Westfälische
Wilhelms-Universität
Münster

Fachbereich 10
Mathematik und
Informatik

Untersuchung stochastischer Optimierung zweiter Ordnung für maschinelles Lernen

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science

Westfälische Wilhelms-Universität Münster
Fachbereich Mathematik und Informatik

Institut für Numerische und Angewandte Mathematik

Alexander Betz

Matr. 405421

Betreuer: Prof. Dr. Benedikt Wirth
Betreuender Assistent: Jun.-Prof. Dr. Manuel Friedrich

Zusammenfassung

Die vorliegende Arbeit befasst sich mit Optimierungsproblemen die typischerweise im Bereich des maschinellen Lernens auftreten und stellt für deren Lösung stochastische Verfahren zweiter Ordnung vor. Die vorgestellten Verfahren reduzieren maßgeblich, insbesondere bei der Anwendung in großen Datensätzen, den sonst prohibitiven Rechenaufwand welcher durch die notwendige Berechnung der Hesse-Matrix sowie der Inversen der Hesse-Matrix einer Verlustfunktion entsteht. Der Rechenaufwand pro Iteration der vorgestellten Verfahren ist dabei mit dem Rechen-aufwand von Verfahren erster Ordnung vergleichbar. So weist der vorgestellte Algorithmus LiSSA Laufzeiten auf, die vergleichbar mit den Laufzeiten von Algorithmen wie SVRG (Stochastic Variance Reduced Gradient), SAGA und SDCA (Stochastic Dual Coordinate Ascent) sind. Mit LiSSA-Sample wird zudem eine Variante von LiSSA vorgestellt, die effiziente Verfahren erster Ordnung mit stochastischen Matrix Sampling Methoden verbindet. Für den Fall, dass die Anzahl der Trainingsdaten sehr viel größer ist, als die Dimension des Optimierungsproblems, wird gezeigt, dass LiSSA-Sample der theoretisch schnellste bekannte Algorithmus ist. Weiterhin wird der Spezialfall betrachtet, wenn die zu untersuchende Funktion selbst-konkordant ist. Dazu wird auch ein Algorithmus eingeführt, der, genauso wie in den Fällen LiSSA und LiSSA-Sample, lineare Konvergenz aufweist. Die Arbeit schließt mit einer Simulation, in der LiSSA anhand eines realen Datensatzes implementiert und die experimentellen Ergebnisse mit den zuvor erarbeiteten theoretischen Aussagen verglichen werden.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

In der Optimierung werden seit geraumer Zeit stochastische Algorithmen erster Ordnung verwendet, um Optimierungsprobleme zu lösen, die typischerweise im Bereich des maschinellen Lernens vorkommen. Kontinuierliche Forschungsanstrengungen zu den Konvergenzeigenschaften von Verfahren erster Ordnung haben in den vergangen Jahren zahlreiche Methoden hervorgebracht, darunter adaptive Regularisierung (siehe [4]), Methoden zur Varianzredukion (siehe [5] und [6]), und der duale Koordinatenaufstieg (siehe [7]). Im Rahmen des maschinellen Lernens wurden dagegen Verfahren zweiter Ordnung vergleichsweise selten untersucht. Grund dafür ist vor allem ein signifikant höherer Rechenaufwand, da neben dem Gradienten sowohl die Hesse-Matrix als auch die Inverse der Hesse-Matrix berechnet werden müssen. Vor allem für Optimierungsprobleme auf großen Datensätzen mit hohen Dimensionen, welche häufig in praktischen Anwendungen vorkommen, stellt ein derart hoher Rechenaufwand eine reale Hürde dar.

Diese Arbeit basiert zu einem Großteil auf dem Paper "Second-Order Stochastic Optimization for Machine Learning in Linear Time" (siehe [1]) und stellt für konvexe Optimierungsprobleme Algorithmen zweiter Ordnung vor. Diese Algorithmen, darunter LiSSA (*Linear (time) Stochastic Second-Order Algortihm*), weisen schnelle Konvergenzraten und gleichzeitig Rechenaufwände auf, die für jeden Iterationsschritt in linearer Zeit erfolgen. Dadurch ergeben sich Laufzeiten, die vergleichbar sind mit oft bevorzugten Verfahren erster Ordnung. Für den Fall, dass die Größe des Datensates m sehr viel größer ist als dessen Dimension d, weist ein weiterer Algorithmus zweiter Ordnung, *LiSSA-Sample*, sogar schnellere Laufzeiten auf als herkömmliche Gradienten-basierte Methoden erster Ordnung (siehe [1]).

Das Optimierungsproblem, dass in dieser Arbeit betrachtet wird, ist das empirische Risiko Minimierungsproblem (ERM, in englisch "empirical risk minimization")):

$$\min_{x\in\mathbb{R}^d} f(x) = \min_{x\in\mathbb{R}^d} \left\{ \frac{1}{m} \sum_{k=1}^{m} f_k(x) + R(x) \right\}, \tag{1.1}$$

mit $f : \mathbb{R}^d \to \mathbb{R}$, wobei jedes $f_k : \mathbb{R}^d \to \mathbb{R}$ eine konvexe Funktion ist und $R : \mathbb{R}^d \to \mathbb{R}$ ein konvexer Regularisierer. Weitere Informationen zur empirischen Risikominimierung finden sich unter anderem in [8], S.55. Das Minimierungsproblem in (1.1) tritt typischerweise in Anwendungen des überwachten maschinellen Lernens auf, wo zu jedem Datenpunkt ein Label vorliegt, welches im Rahmen des jeweiligen Problems dem Datenpunkt seinen Wahrheitsgehalt (beispielsweise auf die Frage "ist in diesem Bild ein Auto zu sehen?") eindeutig zuordnet. Prominente Methoden in diesem Bereich sind etwa die logistische Regression und SVM (Support Vector Machine). In Kapitel 6 wird eine Implementierung des Algorithmus LiSSA gezeigt, bei der die logistische Regressionsfunktion verwendet wird. Bei vielen Anwendungen von ERM ist es üblich, dass die Verlustfunktion $f_k(x)$ die Form $l(x^T v_k, y_k)$ annimmt, wobei (v_k, y_k) das k-te Datentupel aus dem Datensatz angibt. Solch eine Funktion wird generalisiertes lineares Modell genannt (GLM). Diese wurden 1972 in der Statistik von Robert Wedderburn und John Nedler eingeführt um klassische lineare Regressionsmodelle zu verallgemeinern. Diese Mastarbeit widmet sich in den folgenden Kapiteln unter anderem diesen Funktionen. Zur Vereinfachung wird für den Regularisierungsterm $R(x)$ der ℓ_2 Regularisierer gewählt, speziell $R(x) = \lambda||x||^2$, wobei $\lambda > 0$ der Regularisierungsparameter ist. Beim Ansatz zur Lösung des Minimierungsproblems konzentriert sich diese Arbeit auf das Newton-Verfahren. Hierzu wird in der gesamten Arbeit folgende Notation verwendet: $\nabla^{-2} f(x) := \left(\nabla^2 f(x)\right)^{-1}$. Das Newton-Verfahren hat folgende Darstellung:

$$x_{t+1} = x_t - \nabla^{-2} f(x_t) \nabla f(x), \quad t \geq 0. \tag{1.2}$$

Für das Verfahren kann ein quadratisches Konvergenzverhalten gezeigt werden (siehe [9]). Der Rechenaufwand pro Iterationsschritt beträgt $\Omega(md^2 + d^\omega)$. Der erste Term entsteht durch die Berechnung der Hesse-Matrix und der zweite Term durch die Berechnung der Inversen, wobei $\omega \approx 2,37$ die Matrixmultiplikationskonstante ist (siehe [1]). Diese beiden Probleme werden durch einige der in dieser Arbeit vorgestellten Algorithmen gelöst. Insbesondere wird ein Schätzer für die Inverse der Hesse-Matrix definiert, der in Zeit proportional zu $O(d)$ berechnet werden kann. Dieser berechnet die Inverse der Hesse-Matrix auf stochastische Weise und wird im LiSSA Algorithmus verwendet. Durch die Berechnung der Hesse-Matrix auf stochastische Weise wird der Rechenaufwand maßgeblich verringert. Diese Algorithmen sind, wie schon erwähnt, in der Theorie vergleichbar mit Verfahren erster Ordnung. Tabelle 1.1 zeigt einen Überblick und beinhaltet ebenfalls die beiden in späteren Kapiteln vorgestellten Algorithmen LiSSA und LiSSA-Sample. Die Laufzeiten der Algorithmen hängen von verschiedenen Konditionszahlen ab. Diese werden in Kapitel 2 definiert und sind üblicherweise kleiner als $1/\lambda$ (siehe [1]). Wie zu erkennen ist, hat LiSSA eine vergleichbare Laufzeit zu Algorithmen erster Ordnung SVRG, SAGA und SDCA. Im Fall eines Datensatzes mit Dimensionensverhältnissen $m >> d$ ist LiSSA-Sample der theoretisch schnellste bekannte Algorithmus (nach [1], S. 6).

Algorithmus	Laufzeit	Referenz
SVRG, SAGA, SDCA	$(md + O(\hat{\kappa}d)) \log\left(\frac{1}{\varepsilon}\right)$	[5],[15],[16]
LiSSA	$(md + O(\hat{\kappa}_l)S_1 d) \log\left(\frac{1}{\varepsilon}\right)$	Korollar 3.6
AccSDCA, Catalyst, Katyusha	$\tilde{O}(md + d\sqrt{\hat{\kappa}m}) \log\left(\frac{1}{\varepsilon}\right)$	[17],[2],[18]
LiSSA-Sample	$\tilde{O}(md + d\sqrt{\kappa_{sample}\bar{d}}) \log^2\left(\frac{1}{\varepsilon}\right)$	Theorem 4.11

Tabelle 1.1.: Vergleich der Laufzeiten verschiedener Algorithmen. S_1 ist ein Parameter von LiSSA, zu finden in Kapitel 3 in Algorithmus 1. Die Konditionszahlen sind in Kapitel 2 definiert. Entnommen aus [1].

Der verbleibende Teil der Arbeit gliedert sich wie folgt. Nach einer Analyse der Algorithmen LiSSA und LiSSA-Sample in den Kapiteln 3 und 4, werden in Kapitel 5 selbst-konkordante Funktionen betrachtet, welche nicht von Konditionszahlen abhängen. Kapitel 6 bildet letztlich eine Simulationsstudie, in welcher der LiSSA Algorithmus anhand eines Beispieldatensatzes implementiert und die experimentellen Ergebnisse mit den zuvor erarbeiteten theoretischen Aussagen abgeglichen werden.

KAPITEL 2

Mathematische Hilfsmittel

Dieses Kapitel definiert relevante Notationen und beinhaltet Annahmen welche, sobald nicht anders erwähnt, im verbleibenden Teil der Arbeit gelten. Zunächst sei angemerkt, dass Skalare, Vektoren, sowie Matrizen alle im nicht-fett gedruckten Stil vorkommen. Im Folgenden werden einige mathematische Zusammenhänge definiert und Theoreme vorgestellt, die im analytischen Teil der kommenden Kapitel notwendig sind.

Sei $||\cdot||$ die ℓ_2 Norm für Vektoren und die Spektralnorm für Matrizen. Dies ist im Folgenden formal definiert.

Definition (Matrixnorm, siehe [11]). Unter einer Matrixnorm versteht man eine Funktion $||\cdot|| : \mathbb{R}^{d\times d} \to \mathbb{R}$ mit folgenden Eigenschaften:

a) $||A|| \geq 0$ für alle $A \in \mathbb{R}^{d\times d}$,
$||A|| = 0$ genau dann, wenn $A = 0$

b) $||\lambda A|| = |\lambda| \cdot ||A||$ für alle $\lambda \in \mathbb{R}, A \in \mathbb{R}^{d\times d}$

c) $||A + B|| \leq ||A|| + ||B||$ für alle $A, B \in \mathbb{R}^{d\times d}$

d) $||A \cdot B|| \leq ||A|| \cdot ||B||$ für alle $A, B \in \mathbb{R}^{d\times d}$

Definition (Spektralnorm, siehe [11]). Die Spektralnorm für eine Matrix $A \in \mathbb{R}^{dxd}$ ist definiert als $||A|| := \sqrt{\lambda_{\max}(A^\top A)}$, wobei $\lambda_{\max}(A^\top A)$ der größte Eigenwert von $A^\top A$ ist. Falls A symmetrisch ist, gilt $||A|| = \max_i \{|\lambda_i| \mid \lambda_i \text{ Eigenwert von } A\}$.

Weiterhin gilt für alle $A \in \mathbb{R}^{d\times d}$ und $x \in \mathbb{R}^d$ (siehe [11]):

$$||Ax|| \leq ||A|| \cdot ||x||.$$

Unter der Euklidischen Norm versteht man für $x \in \mathbb{R}^d$:

$$||x|| := \left(\sum_{i=1}^{d} |x_i|^2\right)^{1/2}.$$

Fakt (Neumann-Reihe, siehe [12], S. 55). Es wird der Fakt benutzt, dass die Inverse einer Matrix $A \in \mathbb{R}^{dxd}$, mit den Bedingungen $||A|| \leq 1$ und $A \succ 0$, folgende Darstellung hat:

$$A^{-1} = \sum_{i=0}^{\infty} (I - A)^i.$$

Definition (α-strenge Konvexität und β-Glattheit, siehe [1]). Sei f eine konvexe Funktion. Die Funktion f ist α-streng konvex und β-glatt, falls für alle $x, y \in \mathbb{R}^d$ gilt:

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

Dazu werden die folgenden Zusammenhänge bemerkt (siehe [13], S.281): β-Glattheit ist äquivalent dazu, dass der größte Eigenwert der Hesse-Matrix von f kleiner als β in jedem Punkt ist, das heißt es gilt:

$$\nabla^2 f(x) \preceq \beta I \quad \forall x \in \mathbb{R}^d.$$

Außerdem ist α-strenge Konvexität äquivalent zu:

$$\nabla^2 f(x) \succeq \alpha I \quad \forall x \in \mathbb{R}^d.$$

Im Folgenden werden die verschiedenen Konditionszahlen für eine Funktion f definiert (siehe [1]), die in den Laufzeiten der Algorithmen vorkommen. Definiere für eine α-streng konvexe und β-glatte Funktion f die Konditionszahl der Funktion als $\kappa(f) := \frac{\beta}{\alpha}$. Nach Definition entspricht dies der folgenden Konditionszahl:

$$\kappa := \frac{\max_x \lambda_{\max}(\nabla^2 f(x))}{\min_x \lambda_{\min}(\nabla^2 f(x))}.$$

Zu dieser "globalen" Konditionszahl kann eine "lokale" definiert werden:

$$\kappa_l := \max_x \frac{\lambda_{\max}(\nabla^2 f(x))}{\lambda_{\min}(\nabla^2 f(x))},$$

mit $\kappa_l \leq \kappa$. Diese beiden Konditionszahlen sind für beliebige Funktionen f definiert. Da jedoch die in dieser Arbeit betrachtete Funktion die Form $f(x) = \frac{1}{m}\sum_{k=1}^{m} f_k(x)$ hat, werden dazu speziellere Konditionszahlen definiert. Für Funktionen dieser Form wird typischerweise angenommen, dass jede Komponente beschränkt ist durch $\beta_{\max}(x) := \max_k \lambda_{\max}(\nabla^2 f_k(x))$.

