{\text{noise}}}) > -f_{\text{thres}}$ then
9:         RETURN $\widetilde{\boldsymbol{x}}_{t_{\text{noise}}}$
10:    end if
11:    $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta\nabla f(\boldsymbol{x}_t)$
12: end for
13: Output: $\boldsymbol{x}_K$
```

5.4 Coordinate Descent and Stochastic Coordinate Descent

References:

Wright, S.J., Coordinate Descent Algorithms, *Mathematical Programming*, 2015, **151**(1): 3–34.

Nesterov, Y., Efficiency of Coordinate Descent Methods on Huge–Scale Optimization Problems, *SIAM Journal on Optimization*, 2012, **22**(2): 341.

The k–th iteration of Coordinate Descent applied to a function $f : \mathbb{R}^n \to \mathbb{R}$ chooses some index $i_k \in \{1, 2, ..., n\}$ and takes iteration

$$x^{k+1} = x^k + \gamma_k e_{i_k} \ ,$$

where e_{i_k} is the i_k–th unit vector and γ_k is the step.

Example 1. Gauss–Seidel method:

$$\gamma_k = \arg\min_{\gamma} f(x^k + \gamma e_{i_k}) \ .$$

Example 2. Coordinate Gradient Descent method:

$$\gamma_k = -\alpha_k \frac{\partial f}{\partial x_{i_k}}(x^k) e_{i_k}$$

for some $\alpha_k > 0$.

Example 3. If i_k is chosen randomly the method is Stochastic Coordinate Descent.

Descent methods need $f(x^{k+1}) < f(x^k)$ for all k.

6 Second Order Methods

6.1 Newton's method

Second order methods: Produce an optimization sequence $\{x^k\}$ using second–order Taylor expansions (quadratic surrogate)

$$\min_{x\in\mathcal{U}(x^k)} f(x) \approx \min_{x\in\mathcal{U}(x^k)} \left[f(x^k) + \nabla f(x^k)^T(x-x^k) + \frac{1}{2}(x-x^k)^T\nabla^2 f(x^k)(x-x^k) \right] ,$$

where $\mathcal{U}(x^k)$ is a convex open neighborhood of x^k.

If the Hessian $\nabla^2 f(x^k)$ is invertible, the minimum is taken at $x^k - [\nabla^2 f(x^k)]^{-1}\nabla f(x^k)$.

Newton's iteration:

$$x^{k+1} = x^k - [\nabla^2 f(x^k)]^{-1}\nabla f(x^k) \ . \tag{6.1}$$

Algorithm 6.1 Newton's method

```
1: Input: Input Initialization $x^0$.
2: for $k = 0, 1, ..., K-1$ do
3:     Calculate the gradient $\nabla f(x^k)$
4:     Calculate the Hessian $\nabla^2 f(x^k)$ and the inverse $[\nabla^2 f(x^k)]^{-1}$
5:     Update $x^{k+1} = x^k - [\nabla^2 f(x^k)]^{-1}\nabla f(x^k)$
6: end for
7: Output: $x^K$
```

Convergence of Newton's method.

Theorem 6.1 (Local superlinear convergence of the Newton's method)**.** *Suppose that the function f is twice differentiable and the Hessian $\nabla^2 f(x)$ is Lipschitz–continuous with Lipschitz constant $M > 0$, i.e.*

$$\|\nabla^2 f(x) - \nabla^2 f(y)\| \le M\|x-y\| \ .$$

Let x^* be a second–order stationary point in the sense of Theorem 2.7 (b), then if the starting point x_0 is sufficiently close to x^*, then the sequence of iterates from Newton's method (6.1) converges to x^* at a superlinear rate (in particular, this rate is quadratic).

Proof. See the proof of Theorem 3.5 of the textbook: Nocedal, J., Wright, S.J., *Numerical Optimization*, Second Edition, Springer, 2006. □

Drawbacks of Newton's method: (1) Inverse of Hessian $[\nabla^2 f(x)]^{-1}$ hard to calculate for high–dimensional problems; (2) No positive definite guarantee of Hessian $\nabla^2 f(x)$.

6.2 Quasi–Newton: BFGS

Reference: J. E. Dennis, Jr., Jorge J. Moré, Quasi–Newton methods, motivation and theory, *SIAM Review*, Vol. **19**, No. 1 (Jan. 1977), pp. 46–89.

Quasi–Newton method: If the Hessian $\nabla^2 f(x^k)$ is not positive definite or it is hard to calculate, we need to replace it with a positive definite matrix B_k that approximates $\nabla^2 f(x^k)$: $B_k \approx \nabla^2 f(x^k)$. Or we update the inverse of the Hessian matrix $H_k \approx [\nabla^2 f(x^k)]^{-1}$ iteratively, so that we do not have to recalculate the approximation of the inverse of the Hessian matrix $[\nabla^2 f(x^k)]^{-1}$ at each iteration.

Set $B_k = \nabla^2 f(x^k)$. Quasi–Newton condition uses quadratic surrogate around each two consecutive iterations x^k and x^{k+1}, which produces an iteration that calculates B_{k+1} from current iteration. In fact we have

$$f(x^k) \approx f(x^{k+1}) + \nabla f(x^{k+1})^T(x^k - x^{k+1}) + \frac{1}{2}(x^k - x^{k+1})^T \nabla^2 f(x^{k+1})(x^k - x^{k+1})$$

gives

$$\nabla f(x^k) \approx \nabla f(x^{k+1}) + B_{k+1}(x^k - x^{k+1}) \ .$$

Set $y_k = \nabla f(x^{k+1}) - \nabla f(x^k)$ and $s_k = x^{k+1} - x^k$. Then

$$B_{k+1} s_k = y_k \ . \tag{6.2}$$

Equation (6.2) is called the *secant equation*. The secant equation will produce a next iteration B_{k+1} from current iteration, and the update rule from B_k to B_{k+1} should always satisfy the following criteria: (1) it satisfies the secant equation; (2) it keeps symmetry and positive definiteness of B_{k+1}; (3) it is a low rank modification from B_k to B_{k+1}.

The positive definiteness can be directly obtained from the secant equation since we have from (6.2) that $s_k^T B_{k+1} s_k = s_k^T y_k > 0$. This give the *curvature condition*: for any $x, y \in \mathrm{Dom}(f)$ we have

$$(x-y)^T(\nabla f(x) - \nabla f(y)) > 0 \ . \tag{6.3}$$

It can be shown that when f is strongly convex then (6.3) is satisfied. This is left as one problem in Homework 4.

Secant equation is solved by doing minimization such as

$$\min_B \|B - B_k\| \text{ subject to } B = B^T \ , \ Bs_k = y_k \ .$$

Choosing appropriate norm, solution is

DFP (Davidson–Fletcher–Powell) update

$$B_{k+1} = (I - \rho_k y_k s_k^T) B_k (I - \rho_k s_k y_k^T) + \rho_k y_k y_k^T \ , \ \rho_k = \frac{1}{y_k^T s_k} \ . \tag{6.4}$$

Sherman–Morrison–Woodbury formula

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1} u} \ , \ \text{where } u, v \text{ are column vectors} \ .$$

Thus we get (try it!) the DFP iteration for $H_k = B_k^{-1}$ as

$$H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \frac{s_k s_k^T}{y_k^T s_k} \ . \tag{6.5}$$

In a same way but conversely, we have the secant equation for H:

$$H_{k+1} y_k = s_k \ . \tag{6.6}$$

Secant equation is solved by doing minimization such as

$$\min_H \|H - H_k\| \text{ subject to } H = H^T \ , \ H y_k = s_k \ .$$

BFGS (Broyden–Fletcher–Goldfarb–Shanno) update

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T \ , \ \rho_k = \frac{1}{s_k^T y_k} \ . \tag{6.7}$$

Invert it we get

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \ . \tag{6.8}$$

Algorithm 6.2 Quasi–Newton: BFGS method

