Bachelorarbeit

Universität Passau

Fakultät für Informatik und Mathematik

Stochastic gradient descent

Julian Brunthaler

3. Mai 2019

Betreuer:
Prof. Dr. Tomas Sauer

Inhaltsverzeichnis

1 Einleitung

Seit mehreren Jahren gewinnt die Analyse, Prognose und vor allem das Lernen von vorhandenen Daten immer mehr an Bedeutung. Im Besonderen die Gebiete des maschinellen Lernens und des Deep Learnings werden immer populärer. Damit einhergehend erlangen auch Algorithmen zum Lösen der Optimierungsaufgaben, die dort entstehen, mehr an Bedeutung.
Vorliegende Arbeit beschäftigt sich daher mit Lösungsalgorithmen für das sogenannte „Finite Sum Training Problem“, welches in diesen Bereichen sehr häufig auftritt. Hierbei muss eine endliche Summe von Funktionswerten minimiert werden. Dieses Problem ist eine nichtlineare Optimierungsaufgabe und hat zumeist keine Nebenbedingungen. Die Problematik hierbei ist, dass die Anzahl der Daten und somit die Laufzeit vieler bekannter Methoden, wie zum Beispiel das einfache Gradientenverfahren, viel zu hoch ist. Aus diesem Grund wurde das stochastische Gradientenverfahren und andere ähnliche Algorithmen entwickelt. In dieser Ausarbeitung wird nun das stochastische Gradientenverfahren genauer erläutert, zudem werden auch darauf aufbauende Verfahren vorgestellt. Der Aufbau und Inhalt dieser Arbeit orientiert sich an [1].

2 Motivation

Dieses erste Kapitel soll einen kleinen Einblick in die Anwendungsmöglichkeiten von stochastischen Gradientenverfahren geben.
Im Folgenden seien $\mathcal{X}$ und $\mathcal{Y}$ zwei Mengen von Objekten, wobei wir $\mathcal{X}$ die Menge der Eingabedaten und $\mathcal{Y}$ die Menge der Ausgabedaten nennen. Jedes $y \in \mathcal{Y}$ ist eindeutig einem $x \in \mathcal{X}$ zugeordnet, die zusammengehörenden Daten schreiben wir als Paar (x, y).

2.1 Überwachtes Lernen

Überwachtes Lernen ist ein Teilgebiet des maschinellen Lernens, in dem auf der Grundlage bereits vorhandener Datenpaare $(x_1, y_1), ..., (x_n, y_n)$, wobei n die Anzahl der Beobachtungen ist, versucht wird eine Funktion zu finden, die durch Eingabe von Daten eine Vorhersage trifft. Da die richtigen Ergebnisse vorhanden sind, können die prognostizierten Werte mit den tatsächlichen Werten verglichen werden und deswegen spricht man von überwachtem Lernen.
In diesem Zusammenhang trifft man dann entweder auf Regressionsprobleme oder auf Klassifikationsprobleme. Der Unterschied hier ist, dass bei Regressionsproblemen die zu erklärende Variable stetig ist. Bei Klassifikationsproblemen hingegen, liegt die zu erklärende Variable in diskreter Form oder als kategorische Variable vor und kann somit nur endlich viele Werte annehmen. Für beide Aufgaben gibt es verschiedene Lösungsmethoden, welche in den meisten Fällen, das Finden eines Minimums beinhalten. Wie sich das genauer darstellt sehen wir im nächsten Abschnitt.

2.2 Minimierung finiter Summen von Funktionswerten

Eine Vielzahl der in der Praxis auftretenden Optimierungsprobleme bestehen darin, eine endliche Summe von Funktionswerten über eine große Anzahl an Datenpunkten zu minimieren. Ein einfaches Beispiel dafür ist die Kleinste-Quadrate Schätzung, die für

das Lösen eines Linearen Regressionsmodell verwendet wird,

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (x_i^T w - y_i)^2, \tag{2.1}$$

wobei $x_i \in \mathbb{R}^d$ und $y_i \in \mathbb{R}$ die Daten aus der zugeordneten Linearen Regression sind [2]. Ein weiteres ist die Logistische Regression,

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} log(1 + exp(-y_i x_i^T w) \tag{2.2}$$

wobei $x_i \in \mathbb{R}^d$ und $y_i \in \{-1, 1\}$ die Daten aus dem dazugehörenden Klassifikationsproblem sind [2]. Nun wollen wir einen genaueren Blick auf die Entstehung dieser Minimierungsaufgaben werfen.

2.2.1 Empirisches Risiko

Wir haben nun schon zwei Beispiele gesehen, wie es zu Minimierungsaufgaben bei Fragestellungen aus dem überwachten Lernen kommen kann. Um besser zu verstehen, wie diese entstehen, wird im Folgenden kurz auf das Empirische Risiko eingegangen.

Es seien wieder Datenpaare $(x_1, y_1), ..., (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$ gegeben, welche unter Annahme unabhängig und identisch gezogen wurden. Die Aufgabe ist nun für die gegebenen Daten eine Vorhersagefunktion, oder auch Hypothesenfunktion genannt, zu finden, welche für gegebenes $x_i,\ i \in \{1, ..., n\}$ das dazugehörige y_i am besten approximiert. Diese Funktion wird mit h_w bezeichnet und w ist hierbei der Gewichtsvektor, welcher zu bestimmen ist. Damit nun die Genauigkeit der Schätzung gemessen werden kann, misst man den „*Abstand*“ zwischen der Schätzung $h_w(x_i)$ und dem wahren Wert y_i. Um dies zu bewerkstelligen, benötigt man eine Verlustfunktion L. Inzwischen wurden bereits zwei Darstellungen für Verlustfunktionen genannt, nämlich die Quadratische, auch quadratic/square loss genannt,

$$L(h_w(x_i), y_i) = (h_w(x_i) - y_i)^2$$

in Gleichung (2.1) und die logistische Verlustfunktion, auch als logistic loss bekannt,

$$L(h_w(x_i), y_i) = log(1 + exp(-y_i h_w(x_i))) \tag{2.3}$$

in Gleichung (2.2). Bei der Logistischen Regression wird für die Schätzung der Parameter

auch gerne folgende Funktion verwendet,

$$L(h_w(x_i), y_i) = -y_i \log(h_w(x_i)) - (1 - y_i) \log(1 - h_w(x_i)). \tag{2.4}$$

Hierfür muss aber $y_i \in \{0, 1\}$ sein.
Im Gegensatz zu anderen Verlustfunktionen haben diese drei den Vorteil, dass sie differenzierbar und konvex sind und somit für Optimierungsprobleme sehr gut geeignet sind. Für welche man sich schließlich entscheidet hängt stark von der Anwendung ab, denn jede einzelne Verlustfunktion hat andere Stärken und Schwächen.

Nun zurück zum empirischen Risiko. Unter der Annahme, dass die Daten einer gewissen Verteilung $P(x, y)$ unterliegen, kann nun das erwartete Risiko als Summe über alle Datenpunkte, mit der Wahrscheinlichkeit ein solches Datenpaar zu beobachten, multipliziert mit dem dazugehörigen Fehler, definiert werden,

$$R(h) = E[L(h_w(x), y)] = \int \int L(h_w(x), y) dP(x, y).$$

Das erwartete Risiko misst somit die generelle Performance der Vorhersagefunktion für zukünftige Daten [3]. In der Realität jedoch ist über die Verteilung der Daten nichts bekannt, also kann das erwartete Risiko nicht genau gemessen werden. Glücklicherweise kann aber das Empirische Risiko, welches eine Approximation des wahren Risikos ist, berechnet werden. Es ist definiert als

$$R_{emp}(h) = \frac{1}{n} \sum_{i=1}^{n} L(h_w(x_i, y_i)). \tag{2.5}$$

Im Gegensatz zum erwarteten Risiko, misst das empirische Risiko nur die Performance der bekannten Trainingsdaten [3].

Oftmals wird dem empirischen Risiko noch ein Strafterm r, auch Regulierer genannt, hinzugefügt. Dies wird angewendet, um das Verwenden zu vieler erklärender Variablen, auch overfitting genannt, zu bestrafen. Der Grund, warum man overfitting verhindern möchte, liegt darin, dass bei einem Modell mit zu vielen erklärenden Variablen zwar das empirische Risiko sehr klein wird, die Vorhersagegenauigkeit jedoch sehr schlecht wird, da das Modell zu stark an die vorhandenen Daten angepasst ist. Somit eignen sie sich nicht mehr für Prognosen, was ja das eigentliche Ziel wäre.

Definition 2.1. *Sei V ein Vektorraum. Eine Abbildung $\|\cdot\|V :\to \mathbb{R}_+$ wird Norm genannt, wenn folgende Bedingungen erfüllt sind:*

- *Nichtnegativität: $\|v\| \geq 0$ und $\|v\| = 0 \Leftrightarrow v = 0$.*
- *positive Homogenität: Für $v \in V$ und $\alpha \in \mathbb{R}$ gilt, $\|\alpha v\| = |\alpha|\|v\|$.*
- *Dreiecksungleichung: Für $v, w \in V$ gilt, $\|v + w\| \leq \|v\| + \|w\|$.*

Nun zu zwei oft angewandten Beispielen für Strafterme [3]:

$$r(w) = \|w\|_2^2 = \sum_{i=1}^{n} |w_i|^2 \quad und \quad r(w) = \|w\|_1 = \sum_{i=1}^{n} |w_i|.$$

Das Erste ist konkret die quadrierte Standardnorm im $\mathbb{R}^n$, auch euklidische Norm und 2-Norm genannt, des Koeffizientenvektors und wird oft als l^2-Strafterm bezeichnet. Das zweite Beispiel ist die 1-Norm auch Betragsnorm genannt und wird hier oft als l^1-Strafterm bezeichnet. Diese Strafterme werden meist noch mit einem positiven Skalar λ multipliziert, um die Stärke des Strafterms zu kontrollieren. Somit ergibt sich die Minimierungsaufgabe

$$\min_w \frac{1}{n} \sum_{i=1}^{n} L(h_w(x_i, y_i)) + \lambda r(w), \tag{2.6}$$

welche in der Literatur als regularized empirical risk minimization bekannt ist. Von nun an, falls nicht anders erwähnt, wird immer diese Optimierungsaufgabe betrachtet.

3 Iterative Lösungsverfahren

Eine sehr große Herausforderung bei Klassifikationsaufgaben ist, dass die Anzahl der Datenpunkte n, sowie die Anzahl der Variablen d, oftmals sehr groß sind. Deswegen ist es besonders von Bedeutung Algorithmen zur Verfügung zu haben, die auch bei sehr großen Datenmengen den Rechenaufwand und die Laufzeit in Grenzen halten. Eine solche Methode ist das stochastische Gradientenverfahren, welches in diesem Abschnitt vorgestellt wird. Zusätzlich wird als Einstieg das einfache Gradientenverfahren erläutert. Als erstes werden aber noch einige mathematische Begriffe eingeführt, welche für das Verständnis der Verfahren von zentraler Rolle sind. Dabei wird vor allem den Ausführungen von [4], [5], [6] und [7] gefolgt.

Definition 3.1. *Skalarprodukt*
Ein Skalarprodukt oder inneres Produkt auf einem reellen Vektorraum V ist eine positiv symmetrische Bilinearform $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$, das heißt für $x, y, z \in V$ und $\lambda, \mu \in \mathbb{R}$ gelten folgende Bedingungen:

- *Bilinearität:*
  - $\langle \lambda x + \mu y, z \rangle = \lambda \langle x, z \rangle + \mu \langle y, z \rangle$.
  - $\langle x, \lambda y + \mu z \rangle = \lambda \langle x, y \rangle + \mu \langle x, z \rangle$.
- *Symmetrie:* $\langle x, y \rangle = \langle y, x \rangle$.
- *positive Definitheit:*
  - $\langle x, x \rangle \geq 0$
  - $\langle x, x \rangle = 0 \Leftrightarrow x = 0$.

Das Standardskalarprodukt im $\mathbb{R}^d$ ist definiert als,

$$\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i = x^T y. \tag{3.1}$$

Definition 3.2. *Partielle Ableitungen und Gradient*
Sei $f : \mathbb{R}^d \to \mathbb{R}$ eine Funktion und e_j der j-te Einheitsvektor im $\mathbb{R}^d$.

- *Die Funktion f heißt partiell differenzierbar nach der j-ten Koordinate an $x \in \mathbb{R}^d$, wenn der Grenzwert*

$$\frac{\partial f}{\partial x_j}(x) := \lim_{h \to 0} \frac{f(x + he_j) - f(x)}{h}$$

existiert.

- *Der Vektor*

$$\nabla f = \begin{pmatrix} \frac{\partial}{\partial x_1} f \\ \vdots \\ \frac{\partial}{\partial x_d} f \end{pmatrix} (x)$$

heißt Gradient von f an der Stelle x.

Beispiel 3.3. *Betrachten wir das Beispiel der Kleinst-Quadrate Methode aus* (2.1)*. Hier haben wir Funktionen $f_i : \mathbb{R}^d \to \mathbb{R}$ mit $f_i(w) = (x_i w - y_i)^2$, für i=1,...,n. Der Gradient von f_i ist dann gegeben als,*

$$\begin{pmatrix} \frac{\partial f_i}{\partial w_1} \\ \vdots \\ \frac{\partial f_i}{\partial w_d} \end{pmatrix} (w) = \begin{pmatrix} 2x_{i1}^2 w_1 - 2x_{i1} y_i \\ \vdots \\ 2x_{id}^2 w_d - 2x_{id} y_d \end{pmatrix}.$$

Definition 3.4. *Die Menge $X \subset \mathbb{R}^d$ heißt konvex, wenn für alle x, y $\in X$,*

$$(1 - \alpha)x + \alpha y \in X, \quad \alpha \in [0, 1] \tag{3.2}$$

gilt.