Für die oben genannte Form von f wird dann die folgende Konditionszahl definiert:

$$\hat{\kappa} := \frac{\max_x \beta_{\max}(x)}{\min_x \lambda_{\min}(\nabla^2 f(x))}$$

Wie aus Tabelle 1.1 zu erkennen ist, hängen die Laufzeiten von Algorithmen wie SVRG von dieser Konditionszahl ab. Ähnlich wird dazu eine lokale Konditionszahl für $\hat{\kappa}$ definiert:

$$\hat{\kappa}_l := \max_x \frac{\beta_{\max}(x)}{\lambda_{\min}(\nabla^2 f(x))},$$

wobei wieder $\hat{\kappa}_l \leq \hat{\kappa}$ gilt. Für Schranken der Varianz wird außerdem eine Schranke $\alpha_{\min}(x) := \min_k \lambda_{\min}(\nabla^2 f_k(x))$ für die strenge Konvexität jeder Komponente benötigt.

$$\hat{\kappa}_l^{\max} := \max_x \frac{\beta_{\max}(x)}{\alpha_{\min}(x)}$$

Diese Konditionszahl wird beispielsweise in Algorithmus 1, LiSSA, verwendet und kommt dort unter anderem im Parameter S_1 vor.
Es ist anzumerken, dass alle hier definierten Konditionszahlen kleiner als $1/\lambda$ sind (siehe [1]).

Zur Vereinfachung der Analysis werden außerdem einige Annahmen getroffen die, falls nicht anders erwähnt, in der gesamten Arbeit verwendet werden. Zuerst ist die Funktion $f(x)$ von der Form $f(x) = \frac{1}{m}\sum_{k=1}^{m} f_k(x)$. Der Regularisierungsterm $R(x) = \lambda||x||^2$ wird dabei als Teil von $f_k(x)$ aufgefasst. Weiterhin wird die Annahme $\frac{1}{\hat{\kappa}_l} I \preceq \nabla^2 f_k(x) \preceq I$ getroffen. Diese Skalierung ist ohne Beschränkung der Allgemeinheit, da weitere auftretende Fehler in logarithmischen Termen auftreten wegen der linearen Konvergenz (siehe [1]).
Anmerkung: Die Annahme $\frac{1}{\hat{\kappa}_l} I \preceq \nabla^2 f_k(x)$ fehlt in [1]. Diese wird jedoch für die Analysis von LiSSA in Kapitel 3 benötigt, speziell in dem Beweis für die Konvergenzrate (Theorem 3.2 und die dazugehörigen Lemma 3.3 und Lemma 3.4).

Sei weiterhin f α-streng konvex, β-glatt und sei $\hat{\kappa}_l$ die dazugehörige Konditionszahl. Sei außerdem $\nabla^2 f(x)$ Lipschitz-stetig mit Lipschitz-Konstante M, das heißt:

$$||\nabla^2 f(x) - \nabla^2 f(y)|| \leq M||x - y|| \quad \forall\, x, y \in \mathbb{R}^d.$$

Das folgende Theorem ist eine Standardkonzentration für Summen von unabhängigen Matrizen. Dieses Theorem ist eine Abänderung des Matrix Bernstein Theorems, das in [10] (genauer in [10], Seite 23 Theorem 6.1) zu finden ist. Dort ist die Ungleichung für den beschränkten Varianzfall beschrieben. Hier wird eine vereinfachte Variante definiert, bei der in den Voraussetzungen nur die Spektralnorm vorkommmt und beschränkt ist (siehe [1], Seite 9).

Theorem 2.1 (Matrix Bernstein, siehe [10]). Sei X_k eine endliche Folge von unabhängigen hermiteschen Zufallsmatrizen mit Dimensionen d x d. Es gelte

$$\mathbb{E}[X_k] = 0 \text{ und } ||X_k|| \leq R.$$

Definiere $Y = \sum_k X_k$. Dann folgt für alle $t \geq 0$,

$$\mathbf{Pr}(||Y|| \geq t) \leq d \exp\left(\frac{-t^2}{4R^2}\right).$$

Das folgende Theorem wurde in [2] gezeigt und bewiesen.

Theorem 2.2 (siehe [2]). Sei $f(x) = \frac{1}{m}\sum_{k=1}^{m} f_k(x)$ gegeben mit der Konditionszahl κ. Die beschleunigte Version von SVRG via Catalyst findet ein ε-approximiertes Minimum mit Wahrscheinlichkeit $1 - \delta$ in der Zeit

$$\tilde{O}(md + \min(\sqrt{\kappa m}, \kappa)d \log\left(\frac{1}{\varepsilon}\right).$$

Sherman-Morrison Formel (siehe [1]). Die folgende Formel gilt für Rang-1-Änderungen von einer beliebigen Matrix $A \in \mathbb{R}^{dxd}$ und Vektor $v \in \mathbb{R}^d$:

$$\left(A + vv^\top\right)^{-1} = A^{-1} - \frac{A^{-1}vv^\top A^{-1}}{1 + v^\top A^{-1}v}.$$

KAPITEL 3

LiSSA

In diesem Kapitel wird der Algorithmus LiSSA (Linear (time) Stochastic Second-Order Algortihm) und die dazugehörigen Konvergenzeigenschaften beschrieben und erläutert. Außerdem werden die Laufzeiten des Algorithmus gezeigt und als letztes wird eine Sparsity Variante betrachtet, bei der die Inputvektoren dünnbesetzt sind. Zunächst wird jedoch ein Schätzer für die Inverse der Hesse-Matrix einer Funktion definiert und in den Algorithmus eingebaut.

3.1. Schätzer der Inversen einer Hesse-Matrix

Zur Definition des LiSSA Algorithmus bedarf es zunächst einmal der Definition der Inversen einer Matrix. Für die Inverse einer Matrix A mit $||A|| \leq 1$ und $A \succ 0$ gilt:

$$A^{-1} = \sum_{i=0}^{\infty} (I - A)^i \tag{3.1}$$

Ausgehend von dieser Definition, definiere A_j^{-1} als die ersten j Terme in dieser Reihe, das heißt:

$$A_j^{-1} = \sum_{i=0}^{j} (I - A)^i \qquad \Leftrightarrow \qquad A_j^{-1} = I + (I - A)A_{j-1}^{-1}. \tag{3.2}$$

Damit gilt $\lim_{j \to \infty} A_j^{-1} = A^{-1}$. Mithilfe der rekursiven Formulierung in (4.2) kann nun ein unvoreingenommener Schätzer der Inversen der Hesse-Matrix $\nabla^{-2} f$ einer Funktion f beschrieben werden, indem zuerst ein unvoreingenommener Schätzer $\widetilde{\nabla}^{-2} f_j$ für $\nabla^{-2} f_j$ definiert wird.

Definition 3.1. Seien j unabhängige und unvoreingenommene Proben $X_1, ..., X_j$ der Hesse Matrix $\nabla^2 f$ gegeben. Definiere $\widetilde{\nabla}^{-2} f_0, ..., \widetilde{\nabla}^{-2} f_j$ rekursiv wie folgt:

$$\widetilde{\nabla}^{-2} f_0 = I, \quad \widetilde{\nabla}^{-2} f_t = I + (I - X_t)\widetilde{\nabla}^{-2} f_{t-1}, \quad \text{für } t = 1, ..., j. \tag{3.3}$$

Für den Erwartungswert von $\widetilde{\nabla}^{-2} f_j$ gilt

$$\begin{aligned} \mathrm{E}[\widetilde{\nabla}^{-2} f_j] &= \mathrm{E}[I + (I - X_j)\widetilde{\nabla}^{-2} f_{j-1}] \\ &= I + (I - \mathrm{E}[X_j])\mathrm{E}[\widetilde{\nabla}^{-2} f_{j-1}] \\ &= I + (I - \nabla^2 f)\widetilde{\nabla}^{-2} f_{j-1} \\ &= \nabla^{-2} f_j, \end{aligned} \tag{3.4}$$

wobei der letzte Schritt rekursiv erfolgt. Damit gilt $\lim_{j\to\infty} \mathrm{E}[\widetilde{\nabla}^{-2} f_j] = \nabla^{-2} f$ nach Definition der Inversen aus (4.2) und (4.1), sodass man auch im Limes einen unvoreingenommenen Schätzer erhält.
Mithilfe dieser Überlegungen für die Inverse einer Hesse-Matrix kann nun der Algorithmus LiSSA formuliert werden.

3.2. LiSSA Algorithmus

Der LiSSA (Linear (time) Stochastic Second-Order Algorithm) Algorithmus besteht aus zwei Teilen. Im ersten Teil läuft eine beliebige effiziente Methode erster Ordnung FO für T_1 Schritte um eine erste Lösung des Optimierungsproblems zu erreichen, sodass die Genauigkeit in einem bestimmten Bereich liegt. Im zweiten Teil wird ein approximiertes Newton Verfahren durchgeführt, indem anstelle der wahren Hesse-Matrix die Approximation aus Definition 3.1 verwendet wird. Der LiSSA Algorithmus ist als Pseudocode in Algorithmus 1 gegeben.

Algorithmus 1 LiSSA (Linear (time) Stochastic Second-Order Algorithm)

Input: $T,\ f(x) = \frac{1}{m}\sum_{k=1}^{m} f_k(x),\ S_1,\ S_2,\ T$
$x_1 = FO(f(x), T_1)$
for $t = 1$ to T **do**
  for $i = 1$ to S_1 **do**
    $X_{[i,0]} = \nabla f(x_t)$
    for $j = 1$ to S_2 **do**
      Ziehe $\widetilde{\nabla}^2 f_{[i,j]}(x_t)$ gleichmäßig aus $\{\nabla^2 f_k(x_t) \mid k \in \{1, ..., m\}\}$
      $X_{[i,j]} = \nabla f(x_t) + (I - \widetilde{\nabla}^2 f_{[i,j]}(x_t)) X_{[i,j-1]}$
    end for
  end for
  $X_t = \frac{1}{S_1}\sum_{i=1}^{S_1} X_{[i,S_2]}$
  $x_{t+1} = x_t - X_t$
end for
return x_{T+1}

Für den Algorithmus werden zwei Parameter, S_1 und S_2, verwendet um einen Schritt des Newton Verfahrens zu berechnen. S_1 repräsentiert dabei die Anzahl der unvoreingenommenen Schätzer der Inversen der Hesse-Matrix, die gemittelt werden um eine bessere Konzentration für den Schätzer zu erhalten. S_2 gibt die Anzahl an Summanden für den Schätzer an.

3.3. Konvergenzeigenschaften des LiSSA Algorithmus

Im Folgenden wird die Konvergenzeigenschaft des LiSSA Algorithmus gezeigt und bewiesen. Dafür benutzen wir folgende Notation: $FO(M, \hat{\kappa}_l)$ definiert die gesamte Zeit, die ein gewählter Algorithmus erster Ordnung in LiSSA benötigt um eine Lösung der Genauigkeit von mindestens $\frac{1}{4\hat{\kappa}_l M}$ zu erzielen. Die Konvergenzeigenschaft von LiSSA ist in Theorem 3.3 gegeben.

Theorem 3.2. Betrachte Algorithmus 1 und seien die Parameter wie folgt gegeben: $T_1 = FO(M, \hat{\kappa}_l), S_1 \geq \frac{64}{3}\hat{\kappa}_l^{\max}\sqrt{\ln(\frac{d}{\delta})}$ und $S_2 \geq 2\hat{\kappa}_l \ln(4\hat{\kappa}_l)$, mit $\delta \in [0, 1]$. Dann gilt für alle $t \geq 1$ mit Wahrscheinlichkeit mindestens $1 - \delta$:

$$||x_{t+1} - x^*|| \leq \frac{||x_t - x^*||}{2} \tag{3.5}$$

Anmerkung: Der Parameter S_1 hat einen anderen Wert als in [1]. Der Grund dafür ist das Einsetzen des Wertes in γ im letzten Schritt des Beweises in (4.12), was in [1] nicht den gewünschten Vorfaktor ergibt.
Beweis: Da zu Beginn von Algorithmus 1 ein erste-Ordnung Algorithmus für $T_1 = FO(M, \hat{\kappa}_l)$ Schritte läuft um eine Lösung der Genauigkeit von mindestens $\frac{1}{4\hat{\kappa}_l M}$ zu erhalten, gilt:

$$||x_1 - x^*|| \leq \frac{1}{4\hat{\kappa}_l M} \tag{3.6}$$

Wie man mit Definition 3.1 sieht, ist ein einzelner Iterationsschritt von Algorithmus 1 äquivalent zu $x_{t+1} = x_t - \widetilde{\nabla}^{-2} f(x_t) \nabla f(x_t)$, wobei $\widetilde{\nabla}^{-2} f(x_t)$ der Durchschnitt von S_1 unabhängigen Schätzern $\widetilde{\nabla}^{-2} f(x_t)_{S_2}$ ist, das heißt:

$$\widetilde{\nabla}^{-2} f(x_t) := \frac{1}{S_1} \sum_{i=1}^{S_1} \widetilde{\nabla}^{-2} f(x_t)_{S_2}, \tag{3.7}$$

$$\text{mit } \widetilde{\nabla}^{-2} f(x_t)_{S_2} = I + \left(I - \widetilde{\nabla}^2 f_{[i,S_2]}(x_t)\right) \widetilde{\nabla}^{-2} f(x_t)_{S_2 - 1} \tag{3.8}$$

nach Definition 3.1. Für den weiteren Verlauf des Beweises wird folgendes Lemma benötigt, welches im Anschluss bewiesen wird.

Lemma 3.3. Sei $x_{t+1} = x_t - \widetilde{\nabla}^{-2} f(x_t) \nabla f(x_t)$ mit der Notation aus (4.7) und (4.8). Seien S_1 und S_2 aus Algorithmus 1 gegeben wie in Theorem 3.2. Dann gilt für alle $t \geq 1$ mit Wahrscheinlichkeit mindestens $1 - \delta,\ \delta \in [0, 1]$, die folgende Konvergenzrate:

$$||x_{t+1} - x^*|| \leq \gamma ||x_t - x^*|| + M||\nabla^{-2} f(x_t)||||x_t - x^*||^2, \tag{3.9}$$

mit $\gamma = 4\hat{\kappa}_l^{\max} \dfrac{\sqrt{\ln(\frac{d}{\delta})}}{S_1} \ + \ \frac{1}{16}$.