```
1: Input: Input Initialization $x^0$, $\nabla f(x^0)$, $H_0 = [\nabla^2 f(x^0)]^{-1}$, $B_0 = I$, learning rate $\eta > 0$
2: for $k = 0, 1, ..., K-1$ do
3:     Calculate $x^{k+1} = x^k - \eta H_k \nabla f(x^k)$
4:     Calculate $y_k = \nabla f(x^{k+1}) - \nabla f(x^k)$ and $s_k = x^{k+1} - x^k$
5:     Update $H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$
6:     Update $B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$
7: end for
8: Output: $x^K$
```

Superlinear convergence of BFGS method (see Theorem 6.6 in Nocedal, J., Wright, S.J., *Numerical Optimization*, Second Edition, Springer 2006.).

6.3 Stochastic Quasi–Newton

Reference: Byrd, R.H., Hansen, S.L., Nocedal, J. et al, A stochastic quasi–Newton method for large–scale optimization, *SIAM Journal on Optimization*, 2016, 26(2): 1008–1031.

Large–scale optimization

$$F(w) = \frac{1}{N}\sum_{i=1}^{N} f_i(w) \ .$$

Stochastic gradient estimate: $b = |\mathcal{S}| \ll N$, randomly $\mathcal{S} \subset \{1, 2, ..., N\}$, take

$$\hat{\nabla} F(w) = \frac{1}{b}\sum_{i\in\mathcal{S}} \nabla f_i(w) \ .$$

It is necessary to employ a limited memory variant that is scalable in the number of variables, but enjoys only a linear rate of convergence.

Quasi–Newton updating is inherently an overwriting process rather than an averaging process, and therefore the vector $y = \nabla F(w^{k+1}) - \nabla F(w^k)$ must reflect the action of the Hessian of the entire objective F, which is not achieved by differencing stochastic gradients based on small samples.

To achieve stable Hessian approximation one needs to *decouple* the stochastic gradient and curvature estimation calculations.

Algorithm 6.3 Stochastic Quasi–Newton Method

```
1: Input: Input Initialization $w^1$, positive integers $M, L$, and step–length sequence $\alpha^k > 0$
2: Set $t = -1$                      ▷ Records number of correction pairs currently computed
3: $\bar{w}_t = 0$
4: for $k = 1, 2, ..., K-1$ do
5:     Choose a sample $\mathcal{S} \subset \{1, 2, ..., N\}$
6:     Calculate the stochastic gradient $\hat{\nabla} F(w^k) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \nabla f_i(w)$
7:     $\bar{w}_t = \bar{w}_t + w^k$
8:     if $k \leq 2L$ then
9:         $w^{k+1} = w^k - \alpha^k \hat{\nabla} F(w^k)$                ▷ Stochastic gradient iteration
10:    else
11:        $w^{k+1} = w^k - \alpha^k H_t \hat{\nabla} F(w^k)$, where $H_t$ is defined by Algorithm 6.4
12:    end if
13:    if $\mod(k, L) = 0$ then
14:        $t = t + 1$
15:        $\bar{w}_t = \bar{w}_t / L$
16:        if $t > 0$ then
17:            Choose a sample $\mathcal{S}_H \subset \{1, 2, ..., N\}$ to define $\hat{\nabla}^2 F(\bar{w}_t) \equiv \frac{1}{|\mathcal{S}_H|} \sum_{i \in \mathcal{S}_H} \nabla^2 f(w)$
18:            Compute $s_t = \bar{w}_t - \bar{w}_{t-1}$, $y_t = \hat{\nabla}^2 F(\bar{w}_t)(\bar{w}_t - \bar{w}_{t-1})$
                                                  ▷ Compute correction pairs every $L$ iterations
19:        end if
20:        $\bar{w}_t = 0$
21:    end if
22: end for
23: Output: $w^K$
```

Algorithm 6.4 Stochastic Quasi–Newton Method: Hessian Updating