Definition 3.5. *Es seien $D \subset \mathbb{R}^d$ und $X \subset D$ nichtleer und konvex. Die Funktion f: $D \to \mathbb{R}$ heißt konvex auf X, wenn*

$$f((1 - \alpha)x + \alpha y) \leqslant (1 - \alpha) f(x) + \alpha f(y) \tag{3.3}$$

für alle x, y $\in X$ und alle $\alpha \in$ [0, 1] gilt und strikt konvex auf X, wenn

$$f((1 - \alpha)x + \alpha y) < (1 - \alpha) f(x) + \alpha f(y) \tag{3.4}$$

für alle x, y ∈ X und alle α ∈ (0, 1) gilt.

Eine alternative Definition für eine konvexe Funktion ist, dass

$$\langle \nabla f(x) - \nabla f(y), x - y \rangle \geq 0, \quad \text{für alle } x, y \in X \text{ gilt.} \tag{3.5}$$

Beweis. Um diese Behauptung zu beweisen, wird davor noch folgende Aussage gezeigt:

$$f \text{ ist konvex} \Leftrightarrow f(y) \geq f(x) + \nabla f(x)^T (y - x), \quad \text{für alle } x, y \in X. \tag{3.6}$$

Dies wird auch die Bedingung 1. Ordnung für Konvexität genannt.
"⇒"
Aus der Definition der Konvexität erhält man,

$$\begin{aligned} \forall \lambda \in [0,1], x, y \in \mathbb{R}^d : \quad & f(\lambda y + (1-\lambda)x) \leq \lambda f(y) + (1-\lambda) f(x) \\ \Leftrightarrow\ & f(x + \lambda(y - x)) \leq f(x) + \lambda(f(y) - f(x)) \\ \Rightarrow\ & f(y) - f(x) \geq \frac{f(x + \lambda(y-x)) - f(x)}{\lambda}, \ \forall \lambda \in [0,1]. \end{aligned}$$

Für $\lambda \downarrow 0$ ergibt sich dann,

$$f(y) - f(x) \geq \nabla f(x)^T (y - x). \tag{3.7}$$

"⇐"
Es wird angenommen, dass (3.7) für alle $x, y \in \mathbb{R}^d$ gilt. Sei nun $x, y \in \mathbb{R}^d$ beliebig, setze

$$z = \lambda x + (1 - \lambda) y. \tag{3.8}$$

Folglich erhält man,

$$f(x) \geq f(z) + \nabla f(z)^T (x - z) \tag{3.9}$$
$$f(y) \geq f(z) + \nabla f(z)^T (y - z). \tag{3.10}$$

Multipliziert man nun (3.9) mit λ und Gleichung (3.10) mit $(1 - \lambda)$ und addiert sie

anschließend, dann resultiert daraus,

$$\begin{aligned}\lambda f(x) + (1-\lambda)f(y) &\geq f(z) + \nabla f(z)^T(\lambda x + (1-\lambda)y - z)\\ &= f(z)\\ &= f(\lambda x + (1-\lambda)y).\end{aligned}$$

Kombiniert man nun,

$$f(y) \geq f(x) + \nabla f(x)^T(y-x) \text{ und } f(x) \geq f(y) + \nabla f(y)^T(x-y),$$

dann ergibt sich, $\langle \nabla f(x) - \nabla f(y), x - y\rangle \geq 0$ für alle $x, y \in X$. □

Ungleichung (3.5) wird auch die monotone Gradientenbedingung genannt. Ein weiteres Identifikationsmerkmal einer konvexen Funktion ist, dass ihre zweite Ableitung nie negativ ist, also $\nabla^2 f(x) \geq 0$ für alle $x \in X$ gilt. Nun was bedeutet das alles nun genau? Betrachten wir dazu den anschaulichen Fall, also $D \subset \mathbb{R}$, dann bedeutet Konvexität, dass die Verbindungsgerade zu zwei beliebigen Punkten auf dem Funktionsgraphen den Funktionsgraphen nicht zwischen den beiden gewählten Punkten schneidet. Ein Beispiel für eine konvexe Funktion ist die quadratische Funktion $f(x) = x^2$, welche sogar strikt konvex ist.

Definition 3.6. *Stationärer Punkt*
Sei $f : \mathbb{R}^d \to \mathbb{R}$. Ein Punkt $w \in \mathbb{R}^d$ heißt stationärer Punkt, wenn $\nabla f(w) = 0$ gilt.

Eine Funktion $f : D \to \mathbb{R}$ heißt stetig differenzierbar, wenn sie differenzierbar ist und wenn ihre Ableitung stetig ist. Ist f k-mal differenzierbar und f^k stetig
für $k \in \mathbb{N}$, so nennen wir f k-mal stetig differenzierbar und bezeichnen mit $C^k(D)$ die Menge aller k-mal stetig differenzierbaren Funktionen auf D. Ist f nun unendlich oft (stetig) differenzierbar, so nennen wir f eine glatte Funktion, da anschaulich betrachtet dies eine Funktion ohne „*Ecken*“ bzw. ohne Stellen, wo sie nicht differenzierbar ist, darstellt.

Nun legen wir das Augenmerk wieder auf das im vorherigen Kapitel eingeführte Optimierungsproblem,

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} L(h_w(x_i, y_i)) + \lambda r(w), \ \textit{mit } x_i \in \mathbb{R}^d, \ y_i \in \mathbb{R}$$

und schreiben es als,

$$\min_{w\in\mathbb{R}^d} \quad f(w) := \frac{1}{n}\sum_{i=1}^{n} f_i(w),\ mit\ f_i(w) := L(h_w(x_i), y_i) + \lambda r(w). \tag{3.11}$$

Um hier vernünftige Iterationsverfahren konstruieren zu können, müssen einige Grundvoraussetzungen an die zu minimierende Funktion f bzw. an die einzelnen Datenfunktionen f_i gestellt werden [6]. Diese wären:

1. f ist stetig differenzierbar, also $f \in C^1(\mathbb{R}^d)$.

2. Für gegebenes $w \in \mathbb{R}^d$ ist die Niveaumenge,

$$\mathcal{N} := \{w' \in \mathbb{R}^d \ : \ f(w') \leq f(w)\} \tag{3.12}$$

   kompakt.
   Dies bedeutet im $\mathbb{R}^d$, dass die Menge abgeschlossen und beschränkt ist.

3. Der Gradient ∇f ist auf $\mathcal{N}$ Lipschitz-stetig, d.h. es existiert eine konstante $L > 0$, sodass

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2,\ x, y \in \mathcal{N}. \tag{3.13}$$

Erstens benötigt man, um eine Richtung zu finden, in der man sich verbessern kann. Die Lipschitz-Stetigkeit besagt, dass alle Sekanten einer Funktion eine Steigung haben, welche nicht größer als eine Konstante ist. Man kann sagen, diese Forderung verspricht uns, dass die Funktion f halbwegs vernünftig bzw. brauchbar ist. Die zweite Bedingung erläutert uns, dass die Optimierungsaufgabe eine Lösung besitzt. Dies folgt aus folgendem Satz:

Satz 3.7. *Seien $D \subset \mathbb{R}^d$ und $f : D \to \mathbb{R}$ stetig auf D. Ist für ein $w \in D$ die Niveaumenge $\mathcal{N}$ nichtleer und kompakt, dann gibt es ein globales Minimum von f auf D.*

Beweis. Da $\mathcal{N}$ kompakt ist, existiert nach dem Satz von Weierstraß [8] ein $\tilde{x} \in \mathcal{N}$ mit $f(\tilde{x}) \leq f(x)\ \ \forall x \in \mathcal{N}$. Für $x \in D \setminus \mathcal{N}$ ist $f(x) > f(w) \geq f(\tilde{x})$. □

In dieser Arbeit beschränken wir uns auf Aufgaben, in denen alle f_i konvexe und stetig differenzierbare Funktionen sind. Warum konvexe Funktionen so praktisch für Optimierungsaufgaben sind, zeigt der folgende Satz:

Satz 3.8. *Gegeben sei ein Optimierungsproblem ohne Nebenbedingungen*

$$\min_{w \in \mathbb{R}^d} f(w),$$

wo f konvex und differenzierbar ist. Dann ist jeder stationäre Punkt w^* ein globales Minimum.

Beweis. Wegen (3.6) gilt $\forall y \in \mathbb{R}^d$,

$$f(y) \geq f(w^*) + \nabla f(w^*)^T (y - w^*).$$

Da $\nabla f(w^*) = 0$, folgt

$$f(y) \geq f(w^*), \ \text{für alle } y \in \mathbb{R}^d.$$

□

Nun zum Ersten Verfahren.

3.1 Gradientenverfahren

Ein sehr einfacher iterativer Lösungsalgorithmus ist das Gradientenverfahren, auch Verfahren des steilsten Abstiegs genannt. Die Idee hier ist sehr simpel, ausgehend von einem Startwert geht man in jedem Schritt in die Richtung des negativen Gradienten, da dieser den steilsten Abstieg angibt. Die Abstiegsrichtung ist dann gegeben als,

$$-\nabla f(w) = -\frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w). \tag{3.14}$$

Daraus ergibt sich die Rechenvorschrift,

$$w^{t+1} = w^t - \frac{\alpha}{n} \sum_{i=1}^{n} \nabla f_i(w^t) \ , \tag{3.15}$$

wobei α die Schrittweite, auch Lernrate genannt, für jede Iteration t ist. Will man eine Abstiegsrichtung mit Länge 1, was sehr zu empfehlen ist, dann muss

$$y = -\frac{\nabla f(w)}{\|\nabla f(w)\|},$$

als Abstiegsrichtung gewählt werden. Das Gradientenverfahren konvergiert theoretisch immer gegen die optimale Lösung. Dies kann jedoch so langsam sein, dass man numerisch

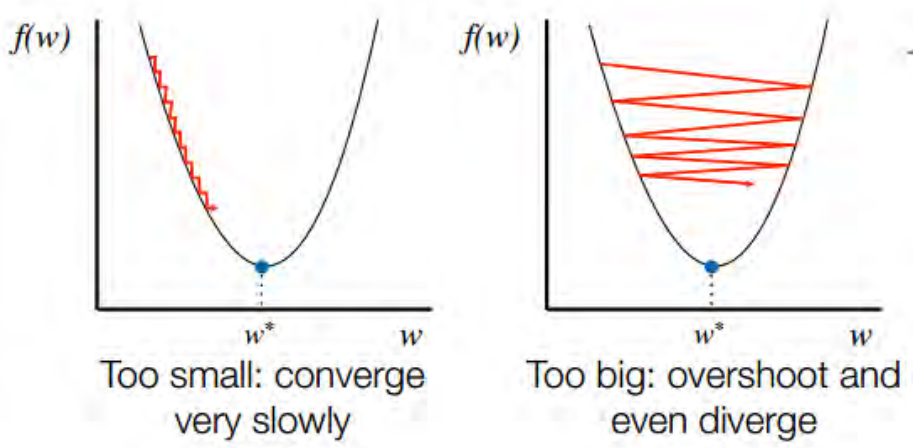

Abbildung 3.1: Vergleich Schrittweiten [9].

nicht mehr von Konvergenz sprechen kann. Hier spielt vor allem die richtige Wahl der Schrittweite α eine Rolle. Eine optimale Schrittweite sollte so gewählt sein, dass sie groß genug ist, um in jedem Schritt eine deutliche Verbesserung zu erzielen. Hiermit ist gemeint, dass man steil genug absteigen soll um sich dem Minimum schnell zu nähern. Jedoch darf die Schrittweite auch nicht zu groß sein, da es sonst sein kann, dass das Minimum übersprungen wird, wie Abbildung (3.1) zeigt. Welche Bedingungen es für die richtige Wahl von α gibt, wird im nächsten Abschnitt noch genau betrachtet.

In der Numerik gibt es mehrere Qualitätsmerkmale für iterative Lösungsverfahren. Die wichtigsten davon sind die numerische Stabilität, die Konvergenzgeschwindigkeit und der Rechenaufwand pro Iteration. Die Stabilität beschreibt die Robustheit gegenüber Störungen bei den Eingabedaten. Dies bedeutet unter anderem, dass sich Rundungsfehler nicht summieren und zu verzerrten Lösungen führen. Wie bereits erwähnt, konvergiert das Gradientenverfahren mit der richtigen Schrittweite gegen die optimale Lösung und gilt somit als numerisch stabil. Die Angabe der Konvergenzgeschwindigkeit erfolgt zumeist mit dem Landau-Symbol $\mathcal{O}$, welches eine asymptotische obere Schranke angibt. Seien g und h zwei beliebige Funktionen, dann bedeutet $h \in \mathcal{O}(g)$, dass $\limsup_{w\to\infty} \left|\frac{h(w)}{g(w)}\right| < \infty$ gilt. Für das Gradientenverfahren gilt,

$$f(w^t) - f(w^*) = \mathcal{O}\left(\frac{1}{t}\right).$$

Das ist eine lineare Konvergenzgeschwindigkeit, das heißt,

$$\limsup_{t\to\infty} \frac{|w^{t+1} - w^*|}{|w^t - w^*|} < 1.$$

Sodann zum Nachteil vom Gradientenverfahren, dem Rechenaufwand pro Iteration. Wie in der Iterationsvorschrift zu sehen ist, muss für jede Beobachtung der Gradient gebildet werden. Eine Gradientenberechnung benötigt d arithmetische Operationen, damit hat man Iterationskosten von $\mathcal{O}(nd)$. Bei Datensätzen mit sehr vielen Beobachtungen und sehr vielen Variablen, also wo n und d groß sind, wird der Rechenaufwand somit extrem groß und die Laufzeit viel zu lange. Hierfür werden dann Verfahren verwendet, deren Kosten pro Iteration nicht mehr von der Anzahl der Beobachtungen n abhängen.