Da alle Voraussetzungen aus Lemma 3.3 erfüllt sind, kann die Konvergenzrate (4.9) nun zu Nutze gemacht werden. Mit der Wahl von S_1 kann γ nach oben abgeschätzt werden:

$$\gamma \leq \frac{1}{4}. \tag{3.10}$$

Aufgrund der Annahme

$$\nabla^2 f(x_t) \succeq \frac{1}{\hat{\kappa}_l} I \ \Leftrightarrow \ \nabla^{-2} f(x_t) \preceq \hat{\kappa}_l I$$

aus Kapitel 2 gilt für die Spektralnorm der inversen Hesse-Matrix:

$$||\nabla^{-2} f(x_t)|| \leq \hat{\kappa}_l. \tag{3.11}$$

Mit (4.6),(4.10) und (4.11) folgt nun mit Lemma 3.3 für die Konvergenzrate (4.9):

$$\begin{aligned} ||x_{t+1} - x^*|| \ &\leq \ \frac{||x_t - x^*||}{4} \ + \ M\hat{\kappa}_l ||x_t - x^*||^2 \\ &\leq \ \frac{||x_t - x^*||}{4} \ + \ \frac{||x_t - x^*||}{4} \\ &= \ \frac{||x_t - x^*||}{2}. \end{aligned} \tag{3.12}$$

Die zweite Ungleichung folgt dadurch, dass $||x_t - x^*|| \leq ||x_1 - x^*||$ für alle $t \geq 1$ gilt. Dies kann per Induktion gezeigt werden. Der Fall $t = 1$ ist trivial. Sei $t = 2$:

$$\begin{aligned} ||x_2 - x^*|| \ &\leq \ \frac{||x_1 - x^*||}{4} + M\hat{\kappa}_l ||x_1 - x^*||^2 \\ &\leq \ \frac{||x_1 - x^*||}{2} \ \leq \ ||x_1 - x^*||. \end{aligned}$$

Induktionsschritt: Es gelte $||x_t - x^*|| \leq ||x_1 - x^*||$ für alle natürlichen Zahlen bis t und zeige den Beweis für $t+1$:

$$\begin{aligned} ||x_{t+1} - x^*|| &\leq \frac{||x_t - x^*||}{4} + M\hat{\kappa}_l||x_t - x^*||^2 \\ &\leq \frac{||x_1 - x^*||}{4} + M\hat{\kappa}_l||x_1 - x^*||^2 \\ &\leq \frac{||x_1 - x^*||}{2} \leq ||x_1 - x^*||, \end{aligned}$$

wobei im zweiten Schritt die Induktionsannahme eingegangen ist. Damit ist Theorem 3.2 bewiesen und die lineare Konvergenzrate des LiSSA Algorithmus gezeigt.
Als nächstes wird Lemma 3.3 bewiesen.
Beweis zu Lemma 3.3: Definiere $\chi(x_t) := \int_0^1 \nabla^2 f(x^* + \tau(x_t - x^*)d\tau$, womit man $\nabla f(x_t) = \chi(x_t)(x_t - x^*)$ erhält. Mit der Definition für x_{t+1} folgt dann:

$$\begin{aligned} ||x_{t+1} - x^*|| &= ||x_t - x^* - \widetilde{\nabla}^{-2} f(x_t)\nabla f(x_t)|| \\ &= ||x_t - x^* - \widetilde{\nabla}^{-2} f(x_t)\chi(x_t)(x_t - x^*)|| \\ &= ||(I - \widetilde{\nabla}^{-2} f(x_t)\chi(x_t))(x_t - x^*)|| \\ &\leq ||I - \widetilde{\nabla}^{-2} f(x_t)\chi(x_t)||||x_t - x^*|| \end{aligned} \tag{3.13}$$

Nach einer Umformulierung der ersten Norm im obigen Term erhält man:

$$\begin{aligned} ||I - \widetilde{\nabla}^{-2} f(x_t)\chi(x_t)|| &= ||I - \nabla^{-2} f(x_t)\chi(x_t) - \left(\widetilde{\nabla}^{-2} f(x_t) - \nabla^{-2} f(x_t)\right)\chi(x_t)|| \\ &\leq ||I - \nabla^{-2} f(x_t)\chi(x_t)|| + ||\left(\widetilde{\nabla}^{-2} f(x_t) - \nabla^{-2} f(x_t)\right)\chi(x_t)|| \end{aligned}$$

Der erste Term kann nun mit der Definition von $\chi(x_t)$ weiter abgeschätzt werden:

$$\begin{aligned} ||I - \nabla^{-2} f(x_t)\chi(x_t)|| &= ||\nabla^{-2} f(x_t) \int_0^1 \nabla^2 f(x_t) - \nabla^2 f(x^* + \tau(x_t - x^*)\ d\tau|| \\ &\leq ||\nabla^{-2} f(x_t)||\left|\left|\int_0^1 \nabla^2 f(x_t) - \nabla^2 f(x^* + \tau(x_t - x^*)\ d\tau\right|\right| \\ &\leq ||\nabla^{-2} f(x_t)||\int_0^1 ||\nabla^2 f(x_t) - \nabla^2 f(x^* + \tau(x_t - x^*)||\ d\tau \end{aligned}$$

Wegen der Lipschitz-Stetigkeit der Hesse Matrix folgt dann:

$$\begin{aligned} ||I - \nabla^{-2} f(x_t)\chi(x_t)|| &\leq ||\nabla^{-2} f(x_t)||\int_0^1 M||x_t - x^* - \tau(x_t - x^*)||\ d\tau \\ &= ||\nabla^{-2} f(x_t)||M||x_t - x^*||\int_0^1 1 - \tau\ d\tau \\ &\leq M||\nabla^{-2} f(x_t)||||x_t - x^*|| \end{aligned} \tag{3.14}$$

Eine Abschätzung des zweiten Terms ergibt

$$\begin{aligned} ||(\widetilde{\nabla}^{-2} f(x_t) - \nabla^{-2} f(x_t))\chi(x_t)|| &\leq ||(\widetilde{\nabla}^{-2} f(x_t) - \nabla^{-2} f(x_t))||||\chi(x_t)|| \\ &\leq ||(\widetilde{\nabla}^{-2} f(x_t) - \nabla^{-2} f(x_t))||, \end{aligned} \tag{3.15}$$

wobei die zweite Ungleichung wegen der Annahme $\nabla^2 f_k(x) \preceq I$ für alle $x \in \mathbb{R}^d$ beziehungsweise $||\nabla^2 f(x)|| \leq 1$ für alle $x \in \mathbb{R}^d$ und

$$\begin{aligned} ||\chi(x_t)|| &\leq \int_0^1 ||\nabla^2 f(x^* + \tau(x_t - x^*)||\, d\tau \\ &\leq \int_0^1 1\, d\tau = 1 \end{aligned}$$

folgt. Für die Abschätzung von (4.15) wird nun ein zusätzliches Lemma benötigt.

Lemma 3.4. Sei $\widetilde{\nabla}^{-2} f(x_t)$ der Durchschnitt von S_1 unabhängigen Schätzern $\widetilde{\nabla}^{-2} f(x_t)_{S_2}$, definiert wie in (4.7) und (4.8). Sei $\nabla^2 f(x_t)$ die Hesse-Matrix von f und sei S_2 aus Algorithmus 1 so gegeben wie in Theorem 3.2, das heißt $S_2 \geq 2\hat{\kappa}_l \ln(4\hat{\kappa}_l)$. Dann gilt:

$$\Pr\left(||\widetilde{\nabla}^{-2} f(x_t) - \nabla^{-2} f(x_t)|| > \gamma\right) \leq \delta, \tag{3.16}$$

mit $\gamma = 4\hat{\kappa}_l^{\max} \frac{\sqrt{\ln(\frac{d}{\delta})}}{S_1} \; + \; \frac{1}{16}$.

Beweis. Als erstes ist anzumerken, dass man aufgrund der Konstruktion des Schätzers $\widetilde{\nabla}^{-2} f(x_t)$ folgenden Erwartungswert erhält:

$$\begin{aligned} E[\widetilde{\nabla}^{-2} f(x_t)] &= E[\frac{1}{S_1} \sum_{i=1}^{S_1} \widetilde{\nabla}^{-2} f(x_t)_{S_2}] \\ &= \frac{1}{S_1} \sum_{i=1}^{S_1} E[\widetilde{\nabla}^{-2} f(x_t)_{S_2}] \\ &= \nabla^{-2} f(x_t)_{S_2} \\ &= \sum_{j=0}^{S_2} (I - \nabla^2 f(x_t))^j, \end{aligned} \tag{3.17}$$

wobei die dritte Äquivalenz nach (4.4) folgt und die vierte die Definition aus (4.2) benutzt. Wegen $\nabla^2 f_k \preceq I$ gilt außerdem $||\nabla^2 f_k|| \leq 1$ und damit $||\nabla^2 f|| \leq 1$, sodass die Definition der

Inversen einer Matrix aus (4.1) benutzt werden kann. Damit folgt:

$$\begin{aligned}\nabla^{-2} f(x_t) &= \sum_{j=0}^{S_2} (I - \nabla^2 f(x_t))^j \; + \sum_{j=S_2+1}^{\infty} (I - \nabla^2 f(x_t))^j \\ &= E[\widetilde{\nabla}^{-2} f(x_t)] \; + \sum_{j=S_2+1}^{\infty} (I - \nabla^2 f(x_t))^j \end{aligned} \tag{3.18}$$

Die Norm aus (4.16) kann nun weiter nach oben abgeschätzt werden, sodass man mit (4.18) erhält:

$$||\widetilde{\nabla}^{-2} f(x_t) - \nabla^{-2} f(x_t)|| \leq ||\widetilde{\nabla}^{-2} f(x_t) - E[\widetilde{\nabla}^{-2} f(x_t)]|| + || \sum_{j=S_2+1}^{\infty} (I - \nabla^2 f(x_t))^j || \tag{3.19}$$

Mit der Annahme $\frac{1}{\hat{\kappa}_l} I \preceq \nabla^2 f_k(x_t)$ beziehungsweise $\frac{1}{\hat{\kappa}_l} I \preceq \nabla^2 f(x_t)$ folgt

$$||I - \nabla^2 f(x_t)|| \leq 1 - \frac{1}{\hat{\kappa}_l}, \tag{3.20}$$

sodass man den zweiten Term aus (4.19) wie folgt weiter abschätzen kann:

$$\begin{aligned} || \sum_{j=S_2+1}^{\infty} (I - \nabla^2 f(x_t))^j || &\leq || \sum_{j=S_2}^{\infty} (I - \nabla^2 f(x_t))^j || \\ &\leq \sum_{j=S_2}^{\infty} ||(I - \nabla^2 f(x_t))||^j \\ &= ||(I - \nabla^2 f(x_t))||^{S_2} \left(\sum_{j=0}^{\infty} ||(I - \nabla^2 f(x_t))||^j \right) \\ &\leq (1 - \frac{1}{\hat{\kappa}_l})^{S_2} \left(\sum_{j=0}^{\infty} \left(1 - \frac{1}{\hat{\kappa}_l}\right)^j \right) \\ &\leq (1 - \frac{1}{\hat{\kappa}_l})^{S_2} \hat{\kappa}_l \\ &\leq \exp\left(-\frac{S_2}{\hat{\kappa}_l}\right) \hat{\kappa}_l. \end{aligned}$$

In die erste Ungleichung geht ein, dass $I - \nabla^2 f(x_t) \succeq 0$ gilt wegen $\nabla^2 f(x_t) \preceq I$ und die dritte geht Abschätzung (4.20) mit ein. Die letzte Ungleichung folgt mit dem Fakt: $(1 + x)^\alpha \leq \exp(\alpha x)$ für alle $x \in \mathbb{R}, \alpha \geq 0$. Mit der Wahl des Parameters $S_2 \geq 2\hat{\kappa}_l \ln(4\hat{\kappa}_l)$ erhält man schließlich für den obigen Term:

$$|| \sum_{j=S_2+1}^{\infty} (I - \nabla^2 f(x_t))^j || \; \leq \frac{1}{16}. \tag{3.21}$$

Für den erstem Term aus (4.19) kann mithilfe von Theorem 2.1, der Matrix Bernstein Ungleichung, gezeigt werden, dass der Schätzer $\widetilde{\nabla}^{-2} f(x_t)$ um seinen Erwartungswert konzentriert ist. Dazu müssen zuerst die Voraussetzungen überprüft werden. Sei

$$\begin{aligned}
Y &:= \widetilde{\nabla}^{-2} f(x_t) - E[\widetilde{\nabla}^{-2} f(x_t)] \\
&= \left(\frac{1}{S_1} \sum_{i=1}^{S_1} \widetilde{\nabla}^{-2} f(x_t)_{S_2} \right) - E[\widetilde{\nabla}^{-2} f(x_t)] \\
&= \frac{1}{S_1} \sum_{i=1}^{S_1} \left(\widetilde{\nabla}^{-2} f(x_t)_{S_2} - E[\widetilde{\nabla}^{-2} f(x_t)] \right) \\
&= \frac{1}{S_1} \sum_{i=1}^{S_1} \left(\sum_{j=0}^{S_2} (I - \widetilde{\nabla}^2 f_{[i,j]}(x_t))^j - \sum_{j=0}^{S_2} (I - \nabla^2 f(x_t))^j \right) \\
&= \sum_{i=1}^{S_1} X_i, \quad \text{mit } X_i := \frac{1}{S_1} \left(\sum_{j=0}^{S_2} (I - \widetilde{\nabla}^2 f_{[i,j]}(x_t))^j - \sum_{j=0}^{S_2} (I - \nabla^2 f(x_t))^j \right).
\end{aligned}$$

In die vierte Äquivalenz ist dabei das Resultat für den Erwartungswert aus (4.17) eingegangen. Dann gilt für die erste Voraussetzung:

$$E[X_i] = \frac{1}{S_1} \left(\sum_{j=0}^{S_2} (I - \nabla^2 f(x_t))^j - \sum_{j=0}^{S_2} (I - \nabla^2 f(x_t))^j \right) = 0.$$

Für die zweite Voraussetzung muss eine Abschätzung für die Spektralnorm der Zufallsvariablen vorgenommen werden. Dazu wird bemerkt (siehe [1] S.13), dass $\widetilde{\nabla}^{-2} f(x_t)_{S_2}$ maximale Spektralnorm hat mit

$$||\widetilde{\nabla}^{-2} f(x_t)_{S_2}|| \leq \sum_{j=0}^{S_2} \left(1 - \frac{1}{\hat{\kappa}_l^{\max}} \right)^j \leq \hat{\kappa}_l^{\max}.$$

Es folgt analog zu vorherigen Rechnungen:

$$\begin{aligned}
||X_i|| &\leq \frac{1}{S_1} \left(||\sum_{j=0}^{S_2} (I - \widetilde{\nabla}^2 f_{[i,j]}(x_t))^j|| + ||\sum_{j=0}^{S_2} (I - \nabla^2 f(x_t))^j|| \right) \\
&\leq \frac{1}{S_1} \left(\hat{\kappa}_l^{\max} + \sum_{j=0}^{S_2} \left(1 - \frac{1}{\hat{\kappa}_l} \right)^j \right) \\
&\leq \frac{1}{S_1} (\hat{\kappa}_l^{\max} + \hat{\kappa}_l) \\
&\leq \frac{2}{S_1} \hat{\kappa}_l^{\max}.
\end{aligned}$$

Nun kann Theorem 2.1 angewendet werden, sodass für alle $\epsilon \geq 0$ gilt:

$$\Pr\left(||\widetilde{\nabla}^{-2} f(x_t) - \nabla^{-2} f(x_t)|| > \epsilon\right) \leq d \exp\left(\frac{-\epsilon^2 S_1^2}{16(\hat{\kappa}_l^{\max})^2}\right). \tag{3.22}$$

Wählt man $\epsilon = 4\hat{\kappa}_l^{\max}\frac{\sqrt{\ln(\frac{d}{\delta})}}{S_1}$, ergibt sich, dass die obere Wahrscheinlichkeit nach oben durch δ begrenzt ist.
Schließlich kann gezeigt werden, dass die Wahrscheinlichkeit in (4.16) nach oben durch δ begrenzt ist. Es folgt mit den Überlegungen aus (4.19),(4.21) und (4.22):

$$\begin{aligned}
\Pr\left(||\widetilde{\nabla}^{-2} f(x_t) - \nabla^{-2} f(x_t)|| > \gamma\right) &\leq \Pr\left(||\widetilde{\nabla}^{-2} f(x_t) - E[\widetilde{\nabla}^{-2} f(x_t)]|| + \frac{1}{16} > \epsilon + \frac{1}{16}\right) \\
&= \Pr\left(||\widetilde{\nabla}^{-2} f(x_t) - E[\widetilde{\nabla}^{-2} f(x_t)]|| > \epsilon\right) \\
&\leq \delta.
\end{aligned}$$

Damit ist nun das Haupttheorem 3.2, die lineare Konvergenzrate von Algorithmus 1, und die dazu erforderlichen Lemmas, Lemma 3.3 und 3.4, bewiesen.

Als nächstes wird der Rechenaufwand des Algorithmus pro Iteration gezeigt (siehe [1]).

Lemma 3.5. Jeder Iterationsschritt von Algorithmus 1 benötigt höchstens die Zeit $\tilde{O}(md + \hat{\kappa}_l^{\max}\hat{\kappa}_l d^2)$. Falls f zusätzlich eine GLM Funktion ist, benötigt jeder Iterationsschritt von Algorithmus 1 die Zeit $\tilde{O}(md + \hat{\kappa}_l^{\max}\hat{\kappa}_l d)$. Dabei versteckt $\tilde{O}$ log Terme von $\hat{\kappa}_l, d$ und $\frac{1}{\delta}$. Anmerkung: Da der Parameter S_1 anders gewählt wird als in [1], taucht in der Zeit $\hat{\kappa}_l^{\max}$ auf anstelle von $(\hat{\kappa}_l^{\max})^2$.