```
1: **Input**: Updating counter $t$, memory parameter $M$, correction pairs $(s_j, y_j)$, $j = t - \widetilde{m} + 1, ..., t$ where $\widetilde{m} = \min\{t, M\}$
2: Set $H = \frac{s_t^T y_t}{y_t^T y_t} I$
3: **for** $j = t - \widetilde{m} + 1, ..., t$ **do**
4:     $\rho_j = \frac{1}{y_j^T s_j}$
5:     Apply BFGS formula
           $$H \leftarrow (I - \rho_j s_j y_j^T) H (I - \rho_j y_j s_j^T) + \rho_j s_j s_j^T$$
6: **end for**
7: **Output**: new matrix $H_t = H$
```

7 Dual Methods

7.1 Dual Coordinate Descent/Ascent

Reference: Chapter 13 & 14 of Nocedal, J., Wright, S.J., *Numerical Optimization*, Second Edition, Springer, 2006.

Constraint optimization problem P_0:

$$\min_w f(w)$$
$$\text{s.t. } g_i(w) \leq 0 \ , \ i = 1, 2, ..., m_1 \ , \ h_j(w) = 0 \ , \ j = 1, 2, ..., m_2 \ .$$

Let the optimal solution be p^*.

Unconstrained Version:

$$\min_w \left(f(w) + \infty \sum_{i=1}^{m_1} \mathbf{1}_{[g_i(w)>0]} + \infty \sum_{j=1}^{m_2} \mathbf{1}_{[h_j(w)\neq 0]} \right) \ .$$

Approximate the indicators by linear functions, obtain the Lagrangian function

$$L(w, \lambda, \nu) \equiv f(w) + \sum_{i=1}^{m_1} \lambda_i g_i(w) + \sum_{j=1}^{m_2} \nu_j h_j(w) \ ,$$

in which $\lambda_i \geq 0$ and $\nu_j \in \mathbb{R}$ are the *Lagrange multipliers.* The variable w is called the *primal* variable.

Set the *Lagrangian dual function*

$$h(\lambda, \nu) \equiv \inf_{\text{acceptable } w} L(w, \lambda, \nu) \ , \ \lambda_i \geq 0 \text{ and } \nu_j \in \mathbb{R} \ .$$

Since for any acceptable w we have $g_i(w) \leq 0$ and $h_i(w) = 0$, so we have

$$h(\lambda, \nu) \leq \inf_{\text{acceptable } w} f(w) = p^* \ .$$

The variables (λ, ν) are called *dual* variables.

Observation: $h(\lambda, \nu)$ is always concave (even when the original problem P_0 is not convex) in the dual variables λ and ν.

Dual Problem D_0 ("primal-dual"):

$$\max_{\lambda,\nu} h(\lambda, \nu)$$
$$\text{s.t. } \lambda_i \geq 0 \ , \ i = 1, 2, ..., m \ .$$

Let the optimal solution be d^*. Then $d^* \leq p^*$.

If $d^* = p^*$ this is called *strong duality.* A sufficient condition for strong duality is Slater condition and a necessary condition for strong duality is KKT condition.

Set $d^* = h(\lambda^*, \nu^*)$. Then solve the unconstrained optimization problem

$$\min_w L(w, \lambda^*, \nu^*) \ .$$

If the solution is acceptable then it is the optimal solution for the original problem.

Algorithm 7.1 Dual Coordinate Ascent