3.1.1 Schrittweitenbedingungen

In diesem Abschnitt soll kurz auf die Wahl einer geeigneten Schrittweite α eingegangen werden, eine genauere Ausführung findet man in [6]. Eine ideale Schrittweite sollte zwei Kriterien erfüllen. Erstens sollte sie groß genug sein, damit in jedem Schritt auch eine ausreichend große Näherung an das Minimum $w*$ erzielt wird, da ansonsten das Verfahren zu langsam konvergieren würde. Zweitens soll die Schrittweite aber nicht zu groß gewählt werden, da man sonst Gefahr läuft, das Minimum zu überspringen.

Definition 3.9. *Ein Vektor $y \in \mathbb{R}^d$ wird Abstiegsrichtung für f genannt, wenn $y^T \nabla f(w) < 0$, für alle $w \in \mathbb{R}^d$ gilt.*

Für Gradientenverfahren werden häufig die Armijo- und die Wolfe-Bedingungen zur Wahl der Schrittweite verwendet. Seien dazu $c_1,\ c_2$ Konstanten, mit $0 < c_1 < c_2 < 1$ und mit y, wird die Abstiegsrichtung bezeichnet.

1. Armijo-Bedingung:
$$f(w + \alpha y) - f(w) \leq \alpha c_1 y^T \nabla f(w)$$

2. Wolfe-Powell-Bedingung:
$$\begin{aligned} f(w + \alpha y) - f(w) &\leq \alpha c_1 y^T \nabla f(w) \\ y^T \nabla f(w + \alpha y) &\geq c_2 y^T \nabla f(w) \end{aligned}$$

3. starke Wolfe-Powell-Bedingungen:
$$\begin{aligned} f(w + \alpha y) - f(w) &\leq \alpha c_1 y^T \nabla f(w) \\ y^T f(w + \alpha y) &\leq c_2 |y^T \nabla f(w)| \end{aligned}$$

Die erste Bedingung soll einen hinreichend großen Abstieg gewähren, die zweite besagt, dass man zu einem Punkt gehen soll, an dem die Funktion nicht mehr so stark abfällt. Die starke Wolfe-Powell Bedingung soll verhindern, dass man so weit geht, dass die Funktion wieder zu stark steigt. Üblicherweise wird c_1 sehr klein und c_2 groß gewählt. Infolgedessen stellt sich noch die Frage, ob solche Schrittweiten überhaupt existieren können.

Satz 3.10. *Sei $f \in C^1(\mathbb{R}^d)$ und nach unten beschränkt, und sei y eine Abstiegsrichtung, dann gibt es für alle $0 < c_1 < c_2 < 1$ Werte von $\alpha \in \mathbb{R}_+$ die die (starken) Wolfe-Bedingungen erfüllen.*

Beweis. Betrachte die Funktion $g(\alpha) := f(w + \alpha y) - f(w) - \alpha c_1 y^T \nabla f(w)$, die nach Annahme an f stetig differenzierbar sein muss. Es gilt $g(0) = 0$ und
$g'(0) = (1-c_1)y^T \nabla f(w) < 0$, da eine y Abstiegsrichtung ist. Daraus folgt, dass $g(\alpha) < 0$ für hinreichend kleines $\alpha > 0$. Da f nach unten beschränkt ist und y eine Abstiegsrichtung ist, gilt $g(\alpha) \to \infty$ für $\alpha \to \infty$. Auf Grund der Stetigkeit von g existiert daher ein $\tilde{\alpha}$ mit $g(\alpha) < 0$ für alle $\alpha \in (0, \tilde{\alpha})$ und

$$0 = g(\tilde{\alpha}) = f(w + \tilde{\alpha} y) - f(w) - \tilde{\alpha} c_1 y^T \nabla f(w),$$

das bedeutet $\tilde{\alpha} > 0$ erfüllt die Armijo-Bedingung. Mit dieser Wahl von $\tilde{\alpha}$ gilt auch,

$$y^T \nabla f(w + \tilde{\alpha} y) - c_1 y^T \nabla f(w) = g'(\tilde{\alpha}) = \lim_{t \to 0_+} \frac{g(\tilde{\alpha}) - g(\tilde{\alpha} - t)}{t} \geq 0,$$

und da $c_1 < c_2$ folgt

$$y^T \nabla f(w + \tilde{\alpha} y) \geq c_1 y^T \nabla f(w) > c_2 y^T \nabla f(w).$$

□

3.2 Stochastisches Gradientenverfahren

Dieser Abschnitt befasst sich mit dem stochastischen Gradientenverfahren, welches im Folgenden meist nur mit SGD(=stochastic gradient descent) abgekürzt wird, und folgt den Ausführungen von [1], [10] und [11]. Das Verfahren wurde 1951 von den beiden Mathematikern Herbert Robbins und Sutton Monro entwickelt. Das Ziel ist hier, die Kosten pro Iteration unabhängig von n werden zu lassen, um den hohen Rechenaufwand

des normalen Gradientenverfahren zu minimieren.

Dazu schätzt man den Gradienten $\nabla f(w)$ in jeder Iteration zufällig durch $\nabla f_j(w)$, mit $j \in \{1, ..., n\}$. Es wird also in jedem Durchlauf nur mehr ein Datenpaar (x_j, y_j) zur Berechnung verwendet. Da $j \in \{1, ..., n\}$ zufällig gewählt wird, ergibt das eine unverzerrte Schätzung des Gradienten von f,

$$\mathbb{E}[\nabla f_j(w)] = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w) = \nabla f(w). \tag{3.16}$$

Folglich erhält man diese Iterationsvorschrift

$$w^{t+1} = w^t - \alpha \nabla f_j(w^t). \tag{3.17}$$

Wie nun unschwer zu erkennen ist, sind die Kosten pro Iteration unabhängig von n. Der Rechenaufwand wird somit, im Vergleich mit dem einfachen Gradientenverfahren, stark verringert, spart man sich ja in jedem Durchlauf $(n-1)$-mal einen Gradienten zu berechnen. Wir haben hier nun Kosten pro Iteration von $\mathcal{O}(d)$ und nicht mehr $\mathcal{O}(nd)$ wie beim Gradientenverfahren.

Diese schöne Verbesserung bringt aber auch einen großen Nachteil mit sich. Im Gegensatz zum normalen Gradientenverfahren ist beim SGD die Schrittrichtung nicht mehr der Mittelwert aller Gradienten, sondern nur mehr ein einzelner Gradient. Deswegen gibt es keine Garantie, dass man sich wirklich mit jeder Iteration dem Minimum nähert. Vielmehr springt das Verfahren ziemlich beliebig hin und her und nähert sich dem optimalen Wert auf diese Weise sehr langsam. Abbildung (3.2) veranschaulicht diese Problematik gut. Links sieht man das normale Gradientenverfahren, rechts die stochastische Variante.

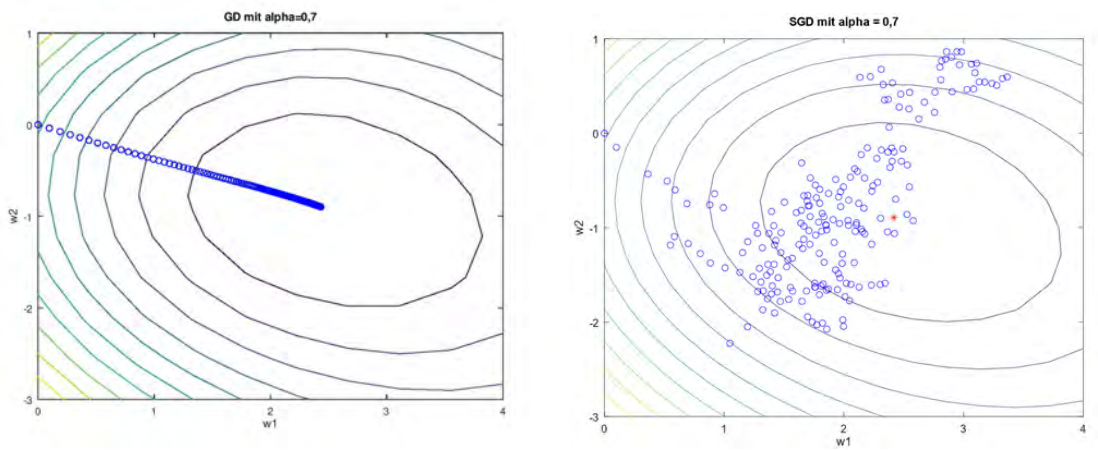

Abbildung 3.2: Vergleich GD und SGD.

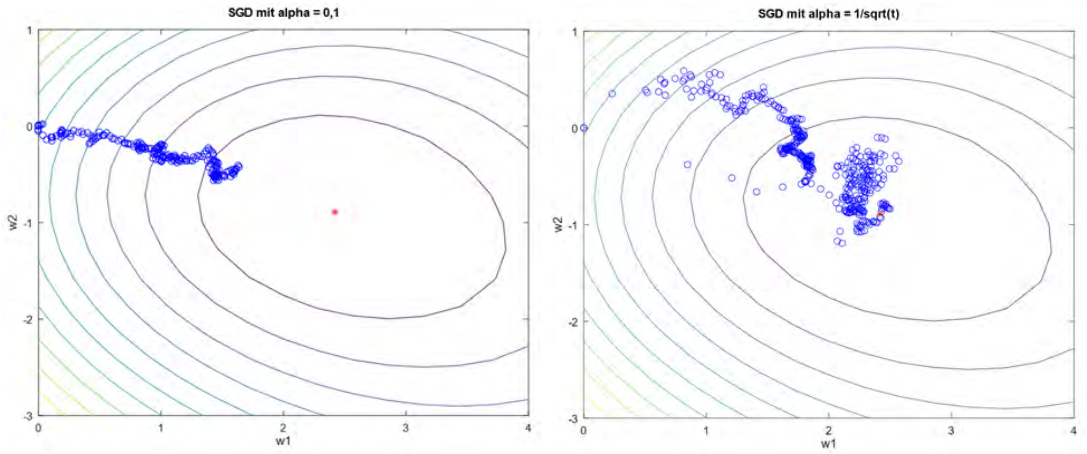

Abbildung 3.3: Vergleich zwischen fester und abfallender Lernrate.

Der rote Stern markiert den optimalen Wert und die blauen Kreise sind die Lösungen aus jeder Iteration. Hier ist deutlich zu erkennen, wie sich die Schrittrichtungen voneinander unterscheiden und dass sie nur im Mittel gleich sind.

Im weiteren wollen wir auch für dieses Verfahren die Schrittweite bzw. Lernrate α genauer untersuchen. Beginnen wir mit der Betrachtung von festen Schrittweiten. Das linke Bild in der Abbildung (3.3) zeigt, was passiert, wenn man α zu klein wählt.

Wie schon beim FG nähern wir uns hier dem Minimum viel zu langsam. Man sollte α also größer wählen, jedoch haben wir folglich mit einem anderem Problem zu kämpfen. Wie zuvor bereits erwähnt und im rechten Bild von Abbildung (3.2) gut zu erkennen, springen die Schrittrichtungen beim SGD ziemlich wild hin und her. Es endet auch nicht damit, je näher wir dem Minimum kommen. Es kann somit sein, dass man dem optimalen Wert schon einmal sehr nahe kommt, im nächsten Durchlauf hingegen wieder weit davon entfernt ist. Das Verfahren konvergiert mit fester Schrittweite somit nicht exakt gegen das Minimum, sondern lediglich gegen eine Kugel mit dem Minimum als Mittelpunkt. Der Radius dieser Kugel ist von α abhängig. Je kleiner α ist, desto kleiner ist auch diese Kugel. Man spricht hier von der Varianz des Verfahrens.

Wie wir nun gesehen haben, haben sowohl eine kleine, als auch eine größere Schrittweite ihre Vor- und Nachteile, am besten ist es daher, wenn man beide Ansätze kombiniert. Dies kann erreicht werden, indem man eine sinkende Schrittweitenfolge verwendet. Hierfür werden meist zwei Bedingungen an die Folge der Schrittweiten $\{a_t\}_{t\in\mathbb{N}}$ gesetzt:

$$\sum_{t=0}^{\infty} \alpha_t = \infty \qquad \textit{und} \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty. \tag{3.18}$$

Die erste Bedingung soll sicherstellen, dass α groß genug ist, um sich auch wirklich dem Minimum zu nähern. Die zweite hingegen, soll den Effekt der Varianz möglichst klein werden lassen. In der Praxis haben sich bei der Wahl der Schrittweitenfolge

$$\alpha_t = \mathcal{O}\left(\frac{1}{t}\right) \text{ und } \alpha_t = \mathcal{O}\left(\frac{1}{\sqrt{t}}\right) \tag{3.19}$$

als sehr brauchbar erwiesen, jedoch ist die optimale Wahl auch stark von der jeweiligen Anwendung abhängig.