Beweis. In jedem Iterationsschritt wird $\nabla f(x_t)$ berechnet, was $O(md)$ Rechenoperationen benötigt. Dann werden zwei Schleifen durchgegangen. In der inneren Schleife kann $\widetilde{\nabla}^2 f_{[i,j]}(x_t) X_{[i,j-1]}$ in $O(d^2)$ Operationen berechnet werden. Dieser Term wird durch die innere Schleife S_2 mal berechnet und aufgrund der äußeren Schleife wiederum S_1 mal. Das heißt es müssen $O(S_1 S_2 d^2)$ Operationen erfolgen. Insgesamt ergibt sich, dass X_t in

$$\begin{aligned}
O(md + S_1 S_2 d^2) &= O(md + \hat{\kappa}_l^{\max}\ln(d/\delta)\hat{\kappa}_l \ln(\hat{\kappa}_l) d^2) \\
&= \tilde{O}(md + \hat{\kappa}_l^{\max}\hat{\kappa}_l d^2)
\end{aligned}$$

Zeit berechnet werden kann.
Falls f eine GLM Funktion ist, haben die Hesse Matrizen $\nabla^2 f_k(x_t)$ die Form $\alpha v_k v_k^T$,
$v \in \mathbb{R}^d$, wobei α ein Skalar ist, der von $v_k^T x$ abhängt. Damit reduziert sich das Matrix-Vektor Produkt $\widetilde{\nabla}^2 f_{[i,j]}(x_t) X_{[i,j-1]}$ auf ein Vektor-Vektor Produkt, sodass es in $O(d)$ Operationen

berechnet werden kann (siehe [1], S.4). Damit erhält man analog zu obigen Rechnungen, dass jeder Iterationsschritt von Algorithmus 1 $\tilde{O}(md + \hat{\kappa}_l^{\max}\hat{\kappa}_l d)$ Zeit benötigt.

Als letztes wird die Laufzeit des Algorithmus gezeigt, welches im folgenden Korollar festgehalten wird (siehe [1]) und eine Schlussfolgerung von Theorem 3.2 ist. Es ist zu beachten, dass hier und in weiteren Theoremen und Korollaren, in denen Wahrscheinlichkeiten vorkommen und Schlussfolgerungen von vorherigen Theoremen sind, verschiedene Werte für δ vorkommen.

Korollar 3.6. Sei f eine GLM Funktion. Dann liefert Algorithmus 1 einen Vektor x_t, sodass mit Wahrscheinlichkeit mindestens $1 - \delta$ gilt:

$$f(x_t) \leq f(x^*) + \epsilon, \tag{3.23}$$

in gesamter Zeit $\tilde{O}(m + \hat{\kappa}_l^{\max}\hat{\kappa}_l)d\ln(\frac{1}{\epsilon})$ für $\epsilon \to 0$. Dabei versteckt $\tilde{O}$ log Terme von $\hat{\kappa}_l, d, M$ und $\frac{1}{\delta}$.
Anmerkung: Aufgrund von Lemma 3.5 taucht in der gesamten Zeit ein $\hat{\kappa}_l^{\max}$ auf anstelle von $(\hat{\kappa}_l^{\max})^2$, wie in [1].

Beweis. Zuerst wird festgehalten, dass die Annahme $\nabla^2 f(x_t) \preceq I$ an die Hesse-Matrix von f äquivalent dazu ist, dass f 1-glatt ist. Das heißt nach der Definition von β-glatten Funktionen in Kapitel 2 erhält man mit $y = x_t,\ x = x^*$ und der Tatsache $\nabla f(x^*) = 0$:

$$f(x_t) - f(x^*) \leq \frac{1}{2}||x_t - x^*||^2.$$

Theorem 3.2 gilt mit Wahrscheinlichkeit $1 - \delta'$. Das rekursive Anwenden von Theorem 3.2 liefert dann:

$$\begin{aligned} f(x_t) - f(x^*) &\leq \frac{1}{2 * 4^{t-1}}||x_1 - x^*||^2 \\ &\leq \frac{1}{4^t * 8\hat{\kappa}_l^2 M^2} \\ &\stackrel{!}{=} \epsilon, \end{aligned}$$

mit Wahrscheinlichkeit $(1 - \delta')^{t-1}$. Die zu zeigende Ungleichung gilt dann mit Wahrscheinlichkeit $1 - \delta$, wobei $\delta = 1 - (1 - \delta')^{t-1}$. Umformen nach der Anzahl der Iterationsschritte

ergibt:

$$\begin{aligned} t &= \frac{1}{\ln(4)} \ln\left(\frac{1}{8\hat{\kappa}_l^2 M^2 \epsilon}\right) \\ &= \frac{1}{\ln(4)} \ln\left(\frac{1}{8\hat{\kappa}_l^2 M^2}\right) + \frac{1}{\ln(4)} \ln\left(\frac{1}{\epsilon}\right) \\ &\leq \left(\ln\left(\frac{1}{8\hat{\kappa}_l^2 M^2}\right) + 1\right) \frac{1}{\ln(4)} \ln\left(\frac{1}{\epsilon}\right) \\ &\leq C \ln\left(\frac{1}{\epsilon}\right), \end{aligned}$$

wobei $C = C(\hat{\kappa}_l, M) > 0$ eine Konstante ist, die von $\hat{\kappa}_l$ und M abhängt. Die gesamte Zeit, die Algorithmus 1 benötigt um das ϵ approximierte Minimum (4.23) zu erreichen, setzt sich dann durch den Rechenaufwand für jeden Iterationsschritt aus Lemma 3.5 und der Anzahl der Iterationen zusammen, das heißt $\tilde{O}(m + \hat{\kappa}_l^{\max} \hat{\kappa}_l) dC \ln(\frac{1}{\epsilon}) = \tilde{O}(m + \hat{\kappa}_l^{\max} \hat{\kappa}_l) d \ln(\frac{1}{\epsilon})$.

3.4. Sparsity Variante

Eine wichtige Eigenschaft für Datensätze aus der Praxis ist, dass obwohl der Input hochdimensionale Vektoren sind, der Großteil der Einträge typischerweise aus Nullen besteht (siehe [1]). Solche Vektoren werden als dünnbesetzt bezeichnet (oder auf englisch "sparse"). Der LiSSA Algorithmus kann sich diese Sparsity Eigenschaft zu Nutze machen. Speziell für GLM Funktionen heißt das, dass das Rang 1 Matrix-Vektor Produkt im Algorithmus in $O(s)$ Zeit anstelle von $O(d)$ berechnet werden kann, wobei s die Sparsity des Inputs $v_k \in \mathbb{R}^d$ ist, das heißt die Anzahl der nicht-null Einträge in v_k ist beschränkt durch s.
Zuerst wird der Rechenaufwand des Algorithmus pro Iteration gezeigt (siehe [1]).

Theorem 3.7. Betrachte Algorithmus 1 und sei f eine GLM Funktion. Sei s außerdem so, dass die Anzahl der nicht-null Einträge im Inputvektor $v_k, k = 1, ..., m$, beschränkt ist durch s. Dann kann jeder Iterationsschritt von Algorithmus 1 in der Zeit $\tilde{O}(ms + \hat{\kappa}_l^{\max} \hat{\kappa}_l s)$ berechnet werden. Dabei versteckt $\tilde{O}$ log Terme von $\hat{\kappa}_l, d$ und $\frac{1}{\delta}$.

Anmerkung: Wie in den vorherigen Theoremen, taucht hier aufgrund der Wahl von S_1 ein $\hat{\kappa}_l^{\max}$ auf anstelle von $(\hat{\kappa}_l^{\max})^2$, wie in [1]. Außerdem fehlt in [1] die Notation $\tilde{O}$.

Beweis. Der Beweis erfolgt per Induktion. Definiere dazu:

$$
\begin{aligned}
c_{j+1} &= 1 + (1-\lambda)c_j, \quad \text{mit } c_0 = 1, \\
d_{j+1} &= (1-\lambda)d_j, \quad \text{mit } d_0 = 1, \\
v_{j+1} &= v_j - \frac{1}{(1-\lambda)d_j}\widetilde{\nabla}^2 \widetilde{f}_{[i,j+1]}(x_t)(c_j \nabla f(x_t) + d_j v_j), \quad \text{mit } v_0 = 0,
\end{aligned}
$$

wobei λ der Regularisierungsparameter ist. Behauptung: Es gilt $X_{[i,j]} = c_j \nabla f(x_t) + d_j v_j$ für $j \geq 0$. Für den Induktionsanfang sei $j = 0$, dann gilt $X_{[i,0]} = c_0 \nabla f(x_t) + d_0 v_0 = \nabla f(x_t)$, was äquivalent zu Algorithmus 1 ist. Induktionsschritt: Es gelte die Behauptung für alle natürlichen Zahlen bis j und zeige den Beweis für $j+1$. Da $f_k(x_t)$ den Regularisierungsterm enthält, wird folgende Notation benutzt: Sei $\widetilde{f}(x) := \frac{1}{m}\sum_{k=0}^{m} f_k(x) - \lambda||x||^2$ die Funktion, die nicht den Regularisierungsterm enthält. Dann gilt: $\widetilde{\nabla}^2 f_{[i,j]}(x_t) = \widetilde{\nabla}^2 \widetilde{f}_{[i,j]}(x_t) + \lambda I$, wobei analog zum Algorithmus $\widetilde{\nabla}^2 \widetilde{f}_{[i,j]}(x_t)$ gleichmäßig von $\{\nabla^2 f_k(x_t) - \lambda I \mid k \in \{1, ..., m\}\}$ ausgewählt wird. Für die Iterationsvorschrift aus Algorithmus 1 folgt dann:

$$
\begin{aligned}
X_{[i,j+1]} &= \nabla f(x_t) + (I - \widetilde{\nabla}^2 f_{[i,j+1]}(x_t)) X_{[i,j]} \\
&= \nabla f(x_t) + (I - \lambda I - \widetilde{\nabla}^2 \widetilde{f}_{[i,j+1]}(x_t)) X_{[i,j]}
\end{aligned} \tag{3.24}
$$

Es ist anzumerken, dass in [1] diese Notation nicht beachtet wurde und mit der ersten Äquivalenz weitergerechnet wurde, sodass sich dort ein Fehler durch den Beweis zieht. Einsetzen der Induktionsvoraussetzung in (4.24) ergibt:

$$
\begin{aligned}
X_{[i,j+1]} &= \nabla f(x_t) + ((1-\lambda)I - \widetilde{\nabla}^2 \widetilde{f}_{[i,j+1]}(x_t))(c_j \nabla f(x_t) + d_j v_j) \\
&= (1 + (1-\lambda)c_j)\nabla f(x_t) + (1-\lambda) d_j v_j - \widetilde{\nabla}^2 \widetilde{f}_{[i,j+1]}(x_t)(c_j \nabla f(x_t) + d_j v_j) \\
&= c_{j+1} \nabla f(x_t) + (1-\lambda) d_j v_{j+1} \\
&= c_{j+1} \nabla f(x_t) + d_{j+1} v_{j+1}.
\end{aligned}
$$

Nun können die Terme genauer betrachtet werden. Die Berechnung der Terme c_{j+1} und d_{j+1} erfordert konstante Zeit. Die Terme $\widetilde{\nabla}^2 \widetilde{f}_{[i,j+1]}(x_t) c_j \nabla f(x_t)$ und $\widetilde{\nabla}^2 \widetilde{f}_{[i,j+1]}(x_t) d_j v_j$ können jeweils in $O(s)$ Zeit berechnet werden. Da beide Terme einen Vektor mit s nicht-null Einträgen ausgeben, erfordert die Berechnung von v_{j+1} die Zeit $O(s)$. Weiterhin kann $\nabla f(x_t)$ in $O(ms)$ Zeit berechnet werden. Da v_0 der Nullvektor ist, folgt außerdem, dass die Anzahl der nicht-null Einträge in v_j höchstens js ist. Damit ergibt sich schließlich, dass die Berechnung von X_t beziehungsweise x_t die Zeit $O(ms + S_1 S_2 s) = \tilde{O}(ms + \hat{\kappa}_l^{\max} \hat{\kappa}_l s)$ erfordert.

Die Laufzeit des Algorithmus wird in Korollar 3.8 festgehalten.

Korollar 3.8. Sei f eine GLM Funktion. Dann liefert Algorithmus 1 einen Vektor x_t,

sodass mit Wahrscheinlichkeit mindestens $1 - \delta$ gilt:

$$f(x_t) \leq f(x^*) + \epsilon, \tag{3.25}$$

in gesamter Zeit $\tilde{O}(ms + \hat{\kappa}_l^{\max}\hat{\kappa}_l s)\ln(\frac{1}{\epsilon})$ für $\epsilon \to 0$. Dabei versteckt $\tilde{O}$ log Terme von $\hat{\kappa}_l, d, M$ und $\frac{1}{\delta}$.
Anmerkung: Aufgrund von Theorem 3.7 taucht in der gesamten Zeit ein $\hat{\kappa}_l^{\max}$ auf anstelle von $(\hat{\kappa}_l^{\max})^2$, wie in [1].

Beweis. Der Beweis verläuft analog zum Beweis von Korollar 3.6. Der einzige Unterschied ist der Rechenaufwand aus Theorem 3.7.

KAPITEL 4

Erweiterungen von LiSSA

In diesem Kapitel werden weitere Algorithmen vorgestellt, darunter LiSSA-Quad und LiSSA-Sample. Die Motivation dahinter ist das Ausnutzen der quadratischen Struktur von Unterproblemen des eigentlichen Minimierungsproblems. Dadurch können Matrix "Sampling" und "Sketching" Verfahren verwendet werden.

4.1. LiSSA-Quad

In diesem Kapitel wird eine Familie von Algorithmen beschrieben, die erste-Ordnung Methoden als Soubroutine mit zweite-Ordnung Methoden verbindet. Speziell wird hier LiSSA-Quad vorgestellt und dessen Laufzeiten gezeigt. Die Idee hinter dem Algorithmus ist, dass das Newton-Verfahren das konvexe Optimierungsproblem reduziert auf ein Problem, bei dem quadratische Unterprobleme gelöst werden müssen. Genauer heißt das, gelöst werden soll das quadratische Unterproblem Q_t, das definiert ist als die Taylor-Entwicklung 2.Ordnung im Entwicklungspunkt x_{t-1}:

$$Q_t(y) := f(x_{t-1}) + \nabla f(x_{t-1})^T y + \frac{y^T \nabla^2 f(x_{t-1}) y}{2}, \tag{4.1}$$

mit $y := x - x_{t-1}$. Diese Überlegungen geben eine alternative Implementation des Schätzers für $\nabla^{-2} f(x)$, der in LiSSA benutzt wird. Wird nämlich das Gradientenverfahren benutzt für obiges Problem, wobei y_t^i der i-te Schritt des Verfahren ist, erhält man:

$$y_t^{i+1} = y_t^i - \nabla Q_t(y_t^i) = (I - \nabla^2 f(x_{t-1}))y_t^i - \nabla f(x_{t-1}) \tag{4.2}$$

Es kann nun gezeigt werden, dass (4.27) äquivalent ist zum Schritt $X_{[i,j]}$ aus Zeile 8 von Algorithmus 1. Lässt man das Gradientenverfahren für $i = n - 1$ Schritte laufen um den Punkt y_t^n und damit x_t zu bestimmen, ergibt sich:

$$y_t^n = x_t - x_{t-1} \;\Leftrightarrow\; x_t = x_{t-1} + y_t^n.$$

Weiterhin wird eine Stichprobe $\widetilde{\nabla}^2 f(x)$ der Hesse-Matrix anstelle der wahren Hesse-Matrix $\nabla^2 f(x)$ benutzt, das heißt $\widetilde{\nabla}^2 f(x)$ wird zufällig von $\{\nabla^2 f_k(x) \mid k \in \{1, ..., m\}\}$ ausgewählt, für ein $x \in \mathbb{R}^d$. Mit dem Einsetzen von (4.27) kann dies nun weiter umgeformt werden:

$$\begin{aligned} x_t &= x_{t-1} - (-y_t^n) \\ &= x_{t-1} - \left(\nabla f(x_{t-1}) - (I - \nabla^2 f(x_{t-1}))y_t^{n-1}\right) \\ &= x_{t-1} - \left(\nabla f(x_{t-1}) + (I - \nabla^2 f(x_{t-1}))(-y_t^{n-1})\right). \end{aligned}$$

Im Vergleich zu Zeile 8 aus Algorithmus 1 sieht man:

$$\begin{aligned} X_{[i,j]} &= \nabla f(x_t) + (I - \widetilde{\nabla}^2 f_{[i,j]}(x_t))X_{[i,j-1]} \\ &= \nabla f(x_t) + (I - \widetilde{\nabla}^2 f(x_t))(-y_{t+1}^{n-1}) \\ &= (-y_{t+1}^n). \end{aligned}$$

Das bedeutet, dass der Schritt $X_{[i,j]}$ aus Zeile 8 von Algorithmus 1 äquivalent ist zum Schritt $(-y_{t+1}^n)$. Damit kann LiSSA also interpretiert werden als ein teilweises stochastisches Gradientenverfahren angewandt auf das quadratische Problem Q_t. Es ist teilweise, da man in (4.27) eine exakte Schätzung für den Gradienten von f hat und eine stochastische Schätzung für die Hesse-Matrix von f. Dies ist wesentlich für die linearen Konvergenzeigenschaften für LiSSA (siehe [1]).
Die obigen Überlegungen deuten die Möglichkeit an eine erste-Ordnung linear konvergente Methode zu benutzen um den Minimierer von Q_t zu approximieren. Betrachte dazu einen beliebigen Algorithmus ALG, der für eine konvexe quadratische Funktion Q_t und einen Fehlerwert ε einen Punkt $y \in \mathbb{R}^d$ berechnet, so dass

$$||y - y_t^*|| \leq \varepsilon \tag{4.3}$$

mit Wahrscheinlichkeit $1 - \delta_{ALG}$ gilt, mit $y_t^* := \underset{y}{\operatorname{argmin}}\, Q_t(y)$. Sei $T_{ALG}(\varepsilon, \delta_{ALG})$ die gesamte Zeit, die notwendig ist für den Algorithmus ALG um den Punkt y zu bestimmen. Die Anwendung erfordert, dass ALG linear konvergent ist, das heißt die benötigte Zeit T_{ALG} sei proportional zu $\log(\frac{1}{\varepsilon})$ mit Wahrscheinlichkeit mindestens $1 - \delta_{ALG}$ (siehe [1]).
Algorithmus 2, LiSSA-Quad, implementiert nun die obigen Überlegungen für einen Algorithmus ALG (siehe [1]).