```
1: Input: Input Initialization $v_0$, $\lambda_0 \geq 0$
2: for $k = 0, 1, ..., K-1$ do
3:     Calculate $w_k$ such that $w_k = \arg\min_w L(w, \lambda_{k-1}, v_{k-1})$
4:     Update $\lambda_{k,i} = \max\{\lambda_{k-1,i} + \eta_k g_i(w_k), 0\}$, $i = 1, 2, ..., m_1$
5:     Update $v_{k,j} = v_{k-1,j} + \eta_k h_j(w_k)$, $j = 1, 2, ..., m_2$
6: end for
7: Output: $w_K$
```

7.2 SDCA

Shalev–Schwarz, S., Zhang, T., Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization, *Journal of Machine Learning Research*, **14**(2013), pp. 567–599.

Let $x_1, ..., x_n$ be vectors in $\mathbb{R}^d$ and $\phi_1, ..., \phi_n$ be a sequence of scalar convex functions, and let $\lambda > 0$ be a regularization parameter. Our goal is to solve the optimization problem $P(w^*) = \min_{w \in \mathbb{R}^d} P(w)$ where

$$P(w) = \frac{1}{n} \sum_{i=1}^{n} \phi_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2 \ .$$

Here $\phi_i(w^T x_i)$ is the loss function for linear models. For example, given labels $y_1, ..., y_n$ in $\{\pm 1\}$, the SVM problem (with linear kernels and no bias term) is obtained by setting $\phi_i(a) = \max\{0, 1 - y_i a\}$; regularized logistic regression is obtained by setting $\phi_i(a) = \log(1 + \exp(-y_i a))$; ridge regression is obtained by setting $\phi_i(a) = (a - y_i)^2$; regression with the absolute–value is obtained by setting $\phi_i(a) = |a - y_i|$; support vector regression is obtained by setting $\phi_i(a) = \max\{0, |a - y_i| - \nu\}$ for some predefined insensitivity parameter $\nu > 0$.

Dual formulation: $D(\alpha^*) = \max_{\alpha \in \mathbb{R}^d} D(\alpha)$ where

$$D(\alpha) = \frac{1}{n} \sum_{i=1}^{n} -\phi_i^*(-\alpha_i) - \frac{\lambda}{2} \left\| \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i x_i \right\|^2$$

where $\phi_i^*(u) = \max_z (zu - \phi_i(z))$.

In fact we have

$$\begin{aligned}
\min_w P(w) &= \min_w \left[\frac{1}{n}\sum_{i=1}^n(\phi_i(w^Tx_i)+\alpha_i(w^Tx_i))+\frac{\lambda}{2}\left(\|w\|^2-\frac{2}{\lambda n}\sum_{i=1}^n\alpha_i w^Tx_i\right)\right]\\
&= \min_w \left[\frac{1}{n}\sum_{i=1}^n(\phi_i(w^Tx_i)+\alpha_i(w^Tx_i))+\frac{\lambda}{2}\left\|w-\frac{1}{\lambda n}\sum_{i=1}^n\alpha_i x_i\right\|^2-\frac{\lambda}{2}\left\|\frac{1}{\lambda n}\sum_{i=1}^n\alpha_i x_i\right\|^2\right]\\
&\geq \min_w \left[\frac{1}{n}\sum_{i=1}^n(\phi_i(w^Tx_i)+\alpha_i(w^Tx_i))\right]-\frac{\lambda}{2}\left\|\frac{1}{\lambda n}\sum_{i=1}^n\alpha_i x_i\right\|^2\\
&\geq \frac{1}{n}\sum_{i=1}^n\min_\omega(\phi_i(w^Tx_i)+\alpha_i(w^Tx_i))-\frac{\lambda}{2}\left\|\frac{1}{\lambda n}\sum_{i=1}^n\alpha_i x_i\right\|^2\\
&\geq \frac{1}{n}\sum_{i=1}^n-\max_z(-\phi_i(z)-\alpha_i z)-\frac{\lambda}{2}\left\|\frac{1}{\lambda n}\sum_{i=1}^n\alpha_i x_i\right\|^2\\
&= \frac{1}{n}\sum_{i=1}^n-\phi_i^*(-\alpha_i)-\frac{\lambda}{2}\left\|\frac{1}{\lambda n}\sum_{i=1}^n\alpha_i x_i\right\|^2 = D(\alpha)
\end{aligned}$$

for any $\alpha \in \mathbb{R}^d$.

So $\min_w P(w) \geq \max_\alpha D(\alpha)$. Set $w(\alpha) = \frac{1}{\lambda n}\sum_{i=1}^n \alpha_i x_i$, then strong duality holds and

$$w(\alpha^*) = w^*\ ,\ P(w^*) = D(\alpha^*)\ .$$

Algorithm 7.2 Stochastic Dual Coordinate Ascent

```
1: Input: Input Initialization $\alpha_0$, $w_0 = w(\alpha_0)$
2: for $k = 0, 1, ..., K-1$ do
3:     Randomly sample $i_k \in \{1, 2, ..., n\}$
4:     Find $\Delta\alpha_{i_k} = \arg\max_z \left\{-\phi_{i_k}^*(-\alpha_{k,i}+z) - \frac{\lambda n}{2}\left\|w_k + \frac{1}{\lambda n}zx_{i_k}\right\|^2\right\}$
5:     Update $\alpha_{k+1} = \alpha_k + \Delta\alpha_{i_k}e_{i_k}$ , $w_{k+1} = w_k + \frac{1}{\lambda n}\Delta\alpha_{i_k}x_{i_k}$
6: end for
7: Output (Average Option): $\bar{\alpha} = \frac{1}{K-K_0}\sum_{i=K_0+1}^K \alpha_i$ and $\bar{w} = w(\bar{\alpha}) = \frac{1}{K-K_0}\sum_{i=K_0+1}^K w_{k-1}$, return $\bar{w}$
8: Output (Random Option): Let $\bar{\alpha} = \alpha_k$ and $\bar{w} = w_k$ for some random $k \in K_0+1, ..., K$, return $\bar{w}$
```

8 Optimization with Constraints

8.1 Subgradient Projection Method, Frank–Wolfe Method

References:

Levitin, E.S., Polyak, B.T., Constrained Minimization Methods, *USSR Computational Mathematics and Mathematical Physics*, 1966, **6**(5), pp. 1–50.

Frank, M., Wolfe, P., An algorithm for quadratic programming, *Naval Research Logistics (NRL)*, 1956, **3**(1–2), pp. 95–110.

The subgradient $\partial f(x_0)$ for a convex function at point x_0 is defined as

$$\partial f(x_0) = \{c \in \mathbb{R} : f(x) - f(x_0) \geq c(x - x_0)\} \ .$$

Algorithm 8.1 Subgradient Projection Method