Algorithm 1 stochastic gradient

Setze $w^1 = 0$ *und wähle* $\alpha_1 > 0$
for $t = 1, 2, ..., T$ **do**
  ziehe zufällig $j \in \{1, ..., n\}$
  berechne $\nabla f_j(w^t)$
  $w^{t+1} = w^t - \alpha_{t+1} \nabla f_j(w^t)$
end for
output w^{t+1}

Beispiel 3.11. *mini-batch SGD*
Natürlich ist es auch möglich mehr als nur eine Beobachtung pro Iteration zur Schätzung von $\nabla f(w)$ *zu verwenden. Dazu sei* $\tau << d$, *und* $C \subset \{1, \cdots, n\}$ *mit* $|C| = \tau$. *Sei* $I_C \in \mathbb{R}^{n \times \tau}$ *die Matrix mit den entsprechenden Einheitsvektoren des* $\mathbb{R}^n$. *Dann sieht die Schätzung des Gradienten wie folgt aus,*

$$DF(w)I_C = [\nabla f_{C1}(w), \cdots, \nabla f_{C\tau}(w)]$$
$$\Rightarrow w^{t+1} = w^t - \frac{\alpha}{\tau} \sum_{j=C_1}^{C_\tau} \nabla f_j(w^t).$$

Bei dieser Variante wird die Varianz der Schätzung um den Faktor $\frac{1}{\tau}$ vermindert, jedoch erhöhen sich dadurch auch die Kosten pro Iteration um den Faktor τ. Wie unschwer zu erkennen ist, erhält man für $\tau = n$ das normale Gradientenverfahren.

Nun haben wir zwei unterschiedliche Verfahren gesehen, die beide ihre Vor- und Nachteile haben. Das Ziel wäre nun eine Methode zu konstruieren, die den Rechenaufwand pro Iteration des stochastischen Gradientenverfahren mit der Konvergenzgeschwindigkeit des einfachen Gradientenverfahren kombiniert. Dies führt uns im nächsten Kapitel zum stochastic average gradient(=SAG). Davor wird hingegen noch ein genauerer Blick auf die Konvergenz des SGD geworfen.

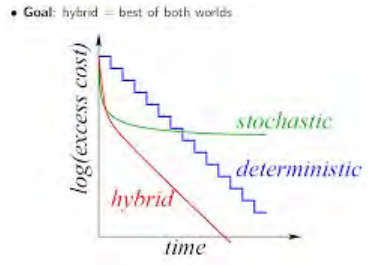

Abbildung 3.4: Vergleich Konvergenzgeschwindigkeit

3.3 Konvergenzanalyse

Es wird wieder angenommen, dass wir die Funktionen f_j aus unserem Opitimierungsproblem (3.11) betrachten und dass alle f_j konvex und stetig differenzierbar sind. Zusätzlich sollten die Gradienten $\nabla f_j(w)$ Lipschitz-stetig mit Konstante L sein. Das bedeutet für alle $x, y \in \mathbb{R}^d$ gilt,

$$\|\nabla f_j(x) - \nabla f_j(y)\|_2 \leq L\|x - y\|_2. \tag{3.20}$$

Darüber hinaus wird die Existenz eines globalen Minimums w^*, das den optimalen Funktionswert annimmt, angenommen. Die hier präsentierten Ergebnisse decken sich weitestgehend mit denen aus [1], [7] und [12].

Definition 3.12. *Eine differenzierbare Funktion $f : \mathbb{R}^d \to \mathbb{R}$ wird stark μ konvex genannt, wenn*

$$f(w) \geq f(y) + \langle \nabla f(y), w - y\rangle + \frac{\mu}{2}\|w - y\|_2^2 \tag{3.21}$$

für ein $\mu > 0$ und für alle w, $y \in \mathbb{R}^d$ gilt. Wenn f zweimal differenzierbar ist, dann ist dies gleichbedeutend mit,

$$\nabla^2 f(w) \geq \mu \mathbf{1}, \quad \mathbf{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^d,$$

auch Bedingung 2. Ordnung für starke Konvexität genannt.

Eine weitere alternative Definition für starke Konvexität ist:

$$h(w) = f(w) - \frac{\mu}{2}\|w\|_2^2 \text{ ist konvex für alle } w \in \mathbb{R}^d. \tag{3.22}$$

Dies folgt aus der Bedingung 1. Ordnung (3.6) für Konvexität von h. Zusätzlich impliziert

die starke Konvexität von f,

$$\langle \nabla f(w) - \nabla f(y), w - y \rangle \geq \frac{\mu}{2} \|w - y\|_2^2.$$

Dies folgt aus der monotonen Gradientenbedingung (3.5) für Konvexität von h,

$$h \text{ ist konvex} \Leftrightarrow \langle \nabla h(w) - \nabla h(y), w - y \rangle \geq 0, \quad \text{für alle } w, y \in \mathbb{R}^d.$$

In der klassischen Analyse des stochastischen Gradientenverfahren wird zusätzlich noch angenommen, dass die stochastischen Gradienten gleichbleibend beschränkt sind,

$$\mathbb{E}[\|\nabla f_j(w^t)\|_2^2] \leq B^2, \ \text{für } B > 0 \text{ und } w^t \in \mathbb{R}^d. \tag{3.23}$$

In [13] wurde aber gezeigt, dass diese Annahme im Widerspruch mit der Annahme der starken Konvexität von f steht. Eine alternative Annahme findet man ebenfalls in [13]. Wir werden diesen Widerspruch hier aber ignorieren. Weshalb dies kein große Problem darstellt, sieht man, wenn man den einfachen Fall $d = 1$, also $f : \mathbb{R} \to \mathbb{R}$ betrachtet. Sei f stark konvex mit einer bestimmten Konstante $\mu > 0$, dann gilt,

$$f''(w) \geq \mu, \quad \text{für alle } w \in \mathbb{R}.$$

Integriert man das obige, ergibt sich mit dem Hauptsatz der Differential- und Integralrechnung,

$$\begin{aligned} \int_a^b f''(w) dw &\geq \int_a^b \mu dw \\ f'(b) - f'(a) &\geq \mu b - \mu a, \qquad w \in [a, b] \subset \mathbb{R}. \end{aligned}$$

Je größer das Intervall wird, desto größer wird auch $\mu b - \mu a$ und somit kann $f'(w)$ beliebig groß bzw. klein werden. Dies lässt sich für $f(w) = w^2$ schön veranschaulichen. Wie unschwer zu erkennen ist, ist der Gradient $f'(w) = 2w$ für $w \mapsto \pm\infty$ unbeschränkt. Bei unserer Opitimierungsaufgabe stört das aber nicht sonderlich, da man ja nur nach Lösungen in einem endlichen Intervall sucht und der Gradient hier beschränkt ist.

Für den Konvergenzbeweis zum Gradientenverfahren benötigen wir noch zwei Abschätzungen für f,

(1) $f(w) \leq f(w') + \nabla f(w')^T (w - w') + \frac{L}{2} \|w - w'\|_2^2, \ \text{für } w, w' \in \mathbb{R}^d,$

(2) $f(w^*) - f(w) \leq -\frac{1}{2L}\|f(w)\|_2^2$, für alle $w \in \mathbb{R}^d$.

Für die erste Ungleichung wird die Konvexität von $g(w) := \frac{L}{2}w^T w - f(w)$ benötigt.

$$0 \overset{(3.5)}{\leq} \langle \nabla f(w) - \nabla f(w'), w - w' \rangle \tag{3.24}$$

$$\leq \|\nabla f(w) - \nabla f(w')\|_2 \|w - w'\|_2 \overset{(3.20)}{\leq} L\|w - w'\|_2^2. \tag{3.25}$$

Der Gradient von g ist gegeben durch

$$\nabla g(w') = Lw' - \nabla f(w') \Leftrightarrow \nabla f(w') = Lw' - \nabla g(w').$$

Setzt man nun $Lw' - \nabla g(w')$ für $\nabla f(w')$ und $Lw - \nabla g(w)$ für $\nabla f(w)$ in (3.24) ein, erhält man,

$$\begin{aligned}
&(Lw - \nabla g(w) - (Lw' - \nabla g(w'))^T (w - w') \leq L\|w - w'\|_2^2 \\
\Leftrightarrow\ &(\nabla g(w') - \nabla g(w))^T (w - w') + L(w - w')^T(w - w') \leq L\|w - w'\|_2^2 \\
\Leftrightarrow\ &(\nabla g(w') - \nabla g(w))^T (w - w') \leq 0 \\
\Leftrightarrow\ &(\nabla g(w) - \nabla g(w'))^T (w - w') \geq 0.
\end{aligned}$$

Somit ist g nach (3.5) konvex. Eine andere Definition der Konvexität besagt,

$$g(w) \geq g(w') + \nabla g(w')^T(w - w'),\ \textit{für alle}\ w, w' \in \mathbb{R}^d.$$

Daraus folgt mit der Definition von g,

$$\begin{aligned}
&\frac{L}{2}w^T w - f(w) \geq \frac{L}{2}w'^T w' - f(w') + (Lw' - \nabla f(w'))^T(w - w') \\
\Leftrightarrow\ &f(w') - f(w) + \frac{L}{2}(w^T w - w'^T w') \geq (Lw' - \nabla f(w'))^T(w - w') \\
\Leftrightarrow\ &f(w) \leq f(w') + \nabla f(w')^T(w - w') + \frac{L}{2}\|w - w'\|_2^2.
\end{aligned}$$

Somit ist (1) gezeigt. Für (2) setzen wir $w = w' - \frac{1}{L}\nabla f(w')$ in (1) ein und erhalten

$$\begin{aligned} f\left(w' - \frac{1}{L}\nabla f(w')\right) &\leq f(w') + \nabla f(w')^T\left(-\frac{1}{L}\nabla f(w')\right) + \frac{L}{2}\| - \frac{1}{L}\nabla f(w')\|_2^2 \\ &= f(w') - \frac{1}{L}\|\nabla f(w')\|_2^2 + \frac{1}{2L}\|\nabla f(w')\|_2^2 \\ &= f(w') - \frac{1}{2L}\|\nabla f(w')\|_2^2. \end{aligned}$$

Da w^* ein globales Minimum von f ist, gilt $f(w^*) \leq f\left(w' - \frac{1}{L}\nabla f(w')\right)$ und somit

$$f(w^*) - f(w') \leq -\frac{1}{2L}\|\nabla f(w')\|_2^2, \text{ für alle } w' \in \mathbb{R}^d. \tag{3.26}$$

Nun zu den ersten Konvergenzaussagen. Als erstes wollen wir noch einen Satz über die Konvergenz des normalen Gradientenverfahren angeben.

Satz 3.13. *Sei f stark μ-konvex und ∇f Lipschitz-stetig mit konstante L. Für gegebenes $w^0 \in \mathbb{R}^d$ und $\frac{1}{L} \geq \alpha > 0$, erfüllen die Iterierten des Gradientenverfahren*

$$\|w^{t+1} - w^*\|_2^2 \leq (1 - \alpha\mu)^{t+1}\|w^0 - w^*\|_2^2.$$

Beweis. Einsetzen der Iterationsvorschrift liefert,

$$\begin{aligned} \|w^{t+1} - w^*\|_2^2 &= \|w^t - w^* - \alpha\nabla f(w^t)\|_2^2 \\ &= \|w^t - w^*\|_2^2 - 2\langle\nabla f(w^t), w^t - w^*\rangle + \alpha\|\nabla f(w^t)\|_2^2 \\ &\overset{(3.21)}{\leq} (1 - \alpha\mu)\|w^t - w^*\|_2^2 - 2\alpha(f(w^t) - f(w^*)) + \alpha^2\|\nabla f(w^t)\|_2^2. \\ &\overset{(3.26)}{\leq} (1 - \alpha\mu)\|w^t - w^*\|_2^2 - 2\alpha(f(w^t) - f(w^*)) + 2\alpha^2 L(f(w^t) - f(w^*)) \\ &= (1 - \alpha\mu)\|w^t - w^*\|_2^2 - 2\alpha(1 - \alpha L)(f(w^t) - f(w^*)). \end{aligned}$$

Da $\frac{1}{L} \geq \alpha$ ist $-2\alpha(1 - \alpha L) \leq 0$, also kann der letzte Term weggelassen werden und man erhält,

$$\begin{aligned} \|w^{t+1} - w^*\|_2^2 &\leq (1 - \alpha\mu)\|w^t - w^*\|_2^2 \\ &= (1 - \alpha\mu)^{t+1}\|w^0 - w^*\|_2^2 \end{aligned}$$

□

Im folgenden Satz wird die Konvergenz des SGD im Falle einer stark μ-konvexen

Funktion f und unter Verwendung einer fixen Schrittweite betrachtet.