Algorithmus 2 LiSSA-Quad

Input: $T,\ f(x) = \frac{1}{m}\sum_{k=1}^{m} f_k(x),\ ALG,\ ALG_{params},\ T_1,\ \varepsilon$

$x_0 = ALG(f(x), T_1)$

for $t = 1$ to T **do**

$Q_t(y) = \nabla f(x_{t-1})^T y + \frac{y^T \nabla^2 f(x_{t-1}) y}{2}$

$y_t^n = ALG(Q_t, \varepsilon^2, ALG_{params})$

$x_t = x_{t-1} + y_t^n$

end for

return x_T

Anmerkung: In [1] kommt hier ein Schreibfehler vor. Der Algorithmus, der benutzt wird um den Punkt y auszurechnen wird ALG genannt, nicht A. Auf Seite 15 des Papers wird dies auch so beschrieben bei Gleichung (5). Aufgrund der Definition von $y = x - x_{t-1}$ erhält man deshalb auch x_t durch eine Umformung, wie hier im Algorithmus zu sehen ist und nicht sofort mit ALG. Die Anzahl der Iterationsschritte n taucht in ALG_{params} auf.
Als nächstes werden die Konvergenzeigenschaften von LiSSA-Quad gezeigt und danach die Laufzeiten, die üblich sind für LiSSA-Quad (siehe [1]).

Theorem 4.1. Sei f von der Form $f(x) = \frac{1}{m}\sum_{k=1}^{m} f_k(x)$ und α-streng konvex. Sei x^* der Minimierer der Funktion $f(x)$ und sei x_t definiert wie in Algorithmus 2. Der Algorithmus ALG genüge der Bedingung (4.3) mit Wahrscheinlichkeit $1 - \delta_{ALG}$ mit einer angemessenen Auswahl an Parametern ALG_{params}. Seien die Parameter von Algorithmus 2 wie folgt gewählt: $T = \log(\log(\frac{1}{\varepsilon}))$, $T_1 = ALG(M, \alpha)$, $\delta_{ALG} = \frac{\delta}{T}$, wobei ε der finale zu erzielende Fehler ist. Dann folgt nach T Iterationsschritten mit Wahrscheinlichkeit $1 - \delta$:

$$\min_{t=\{1,\dots,T\}} ||x_t - x^*|| \leq \varepsilon.$$

Genauer heißt das, LiSSA-Quad(ALG) (also LiSSA-Quad angewendet mit Algorithmus ALG) gibt einen Punkt x wieder, so dass

$$||x - x^*|| \leq \varepsilon$$

gilt in gesamter Zeit $O(T_{ALG}(\varepsilon, \delta_{ALG}\text{loglog}(\frac{1}{\varepsilon})$ mit Wahrscheinlichkeit mindestens $1 - \delta$ für $\varepsilon \to 0$.
Anmerkung: Da im Beweis in [1] auf S.16 die Bedingung $||x_0 - x^*|| \leq \sqrt{\frac{\alpha}{M}}$ benutzt wird, wird hier der Iterationsschritt $T_1 = ALG(M, \alpha)$ gewählt mit der Notation, die analog ist zu Theorem 3.2. Das heißt sei $ALG(M, \alpha)$ die Zeit, die der Algorithmus ALG benötigt um eine Genauigkeit von $\sqrt{\frac{\alpha}{M}}$ zu erzielen. Dann wähle in der Voraussetzung $T_1 = ALG(M, \alpha)$, so dass $||x_0 - x^*|| \leq \sqrt{\frac{\alpha}{M}}$.

Beweis. Der Algorithmus *ALG* erzielt eine Genauigkeit von ε^2 für jede quadratische Funktion Q_t, mit $t = 1, ..., T$. Sei $x_t^* := \underset{y}{\operatorname{argmin}}\ Q_t(y)$.
Anmerkung: Es ist zu beachten, dass x_t^* in [1] nicht definiert wurde, sondern nur y_t^*. Der Term x_t^* wurde nur in [1] Version 3, Seite 12 definiert und dann wahrscheinlich so in [1] übernommen, obwohl das Q_t dort anders gewählt wurde. Sei also $x_t^* := x_t + y_{t+1}^*$ wie in [1] Version 3, Seite 12 definiert. Es wird festgehalten, dass dann gilt:

$$\begin{aligned} x_t^* &= x_t + \underset{y}{\operatorname{argmin}}\ Q_{t+1}(y) \\ &= x_t - \nabla^{-2} f(x_t) \nabla f(x_t) \end{aligned} \tag{4.4}$$

Für $\delta_{ALG} = \frac{\delta}{T}$ folgt dann mit der Bonferroni-Ungleichung für alle $t \leq T$:

$$||x_{t+1} - x_t^*|| \leq \varepsilon^2 \tag{4.5}$$

mit Wahrscheinlichkeit $1 - \delta$. Es wird nun angenommen, dass für $t < T$, $||x_t - x^*|| \geq \varepsilon$ gilt, denn für $||x_t - x^*|| \leq \varepsilon$ ist der Beweis trivial. Dann erhält man für alle $t \leq T$:

$$\begin{aligned} ||x_{t+1} - x^*|| &= ||x_{t+1} - x_t^* + x_t^* - x^*|| \\ &\leq ||x_t^* - x^*|| + ||x_{t+1} - x_t^*|| \\ &\leq ||x_t - \nabla^{-2} f(x_t) \nabla f(x_t) - x^*|| + ||x_{t+1} - x_t^*||, \end{aligned}$$

wobei der letzte Schritt wegen (4.4) folgt. Dieser Term kann nun weiter abgeschätzt werden, indem analog die Analysis aus dem Beweis zu Lemma 3.3 benutzt wird:

$$||x_{t+1} - x^*|| \leq \frac{M}{2} ||\nabla^{-2} f(x_t)|| ||x_t - x^*||^2 + ||x_{t+1} - x_t^*||$$

Da f α-streng konvex ist, gilt für die Hesse-Matrix $\nabla^2 f(x_t) \succeq \alpha$ und damit $||\nabla^{-2} f(x_t)|| \leq \frac{1}{\alpha}$. Mit der Ungleichung (5.5) folgt dann:

$$\begin{aligned} ||x_{t+1} - x^*|| &\leq \frac{M}{2\alpha} ||x_t - x^*||^2 + \varepsilon^2 \\ &\leq \left(\frac{M}{2\alpha} + 1 \right) ||x_t - x^*||^2, \end{aligned}$$

wobei die letzte Ungleichung wegen der getroffenen Annahme folgt.
Im Beweis in [1] taucht irrtümlicherweise der Vorfaktor $\frac{1}{4}$ auf. Dieser sollte aber $\frac{1}{2}$ sein, da er analog zu der Analysis aus dem Beweis zu Lemma 3.3 folgt. Eine rekursive Anwendung

der obigen Ungleichung ergibt:

$$\begin{aligned} ||x_{t+1} - x^*|| &\leq \left(\frac{M}{2\alpha} + 1\right)^{2^{t+1}-1} ||x_0 - x^*||^{2^{t+1}} \\ &\leq \left(\frac{M}{2\alpha} + 1\right)^{2^{t+1}-1} \cdot \sqrt{\frac{\alpha}{M}}^{2^{t+1}}, \end{aligned}$$

wobei im letzten Schritt die Anfangsbedingung mit eingegangen ist. Einsetzen des Wertes von T aus dem Theorem liefert für $t = T - 1$:

$$||x_T - x^*|| \leq \left(\frac{M}{2\alpha} + 1\right)^{-1} \cdot \left(\frac{1}{2} \cdot \sqrt{\frac{M}{\alpha}} + \sqrt{\frac{\alpha}{M}}\right)^{2^{\log(\log(\frac{1}{\varepsilon}))}}$$

Mit der Regel für Basisumrechnungen von Logarithmen kann der obige Term weiter abgeschätzt werden, sodass man schließlich

$$||x_T - x^*|| \leq \varepsilon$$

erhält mit Wahrscheinlichkeit $1 - \delta$. Die gesamte Zeit, die der Algorithmus benötigt um diesen Fehler zu erreichen, setzt sich durch die Zeit pro Iterationsschritt multipliziert mit der Anzahl an Iterationsschritten zusammen und ist somit $O(T_{ALG}(\varepsilon, \delta_{ALG})\log(\log(\frac{1}{\varepsilon}))$.

Falls die Funktion f eine GLM Funktion ist, können weitere Bemerkungen an das Subproblem Q_t gemacht werden. Dann kann $\nabla Q_t(y)$ in jedem Punkt y in Zeit linear zu d berechnet werden. Genauer heißt das, dass ein kompletter Gradient von Q_t in der Zeit $O(md)$ berechnet werden und ein stochastischer Gradient (das heißt wie zuvor beschrieben eine stochastische Schätzung für die Hesse-Matrix von f) in der Zeit $O(d)$. Damit ist eine intuitive Wahl für den Algorithmus ALG in Algorithmus 2 ein erste-Ordnung Algorithmus, der linear konvergent ist. Beispiel hierfür sind SVRG, SDCA und Acc-SDCA (Accelerated SDCA). Damit wird die Notation benutzt, dass LiSSA-Quad(FO) eine Familie von erste-Ordnung Algorithmen FO darstellt, welche für ALG gewählt werden. Die Laufzeiten von LiSSA-Quad(FO) sind dann vergleichbar mit denen der eigentlichen Algorithmen FO, bis auf logarithmische Faktoren (siehe [1]). Im folgenden Korollar wird die Laufzeit von LiSSA-Quad(FO) festgehalten, wenn für FO der Algorithmus Acc-SVRG gewählt wird (siehe [1]).

Korollar 4.2. Sei $f(x)$ eine GLM Funktion. Wenn für ALG der Algorithmus Acc-SVRG (siehe [2]) gewählt wird, dann, für eine angemessene Parameterwahl, gibt LiSSA-Quad einen Punkt x wieder, sodass

$$f(x) - f(x^*) \leq \varepsilon$$

gilt mit Wahrscheinlichkeit mindestens $1-\delta$ in gesamter Zeit
$\widetilde{O}\left(m+\min\{\sqrt{\hat{\kappa}_l m}, \hat{\kappa}_l\}\right) d\log\left(\frac{1}{\varepsilon}\right)\log\left(\log\left(\frac{1}{\varepsilon}\right)\right)$. Hier versteckt $\widetilde{O}$ logarithmische Terme von $\hat{\kappa}_l, d, \delta$, aber nicht in ε.

Die in Korollar 5.2 auftauchende Laufzeit hängt von der Konditionszahl $\hat{\kappa}_l$ ab. Diese liefert potentiell eine bessere Laufzeit im Vergleich zur globalen Konditionszahl $\hat{\kappa}$, da, wie in Kapitel 2 beschrieben, die Laufzeiten von Algorithmen wie SVRG von der globalen Konditionszahl $\hat{\kappa}$ abhängen und nach Definition der Konditionszahlen der Zusammenhang $\hat{\kappa}_l \leq \hat{\kappa}$ gilt. In Anwendungsfällen in der Praxis kann dies dann zu verbesserten Laufzeiten für LiSSA-Quad(FO) führen im Vergleich zu den erste-Ordnung Algorithmen FO (siehe [1]).

4.2. LiSSA-Sample

Das vorherige Kapitel 4.1 ist von dem ursprünglichen konvexen Optimierungsproblem, dass in Kapitel 3 behandelt wurde, zu dem Optimierungsproblem von quadratischen Funktionen übergangen. Dieses Kapitel führt LiSSA-Sample ein, was auf LiSSA-Quad, also Algoithmus 2, basiert. Dabei wird der Algorithmus *ALG* mit Algorithmus 4 ersetzt, welcher in diesem Kapitel eingeführt und beschrieben wird. Die Idee hinter LiSSA-Sample ist, dass für quadratische Funktionen Verfahren existieren, bei den anstelle der gesamten Matrix Stichproben gezogen werden.

4.2.1. Einführung zu Fast Quadratic Solver

Die Motivation hinter LiSSA-Sample ist, dass für große Datensätze bessere Laufzeiten für beschleunigten ("accelerated") erste-Ordnung Verfahren (wie zum Beispiel Acc-SVRG) erzielten werden können, falls also $\kappa > m >> d$. Die Konditionszahl κ wird in den einzelnen Theoremen speziell definiert. Zuerst wird nun die Idee hinter Algorithmus 4 erklärt, welcher "Fast Quadratic Solver" (also schneller quadratischer Löser) genannt wird.
Der Unterschied zu den Überlegungen in Kapitel 3 ist wie folgt: In den vorherigen beiden Algorithmen wurde anstelle der wahren Hesse-Matrix eine Matrix $\widetilde{\nabla}^2 f(x)$ zufällig aus der Menge $\{\nabla^2 f_k(x) \mid k \in \{1, ..., m\}\}$ gewählt. Für Algorithmus 4 wird eine Stichprobe einer bestimmten Größe gezogen. Dies wird weiter unten genauer erklärt und unter anderem in Theorem 5.7.
Zum besseren Verständnis betrachte zuerst eine Matrix $A = \sum_{i=1}^{m} v_i v_i^T = VV^T$, $A \in \mathbb{R}^{dxd}$, wobei die i-te Spalte von $V \in \mathbb{R}^{dxd}$ der Vektor $v_i \in \mathbb{R}^d$ ist. Die Berechnung kann umformuliert werden zu einer Minimierung der konvexen quadratischen Funktion $Q(y) = \frac{y^T A y}{2} + b^T y$ und kann nach Theorem 2.2 mit Genauigkeit ε in gesamter Zeit $(m + \sqrt{\kappa(A)m}) d\log\left(\frac{1}{\varepsilon}\right)$ erfolgen. Im Falle $m > d$ verbessert Algorithmus 4 die Laufzeiten (siehe [1]).