```
1: Input: Input Initialization $x_0$; Convex Set $\mathcal{X}$; learning rate $\eta > 0$
2: for $k = 0, 1, ..., K-1$ do
3:     Randomly sample a subgradient $g_k \in \partial f(x_k)$
4:     Update $y_{k+1} = x_k - \eta g_k$
5:     Project $x_{k+1} = \Pi_{\mathcal{X}}(y_{k+1})$, $\Pi_{\mathcal{X}}(y) = \arg\min_{x \in \mathcal{X}} \|y - x\|$
6: end for
7: Output: $x_K$
```

The projection $\Pi_{\mathcal{X}}(y)$ can be hard to calculate in practical cases. However, subgradient projection = first gradient descent then projection; Frank–Wolfe = combine gradient descent with projection to constraint.

How to do that? GD is an optimal choice of search direction via first–order Taylor expansion:

$$f(x + td) = f(x) + t\nabla f(x + \gamma td)^T d \text{ for some } \gamma \in (0, 1) \ .$$

If $d^T \nabla f(x) < 0$, then d is a descent direction.

"Steepest Descent": $d_{\text{optimal}} = \arg \inf_{\|d\|=1} d^T \nabla f(x) = -\dfrac{\nabla f(x)}{\|\nabla f(x)\|}$.

Want to incorporate the constraint information: how to do that?

$$d_{\text{optimal}} = \arg \inf_{d \in \mathcal{X}} d^T \nabla f(x) \ .$$

Algorithm 8.2 Frank–Wolfe Method

```
1: Input: Input Initialization $x_0 \in \mathcal{X}$; Convex Set $\mathcal{X}$; weights $\eta_k > 0$
2: for $k = 0, 1, ..., K-1$ do
3:     Calculate $\nabla f(x_k)$
4:     Update $d_k = \arg\inf_{d \in \mathcal{X}} d^T \nabla f(x_k)$
5:     Update $x_{k+1} = (1-\gamma_k)x_k + \gamma_k d_k$
6: end for
7: Output: $x_K$
```

8.2 ADMM

Reference: Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers, *Foundations and Trends in Machine Learning*, Vol. 3, No. 1 (2010), pp. 1–122.

Equality–constrained convex optimization problem:

$$\text{minimize } f(x) \text{ subject to } Ax = b \ .$$

Augmented Lagrangian.

$$L_\rho(x,y) = f(x) + y^T(Ax-b) + \frac{\rho}{2}\|Ax-b\|_2^2 \ .$$

The augmented Lagrangian can be viewed as the (unaugmented) Lagrangian associated with the problem

$$\text{minimize } f(x) + \frac{\rho}{2}\|Ax-b\|_2^2 \text{ subject to } Ax = b \ .$$

This problem can be solved by dual gradient ascent:

$$\begin{aligned} x^{k+1} &\equiv \arg\min_x L(x, y^k) \ , \\ y^{k+1} &\equiv y^k + \rho(Ax^{k+1} - b) \ . \end{aligned}$$

Alternating Direction Method of Multipliers (ADMM).

Optimization problem has the form

$$\text{minimize } f(x) + g(z) \text{ subject to } Ax + Bz = c \ .$$

Optimal value

$$p^* = \inf\{f(x) + g(z); Ax + Bz = c\} \ .$$

Augmented Lagrangian can be formed as

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2 \ .$$

This problem can be solved by dual gradient ascent:

$$\begin{aligned}(x^{k+1}, z^{k+1}) &\equiv \arg\min_{x,z} L_\rho(x, z, y^k) \ , \\ y^{k+1} &\equiv y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \ .\end{aligned}$$

Unlike the classical dual gradient ascent which minimizes the augmented Lagrangian jointly with respect to the two primal variables x and z, ADMM updates x and z in an alternating or sequential fashion, which accounts for the term *alternating direction.*

Algorithm 8.3 Alternating Direction Method of Multipliers

```
1: Input: Input Initialization $(x^0, z^0)$, dual variable $y^0$, parameter $\rho > 0$
2: for $k = 0, 1, ..., K-1$ do
3:     Calculate $x^{k+1} \equiv \arg\min_x L_\rho(x, z^k, y^k)$
4:     Calculate $z^{k+1} \equiv \arg\min_z L_\rho(x^{k+1}, z, y^k)$
5:     Calculate $y^{k+1} = y^k + \rho(Ax^{k+1} + Bz^{k+1} - c)$
6: end for
7: Output: $(x^K, z^K)$
```

9 Adaptive Methods

Adapted algorithms are those training algorithms that update the parameters under training using not only the current gradient information but also all past gradient information, as well as adaptively change the learning rate. These algorithms are sometimes also called to be belonging to the Ada–Series.

9.1 SGD with momentum

Reference: Sutskever, I., Martens, J., Dahl, G., Hinton, G., On the importance and momentum in deep learning, *ICML*, 2013.

Algorithm 9.1 SGD with momentum (= stochastic heavy ball method)

```
1: Input: Input Initialization $w_0 = w_{-1}$ and $v_{-1} = 0$
2: for $k = 0, 1, ..., K - 1$ do
3:     Randomly sample minibatch $\mathcal{S}_i \subset \{1, 2, ..., n\}$ such that $|\mathcal{S}_i| = m$
4:     Update $v_{k+1} = \mu v_k - \eta \frac{1}{m} \sum_{i \in \mathcal{S}} \nabla f_i(w_k)$
5:     Update $w_{k+1} = w_k + v_{k+1}$
6: end for
7: Output: $w_K$
```

9.2 AdaGrad, RMSProp, AdaDelta

Reference:

Duchi, J., Hazan, E., Singer, Y., Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, *Journal of Machine Learning Research*, 2011, **12**(July), pp. 2121–2159.

AdaGrad not just accumulate historical information (such as square gradient information) but also adjust the learning rate (stepsize) according to the magnitude of historical gradient information at different dimensions, so that those dimensions with relatively small magnitudes of the gradient have larger learning rates. In this way, the algorithm favors some dimensions that have currently small magnitude of the gradient but contributes importantly to the optimization search path. This is more suitable to optimize neural network models that possess many local minimizers and saddle points. Empirical evidence shows that AdaGrad can stabilize SGD. The disadvantage of AdaGrad comes from the term $\sqrt{g_{k+1}}$ in the denominator, since as it accumulates during the training process the learning rates will gradually tend to 0 and this leads to early stopping.

Algorithm 9.2 AdaGrad