Satz 3.14. *Falls* $\frac{1}{\mu} \geq \alpha \geq 0$*, dann erfüllen die Iterierten des SGD*

$$\mathbb{E}[\|w^t - w^*\|_2^2] \leq (1 - \alpha\mu)^t \mathbb{E}[\|w^0 - w^*\|_2^2] + \frac{\alpha}{\mu} B^2. \tag{3.27}$$

Beweis.

$$\begin{aligned}\|w^{t+1} - w^*\|_2^2 =& \|w^t - w^* - \alpha \nabla f_j(w^t)\|_2^2 \\ =& \|w^t - w^*\|_2^2 - 2\alpha \langle \nabla f_j(w^t), w^t - w^* \rangle + \alpha^2 \|\nabla f_j(w^t)\|_2^2.\end{aligned}$$

Nehmen wir den Erwartungswert bezüglich j, dann ergibt sich,

$$\begin{aligned}\mathbb{E}_j[\|w^{t+1} - w^*\|_2^2] &= \|w^t - w^*\|_2^2 - 2\alpha \langle \nabla f(w^t), w^t - w^* \rangle + \alpha^2 \mathbb{E}_j \|[\nabla f_j(w^t)\|_2^2] \\ &\leq \|w^t - w^*\|_2^2 - 2\alpha \langle \nabla f(w^t), w^t - w^* \rangle + \alpha^2 B^2.\end{aligned}$$

Hier haben wir Annahme (3.23) und $\mathbb{E}[\nabla f_j(w)] = \nabla f(w)$ verwendet. Da f stark μ-konvex ist, gilt

$$2\langle \nabla f(w^t), w^t - w^* \rangle \geq \mu \|w^t - w^*\|_2^2.$$

Somit erhalten wir,

$$\begin{aligned}\mathbb{E}[\|w^{t+1} - w^t\|_2^2] &\leq \mathbb{E}[\|w^t - w^*\|_2^2] - \alpha\mu \mathbb{E}[\|w^t - w^*\|_2^2] + \alpha^2 B^2 \\ &= (1 - \alpha\mu)\mathbb{E}[\|w^t - w^*\|_2^2] + \alpha^2 B^2.\end{aligned}$$

Mit Induktion folgt dann,

$$\mathbb{E}[\|w^{t+1} - w^t\|_2^2] \leq (1 - \alpha\mu)^{t+1} \|w^0 - w^*\|_2^2 + \sum_{i=0}^{t} (1 - \alpha\mu)^i \alpha^2 B^2. \tag{3.28}$$

Verwenden der Geometrischen Summe $\sum_{i=0}^{t} (1 - \alpha\mu)^i = \frac{1-(1-\alpha\mu)^{t+1}}{\alpha\mu} \leq \frac{1}{\alpha\mu}$ liefert

$$\mathbb{E}[\|w^{t+1} - w^t\|_2^2] \leq (1 - \alpha\mu)^{t+1} \|w^0 - w^*\|_2^2 + \alpha^2 B^2. \tag{3.29}$$

□

Um nun optimale Konvergenz zu erreichen sollte $\alpha \approx \frac{1}{\mu}$ sein, damit der erste Term

wegfällt. Jedoch sollte α auch nahe genug bei 0 sein, um den zweiten Term möglichst klein zu halten. Wie bereits erwähnt, ist es besser beim SGD eine abfallenden Schrittweitenfolge zu verwenden. Dann erhält man für die Konvergenz folgendes:

Satz 3.15. *Falls $\alpha_t = \frac{1}{t\mu}$, dann erfüllen die Iterierten des SGD*

$$\mathbb{E}[\|w^t - w^*\|_2^2] \leq \frac{4B^2}{t\mu^2}. \tag{3.30}$$

Beweis. Die starke Konvexität von f impliziert, dass für alle $w^1 \in \mathbb{R}^d$,

$$\langle \nabla f(w^1) - \underbrace{\nabla f(w^*)}_{=0}, w^1 - w^* \rangle \geq \frac{\mu}{2}\|w^1 - w^*\|_2^2$$

gilt, mit der Cauchy-Schwarz Ungleichung folgt dann,

$$\|\nabla f(w^1)\|_2^2 \geq \frac{\mu^2}{4}\|w^1 - w^*\|_2^2. \tag{3.31}$$

Zusätzlich haben wir,

$$\begin{aligned}
\mathbb{E}[\|\nabla f_j(w^1)\|_2^2 &= \mathbb{E}[\|\nabla f(w^1) + (\nabla f_j(w^1) - \nabla f(w^1)\|_2^2] \\
&= \mathbb{E}[\|\nabla f(w^1)\|_2^2] + \mathbb{E}[\|\nabla f_j(w^1) - \nabla f(w^1)\|_2^2] + 2\mathbb{E}[\langle \nabla f_j(w^1) - \nabla f(w^1), \nabla f(w^1)\rangle] \\
&\geq \mathbb{E}[\|\nabla f(w^1)\|_2^2].
\end{aligned}$$

Dies kombiniert mit (3.31) ergibt für alle t,

$$\mathbb{E}[\|w^1 - w^*\|_2^2] \leq \frac{4}{\mu^2}\mathbb{E}[\|\nabla f_j(w^t)\|_2^2] \leq \frac{4B^2}{\mu^2}. \tag{3.32}$$

Unter der Annahme eines unverzerrten Schätzers und (3.23) liefert die SGD Rechenvorschrift,

$$\mathbb{E}[\|w^{t+1} - w^*\|_2^2] \leq \mathbb{E}[\|w^t - w^*\|_2^2] - 2\alpha\mathbb{E}[\langle \nabla f(w^t), w^t - w^*\rangle] + \alpha_{t+1}^2 B^2. \tag{3.33}$$

Die starke Konvexität von f besagt, dass

$$\langle \nabla f(w^t), w - w*\rangle \geq \mu\|w^t - w^*\|_2^2$$

und folglich,

$$\mathbb{E}[\langle \nabla f(w^t), w^t - w^* \rangle] \geq \mu \mathbb{E}[\|w^t - w^*\|_2^2].$$

Dies eingesetzt in (3.33) liefert,

$$\mathbb{E}[\|w^{t+1} - w^*\|_2^2] \leq (1 - 2\alpha_{t+1}\mu)\mathbb{E}[\|w^t - w^*\|_2^2] + \alpha_{t+1}^2 B^2. \tag{3.34}$$

Nun lässt sich der Satz mit vollständiger Induktion beweisen.
$t = 1$:

$$\begin{aligned}\mathbb{E}[\|w^2 - w^*\|_2^2] &\overset{\alpha_t = \frac{1}{t\mu^2}}{\leq} \left(1 - \frac{2}{2}\right)\mathbb{E}[\|w^1 - w^*\|_2^2] + \frac{B^2}{2\mu} \\ &= \frac{B^2}{2\mu^2} \leq \frac{4B^2}{2\mu^2}.\end{aligned}$$

Der Basisfall ist somit bereits korrekt. Die Induktionshypothese ist (3.30). Nun sei $t \in \mathbb{N}$ beliebig.
$t \mapsto t + 1$:

$$\begin{aligned}\mathbb{E}[\|w^{t+1} - w^*\|_2^2] &\leq \left(1 - \frac{2}{t+1}\right)\mathbb{E}[\|w^t - w^*\|_2^2] + \frac{B^2}{(t+1)\mu^2} \\ &\overset{I.H.}{\leq} \left(1 - \frac{2}{t+1}\right)\frac{4B^2}{t\mu^2} + \frac{B^2}{(t+1)\mu^2} \\ &\leq \frac{4B^2}{t\mu^2} - \frac{8B^2}{(t^2+t)\mu^2} + \frac{B^2}{(t+1)\mu^2} \leq \frac{4B^2}{(t+1)\mu^2}.\end{aligned}$$

□

4 Varianzreduzierende Verfahren

Das stochastische Gradientenverfahren löst zwar das Problem der hohen Kosten pro Iteration, jedoch konvergiert es viel zu langsam gegen die optimale Lösung. Wie bereits erwähnt, ist die hohe Varianz der Schätzung des Gradienten $\nabla f(w)$ durch $\nabla f_j(w)$ problematisch. Sie kann zwar durch die Verwendung einer sinkenden Schrittweitenfolge reduziert werden, jedoch muss bedacht werden, dass diese Folge der Schrittweiten für unterschiedliche Anwendungen immer wieder angepasst werden muss. Aus diesem Grund wurden in den letzten Jahren alternative Methoden zur Varianzreduzierung entwickelt. Das Kapitel orientiert sich an den Ausführungen von [1], [2] und [14].

4.1 Stochastic Average Gradient

Das Erste Verfahren, das wir betrachten, ist der stochastic average gradient descent, kurz SAG. Dieser Algorithmus wurde erst vor ein paar Jahren von M. Schmidt, N. Le Roux und F. Bach entwickelt und gilt als bahnbrechende Methode in der stochastischen Optimierung. Er kombiniert die Konvergenzgeschwindigkeit des normalen Gradientenverfahrens und die Kosten pro Iteration des SGD. Bevor genauer auf das Verfahren eingegangen wird, werden noch ein paar Definition benötigt.

Definition 4.1. *Jacobi-Matrix*
Sei $f : U \to \mathbb{R}^n$, $U \subseteq \mathbb{R}^d$ offen, eine Funktion, deren partiellen Ableitungen alle existieren, wobei $f = (f_1, ..., f_n)$, dann bezeichnet man für $x \in U$ die Matrix

$$Df(x) := Jf(x) := \left[\frac{\partial f_j}{\partial x_k}(x) : \begin{array}{l} j = 1, ..., n \\ k = 1, ..., d \end{array} \right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_d} \end{bmatrix}$$

als Jacobimatrix von f an der Stelle x.

Betrachten wir wieder unsere Optimierungsaufgabe (3.11) und schreiben nun alle

$f_j(w)$ als Spalte in eine Matrix, dann erhalten wir

$$F(w) \stackrel{def}{=} (f_1(w), ..., f_n(w)).$$

Die Jacobimatrix von F an der Stelle w wäre nun $JF(w) = (\nabla f_1(w)^T, ..., \nabla f_n(w)^T)$, hier schreiben wir hingegen $DF(w)$ als die Transponierte von $JF(w)$ und bezeichnen sie mit J, also

$$J = DF(w) = (\nabla f_1(w), ..., \nabla f_n(w)) \in \mathbb{R}^{d \times n}.$$

Somit ergibt sich

$$\nabla f(w) = \frac{1}{n} DF(w)\mathbf{1}, \quad wobei\ \mathbf{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^n.$$

Nun wieder zurück zum Verfahren. Der Unterschied zum SGD ist beim SAG, dass wir nicht mehr direkt $\nabla f_j(w^t)$ zur Schätzung von $\nabla f(w^t)$ verwenden. Nun wird mit $\nabla f_j(w^t)$ in jedem Schritt eine Schätzung g^t vom wahren Gradienten aktualisiert,

$$g^t \approx \nabla f(w^t).$$

Dafür verwenden wir die Matrix $J^t \approx DF(w^t)$. Es wird nun in jeder Iteration zufällig ein Index j gewählt und

$$J_i^t = \begin{cases} \nabla f_i(w^t), & \text{für } i = j \\ J_i^{t-1}, & \text{für } i \neq j \end{cases} \tag{4.1}$$

gesetzt. J_i^t bezeichnet hierbei die i-te Spalte von J^t. Damit ergibt sich für die SAG Iteration

$$w^{t+1} = w^t - \frac{\alpha}{n} \sum_{i=1}^{n} J_i^t = w^t - \alpha g^t. \tag{4.2}$$

In der Matrix J^t sind somit die bereits verwendeten Gradienten $\nabla f_j(w^k)$,
$k = 1, ..., t-1$, aus den ersten $(t-1)-$Schritten abgespeichert. Bei jeder Iteration wird nun der Durchschnitt dieser Gradienten gebildet und als Schätzung für $\nabla f(w^t)$ verwendet. Diese Berechnungen der Mittelwerte ist natürlich wieder teurer je größer n ist, ist also nicht wie gewollt von n unabhängig. Deswegen schreiben wir die Iteration

geschickt um zu

$$w^{t+1} = w^t - \alpha \left(\underbrace{\frac{J_j^t}{n} - \frac{J_j^{t-1}}{n} + \underbrace{\frac{1}{n}\sum_{i=1}^{n} J_i^{t-1}}_{alterDurchschnitt}}_{neuerDurchschnitt} \right) \tag{4.3}$$

und speichern einfach den alten Durchschnitt ab. Dann wird er bei jedem Schritt nur um den zufällig gezogenen Gradienten $J_j^t = \nabla f_j(w^t)$ aktualisiert und der alte Wert für $\nabla f_j(w^k) = J_j^{t-1}$ wird gelöscht. Somit haben wir wieder dieselben Kosten pro Iteration wie beim SGD.

Algorithm 2 SAG mit Schrittweite α

$g = 0,\ w^0 = 0,\ J_i = 0\ f\ddot{u}r\ i = 1, ..., n$
for $t = 1, 2, ..., T$ **do**
  $Ziehe\ zuf\ddot{a}llig\ i \in \{1, ..., n\}$
  $g = g - J_i + \nabla f_i(w^t)$
  $J_i = \nabla f_i(w^t)$
  $w^t = w^t - \frac{\alpha}{n} g$
end for

Das SAG Verfahren bietet leider nicht nur Vorteile, der große Nachteil gegenüber dem SGD ist, dass die Schätzungen für die Gradienten nicht mehr unverzerrt sind. Diese Verzerrung, oft auch Bias genannt, sehen wir, wenn wir die stochastischen Gradienten in der SAG Update-Regel aus (4.3) betrachten,

$$\underbrace{J_j^t - J_j^{t-1} + \sum_{i=1}^{n} J_i^{t-1}}_{X:=}.$$

Es gilt nun

$$\mathbb{E}\left[J_j^t\right] = \mathbb{E}\left[\nabla f_j(w^t)\right] = \nabla f(w^t).$$

Die Gültigkeit der ersten Gleichung folgt einfach aus der Definition für J_j^t und die zweite Gleichheit wissen wir bereits aus dem letzten Kapitel. Es ist unschwer zu sehen, dass

$$\mathbb{E}\left[J_j^{t-1} + \sum_{i=1}^{n} J_i^{t-1}\right] \neq 0.$$

Somit ist $\mathbb{E}[X] \neq \nabla f(w^t)$ und wir haben einen verzerrten Schätzer. Im nächsten Kapitel werden wir eine Möglichkeit sehen, wie wir einen unverzerrten Schätzer erhalten,

doch davor gehen wir noch kurz auf einen Satz zur Konvergenz des SAG Algorithmus für konvexe und stark konvexe Funktionen ein. Wir nehmen dafür wieder an, dass ∇f Lipschitz-stetig mit konstante L ist und dass genau eine optimale Lösung w^* existiert. Zusätzlich definieren wir $\overline{w}^t := \frac{1}{t}\sum_{i=1}^{t-1} w^i$ und $\sigma^2 := \frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(w^*)\|^2$.