Als nächstes wird nun das Vorgehen in Algorithmus 4 und Algorithmus 3, wobei dieser für Algorithmus 4 verwendet wird, speziell beschrieben (siehe [1]):

1. Als erste wird eine Spektralapproximation B von A berechnet. Diese hat die Form $B = \sum_{i=1}^{O(d\log(d))} u_i u_i^T$ mit $B \preceq A \preceq 2B$. Dies erfolgt durch Verfahren, bei denen Stichproben einer Matrix gezogen werden wie zum Beispiel in [3]. Das Verfahren benötigt das Lösen von $O(d\log(d))$ großen linearen Gleichungssystem, was mit Accelerated SVRG (der beschleunigten Variante von SVRG) passiert.
2. Als nächstes wird $BA^{-1}y$ berechnet, indem die quadratische Funktion $Q(y) := \frac{y^T AB^{-1}y}{2} + b^T y$ minimiert wird. Die Matrix B wird dabei als Vorkonditionierung benutzt. Die Funktion ist gut konditioniert und kann zum Beispiel mit dem Gradientenverfahren minimiert werden. Um den Gradienten zu berechnen, wird wieder Accelerated SVRG benutzt um ein lineares System in B zu lösen.
3. Als letztes wird $A^{-1}b = B^{-1}BA^{-1}b$ berechnet, indem wieder Accelerated SVRG benutzt wird um ein lineares System in B zu lösen.

Es ist zu beachten (siehe [1]), dass aufgrund der Definition der Funktion $f(x) = \sum_{k=1}^{m} f_k(x)$ der Regularisierungsterm λI in den Funktionen $f_k(x)$ mit drin ist. Deshalb ist Hesse-Matrix nicht unbedingt eine Summe aus Rang 1 Matrizen. Eine Möglichkeit ist, die Identitätsmatrix als eine Summe von Rang 1 Matrizen zu zerlegen. Das in den drei Punkten beschriebene Verfahren verlangt jedoch, dass jede Auswahl in der Stichprobe auch gut konditioniert ist, damit lineare Systeme mit Accelerated SVRG gelöst werden können. Das bedeutet, dass die ausgewählten Matrizen in der Stichprobe so aussehen müssen wie die ausgewählten Matrizen in der Stichprobe von $\nabla^2 f(x) = \sum_{k=1}^{m} \nabla^2 f_k(x)$. Aus diesem Grund werden die Verfahren von sogenannten "Leverage scores", die in [3] beschrieben wurden, erweitert und neu für den Fall definiert, wenn die betrachtete Matrix $A = \sum_{i=1}^{m} A_i$ aus einer Summe von positiv semidefiniten Matrizen besteht und nicht nur aus Rang 1 Matrizen. Dazu werden dann auch einige der Theoreme aus [3] neu definiert und neu bewiesen. Aufgrund der Wahl der positiv semidefiniten Matrizen A_i als Rang 1 Matrizen plus Identitätsmatrix multipliziert mit dem Regularisierungsterm sieht man in folgenden Theoremen, dass die Recheneffizienz beibehalten wird (siehe [1]).

4.2.2. Mathematische Hilfsmittel

In diesem Kapitel werden einige Definitionen, Lemmas und Theoreme definiert und gezeigt, die für den Rest des Kapitels und für die Algorithmen benötigt werden. Darunter:

1. Sei $A = \sum_{i=1}^{m} A_i,\ A \in \mathbb{R}^{d\text{x}d}$, eine positiv semidefinite Matrix, wobei die Matrizen A_i auch positiv semidefinit sind.
2. Für $B \in \mathbb{R}^{d\text{x}d}$ definiere $A \cdot B := Tr(B^T A)$, wobei mit Tr die Spur gemeint ist.

3. Für zwei Matrizen $A \in \mathbb{R}^{dxd}$ und $B \in \mathbb{R}^{dxd}$ ist B eine λ-Spektralapproximation von A, wenn $\frac{1}{\lambda} \preceq B \preceq A$.
4. Für eine Matrix A ist mit A^+ die Moore-Penrose Pseudoinverse gemeint.

Außerdem werden die sogennanten "Leverage Scores" definiert.
Definition 4.3. (Generalisierte Matrix Leverage Scores). Definiere

$$\tau_i(A) := A^+ \cdot A_i$$
$$\tau_i^B(A) := B^+ \cdot A_i.$$

Weiterhin gelten die folgenden zwei Fakten:
Fakt 4.4. Es gilt:

$$\sum_{i=1}^{n} \tau_i(A) = Tr(\sum A^+ A_i) = Tr(A^+ A) = \text{rank}(A) \leq d.$$

Fakt 4.5. Wenn B eine λ-Spektralapproximation von A ist, dann gilt:

$$\tau_i(A) \leq \tau_i^B(A) \leq \lambda \tau_i(A).$$

Wenn A die Form $A = \sum_{i=1}^{m} A_i$ hat, wird eine Stichprobe von A der Größe r wie folgt definiert: Betrachte eine Untermenge von Indizes der Größe r, $I = \{i_1, ..., i_r\} \subseteq \{1, ..., m\}$. Für jede so eine Stichprobe I mit einem Gewichtsvektor $w \in \mathbb{R}^r$ kann folgende Matrix definiert werden:

$$\text{Sample}(w, I) := \sum_{j \in I} w_j A_j. \tag{4.6}$$

Wenn die Stichprobe nicht gewichtet ist, zum Beispiel, wenn w der Einsvektor ist, also ein Vektor mit einer eins in jedem Eintrag, wird (4.6) als Sample(I) geschrieben. Als nächstes kann dazu die Konditionszahl definiert werden:

$$\kappa_{\text{Sample}}(A, r) := \max_{I:|I| \geq r} \kappa(\text{Sample}(I)). \tag{4.7}$$

Dies ist der schlechteste Fall für die Konditionszahl von einer nicht gewichteten Stichprobe der Größe r der Matrix A. Es ist anzumerken, dass wegen der Definition von κ_{Sample} für Hesse-Matrizen von Funktionen, die hier interessant sind, immer größer als $\frac{1}{\lambda}$ ist, wobei λ der Regularisierungsparameter ist (siehe [1]). Für den Rechenaufwand der Algorithmen ist die Konditionszahl $\kappa_{\text{Sample}}(A, O(d \log(d))$ von Interesse. Nun werden einige Lemmata und Theoreme definiert und gezeigt.
Das folgende Lemma (siehe [1]) ist eine Generalisierung von Lemma 4 in [3].

Lemma 4.6. (Spektralapproximation via Leverage Score Stichprobenauswahl). Für einen

Fehlerwert $0 < \varepsilon < 1$, sei u ein Vektor mit zu großen Leverage Score Schätzern, das heißt $\tau_i(A) \leq u_i$ für alle $i \in \{1, ..., m\}$. Sei α eine Stichprobenauswahlrate und sei c eine positive fixierte Konstante. Für jede Matrix A_i wird eine Wahrscheinlichkeit zur Stichprobenauswahl $p_i(\alpha) = \min\{1, \alpha u_i c \log(d)\}$ definiert. Sei I eine zufällige Stichprobenauswahl von Indizes, die aus $\{1, ..., m\}$ gezogen wurde, in dem jeder Index mit einer Wahrscheinlichkeit $p_i(\alpha)$ gezogen wird. Definiere den Gewichtsvektor $w(\alpha)$ durch $w(\alpha)_i = \frac{1}{p_i(\alpha)}$. Mit der Definition (4.6) gilt:

$$\text{Sample}(w(\alpha), I) = \sum_{i=1}^{m} \frac{1}{p_i(\alpha)} A_i \mathbf{1}_{x_i \sim p_i(\alpha)}(x_i = 1),$$

wobei x_i eine Bernoulli Zufallsvariable mit Wahrscheinlichkeit $p_i(\alpha)$ ist.
Wenn $\alpha = \varepsilon^{-2}$ gesetzt wird, dann wird $S = Sample(w(\alpha), I)$ geformt durch höchstens $\sum_i \min\{1, \alpha u_i c \log(d)\} \leq \alpha c \log(d) ||u_i||_1$ Einträge in obiger Summe. Außerdem gilt dann, dass $\frac{1}{1+\varepsilon} S$ eine $\frac{1+\varepsilon}{1-\varepsilon}$ Spektralapproximation für A mit Wahrscheinlichkeit mindestens $1 - d^{-c/3}$ ist.

Beweis. Der Beweis verläuft ähnlich zu dem Beweis von Lemma 4 in [3]. Dazu benutzen wie folgendes Lemma zur Matrixkonzentration, welches als Lemma 11 in [3] auftaucht.

Lemma A.1 (siehe [3]). Seien $Y_1 \dots Y_k$ unabhängige positiv semidefinite Zufallsmatrizen mit Dimensionen $d \times d$. Definiere $Y = \sum Y_i$ und $Z = \mathbb{E}[Y]$. Falls $Y_i \preceq R \cdot Z$, dann gilt:

$$\mathbf{Pr}\left[\sum Y_i \preceq (1-\varepsilon)Z\right] \leq d e^{\frac{-\varepsilon^2}{2R}},$$
$$\mathbf{Pr}\left[\sum Y_i \succeq (1+\varepsilon)Z\right] \leq d e^{\frac{-\varepsilon^2}{3R}}.$$

Anmerkung: In [1] (Lemma A.1) fehlt hier in den Exponentialfunktionen ein negatives Vorzeichen, da in dem ursprünglichen Lemma diese auch vorkommen und man ansonsten die gewünschte Wahrscheinlichkeit $1 - d^{-\frac{\varepsilon}{3}}$ nicht erhält.
Sei $Y_i = \frac{A_i}{p_i}, i = 1, ..., m$, mit Wahrscheinlichkeit p_i und 0 sonst. Es gilt dann:

$$Z = \mathbb{E}[Y] = \mathbb{E}\left[\sum_{i=1}^{m} Y_i\right] = \sum_{i=1}^{m} \mathbb{E}[Y_i] = \sum_{i=1}^{m} A_i = A.$$

Um Lemma A.1 anwenden zu können, muss gezeigt werden, dass

$$Y_i \preceq \frac{1}{c \log d \varepsilon^{-2}} A \quad \forall i \in \{1, ..., m\} \tag{4.8}$$

gilt. Für die Wahrscheinlichkeit p_i muss nun eine Fallunterscheidung vorgenommen werden.

Sei zuerst $p_i < 1$: Dann gilt:

$$\begin{aligned} p_i < 1 &\Leftrightarrow \alpha u_i c\log(d) < 1 \\ &\Leftrightarrow u_i < \frac{1}{\alpha c\log(d)} \\ &\Leftrightarrow u_i < \frac{1}{c\log(d)}. \end{aligned}$$

Der letzte Schritt folgt wegen $0 < \varepsilon < 1$ und damit $\alpha = \varepsilon^{-2} \geq 1$. Damit gilt für die Wahl von Y_i:

$$Y_i = \frac{A_i}{p_i} = \frac{A_i}{c\log(d)\varepsilon^{-2}u_i} \preceq \frac{A_i}{\tau_i(A)c\log(d)\varepsilon^{-2}}. \tag{4.9}$$

Der letzte Schritt folgt wegen der Bedingung $\tau_i(A) \leq u_i$ aus den Voraussetzungen. Nun wird gezeigt, dass

$$\frac{A_i}{\tau_i(A)} \preceq A. \tag{4.10}$$

gilt. Das heißt, es muss $x^\top A_i x \leq \tau_i(A) x^\top A x$ für alle x gezeigt werden. Die Matrizen A_i sind positiv semidefinit, also gilt $x^T A_i x \geq 0$. Für $x^\top A x = 0$ folgt dann $x^\top A_i x = 0$ für alle $i \in \{1, ..., m\}$. Sei also ohne Beschränkung der Allgemeinheit $x = A^{+/2}y$ für $y \in \mathbb{R}^d$. Dann muss gezeigt werden:

$$y^T A^{+/2} A_i A^{+/2} y \leq \tau_i(A)||y||^2. \tag{4.11}$$

Wegen der positiven Semidefinitheit gilt für den größten Eigenwert:

$$\lambda_{\max}(A^{+/2}A_iA^{+/2}) \leq Tr(A^{+/2}A_iA^{+/2}) = Tr(A^+A_i) = \tau_i(A),$$

womit (4.11) gezeigt ist und damit (4.10). Für (4.9) ergibt sich dann mit (4.10):

$$Y_i \preceq \frac{1}{c\log(d)\varepsilon^{-2}} A.$$

Im zweiten Fall $p_i = 1$ gilt obiges nicht. Jedoch ist das Auswählen von $Y_i = A_i$ dasselbe wie das Auswählen und Summieren von $c\log d\varepsilon^{-2}$ Zufallsvariablen $Y_i^{(1)}, ..., Y_i^{((c\log d\varepsilon^{-2}))}$, alle äquivalent zu $\frac{A_i}{c\log d\varepsilon^{-2}}$ mit Wahrscheinlichkeit 1, sodass

$$Y^{(j)} \preceq \frac{1}{c\log(d)\varepsilon^{-2}} A_i$$

gilt. Dann kann Y_i in Lemma A.1 mit diesen kleineren Zufallsvariablen ersetzt werden. Dies verändert nicht $Z = \mathbb{E}[Y]$, zeigt aber die Konzentration.
Aber $Y_i = A_i$ kann mit $c\log d\varepsilon^{-2}$ Variablen ersetzt werden, die alle äquivalent zu $\frac{A_i}{c\log d\varepsilon^{-2}}$

sind, welche alle mit Wahrscheinlichkeit 1 gezogen werden. Dies verändert $\mathbb{E}[\sum Y_i]$ nicht, zeigt aber die Konzentration.
Damit ist (4.8) gezeigt und es kann Lemma A.1 mit $R = \frac{1}{c\log(d)\varepsilon^{-2}}$ und $Z = A$ angewandt werden. Somit erhält man:

$$(1-\varepsilon)A \preceq \sum_{i=1}^{m} Y_i \preceq (1+\varepsilon)A. \tag{4.12}$$

mit Wahrscheinlichkeit mindestens

$$1 - d\exp\left(\frac{-c\log(d)\varepsilon^{-2}\varepsilon^2}{3}\right) = 1 - d^{1-\frac{c}{3}}.$$

Nach Definition von S ist $S = \sum_{i=1}^{m} Y_i$, sodass eine Umformung von (4.12) ergibt:

$$\frac{(1-\varepsilon)}{(1+\varepsilon)}A \preceq \frac{1}{1+\varepsilon}S \preceq A,$$

was bedeutet, dass $\frac{1}{1+\varepsilon}S$ eine $\frac{1+\varepsilon}{1-\varepsilon}$ Spektralapproximation für A ist. Mit einer klassischen Chernoff-Ungleichung folgt, dass S $\sum_i \min\{1, \alpha u_i c\log(d)\} \leq \alpha c\log(d)||u_i||_1$ Einträge hat, womit der Beweis abgeschlossen ist. Als nächstes wird ein Theorem zu gleichmäßiges Sampling gezeigt (siehe [1]).