```
1: Input: Input Initialization $w_0$ and $g_0 = 0$
2: for $k = 0, 1, ..., K - 1$ do
3:     Update $g_{k+1} = g_k + (\nabla f(w_k))^2$
4:     Update $w_{k+1} = w_k - \frac{\eta \nabla f(w_k)}{\sqrt{g_{k+1}} + \varepsilon_0}$, $\varepsilon_0 > 0$
5: end for
6: Output: $w_K$
```

Tieleman, T., Hinton, G., Lecture 6. 5–RmsProp: Divide the Gradient by A Running Average of its Recent Magnitude, *COURSERA, Neural Networks for Machine Learning*, 2012, **4**(2), pp. 26–31.

RMSProp is similar to AdaGrad but also makes use of the idea of SGD with momentum, this gives an exponential weight–decay average of historical information. RMSProp does a weighted average of both g_k and $(\nabla f(w_k))^2$, so that the weights in historical information will be small and thus it favors more on current gradient information. This avoids the monotonic decay of the learning rate (stepsize) along with the training process.

Algorithm 9.3 RMSProp

```
1: Input: Input Initialization $w_0$ and $g_0 = 0$
2: for $k = 0, 1, ..., K - 1$ do
3:     Update $g_{k+1} = \gamma g_k + (1 - \gamma)(\nabla f(w_k))^2$
4:     Update $w_{k+1} = w_k - \frac{\eta \nabla f(w_k)}{\sqrt{g_{k+1}} + \varepsilon_0}$, $\varepsilon_0 > 0$
5: end for
6: Output: $w_K$
```

Zeiler, M.D., AdaDelta: An Adaptive Learning Rate Method, arXiv:1212.5701, 2012.

AdaDelta, based on RMSProp, adjust the accumulated historical information according to the magnitude of gradients.

Algorithm 9.4 AdaDelta

```
1: Input: Input Initialization $w_0$ and $u_0 = 0$, $v_0 = 0$ and $g_0 = 0$
2: for $k = 0, 1, ..., K-1$ do
3:     Update $g_{k+1} = \gamma g_k + (1-\gamma)(\nabla f(w_k))^2$
4:     Update $v_{k+1} = \gamma v_k + (1-\gamma)u_k^2$
5:     Update $u_{k+1} = -\frac{\sqrt{v_k + \varepsilon_0}\nabla f(w_k)}{\sqrt{g_{k+1}} + \varepsilon_0}$
6:     Update $w_{k+1} = w_k + u_{k+1}$
7: end for
8: Output: $w_K$
```

9.3 Adam

Reference:

Kingma, D.P., Ba, J.L., ADAM: A method for stochastic optimization, *ICLR*, 2015.

Reddi, S.J., Kale, S., Kumar, S., On the convergence of ADAM and Beyond, *ICLR*, 2018.

Adam combines all ingredients in SGD–momentum, AdaGrad, RMSProp, AdaDelta: (1) The update direction is determined by accumulated historical gradient information; (2) Correct the stepsize using accumulated square gradient information; (3) Exponential decay of accumulated information. Adam has been widely used in training neural networks.

Algorithm 9.5 Adam

```
1: Input: $\alpha$: stepsize; $\beta_1, \beta_2 \in [0,1)$: Exponential decay rates for the moment estimates; $f(\theta)$: stochastic objective function with parameters $\theta$; $\theta_0$: initial parameter vector; $m_0 \leftarrow 0$ (initialize first moment vector); $v_0 \leftarrow 0$ (initialize second moment vector); $t \leftarrow 0$ (initialize timestep)
2: while $\theta_t$ not converged do
3:     $t \leftarrow t + 1$
4:     $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
5:     $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ (Update biased first moment estimate)
6:     $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
7:     $\hat{m}_t \leftarrow m_t/(1-\beta_1^t)$ (Compute bias–corrected first moment estimate)
8:     $\hat{v}_t \leftarrow v_t/(1-\beta_2^t)$ (Compute bias–corrected second raw moment estimate)
9:     $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \varepsilon)$ (Update parameters)
10: end while
11: Output: $\theta_t$ (Resulting parameters)
```

10 Optimization with Discontinuous Objective Function

10.1 Proximal operators and proximal algorithms

Reference: Parikh, N., Boyd, S., Proximal Algorithms, *Foundations and Trends in Optimization*, Vol. 1, No. 3 (2013), pp. 123–231.

Also see http://stanford.edu/~boyd/papers/pdf/prox_slides.pdf

Let $R = R(v)$ be a convex function. With respect to the convex function R the proximal operator $\text{prox}_R(w) : \mathbb{R}^d \to \mathbb{R}^d$ is defined as

$$\text{prox}_R(w) = \arg\min_v \left(R(v) + \frac{1}{2}\|w - v\|^2 \right) .$$

If $R(w) = 0$, then $\text{prox}_0(w) = w$; if $R(w) = aI_C(w)$ (where $C \subseteq \mathbb{R}^d$ is usually a convex set, and the indicator function is defined as in (2.5)), then $\text{prox}_{aI_C}(w) = \arg\min\limits_{v\in C} \|w - v\|^2$ degenerates to a projection operator.

We have

$$p = \text{prox}_R(w) \Longleftrightarrow w - p \in \partial R(p) .$$

Also

$$\text{prox}_{aR}(w) = \arg\min_v \left(R(v) + \frac{1}{2a}\|w - v\|^2 \right) .$$

Convex optimization problem

$$\min_{w\in\mathcal{W}} f(w) = l(w) + R(w) ,$$

where $l(w)$ is a convex differentiable function and $R(w)$ is non–differentiable convex function (such as L_1–regularization term).

Algorithm 10.1 Proximal Gradient Descent