Satz 4.2. *(Schmidt, Le Roux, Bach)*
Für konvexes f und konstanter Schrittweite $\alpha_t = \frac{1}{16L}$ erfüllt die SAG Iteration für $t \geq 1$

$$\mathbb{E}\left[\nabla f(\overline{w}^t)\right] - f(w^*) \leq \frac{32n}{k} C_0,$$

wo wir für die Initialisierung $J_i^0 = 0$

$$C_0 = \nabla f(w^0) - \nabla f(w^*) + \frac{4L}{n}\|w^0 - w^*\|^2 + \frac{\sigma^2}{16L}$$

erhalten und für $J_i^0 = \nabla f_i(w^0) - \nabla f(w^0)$

$$C_0 = \frac{3}{2}\Big(\nabla f(w^0) - \nabla f(w^*)\Big) + \frac{4L}{n}\|w^0 - w^*\|^2$$

erhalten. Für stark μ-konvexes f gilt

$$\mathbb{E}\left[\nabla f(\overline{w}^t)\right] - f(w^*) \leq \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8n}\right\}\right)^t C_0.$$

Für den komplizierten und sehr aufwendigen Beweis wird auf [**miwisag**] verwiesen.

4.2 Jacobian Sketching Algorithm

Die verzerrte Schätzung des Gradienten ist ein sehr großer Nachteil des SAG, daher wurden bereits mehrere neue Methoden entwickelt, die den SAG-Algorithmus so modifizieren, dass er unverzerrte Schätzungen berechnet. Zwei der bekanntesten Verfahren davon sind der SAGA-Algorithmus von A. Defazio, F. Bach und S. Lacoste-Julien und der SVRG-Algorithmus von R. Johnson und T. Zhang. In diesem Abschnitt wird ein weitere Alternative vorgestellt, welche ebenfalls den SAG-Algorithmus so modifiziert, dass er nicht mehr verzerrte Schätzungen berechnet.

Der Startpunkt für die neue Methode ist folgende triviale Beobachtung

$$\frac{1}{n} J\mathbf{1} = \nabla f(w), \tag{4.4}$$

wobei **1** der Vektor aus lauter Einsen im $\mathbb{R}^n$ ist. Dies ist noch wenig hilfreich, da J nicht verfügbar ist, denn die Berechnung für $J \approx DF(w)$ würde $\mathcal{O}(nd)$ Operationen kosten. Ersetzen wir **1** mit einem beliebigen Einheitsvektor e_j, so erhalten wir,

$$Je_j = DF(w)e_j = \nabla f_j(w). \tag{4.5}$$

Dies ist nichts Anderes als das schon bekannte stochastische Gradientenverfahren. Es löst zwar das Problem des hohen Rechenaufwandes, hat aber den Nachteil, dass nicht von den Informationen aus den vorherigen Iterationen gelernt wird. Das Ziel wäre es, die Schätzung J so zu berechnen, dass $J = DF(w)$ gilt, ohne die hohen Kosten der direkten Berechnung zu tragen. Dafür nehmen wir eine dünnbesetzte Matrix $S \in \mathbb{R}^{nx\tau}$, welche zufällig aus einer bestimmten Verteilung $\mathcal{D}$ gezogen wird und erhalten

$$JS = DF(w)S \in \mathbb{R}^{dx\tau}, \quad \tau << d. \tag{4.6}$$

Dies ist nun eine Verallgemeinerung von (4.4) und (4.5). Es ist klar, dass die wahre Jacobi-Matrix (4.6) löst. Aber im Allgemeinen, wenn τ sehr viel kleiner als d ist, hat die Gleichung unendlich viele Lösungen für J.

Beispiel 4.3. *Batch SGD Sketch*

Seien $n = 6$, $\tau = 3$, $S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, *dann ergibt sich* $DF(w)S = [\nabla f_1(w), \nabla f_3(w), \nabla f_6(w)]$.

Beispiel 4.4. *Averaging Batch*

$S = \begin{pmatrix} a_1 \\ 0 \\ a_3 \\ a_4 \end{pmatrix}$, *mit* $a_1, a_3, a_4 \in \mathbb{R}$, *dann ist* $DF(w)S = \sum_{i=1}^{4} a_i \nabla f_i(w)$.

Die Idee beim Jacobian Sketching ist nun, dass bei jeder Iteration die Lösung J^t gewählt wird, die am nächsten zu der vorherigen Lösung J^{t-1} ist. Um diese Abstandsmessung vorzunehmen, wählt man die Frobeniusnorm, welche eine Matrixnorm ist und

wird für eine reelle Matrix $A \in \mathbb{R}^{mxn}$ folgendermaßen definiert,

$$\|A\|_F := \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_i j|^2}.$$

Um die Lösung für (4.6) zu erhalten, muss somit folgende Optimierungsaufgabe gelöst werden:

$$J^t = arg\min_{J\in\mathbb{R}^{d\times n}} \|J - J^{t-1}\|_F^2 \tag{4.7}$$

unter der Nebenbedingung

$$JS = DF(w^t)S.$$

Die Lösung für J^t ist gegeben als,

$$J^t = J^{t-1} - (J^{t-1} - DF(w^t))S(S^TS)^{-1}S^T. \tag{4.8}$$

Um diese Aussage beweisen zu können, benötigen wir das Verfahren der Lagrange-Multiplikatoren, welches eine Methode zur Lösung von Optimierungsproblemen mit Gleichheitsnebenbedingungen ist. Kurz erklärt funktioniert das Verfahren folgendermaßen: Gegeben ist eine stetig differenzierbare Funktion $f : D \to \mathbb{R}$, mit $D \subseteq \mathbb{R}^d$ offen, welche entweder maximiert oder minimiert werden soll, unter den Nebenbedingungen $g_1(x) = c_1, ..., g_k(x) = c_k$ mit $x \in D,\ c_i \in \mathbb{R},\ k \in \mathbb{N}$, und $i = 1, ..., k$. Die Vorgehensweise sieht nun wie folgt aus:

- Bilde die Lagrange-Funktion $L(x, \lambda) = f(x) + (\lambda_1, \cdots, \lambda_k)\begin{pmatrix} c_1 - g_1(x) \\ \vdots \\ c_k - g_k(x) \end{pmatrix}$.

- Finde die stationären Punkte von L: Löse $\nabla L(x, \lambda_1, \cdots, \lambda_k) = 0$.

- Untersuche welche Stellen Maxima/Minima sind.

Die Werte $\lambda_1, \cdots, \lambda_k$ werden Lagrange-Multiplikatoren genannt. Ein mathematisch genauer Satz zu den notwendigen Bedingungen für die Existenz der Lagrange-Multiplikatoren findet sich unter anderem in [15]. Nun zum Beweis von (4.8).

Beweis. Die Lagrange-Funktion ist,

$$\begin{aligned} L(J,Y) &:= \|J - J^{t-1}\|_F^2 + \langle Y, (DF(w^t) - J)S\rangle \\ &= \|J - J^{t-1}\|_F^2 + \langle YS^T, DF(w^t) - J\rangle \end{aligned}$$

ableiten nach J ergibt, $2(J - J^{t-1}) - YS^T$ und Null setzen liefert,

$$YS^T = 2(J - J^{t-1}). \tag{4.9}$$

Von rechts mit $S(S^TS)^{-1}$ multiplizieren ergibt,

$$Y = 2(DF(w^t) - J^{t-1})S(S^TS)^{-1}. \tag{4.10}$$

Einsetzen von (4.10) in (4.9) liefert die Lösung $J^t = J^{t-1} - (J^{t-1} - DF(w^t)) \underbrace{S(S^TS)^{-1}S^T}_{P:=}$. □

Jetzt wissen wir die Jacobi-Schätzung für jeden Schritt und können das verwenden, um eine Schätzung für den Gradienten zu erhalten,

$$g^t = \frac{1}{n} J^t \mathbf{1}.$$

Unglücklicherweise liefert das Einsetzen von J^t anstatt $DF(w^t)$ wieder keine unverzerrte Schätzung. Der Grund dafür ist, dass diese Vorgehensweise, genauso wie die des SAG-Algorithmus, die Varianz der Schätzung zu aggressiv minimiert und dabei den Bias ignoriert. Daher wird ein sogenannter stochastischer Relaxationsparameter θ eingeführt,

$$g^t = \frac{\theta}{n} J^t \mathbf{1} - \frac{1-\theta}{n} J^{t-1}\mathbf{1} = \frac{1}{n} J^{t-1}\mathbf{1} - \frac{\theta}{n}\left(J^{t-1} - DF(w^t)P\right)\mathbf{1}. \tag{4.11}$$

Der Parameter soll dann einen Trade-off zwischen Varianz und Verzerrung herstellen. Je näher θ bei eins ist, desto niedriger ist die Varianz und desto höher ist der Bias. Im nächsten Satz werden wir dann sehen, für welches θ die Schätzung (4.11) unverzerrt ist, doch davor werden noch ein paar mathematische Begriffe benötigt.

Definition 4.5. *Eigenwert und Eigenvektor*
Die Zahl $\lambda \in \mathbb{C}$ heißt Eigenwert einer Matrix $A \in \mathbb{R}^{nxn}$ mit dem zugehörigen Eigenvektor $x \in \mathbb{C}^n,\ x \neq 0$, wenn

$$Ax = \lambda x.$$

Man bezeichnet dann (λ, x) als ein Eigenpaar von A.

Die Matrix $P = S(S^TS)^{-1}S^T$ wird eine stochastische Matrix genannt, da ihre Einträge Zufallsvariablen sind, die von der für S gewählten Verteilung $\mathcal{D}$ abhängen. Für solche Matrizen ist der Erwartungswert definiert als

$$\mathbb{E}[P] = \begin{pmatrix} \mathbb{E}(p_{11}) & \cdots & \mathbb{E}(p_{1n}) \\ \vdots & \ddots & \vdots \\ \mathbb{E}(p_{n1}) & \cdots & \mathbb{E}(p_{nn}) \end{pmatrix}.$$

Mit $\mathbb{E}_S[P]$ wird der Erwartungswert von P gegeben S definiert.
Nun haben wir alle Hilfsmittel und können zum vorher bereits angekündigten Satz zurückkommen.

Satz 4.6. *Wenn $(\frac{1}{\theta}, \mathbf{1})$ ein Eigenpaar von $\mathbb{E}[P_S]$ ist, dann gilt*

$$\mathbb{E}_S[g^t] = \nabla f(w^t), \tag{4.12}$$

folglich ist g^t ein unverzerrter Schätzer.

Beweis. Der Erwartungswert von (4.11) gegeben S ist,

$$\begin{aligned} \mathbb{E}_S[g^t] &= \frac{1}{n}J^{t-1}\mathbf{1} - \frac{\theta}{n}(J^{t-1} - DF(w^t))\mathbb{E}_S[P]\mathbf{1} \\ &= \frac{1}{n}J^{t-1}\mathbf{1} - \frac{\theta}{n}(J^{t-1} - DF(w^t))\mathbb{E}_S[S(S^TS)^{-1}S^T]\mathbf{1} \\ &= \frac{1}{n}J^{t-1}\mathbf{1} - \frac{\theta}{n\theta}(J^{t-1} - DF(w^t)\mathbf{1} \\ &= \frac{1}{n}J^{t-1}\mathbf{1} - \frac{1}{n}J^{t-1}\mathbf{1} + \frac{1}{n}DF(w^t)\mathbf{1} \\ &= \nabla f(w^t). \end{aligned}$$

□

Als nächstes folgen zwei Beispiele für die Bestimmung des Parameters θ.

Beispiel 4.7. *Sei $\mathbb{P}[S = e_i] = \frac{1}{n}$, für $i = 1, ..., n$. Dann gilt*

$$\mathbb{E}[P]\mathbf{1} = \mathbb{E}[S(S^TS)^{-1}S^T]\mathbf{1} = \frac{1}{n}\mathbf{1}.$$

Beweis. Dies lässt sich leicht durch nachrechnen zeigen:

$$\begin{aligned}\mathbb{E}[S(S^TS^{-1})S^T]\mathbf{1} &= \sum_{i=1}^{n} \frac{1}{n}\frac{e_ie_i^T}{e_i^Te_i} \\ &= \frac{1}{n}\sum_{i=1}^{n} e_ie_i^T\mathbf{1} \\ &= \frac{1}{n}\mathbf{I}_n\mathbf{1} = \frac{1}{n}\mathbf{1}.\end{aligned}$$

□

Hier wurde verwendet, dass $e_i^Te_i$ gleich eins ist und dass $\sum_{i=1}^{n} e_ie_i^T$ die Einheitsmatrix $\mathbf{I}_n$ ergibt. Mit Hilfe des letzten Satzes sehen wir, dass in diesem Beispiel $\theta = n$ ist, da $\frac{1}{n}$ der Eigenwert mit Eigenvektor $\mathbf{1}$ von $\mathbb{E}[P]$ ist.