Theorem 4.7 (Gleichmäßiges Sampling). Sei $A = \sum_{i=1}^{m} A_i$ definiert wie zuvor. Die Matrix $S = \sum_{j=1}^{r} X_j$ sei geformt, indem gleichmäßig r Matrizen $X_1 \dots X_r \sim \{A_i\}$ gezogen werden ohne Wiederholung. Definiere

$$\tilde{\tau}_i^S(A) := \begin{cases} \tau_i^S(A) & \text{if } \exists j \text{ s.t. } X_j = A_i \\ \tau_i^{S+A_i}(A) & \text{sonst} \end{cases}.$$

Dann ist $\tilde{\tau}_i^S(A) \geq \tau_i(A)$ für alle $i \in \{1, ..., m\}$ und es gilt:

$$\mathbb{E}\left[\sum_{i=1}^{m} \tilde{\tau}_i^S(A)\right] \leq O\left(\frac{md}{r}\right).$$

Anmerkung: In [1] (Lemma 5.7) taucht kein $O(\cdot)$ auf in der zu zeigenden Abschätzung, aber im Beweis. Deshalb ist dies vermutlich ein Schreibfehler.
Beweis. Der Beweis (siehe [1]) verläuft ähnlich zu dem Beweis zu Theorem 1 in [3]. Definiere zuerst S_i wie folgt:

$$S_i := \begin{cases} S & \text{wenn } \exists j s.t. X_j = A_i \\ S + A_i & \text{sonst} \end{cases}$$

Wegen $S_i \preceq A$ gilt $A^+ \preceq S_i^+$. Damit folgt der Zusammenhang: $\tau_i(A) \leq \tilde{\tau}_i^S(A)$. Um den Erwartungswert abzuschätzen, wird die Summe in zwei Teile zerlegt:

$$\sum_i \tilde{\tau}_i^S(A) = \sum_{i \in S} \tilde{\tau}_i^S(A) + \sum_{i \notin S} \tilde{\tau}_i^S(A).$$

Der erste Term ist eine Summe von Leverage Scores von S. Mit Fakt 4.4 folgt, dass dieser Term nach oben durch d beschränkt ist. Der zweite Term wird mit folgenden Überlegungen weiter abgeschätzt: Betrachte einen Zufallsprozess, der zuerst S, dann ein zufälliges $i \notin S$ auswählt and $\tilde{\tau}_i^S(A)$ wiedergibt. Es sind immer genau $m - r$ solche i, sodass der Wert, der durch diesen Zufallsprozess wiedergegeben wird, äquivalent ist zu $\frac{1}{m-r}\mathbb{E}\big[\sum_{i \notin S} \tilde{\tau}_i^S(A)\big]$. Außerdem ist dieser Zufallsprozess äquivalent dazu, dass die Menge S der Größe $r + 1$ zufällig ausgewählt wird, und dann wird $i \in S'$ zufällig ausgewählt und sein Leverage Score wiedergegeben. Damit ist es in Erwartung äquivalent zu dem durchschnittlichen Leverage Score in S. Weiterhin hat S' die Größe $r + 1$ und and dessen Leverage Scores summieren sich zu dessen Rang auf. Damit kann dessen Leverage Score mit $\frac{d}{r+1}$ abgeschätzt werden. Ingesamt ergibt sich für den Erwartungswert:

$$\begin{aligned}
\mathbb{E}\Big[\sum_{i=1}^{m} \tilde{\tau}_i^S(A)\Big] &\leq d + (m-r)\frac{d}{r+1} \\
&= \frac{d(m+1)}{r+1} \\
&\leq O\Big(\frac{md}{r}\Big).
\end{aligned}$$

In [3] werden Rang 1 Matrizen betrachten. Da aber hier die Matrix A andere Eigenschaften hat, ist nicht sofort klar, wie man $\tilde{\tau}_i^S(A)$ effizient berechnen kann, da die Sherman-Morrison Formel nicht direkt anwendbar ist. In dem hier betrachteten Fall ist $A_i = v_i^T v_i + \lambda I$. Als nächstes werden dazu ähnliche Schätzungen getroffen in dem Fall, wenn die Matrizen A_i so gewählt sind (siehe [1]).

Theorem 4.8. Sei eine beliebige Matrix $A = \sum_{i=1}^m A_i$ gegeben, mit $A_i = v_i v_i^\top + \lambda I$. Die Matrix $S = \sum_{j=1}^r X_j$ wird geformt, indem gleichmäßig r Matrizen $X_1 \dots X_r \sim \{A_i\}$ gezogen werden ohne Wiederholung. Definiere

$$\hat{\tau}_i^S(A) := \begin{cases} v_i^\top S^+ v_i + \frac{d}{r} & \text{falls } \exists j \text{ s.t. } X_j = A_i \\ \frac{1}{1 + \frac{1}{v_i^\top (S+\lambda I)^+ v_i}} + \frac{d}{r} & \text{sonst} \end{cases}$$

Dann gilt $\hat{\tau}_i^S(A) \geq \tau_i(A)$ für alle $i \in \{1, ..., m\}$ und

$$\mathbb{E}\Big[\sum_{i=1}^{m} \hat{\tau}_i^S(A)\Big] \leq O\Big(\frac{md}{r}\Big).$$

Anmerkung: Der in [1] (S.21) definierte Leverage Score $\hat{\tau}_i^S(A)$ ist so nicht ganz richtig, da $v_i^\top S^+ v_i^T$ und $v_i^\top (S + \lambda I)^+ v_i^T$ Dimensionsprobleme geben. Dies wurde hier korrigiert.

Beweis. Es ist zu zeigen:

$$\frac{d}{r} \geq \hat{\tau}_i^S(A) - \tilde{\tau}_i^S(A) \geq 0. \tag{4.13}$$

Dann folgt nämlich mit Theorem 4.8 die Behauptung.
Betrachte den ersten Fall, das heißt $\exists j$ s.t. $X_j = A_i$. Nach Definition von $\tilde{\tau}_i^S(A)$ gilt dann:

$$\tilde{\tau}_i^S(A) = \tau_i^S(A) = S^+ \cdot A_i = S^+ \cdot (v_i v_i^\top + \lambda I) = v_i^\top S^+ v_i + \lambda S^+ \cdot I,$$

wobei die letzte Ungleichung mit Fakt 4.4 folgt. Wegen $S \succeq r\lambda I$ und damit $S^+ \preceq (r\lambda I)^+$ folgt:

$$\tilde{\tau}_i^S(A) \leq v_i^\top S^+ v_i + \lambda (r\lambda I)^+ \cdot I = v_i^\top S^+ v_i^\top + \frac{d}{r} = \hat{\tau}_i^S(A).$$

Nach Definition gilt dann:

$$0 \leq \hat{\tau}_i{}^S(A) - \tilde{\tau}_i{}^S(A) \leq \frac{d}{r}.$$

Für den zweiten Fall gilt:

$$\begin{aligned}
\tilde{\tau}_i{}^S(A) &= \tau_i{}^{S+A_i}(A) = (S + A_i)^+ \cdot A_i \\
&= (S + A_i)^+ \cdot A_i \\
&= (S + \lambda I + v_i v_i^\top)^+ \cdot (v_i v_i^T + \lambda I) \\
&= \underbrace{v_i^T (S + \lambda I + v_i v_i^\top)^+ v_i}_{A} + \underbrace{\lambda (S + \lambda I + v_i v_i^\top)^+ \cdot I}_{B},
\end{aligned}$$

wobei der letzte Schritt analog zu vorherigen Rechnungen ist. Wegen der Bedingung $S + \lambda I + v_i v_i^\top \succeq r\lambda I$ folgt für den zweiten Term:

$$B \preceq \frac{d}{r}.$$

Mit der Sherman-Morrison Formel erhält man für den ersten Term:

$$\begin{aligned} A &= v_i^\top \left((S+\lambda I)^+ - \frac{(S+\lambda I)^+ v_i v_i^\top (S+\lambda I)^+}{1 + v_i^T (S+\lambda I)^+ v_i} \right) v_i \\ &= \frac{1}{1 + \frac{1}{v_i^\top (S+\lambda I)^+ v_i}}. \end{aligned}$$

Ingesamt erhält man also:

$$\tilde{\tau}_i^S(A) \leq \frac{1}{1 + \frac{1}{v_i^\top (S+\lambda I)^+ v_i}} + \frac{d}{r} = \hat{\tau}_i^S(A).$$

Damit ist (4.13) insgesamt gezeigt. Eine Anwendung von Theorem 4.7 liefert dann:

$$\tau_i(A) \leq \tilde{\tau}_i^S(A) \leq \hat{\tau}_i^S(A)$$

Für den Erwartungswert erhält man mit Theorem 4.7 und (4.13):

$$\mathbb{E}\left[\sum_{i=1}^m \hat{\tau}_i^S(A) \right] \leq \mathbb{E}\left[\sum_{i=1}^m \tilde{\tau}_i^S(A) + \frac{d}{r} \right] \leq O\left(\frac{md}{r} \right) + \frac{d}{r} = O\left(\frac{md}{r} \right).$$

4.2.3. Laufzeitanalyse der Algorithmen

In diesem Kapitel werden die Algorithmen 3 und 4, also Fast Quadratic Solver und Repeated Halving, definiert. Weiterhin wird die Laufzeit von Algorithmus 4 sowie der Rechenaufwand von Algorithmus 3 gezeigt. Die Vorgehensweise der Algorithmen 3 und 4 wurden zu Beginn von Kapitel 4.2.1 beschrieben.

Algorithmus 3 Repeated Halving

Input: $A = \sum_{i=1}^m (v_i v_i^\top + \lambda I)$

Output: B an $O(d \log(d))$ size weighted sample of A and $B \preceq A \preceq 2B$

Take a uniformly random unweighted sample of size $\frac{m}{2}$ of A to form A'

if A' has size $> O(d \log(d))$ **then**

Recursively compute an 2-spectral approximation $\tilde{A}'$ of A'

end if

Compute estimates γ_i of generalized leverage scores $\{\hat{\tau}_i^{A'}(A)\}$ s.t. the following are satisfied

$$\gamma_i \geq \hat{\tau}_i^{A'}(A)$$

$$\sum \gamma_i \leq \sum 16 \hat{\tau}_i^{A'}(A) + 1$$

Use these estimates to sample matrices from A to form B

Algorithmus 4 Fast Quadratic Solver (FQS)

Input: $A = \sum_{i=1}^{m}(v_i v_i^\top + \lambda I), b, \varepsilon$
Output: $\tilde{v}$ s.t. $||A^{-1}b - \tilde{v}|| \leq \varepsilon$
Compute B s.t. $2B \succeq A \succeq B$ using Repeated Halving (Algorithm 3)
$Q(y) = \frac{y^\top A B^{-1} y}{2} + b^\top y$
Compute $\hat{y}$ such that $||\hat{y} - \text{argmin} Q(y)|| \leq \frac{\epsilon}{4||B^{-1}||}$
Output $\hat{v}$ such that $||B^{-1}\hat{y} - \tilde{v}|| \leq \varepsilon/2$

Zunächst gilt folgendes Theorem für Algorithmus 3 (siehe [1]).

Theorem 4.9. Repeated Halving (Algorithmus 3) liefert eine $\tilde{O}(d)$ große Stichprobe B, so dass $B \preceq A \preceq 2B$ und kann in gesamter Zeit

$$\tilde{O}(md + d^2 + \sqrt{\kappa_{\text{Sample}}(O(d\log(d))d)})$$

Dabei enthält $\tilde{O}$ logarithmische Faktoren von $m, d, ||A||_F$. Außerdem ist anzumerken, dass die Frobeniusnorm von A beschränkt ist durch $d||A||$, was hier in diesem Fall durch d beschränkt ist.
Beweis. Der Beweis ist in [1] zu finden (S.24, Beweis zu Theorem 23).

Es werden weitere folgende Theoreme formuliert und bewiesen.

Theorem 4.10. Sei der Vektor $b \in \mathbb{R}^d$ und die Matrix $A = \sum_{i=1}^{m} A_i$ gegeben, wobei A_i die Form $A_i = v_i v_i^\top + \lambda I$ hat für einen Vektor $v_i \in \mathbb{R}^d, ||v_i|| \leq 1$ und einen fixierten Parameter $\lambda > 0$. Dann berechnet Algorithmus 4 einen Vektor $\tilde{v}$, so dass $||A^{-1}b - \tilde{v}|| \leq \varepsilon$ gilt mit Wahrscheinlichkeit mindestens $1 - \delta$ in gesamter Zeit.

$$\tilde{O}\left(md\log\left(\frac{1}{\varepsilon}\right) + \left(d + \sqrt{\kappa_{sample}(A)d}\right)d\log^2\left(\frac{1}{\varepsilon}\right)\right).$$

Dabei enthält $\tilde{O}$ logarithmische Faktoren von $m, d, \kappa_{sample}(A), ||b||, \delta$. $\kappa_{sample}(A)$ ist die Konditionszahl von einem $O(d\log(d)$ großen Stichprobe von A und wurde in (4.7) definiert.

Algorithmus 4 kann nun einen approximierten Newton-Schritt für $A = \nabla^2 f(x)$ und $b = \nabla f(x)$ berechnen. Damit ist LiSSA-Sample eine Variante von LiSSA-Quad, bei der für den Algorithmus ALG der Algorithmus 4 gewählt wird und ein beliebiger erste Ordnung Algorithmus in der Initialphase.

Beweis zu Theorem 4.10 Anmerkung: In dem Beweis in [1] (S. 23, Beweis zu Theo-

rem 15) wird behauptet, dass $\text{argmin}_y Q(y) = BA^{-1}b$ gilt und

$$\nabla Q(y) = AB^{-1}y - b.$$

Die Funktion $Q(y)$ ist im Algorithmus aber definiert als $Q(y) = \frac{y^T AB^{-1}y}{2} + b^T y$, sodass sich ein Vorzeichenfehler im zweiten Term ergibt. Da der Beweis in [1] aber mit der obigen Definition von Minimum und Minimierer verläuft, werden diese hier für den Verlauf des Beweises angenommen.
Als erstes wird die Wohldefiniertheit gezeigt. Dafür gilt:

$$||B^{-1}\hat{y} - A^{-1}b|| = ||B^{-1}(\hat{y} - \text{argmin}_y Q(y))|| \leq \varepsilon/2,$$

wobei hier Zeile 5 von Algorithmus 4 benutzt wird. Dann gilt mit der Dreiecksungleichung:

$$||A^{-1}b - \tilde{v}|| \leq ||B^{-1}\hat{y} - A^{-1}b|| + ||B^{-1}\hat{y} - \tilde{v}|| \leq \varepsilon$$

Im Folgenden wird die Laufzeit untersucht. Vorab gilt: Da B eine 2-Spektralapproximation von A ist, gilt $\kappa(B) \leq 2\kappa(A)$. Außerdem ist $Q(y)$ nach Definition 1-streng konvex und 2-glatt. Als erstes betrachten wir die Laufzeit (Rechenaufwand) für Schritt 5 (Zeile 5) des Algorithmus. Definiere

$$y_0 := 0,\ \tilde{\varepsilon} := \left(\frac{\varepsilon}{4||B^{-1}||}\right)^2$$

und G_Q als obere Schranke für den Gradienten von $Q(y)$. Nun kann ein Gradientenverfahren für $Q(y)$ formuliert werden. Wähle y_{t+1} dann so, dass gilt:

$$y_{t+1} = y_t - \eta(Av_t - b),$$

wobei v_t so gewählt ist, dass:

$$||B^{-1}y_t - v_t|| \leq \frac{\tilde{\varepsilon}}{100||A||G_Q}.$$

Definiere $h_t := Q(y_t) - \min_y Q(y_t)$. Zu zeigen ist nun für alle $t > 0$:

$$h_t \leq \max\{\tilde{\varepsilon}, 0.9^t h_0\}.$$

Definiere außerdem das "wahre" Gradientenverfahren:

$$z_{t+1} = y_t - \eta\nabla Q(y_t)$$

Es folgt:

$$\begin{aligned}||z_{t+1} - y_{t+1}|| &= ||\eta A(v_t - B^{-1}y_t)|| \\ &\leq |\eta|||A||||(B^{-1}y_t - v_t)|| \\ &\leq \frac{\tilde{\varepsilon}}{10G_Q},\end{aligned}$$

mit der Annahme $|\eta| \leq 1$. Nun gilt:

$$\begin{aligned}h_{t+1} - h_t &= Q(y_{t+1}) - Q(y_t) \\ &\leq \langle \nabla Q(y_t), y_{t+1} - y_t \rangle + \frac{\beta}{2}||y_{t+1} - y_t||^2 \\ &= \langle \nabla Q(y_t), z_{t+1} - y_t \rangle + \langle \nabla Q(y_t), y_{t+1} - z_{t+1} \rangle + \frac{\beta}{2}||z_{t+1} - y_t + y_{t+1} - z_{t+1}||^2 \\ &\leq -\eta||\nabla Q(y_t)||^2 + \langle \nabla Q(y_t), y_{t+1} - z_{t+1} \rangle + \beta\eta^2||\nabla Q(y_t)||^2 + \beta||y_{t+1} - z_{t+1}||^2 \\ &\leq \frac{1}{\beta}||\nabla Q(y_t)||^2 + \frac{2}{10\beta}((||\nabla Q(y_t)|| + 1)\tilde{\varepsilon}) \\ &\leq -\frac{\alpha}{\beta}h_t + \frac{\tilde{\varepsilon}}{50\beta},\end{aligned}$$

wobei der zweite Schritt durch die β-Glattheit von $Q(y)$ folgt und die restlichen Schritte durch Umformungen. Weiterhin sind $\alpha = 1$ and $\beta = 2$ die dazugehörigen Parameter von $Q(y)$. Dann folgt:

$$h_{t+1} - h_t \leq -\frac{1}{2}h_t + \frac{\tilde{\varepsilon}}{100} \Leftrightarrow h_{t+1} \leq 0.75h_t + 0.08\tilde{\varepsilon}.$$