```
1: Input: Input Initialization $w_0$, convex set $\mathcal{W}$, learning rates $\eta_k > 0$
2: for $k = 0, 1, ..., K - 1$ do
3:     Calculate $\nabla l(w_k)$
4:     Update parameter $v_{k+1} = w_k - \eta_k \nabla l(w_k)$
5:     Apply proximal operator $w_{k+1} = \text{prox}_{\eta_k R}(v_{k+1})$
6: end for
7: Output: $w_K$
```

11 Optimization and Generalization

11.1 Generalization Bounds in Statistical Machine Learning

Consider the supervised learning framework. Let the training sample (x_1, y_1), ..., (x_N, y_N) follow the same distribution $(x, y) \sim F(x, y)$ under probability measure $\mathbf{P}$ (or $\mathbf{P}_{(x,y)}$, and the corresponding expectation is $\mathbf{E}$ or $\mathbf{E}_{(x,y)}$). Let the loss functions be $\ell_i(\theta) = \ell((x_i, y_i), \theta)$, $\theta \in \mathbb{R}^d$, $i = 1, 2, ..., N$. Define the empirical loss function

$$L_N(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell_i(\theta) \ . \tag{11.1}$$

Let the local minimizer of $L_N(\theta)$ be θ_N^*, which may not be unique (in this case we further denote them as $\theta_N^{*,1}, ..., \theta_N^{*,k}$). Then the empirical loss function evaluated at θ_N^* is $L_N(\theta_N^*)$. The *generalization gap* is defined to be

$$\text{Gap}_N \equiv \left|L_N(\theta_N^*) - \mathbf{E}_{(x,y)}\ell((x,y),\theta_N^*)\right| = \left|\frac{1}{N}\sum_{i=1}^{N} \ell((x_i, y_i), \theta_N^*) - \mathbf{E}_{(x,y)}\ell((x,y),\theta_N^*)\right| \ . \tag{11.2}$$

We further define the *two-sided generalization gaps* as

$$\text{Gap}_N^+ \equiv L_N(\theta_N^*) - \mathbf{E}_{(x,y)}\ell((x,y),\theta_N^*) \ , \tag{11.3}$$

$$\text{Gap}_N^- \equiv \mathbf{E}_{(x,y)}\ell((x,y),\theta_N^*) - L_N(\theta_N^*) \ . \tag{11.4}$$

In classical statistical learning theory, the basic estimates of generalization gap bounds [1] are bounding Gap_N in (11.2) in terms of features of the family of functions that consists of the loss functions $\ell_i(\theta)$ in which we vary the parameters θ and the training data points (x_i, y_i), $i = 1, 2, ..., N$. Typical such features are the *Vapnik-Chervonenkis dimension* (VC dimension) and the *Rademacher complexity*.

(a) VC dimension. Let the loss function be $\ell((x, y), \theta) = \mathbf{1}(h(x, \theta) \neq y)$ for a family of functions $h(x, \theta)$, $\theta \in \Theta$. Then we define the *growth function* as

$$\Pi_\Theta(N) = \max_{x_1,...,x_N,\theta\in\Theta} |\{(h(x_1, \theta), ..., h(x_N, \theta)), \theta \in \Theta\}| \ . \tag{11.5}$$

Then we have the following bound for the gap Gap_N in (11.2):

$$\mathbf{P}\left(\text{Gap}_N > \varepsilon\right) \leq 4\Pi_\Theta(2N)\exp\left(-\frac{N\varepsilon^2}{8}\right) \ . \tag{11.6}$$

The *VC dimension* is then defined using the growth function

[1] see Vapnik, V., *Statistical Learning Theory*, Wiley-Interscience, first edition, 1988.

$$VC(\Theta) = \max\{N : \Pi_\Theta(N) = 2^N\} \ . \tag{11.7}$$

Suppose $VC(\Theta) = d$, then for any $N > d$, $0 < \delta < 1$ we obtain the generalization gap estimate in terms of VC-dimension as

$$\mathbf{P}\left(\text{Gap}_N^- \leq \sqrt{\frac{8d \ln \frac{2eN}{d} + 8 \ln \frac{4}{\delta}}{N}}\right) \geq 1 - \delta \ . \tag{11.8}$$

The above bound is distribution-free and data-independent. The notion of VC dimension fails to be able to explain the effectiveness of over-parameterized deep neural network models, since in that case the VC dimension blows up yet the generalization gap remains reasonably small.

(b) Rademacher complexity. Let the loss function be $\ell((x, y), \theta) = \mathbf{1}(h(x, \theta) \neq y)$ for a family of functions $h(x, \theta)$, $\theta \in \Theta$. Given the training data points $(x_1, y_1), ..., (x_N, y_N)$, the empirical Rademacher complexity is defined as

$$\widehat{R}_{x_1,...,x_N}(\Theta) = \mathbf{E}_{\sigma_1,...,\sigma_N}\left[\sup_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \sigma_i h(x_i, \theta)\right] \ , \tag{11.9}$$

where the Rademacher variables $\sigma_1, ..., \sigma_N$ independently take values 1 or -1 with probabilities $1/2$. Then we have the gap estimate

$$\mathbf{P}\left(\text{Gap}_N^- \leq 2\widehat{R}_{x_1,...,x_N}(\Theta) + 3\sqrt{\frac{\ln \frac{2}{\delta}}{2N}}\right) \geq 1 - \delta \ . \tag{11.10}$$

Again, the above generalization error bound only makes use of the whole parameter space $\theta \in \Theta$ instead of specifying the information provided by the training algorithm, such as SGD. Indeed, if we run the optimization process like SGD, which can be regarded as a search algorithm of θ_N^*, we may end up in finding among many optimizers $\theta_N^{*,1}, ...\theta_N^{*,k}$ a specific one that achieves good generalization properties.

11.2 Training Matters: Connecting Optimization with Generalization