Beispiel 4.8. *Sei $C \subset \{1, ..., n\}$ mit $|C| = \tau$ und $\mathbb{P}[S = I_C] = \frac{1}{\binom{n}{\tau}}$. In diesem Fall gilt $\theta = \frac{n}{\tau}$. Die Matrix I_C ist hierbei eine $nx\tau$-Matrix mit den Eigenvektoren $e_j,\ j \in C$, des $\mathbb{R}^n$ als Spalten. Die Spalten sind hierbei der Reihe der Eigenvektoren nach sortiert, also dass für $\tau = n$ die Einheitsmatrix entsteht.*

Beweis. Es ist $I_C^TI_C = I_\tau$ für alle beliebig gewählten Mengen C mit $|C| = \tau$. Für $I_CI_C^T$ erhält man eine nxn-Diagonalmatrix X^τ, deren $i - tes$ Diagonalelement 1 ist, wenn $i \in C$, und 0, wenn $i \notin C$. Damit erhalten wir,

$$\begin{aligned}\mathbb{E}[S(S^TS)^{-1}S^T]\mathbf{1} &= \sum_{i=1}^{\binom{n}{\tau}} \frac{1}{\binom{n}{\tau}} S_iI_\tau S_i^T\mathbf{1} \\ &= \frac{1}{\binom{n}{\tau}}\sum_{i=1}^{\binom{n}{\tau}} X_i^\tau\mathbf{1} \\ &= \frac{1}{\binom{n}{\tau}}\binom{n}{\tau}\frac{\tau}{n}I_n\mathbf{1} \\ &= \frac{\tau}{n}\mathbf{1}.\end{aligned}$$

Somit ist $\frac{1}{\theta} = \frac{\tau}{n}$ ein Eigenwert mit zugehörigem Eigenvektor $\mathbf{1}$ von $\mathbb{E}[P]$. □

Algorithm 3 Jacobian Sketching Algorithm

Wähle $\alpha > 0,\ \theta > 0,\ w^1 \in \mathbb{R}^d$ *und* $\mathcal{J}^1 \in \mathbb{R}^{d \times n}$.
for $t = 1, ..., T$ **do**
Ziehe zufälliges $\mathcal{S} \sim \mathcal{D}$.
Berechne sketch $DF(w^t)S$
$J^t = J^{t-1} - (J^{t-1} - DF(w^t))S(S^TS)^{-1}S^T$
$g^t = \frac{\theta}{n} J^t \mathbf{1} + \frac{1-\theta}{n} J^{t-1}\mathbf{1}$
$w^{t+1} = w^t - \alpha g^t$
end for

5 Simulationsbeispiele und Implementierung

In diesem abschließenden Kapitel werden, in der Hoffnung ein besseres Verständnis für die Optimierungsverfahren zu geben, einige Beispielanwendungen für das stochastische Gradientenverfahren simuliert. In jedem Beispiel wird der SGD mit dem einfachen Gradientenverfahren und dem SAG verglichen. Als Simulationsdaten dienen der „fourclass“ Datensatz aus der Open-Source-Bibliothek LIBSVM [16] und selbst generierte Datensätze. Am Ende dieses Kapitels befinden sich die Quellcodes für die Implementierung der Algorithmen in Octave.

5.1 Lineare Regression

Zu Beginn ein ganz einfaches Beispiel. Wir betrachten hierfür zwei Fälle einer Kleinst-Quadrate Schätzung mit jeweils 50000 Beobachtungen. Im ersten Fall haben wir lediglich eine erklärende Variable. Dies ist zwar wenig realistisch, aber für einen ersten Vergleich der Verfahren ausreichend. Zusätzlich lässt sich dieser Fall auch besser graphisch darstellen. Erwähnt muss auch noch sein, dass wir hier trotzdem zwei Parameter zu schätzen haben, da das zugrunde liegende lineare Regressionsmodell mit einem konstanten Glied, Intercept genannt, geschätzt wird. Dadurch hat unsere Datenmatrix X zwei Spalten, wobei die erste aus lauter Einsen besteht. Die zu minimierende Funktion sieht wie folgt aus,

$$f(w) = \sum_{i=1}^{50000} (y_i - x_{i1} * w_1 + x_{i2} * w_2)^2. \tag{5.1}$$

Die Funktion f wird in diesem Kontext auch gerne Kostenfunktion genannt. Wie zu sehen ist, verwenden wir für dieses Modell keinen Regulierungsterm. Der Plot der Kostenfunktion bestätigt, dass wir hier eine relativ einfaches Minimierungsproblem vorfinden. Wie zu sehen ist, ist es hauptsächlich von Bedeutung den richtigen Wert für w_2 zu finden, um die Funktion zu minimieren. Der Intercept-Term w_1 ist weniger von Bedeu-

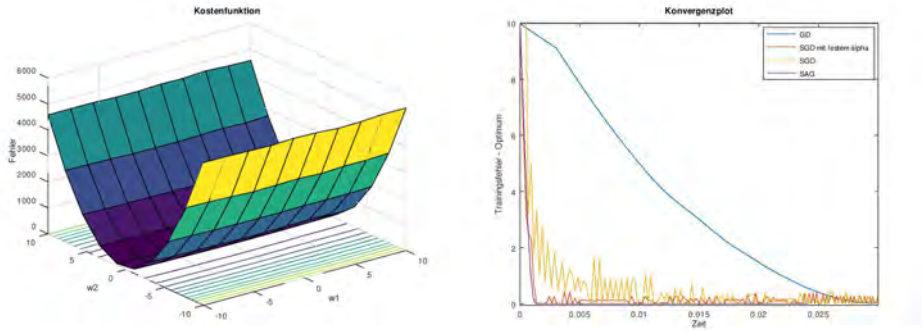

Abbildung 5.1: Kostenfunktion und Konvergenzplot für die Kleinst-Quadrate Schätzung mit $n = 50000$ und $d = 2$.

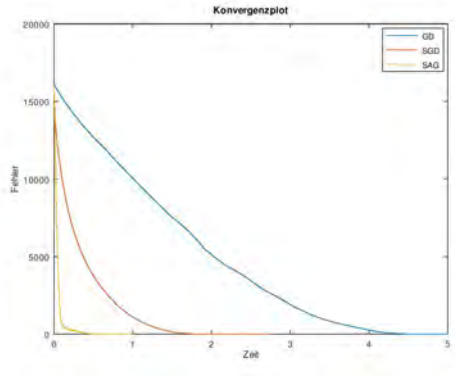

Abbildung 5.2: Konvergenzplot für die Kleinst-Quadrate Schätzung mit n=50000 und d=100.

tung. Das rechte Bild von (5.1) zeigt die Konvergenzgeschwindigkeit der Algorithmen. Auf der x-Achse ist die Zeit in Sekunden gegeben und auf der y-Achse der Fehler minus dem minimalen, also dem kleinstmöglichen Fehler. Zusätzlich wird hier noch der SGD-Algorithmus mit fester Schrittweite betrachtet, um noch einmal zu verdeutlichen, dass dieser nie exakt gegen die Minimalstelle konvergiert. Er ist zwar am Anfang schneller als der SGD-Algorithmus mit abfallender Schrittweitenfolge, im Gegensatz dazu werden seine Schwingungen aber nie kleiner.
Als nächstes betrachten wir die Kleinst-Quadrate Schätzung mit 100 Variablen, um zu zeigen, dass auch auf die Größe von d geachtet werden muss. Wie man nun in Abbildung (5.2) sehen kann, benötigen die Algorithmen nun schon deutlich länger zum Lösen der Aufgabe. Hier sieht man auch, dass das normale Gradientenverfahren für solch große Datensätze nicht mehr geeignet ist.

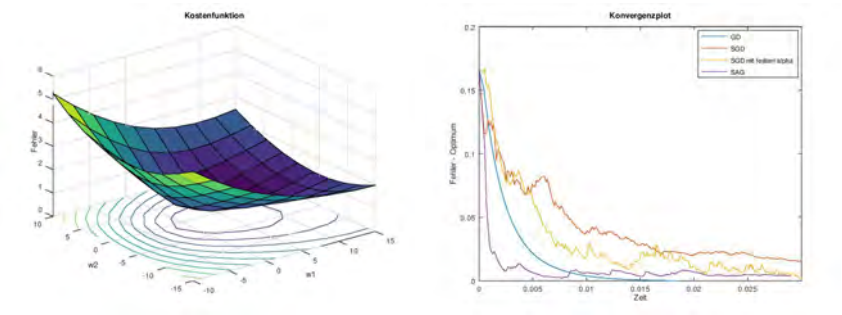

Abbildung 5.3: Kostenfunktion und Konvergenzplot für (5.2).

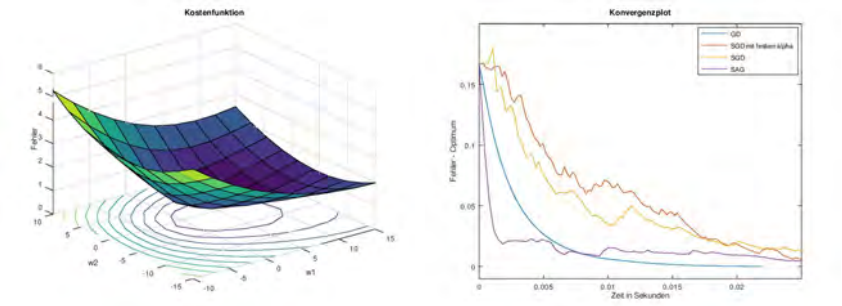

Abbildung 5.4: Kostenfunktion und Konvergenzplot für (5.3).

5.2 Logistische Regression

In diesem Abschnitt werden die Algorithmen nun am fourclass-Datensatz der Open-Source-Bibliothek LIBSVM getestet. Dieser hat n Beobachtungen und 2 erklärende Variablen. Da y hier eine binäre Variable ist, wird das Verfahren der Logistischen Regression verwendet. Aufgrund dessen, dass es dafür zwei geläufige Verlustfunktionen gibt, welche beide in Abschnitt 2.2.1 erwähnt sind, werden die Algorithmen an beiden Funktionen getestet. Im Ersten Fall benutzen wir (2.4) und erhalten,

$$\min_{w\in\mathbb{R}^d} f(w) = \frac{1}{862}\left(-y\log(x^T w) - (1-y)\log(1-x^T w)\right), \tag{5.2}$$

wobei $y \in \{0,1\}$. Im Zweiten wird (2.3) mit $y \in \{-1,1\}$ verwendet und es ergibt sich,

$$\min_{w\in\mathbb{R}^d} f(w) = \frac{1}{862} log(1 + exp(-yx^T w)). \tag{5.3}$$

Wie in Abbildung (5.3) und (5.4) zu sehen ist, liefern die Algorithmen für beide Funktionen fast identische Ergebnisse, lediglich am Anfang sind die Algorithmen für (5.2) etwas schneller.