When applying the classical gap estimates such as (11.8) and (11.10) to deep learning models, the immense size of the number of parameters will make the complexity measurements such as VC-dimensional and Rademacher complexity very large, and thus estimates (11.8) and (11.10) become very rough and useless in practice. This phenomenon is called the "curse of dimensionality", indicating that deep models may overfit the training data and cannot generalize well on test data. However, in practical experiments people found that deep learning models almost never have this bad generalization, and instead, models that are trained by such optimization methods as SGD generalizes very well on test data, with test accuracies often reaching 80-85%+. This seemingly mysterious phenomenon has been one of the major challenges of the theory of deep learning. Currently (at the writing of these lecture notes, in the middle of 2020) we have made only shallow understanding of this phenomenon. It is commonly believed that such strange behavior may be due to the interaction of two effects: the *implicit regularization* of the optimization dynamics and the *network architecture*.

A few papers/preprints suggested:

Cao, Y. and Gu, Q., Generalization Bounds of Stochastic Gradient Descent for Wide and Deep Neural Networks, *NeurIPS*, 2019.

Mei, S. and Montanari, A., The generalization error of random features regression: Precise asymptotics and double descent curve, arXiv:1908.05355 [math.ST]

Jacot, A., Gabriel, F., and Hongler, C., Neural Tangent Kernel: Convergence and generalization in neural networks, NIPS, 2018.

Wu, J., Hu, W., Xiong, H., Huan, J., Braverman, V. and Zhu, Z., On the Noisy Gradient Descent that Generalizes as SGD, arXiv:1906.07405 [cs.LG]

11.3 Experiments: Real Data Sets

• CIFAR-10 is a common benchmark in machine learning for image recognition. The CIFAR-10 dataset consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

The CIFAR-10 and CIFAR-100 datasets are introduced at this webpage

http://www.cs.toronto.edu/~kriz/cifar.html

• Convolutional Neural Networks: see

http://cs231n.github.io/convolutional-networks/

• Use TensorFlow to train and evaluate a convolutional neural network (CNN) on both CPU and GPU. We also demonstrate how to train a CNN over multiple GPUs. Availabe at

https://github.com/tensorflow/models/tree/master/tutorials/image/cifar10

https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py

Detailed instructions on how to get started available at:

https://www.tensorflow.org/tutorials/images/deep_cnn

• Make use of Google-Colab for GPU/TPU support. See https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d

• The MNIST database of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. See

http://yann.lecun.com/exdb/mnist/

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

• More Datasets and Error/Accuracy results in recent Papers. See

http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html.

A Big-O and Little-O notations

Big–O and Little–O notations are used very often in asymptotic analysis, in particular to express asymptotic rates of convergence in the analysis of algorithms.

See https://en.wikipedia.org/wiki/Big_O_notation#Infinite_asymptotics

The symbol $O(x)$, pronounced "Big–O of x", is one of the Landau symbols and is used to symbolically express the asymptotic behavior of a given function.

In particular, if n is an integer variable which tends to infinity and x is a continuous variable tending to some limit, if $\phi(n)$ and $\phi(x)$ are positive functions, and if $f(n)$ and $f(x)$ are arbitrary functions, then it is said that $f \in O(\phi)$ provided that $|f| < A\phi$ for some constant A and all values n and x.

Note that Big–O notation is the inverse of Big–Ω notation, i.e., that

$$f(n) \in O(\phi(n)) \Longleftrightarrow \phi(n) \in \Omega(f(n)).$$

Additionally, Big–O notation is related to Little–O notation in that $f \in o(\phi)$ is stronger than and implies $f \in O(\phi)$.

The symbol $o(x)$, pronounced "Little–O of x," is one of the Landau symbols and is used to symbolically express the asymptotic behavior of a given function.

In particular, if n is an integer variable which tends to infinity and x is a continuous variable tending to some limit, if $\phi(n)$ and $\phi(x)$ are positive functions, and if $f(n)$ and $f(x)$ are arbitrary functions, then it is said that $f \in o(\phi)$ provided that $\frac{f}{\phi} \to 0$. Thus, $\phi(n)$ or $\phi(x)$ grows much faster than $f(n)$ or $f(x)$.

Note that Little–O notation is the inverse of Little–Ω notation, i.e., that

$$f(n) \in o(\phi(n)) \Longleftrightarrow \phi(n) \in \omega(f(n)) \ .$$

Additionally, Little–O notation is related to Big–O notation in that $f \in o(\phi)$ is stronger than and implies $f \in O(\phi)$.