5.3 Quellcodes

Bei der Implementierung der Algorithmen und den dazugehörenden Funktionen wurde sich an [17] orientiert. Das Gradientenverfahren und der SAG sind hier ohne eine spezielle Liniensuche für die Schrittweite α programmiert. Der SGD und SAG sind so implementiert, dass anstatt einer beliebigen Ziehung eines Datenpaares in jedem Schritt, ein zuvor erstellter Zufallsvektor, mit zufälligen Indizes, durchlaufen wird. Dies hat den selben Effekt wie zufälliges Ziehen und wird gemacht, da die Algorithmen so eine kürzere Rechenlaufzeit haben.

```
function [x_tr,y_tr,x_te,y_te,w_opt] = linreg_data_gen (n, d)
% Datengenerator fuer Lineare Regression.
% Inputs:
%       n               Anzahl der Beobachtungen
%       d               Anzahl der Variablen
% Outputs:
%       x_tr            Trainingsdatenmatrix der Dimension nxd
%       y_tr            Trainingsdatenvektor der Dimension nx1
%       x_te            Testdatenmatrix (nxd)
%       y_te            Testdatenvektor  (nx1)
%       w_opt           wahrer Parametervektor ((d+1)x1)
  w_opt = 2 * randn(d+1,1);        % optimale Parameter
  % Trainingsdaten
  x_tr = 10 * randn(n,d);
  x_tr = [ones(n,1),x_tr];         % Hinzufuegen des Intercept/Bias
  noise = 0.1 * rand(n,1);         % Stoerterm
  y_tr = x_tr * w_opt + noise;
  % Testdaten
  x_te = 10 * randn(n,d);
  x_te = [ones(n,1),x_te];
  noise = 0.1 * rand(n,1);
  y_te = x_te * w_opt + noise;
endfunction
```

```
function sol = sigmoid (z)
  sol=1./(1+exp(-z));
endfunction
```

```
classdef lin_reg_l2
%Inputs: x.....Trainingsdaten der Matrix x mit Dimension nxd
%        y.....Trainingsdaten des Vektors y mit Dimension nx1
%                (y_i=1 oder y_i=0)
%        lambda....l2-Regularisierungsparameter
%Outputs: Problem...die zu minimierende Funktion f und deren Gradient g
%Minimierungsproblem ist
%       min f(w) = 1/n * sum_i^n f_i(w),
%wobei
%       f_i(w)= (x_i*w-y_i)^2 + lambda/2 * norm(w)^2;
  properties
    x;
    y;
    lambda;
    n;
    d;
  end
  methods
    function obj = lin_reg_l2(x,y,lambda)
      obj.x=x;
      obj.y=y;
      obj.lambda=lambda;
      obj.n=length(y);
      obj.d=length(x(1,:));
    end
    function f = cost(obj,w)
      f=1/(2*obj.n) *sum((obj.x*w-obj.y).^2) + obj.lambda/2 *norm(w)^2;
    end
    function g = grad(obj,w,indices)
      residuen = obj.x(indices,:)*w - obj.y(indices);
      g = obj.x(indices,:)'* residuen /length(indices) + obj.lambda*w;
    end
  end
end
```

```
classdef log_reg_l2
%Inputs:  x.....Trainingsdaten der Matrix x mit Dimension nxd
%        y.....Trainingsdaten des Vektors y mit Dimension nx1
%                (y_i=1 oder y_i=0)
%        lambda....l2-Regularisierungsparameter
%Outputs: Problem...die zu minimierende Funktion f und deren Gradient g
% Minimierungsproblem ist
%    min f(w) = 1/n * sum_i^n f_i(w),
```

```
%wobei
%     f_i(w) = -y_i*log(1/(1+exp(-x_i*w)))-(1-y_i)*log(1-1/(1+exp(-x_i*w)))
%              + lambda/2*norm(w)^2.
  properties
    x;
    y;
    lambda;
    n;
    d;
  end
  methods
    function obj = log_reg_l2(x,y,lambda)
      obj.x=x;
      obj.y=y;
      obj.lambda=lambda;
      obj.n=length(y);
      obj.d=length(x(1,:));
    end
    function f = cost(obj, w)
      h=sigmoid(obj.x * w);
      % um log(0) zu vermeiden
      h=h + (h<eps).*eps;
      f_nr = -(obj.y'*log(h) + (ones(obj.n,1)-obj.y)'
             *log(ones(obj.n,1)-h))/obj.n;
      f= f_nr + 0.5 * obj.lambda * (norm(w).^2);
    end
    function g = grad(obj, w, indices)
      z=obj.x(indices,:) * w;
      g_nr=obj.x(indices,:)'*(sigmoid(z)-obj.y(indices))/length(indices);
      g= g_nr +obj.lambda*w;
    end
    function gb = grad_batch(obj, w, indices)
      g=zeros(obj.d,length(indices));
      z=obj.x(indices,:)*w;
      for i=length(indices)
        g(:,i)=obj.x(i,:)'*(sigmoid(z(i))-obj.y(i))+obj.lambda*w;
      end
      gb=mean(g,2);
    end
  end
end
```

```
classdef log_reg_2
%Inputs: x.....Trainingsdaten der Matrix x mit Dimension nxd
%        y.....Trainingsdaten des Vektors y mit Dimension nx1
%                (y_i=1 oder y_i=0)
%        lambda....l2-Regularisierungsparameter
%Outputs: Problem...die zu minimierende Funktion f und deren Gradient g
% Minimierungsproblem ist
%       min f(w) = 1/n * sum_i^n f_i(w),
%wobei
%       f_i(w) = log(1+exp(-y_i'.*(x_i*w))) + lambda/2 * norm(w)^2.
  properties
    x;
    y;
    lambda;
    n;
    d;
  end
  methods
    function obj = log_reg_2(x,y,lambda)
      obj.x=x;
      obj.y=y;
      obj.lambda=lambda;
      obj.n=length(y);
      obj.d=length(x(1,:));
    end
    function f = cost(obj, w)
      h =sigmoid(exp(-obj.y.*obj.x*w));
      h = h + (h<eps).*eps;
      f = sum(log(ones(obj.n,1)+h))/obj.n + obj.lambda/2 *norm(w)^2;
    end
    function g = grad(obj, w, indices)
      h = exp(obj.y(indices).*obj.x(indices,:)*w);
      g_nr = -ones(1,length(indices))*((obj.y(indices).*obj.x(indices,:))
             ./(ones(length(indices),1)+h))/length(indices);
      g= g_nr' + obj.lambda*w;
    end
  end
end
```

```
function cost = costfunction (problem, w)
%Inputs:            problem...Daten, Funktion, Gradient
%                   w.........Werte die in die Funktion eingesetzt werden,
%                             kann ein Vektor, oder eine Matrix sein.
%Outputs            cost......Fehler bzw. Kosten fuer verwendeten Wert w,
%                             Vektor der Laenge iter.
  n=problem.n;
  iter=length(w(1,:));
  J=zeros(iter,1);
  for k=1:iter
    J(k)=problem.cost(w(:,k));   %Fehler fuer w aus dem k-ten Schritt.
  end
  cost=J;
endfunction
```

```
function [sol,time] = gd (problem, alpha, num_iter)
%Inputs:     problem...Daten, Funktion, Gradient
%            alpha.....Schrittweite/Lernrate
%            num_iter..Anzahl der Iterationen
%Outputs:    sol.......Matrix(dxnum_iter) mit den berechneten Werte fuer w
%            time......Vektor(num_iterx1) der die Rechenzeit abspeichert
  n=problem.n;                          %Anzahl der Beobachtungen
  d=problem.d;                          %Anzahl der Variablen
  w=zeros(d,num_iter+1);                %speichern der Loesungen
  T=zeros(num_iter+1,1);                %speichern der Rechenzeit/Iteration
  for t=1:num_iter
    tic();
    grad=problem.grad(w(:,t),[1:n]);
    if (norm(grad)>1)                   %Skalieren, um zu gross werdende
      grad=grad/norm(grad);             %Gradienten zu vermeiden
    end
    w(:,t+1)=w(:,t) - alpha*grad;
    T(t+1)=T(t)+toc();
  end
  sol=w;
  time=T;
endfunction
```

```
function [sol, time ] = sgd (problem, num_iter)
%Inputs:          problem...Daten, Funktion, Gradient
%                 num_iter..Anzahl der Iterationen
%Outputs:         sol.......Matrix(dxnum_iter) mit den
%                           berechneten Werte fuer w
%                 time......Vektor(num_iterx1) der Rechenzeit abspeichert
  n=problem.n;                    %Groesse des Datensatzes
  d=problem.d;                    %Anzahl der Variablen
  w=zeros(d,num_iter+1);          %speichern der Loesungen aus jedem Schritt
  T=zeros(num_iter+1,1);          %Vektor zum Abspeichern der Rechenzeit
  indizes=randi(n,num_iter,1);    %Vektor mit den zufaelligen Indizes
  for t=1:num_iter
    tic();
    alpha=1/sqrt(t);                %Schrittweitenfolge alpha
    j=indizes(t);                   %j ist nun der zufaellig gewaehlte Index
    grad=problem.grad(w(:,t),j);
    if (norm(grad)>1)
      grad=grad/norm(grad);         %Laenge des Gradienten skalieren
    end
    w(:,t+1)=w(:,t) - alpha*grad;
    T(t+1)=T(t)+toc();
  end
  sol=w;
  time=T;
endfunction
```

Bei der Berechnung des Mittelwertes des Gradienten in Zeile 20, wird nicht durch n dividiert, da dies ja nicht wirklich der Anzahl der verwendeten Gradienten entspricht.

```
1  function [sol, time] = sag (problem, alpha, num_iter)
2  %Inputs:          problem...Daten, Funktion, Gradient
3  %                 alpha.....Schrittweite/Lernrate
4  %                 num_iter..Anzahl der Iterationen
5  %Outputs:         sol.......Matrix(dxnum_iter) mit den Werten fuer w
6  %                 time......Vektor(num_iterx1) Rechenzeiten
7    n=problem.n;                    %Anzahl der Datenpaare
8    d=problem.d;                    %Anzahl der Variablen
9    w=zeros(d,num_iter+1);          %Speichern der Loesungen ausjedem Schritt
10   T=zeros(num_iter+1,1);          %Vektor zum Abspeichern der Rechenzeit
11   grad_array=zeros(d,n);          %(dxn)-Matrix speichert alle Gradienten
12   grad_ave=zeros(d,1);            %Speichern des Mittelwertes der Grad.
13   indizes=randi(n,num_iter,1);    %Vektor mit zufaelligen Indizes
14   for t=1:num_iter
15      tic();
16      %zufaelliger Index
17      j=indizes(t);
18      grad=problem.grad(w(:,t),j);
19      %Berechnen des neuen Durchschnittes
20      grad_ave= (grad_ave + grad - grad_array(:,j))/t;
21      %den neuen Gradienten abspeichern
22      grad_array(:,j)=grad;
23      %Gradientenlaenge auf 1 skalieren
24      if (norm(grad_ave)>1)
25        grad_ave=grad_ave/norm(grad_ave);
26      end
27      w(:,t+1)=w(:,t) - alpha * grad_ave;
28      T(t+1)=T(t)+toc();
29   end
30   sol=w;
31   time=T;
32 endfunction
```

6 Zusammenfassung

Um Maschinen anzulernen benötigt man Modelle, welche mit einer riesigen Anzahl an Daten gefüttert werden müssen. Für das Berechnen der Parameter dieser Modelle sind deterministische Verfahren, aufgrund des hohen Rechenaufwandes, nicht geeignet. Deswegen werden stochastische Methoden, wie das in dieser Arbeit vorgestellte stochastische Gradientenverfahren, verwendet. Dieses hat den Vorteil, den Aufwand pro Iterationsschritt unabhängig von der Anzahl der Beobachtungen zu machen. Jedoch geht damit der Nachteil einher, dass das Verfahren nicht exakt gegen die Lösung konvergiert. Um das zu beheben, sollte unbedingt eine abfallende Schrittweitenfolge verwendet werden.

Deterministische und stochastische Verfahren können auch kombiniert werden, um die Vorteile beider Seiten zu erlangen. Eine Möglichkeit dafür ist der stochastic average gradient. Dieser berechnet in jedem Schritt zufällig den Gradienten einer Beobachtung, speichert ihn ab und verwendet, zur Schätzung der gesuchten Parameter, den Mittelwert aller bisher berechneten Gradienten. Ein weiterer Vorteil hierbei ist, dass keine abfallende Schrittweitenfolge verwendet werden muss. Der Nachteil, welcher hier einhergeht, ist die verzerrte Berechnung der Gradienten. Das bedeutet, ihr Erwartungswert ist ungleich dem Wert des wahren Gradienten. Trotz dieser Einschränkung gilt der SAG als bahnbrechende Entwicklung in der stochastischen Optimierung. Aufbauend auf diesem wurden in den letzten Jahren mehrere weitere Algorithmen entwickelt, wie zum Beispiel der am Ende dieser Arbeit vorgestellte Jacobian-sketching Algorithmus.

Abbildungsverzeichnis

Literatur

[1] Robert M. Gower. “Introduction to machine learning and stochastic optimization”. Spring School on Optimization and Data Science. 2017.

[2] M. Schmidt, N. Le Roux und F. Bach. “Minimizing Finite Sums with the Stochastic Average Gradient”. In: *Mathematical Programming* (2016).

[3] Francis Bach. “Stochastic gradient methods for machine learning”. INRIA Ecole Normale Superieure. 2012.

[4] Tomas Sauer. “Analysis 1”. Vorlesungsskript Universität Passau. 2016.

[5] Tomas Sauer. “Analysis 2”. Vorlesungsskirpt Universität Passau. 2017.

[6] Tomas Sauer. “Optimierung”. Vorlesungsskript Universität Passau. 2013.

[7] Robert M. Gower. “Convergence Theorems for Gradient Descent”. 2018.

[8] Otto Forster. *Analysis 2*. 8. Aufl. Vieweg+Teubner, 2008. ISBN: 978-3-8348-9541-7.

[9] Imgur, Hrsg. URL: https://i.imgur.com/uqKsueE.jpg (besucht am 12. 01. 2019).

[10] Pradeep Ravikumar. *Stochastic Optimization Methods*. Hrsg. von Carnegie Mellon School of Computer Science. 2018. URL: http://www.cs.cmu.edu/~pradeepr/convexopt/Lecture_Slides/stochastic_optimization_methods.pdf (besucht am 12. 02. 2019).

[11] Suvrit Sra. “Optimization for Machine Learning”. PKU Summer School on Data Science. 2017.

[12] A. Rakhlin, O. Shamir und K. Sridharan. “Making Gradient Descent Optimal for Strongly Convex Stochastic Optimization”. In: *arXiv:1109.5647v7* (2012).

[13] Lam M. Nguyen u. a. “SGDandHogwild! ConvergenceWithouttheBoundedGradientsAssumption”. In: *arXiv:1802.03801v2* (2018).

[14] R. Gower, P. Richtarika und F. Bach. “Stochastic Quasi-Gradient Methods: Variance Reduction via Jacobian Sketching”. In: *arXiv:1805.02632v1* (2018).

[15] Johann Brandstetter. *Satz ueber Lagrange-Multiplikatoren*. Hrsg. von Universität Wien. URL: https://homepage.univie.ac.at/johann.brandstetter/mathe2_folien/lagrafo.pdf (besucht am 25. 03. 2019).

[16] Chih-Chung Chang und Chih-Jen Lin. *LIBSVM - A Library for Support Vector Machines*. 2017.

[17] Hiroyuki-Kasai. *SGDLibrary*. Hrsg. von GitHub. 2018.

[18] R. Gower und P. Richtarik. "Randomized Iterative Methods for Linear Systems". In: *SIAM Journal on Matrix Analysis and Applications* (2015).

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Bachelorarbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle wörtlich oder sinngemäß übernommenen Ausführungen wurden als solche gekennzeichnet. Weiterhin erkläre ich, dass ich diese Arbeit in gleicher oder ähnlicher Form nicht bereits einer anderen Prüfungsbehörde vorgelegt habe.

Datum	Unterschrift