Mit der Induktionsannahme $h_t \leq \max\{\tilde{\varepsilon}, 0.9^t h_0\}$, kann nun gezeigt werden:

$$h_{t+1} \leq \max\{\tilde{\varepsilon}, 0.9^{t+1}h_0\}. \tag{4.14}$$

Dies kann mit einer Fallunterscheidung gezeigt werden. Sei zuerst $h_t \leq \tilde{\varepsilon}$. Dann folgt mit Induktionsannahme:

$$h_{t+1} \leq 0.75\tilde{\varepsilon} + 0.08\tilde{\varepsilon} \leq \tilde{\varepsilon}.$$

Sei nun $h_t \leq 0.9^t h_0$. Dann folgt mit Induktionsannahme:

$$h_{t+1} \leq 0.75 \cdot 0.9^t h_0 + 0.08 \cdot 0.9^t h_0 \leq 0.9^{t+1}h_0.$$

Damit ist (4.14) gezeigt.
Mit (4.14) folgt dann, wenn $t \geq O(\log(\frac{1}{\tilde{\varepsilon}}))$, dass h_t kleiner gleich oder gleich $\tilde{\varepsilon}$ ist. Mit der

1-strengen Konvexität von $Q(y)$ folgt, wenn $h_t \leq \tilde{\varepsilon}$:

$$||y_t - \text{argmin}_y Q(y)|| \leq \sqrt{\tilde{\varepsilon}} \leq \frac{\varepsilon}{4||B||^{-1}}. \tag{4.15}$$

Nach den Überlegungen in [1] (S.24) gilt dann: Die Laufzeit von (4.15) ist beschränkt durch die Zeit, bei der ein Vektor mit A multipliziert wird, was $O(md)$ Zeit in Anspruch nimmt und die Zeit, die benötigt wird um v_t zu bestimmen, wobei hier ein lineares System in B gelöst werden muss (in jedem Schritt). In Schritt 6 des Algorithmus muss wieder ein lineares System in B gelöst werden. Zusammenfassend erhält man die gesamte Zeit

$$\tilde{O}(md + LIN(B, \hat{\varepsilon}))\log\left(\frac{1}{\tilde{\varepsilon}}\right),$$

mit $\hat{\varepsilon} = \frac{\varepsilon}{400||A||||B||^{-1}G_Q} = \Omega\left(\frac{\varepsilon}{\kappa(A)G_Q}\right)$. Schließlich kann die Zeit $LIN(B, \varepsilon)$ durch $\tilde{O}(d^2 + d\sqrt{\kappa(A)d})\log(1/\varepsilon)$ beschränkt werden, indem der Algorithmus ACC-SVRG benutzt wird um das lineare System zu lösen und indem bemerkt wird, dass B ein $O(d\log(d))$ große 2-Spektralapproximation Stichprobe von A. Da die Norm des Gradienten $\nabla Q(y)$ nur entlang des Pfades des Gradientenabstiegs kleiner wird, kann die Schranke G_Q als eine Schranke für den Gradienten zu Beginn des Algorithmus gewählt werden. Das heißt es genügt, wenn $G_Q \leq ||b||$. Dies beendet den Beweis (nach [1]).

Als nächstes wird ein Korollar für die Laufzeit von LiSSA-Sample formuliert.

Theorem 4.11. Sei $f(x) = \sum_i f_i(x)$ eine GLM Funktion. Dann liefert LiSSA-Sample einen Punkt x, sodass gilt:

$$||x - x^*|| \leq \varepsilon$$

mit Wahrscheinlichkeit $1 - \delta$ in gesamter Zeit

$$\tilde{O}\left(\left(md\log\left(\frac{1}{\varepsilon}\right) + \left(d + \sqrt{\kappa_{sample}(\nabla^2 f)d}\right)d\log^2\left(\frac{1}{\varepsilon}\right)\right)\log\left(\log\left(\frac{1}{\varepsilon}\right)\right)\right).$$

Dabei enthält $\tilde{O}$ logarithmische Faktoren von $m, d, \kappa_{sample}(\nabla^2 f), G, \delta$.

Beweis. Der Beweis folgt unmittelbar mit Theorem 4.1 und Theorem 4.10. Die Voraussetzungen von Theorem 4.1 an die Funktion f sind nämlich erfüllt. Damit folgt: LiSSA-Quad(ALG) gibt einen Punkt x wieder, sodass

$$||x - x^*|| \leq \varepsilon$$

gilt in gesamter Zeit $O(T_{ALG}(\varepsilon, \delta_{ALG})\log(\log(\frac{1}{\varepsilon}))$ mit Wahrscheinlichkeit mindestens $1 -$

δ. Wird für den Algorithmus ALG nun LiSSA-Sample gewählt, das heißt Algorithmus 4, kann Theorem 4.10 angewendet werden, mit $A = \nabla^2 f(x)$ und $b = \nabla f(x)$. Dann wird der Rechenaufwand T_{ALG} durch die aus Theorem 4.10 ersetzt. Es folgt die Behauptung.

KAPITEL 5

Selbst-konkordante Funktionen

Dieses Kapitel befasst sich mit dem Spezialfall, das die betrachtete Funktion selbst-konkordant ist. Solche Funktionen sind eine Unterklasse von konvexen Funktionen und wurden in der Literatur zur konvexen Optimierung ausgiebig behandelt (siehe [14]). Es wird ein Algorithmus eingeführt, genannt ellipsoidCover, welcher lineare Konvergenz aufweist und Laufzeiten hat, die unabhängig von der Konditionszahl sind.

5.1. Mathematische Hilfsmittel

Als erste wird die Definition für selbst-konkordante Funktionen angegeben.

Definition 5.1. (Self-Concordant Functions). Sei $\mathcal{K} \subseteq \mathbb{R}^n$ eine nichtleere offene konvexe Menge. Sei $f : \mathcal{K} \to \mathbb{R}$ eine $\mathcal{C}^3$ konvexe Funktion. Dann sagt man f ist selbst-konkordant, wenn

$$|\nabla^3 f(x)[h,h,h]| \leq \ 2(h^\top \nabla^2 f(x) h)^{3/2},$$

mit

$$\nabla^k f(x)[h_1,...,h_k] := \frac{\partial^k}{\partial t_1 \dots \partial t_k}|_{t_1=...=t_k=0} f(x + t_1 h_1 + \cdots + t_k h_k).$$

Um dem Algorithmus 5, ellipsoidCover, und dessen Konvergenztheorem vollständig präsentieren zu können, müssen zuerst einige Definitionen und Zusammenhänge für selbst-konkordante Funktionen beschrieben werden. Ein wichtiger Bestandteil für die Analysis selbst-konkordanter Funktionen ist das Dikin Ellipsoid.

Definition 5.2. (Dikin Ellipsoid). Das Dikin Ellipsoid mit Radius r um einen Punkt x zentriert, ist definiert als:

$$W_r(x) := \{y \mid ||y - x||_x \leq r\},$$

mit der Notation. Für einen Vektor $h \in \mathbb{R}^d$ gilt $||h||_x := \left(\nabla^2 f(x)[h,h]\right)^{\frac{1}{2}}$. Eine wichtige Eigenschaft von selbst-konkordanten Funktionen ist die Tatsache, dass die Funktion im Dikin Ellipsoid gut konditioniert ist und dass die Funktion glatt ist (siehe [1]). Nun werden einige Hilfslemmas definiert. Diese sind in [14] zu finden mitsamt derer Beweise.

Lemma 5.3. (siehe [14]). Für alle $h \in \mathbb{R}^d$ mit $||h||_x < 1$ gilt:

$$(1 - ||h||_x)^2 \nabla^2 f(x) \preceq \nabla^2 f(x+h) \preceq \frac{1}{(1-||h||_x)^2} \nabla^2 f(x).$$

Lemma 5.4. (siehe [14]). Für alle $h \in \mathbb{R}^d$ mit $||h||_x < 1$ gilt:

$$f(x+h) \leq f(x) + \langle \nabla f(x), h \rangle + \rho(||h||_x)$$

mit $\rho(x) := -\ln(1-x) - x$. Aus der Analysis des Newton Verfahrens wird außerdem das sogenannte "Newton decrement" $\lambda(x) := \sqrt{\nabla f(x)^T \nabla^{-2} f(x) \nabla f(x)}$ definiert.

Lemma 5.5 (siehe [14]). Falls $\lambda(x) < 1$, dann gilt:

$$||x - x^*||_x \leq \frac{\lambda(x)}{1 - \lambda(x)}.$$

Dieses Lemma besagt, dass in dem Fall, wenn das Newton decrement kleiner als eins ist, das Minimum der Funktion in dem Dikin Ellipsoid liegt (vgl. [1]).

5.2. Konditionszahlunabhängige Algorithmen

In diesem Kapitel wird der Algorithmus ellipsoidCover eingeführt und beschrieben. In Theorem 5.9 wird gezeigt, dass dieser eine lineare Konvergenz aufweist. Außerdem wird gezeigt, dass die Laufzeit unabhängig von der Konditionszahl ist.

Algorithmus 5 ellipsoidCover

Input: Self-concordant $f(x) = \sum_{i=1}^m f_i(x), T, r \in 0, 1$, initial point $x_1 \in \mathcal{K}, \gamma > 1, S_1, \eta, T$
Initialize $x_{\text{curr}} = x_1$
while $\lambda(x_{\text{curr}}) > \frac{1}{1+r}$ **do**
  $step = \gamma(1 + \lambda(x_{\text{curr}}))$
  $x_{\text{curr}} = x_{\text{curr}} - \frac{1}{step} \nabla^{-2} f(x_{\text{curr}}) \nabla f(x_{\text{curr}})$
end while
$x_{T+1} = \textbf{N-SVRG}(W_r(x), f(x), \nabla^2 f(x_{\text{curr}}), S_1, \eta, T)$
return x_{T+1}

Der Algorithmus ellipsoidCover besteht aus einer Schleife, bei der in jedem Iterationsschritt überprüft wird, ob der derzeitige Iterationsschritt im Dikin Ellipsoid liegt. Solange dies nicht der Fall ist, wird das Newton Verfahren angewandt. Es kann gesehen werden, dass sich die Funktion nach einer konstanten Anzahl an Schritten des Newton Verfahrens in einem Ellipsoid aufhält und die Funktion gut konditioniert ist in Bezug auf die Norm im Ellipsoid (siehe [1]). Wenn dies der Fall ist, kann ein gewünschter erste Ordnung Algorithmus gelaufen werden. Hier wird SVRG gewählt, welcher in Algorithmus 6 definiert ist. Dabei wird für Algorithmus 5 eine Variante für allgemeinere Normen definiert, N-SVRG genannt. Für N-SVRG wird außerdem in Theorem 5.8 die Konvergenzrate gezeigt.

Algorithmus 6 N-SVRG

Input: Convex set $\mathcal{K}$, $f(x) = \frac{1}{m}\sum_{k=1}^{m} f_k(x)$, Norm $\mathcal{A} \succeq 0$, update frequency S_1, and learning rate η
Initialize $\tilde{x}_0$
for $s = 1, 2 \dots$ **do**
  $\tilde{x} = \tilde{x}_{s-1}$
  $\tilde{\mu} = \frac{1}{m}\sum_{k=1}^{m} \nabla f_k(\tilde{x})$
  $x_0 = \tilde{x}$
  for $t = 1$ to S_1 **do**
    Randomly choose $i_t \in \{1 \dots m\}$
    $g = \nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{\mu}$
    $x_t = \prod_{\mathcal{K}}^{A}(x_{t-1} - \eta A^{-1} g)$
  end for
  $\tilde{x}_s = x_t$ for randomly chosen $t \in \{1, \dots, m\}$
end for

Bevor nun das Haupttheorem zu dem Konvergenzverhalten von ellipsoidCover formuliert wird, müssen drei weitere Lemma definiert werden.

Lemma 5.6. (siehe [1]) Sei die Funktion f selbst-konkordant in $\mathcal{K}$. Sei $0 < r < 1$ und $x \in \mathcal{K}$. Sei $W_r(x)$ das Dikin Ellipsoid mit Radius r Weiterhin sei $\alpha = (1-r)^2$ und $\beta = (1-r)^{-2}$. Dann gilt für alle h mit $x + h \in W_r(x)$:

$$\alpha \nabla^2 f(x) \preceq \nabla^2 f(x+h) \preceq \beta \nabla^2 f(x).$$

Lemma 5.7. (siehe [1]) Sei die Funktion f selbst-konkordant in $\mathcal{K}$. Sei $0 < r < 1$ und $\gamma \geq 1$. Betrachte die Schritte des gedämpften Newtonverfahren von Algorithmus 5 (die Schleife) mit Anfangspunkt x_1. Dann gilt: Die Anzahl der Schritte t in Algorithmus 5, bevor der Minimierer von f im Dikin Ellipsoid mit Radius r der aktuellen Iteration, das heißt

$x^* \in W_r(x_t)$, ist höchstens $t = \frac{f(x_1)-f(x^*)}{\nu}$, wobei ν eine Konstante ist, die von γ, r abhängt.

Beweis. Sei $\lambda(x)$ das Newton decrement zu x. Es wird nun eine Fallunterscheidung vorgenommen. Sei $\lambda(x) \leq \frac{r}{1+r}$. Mit $0 < r < 1$ ist dann auch $\lambda(x) < 1$. Dann kann Lemma 5.5 angewandt werden und es gilt:

$$||x - x^*||_x \leq \frac{\lambda(x)}{1 - \lambda(x)} \leq r.$$

Für den Punkt $x = x_t$ und $\lambda(x_t) \leq \frac{r}{1+r}$ erhält man dann:

$$||x - x^*||_x \leq r.$$

Das heißt der Minimierer ist im Dikin Ellipsoid, also $x^* \in W_r(x_t)$. Sei also $\lambda(x_t) > \frac{r}{1+r}$. Nur dann ist x^* vielleicht nicht in $W_r(x_t)$. Mit der Definition der Iterationsvorschrift

$$x_{t+1} = x_t - \frac{1}{\gamma(1 + \lambda(x_t))} \nabla^{-2} f(x_t) \nabla f(x_t),$$

erhält man mit Lemma 5.4, $h = x_{t+1} - x_t$ und $x = x_t$:

$$f(x_{t+1}) \leq f(x_t) - \frac{1}{\gamma(1 + \lambda(x_t))} \nabla f(x_t)^\top ([\nabla^2 f(x_t)]^{-1} \nabla f(x_t)) + \rho\Big(||x_{t+1} - x_t||\Big)$$

Mit der Definition von $\lambda(x)$ ergibt sich:

$$\begin{aligned} f(x_{t+1}) &\leq f(x_t) - \frac{(\lambda(x_t))^2}{\gamma(1 + \lambda(x_t))} + \rho\Big(\frac{\lambda(x_t)}{\gamma(1 + \lambda(x_t))}\Big) \\ &= f(x_t) - \frac{(\lambda(x_t))^2}{\gamma(1 + \lambda(x_t))} - \ln\left(1 - \frac{\lambda(x_t)}{\gamma(1 + \lambda(x_t))}\right) - \frac{\lambda(x_t)}{\gamma(1 + \lambda(x_t))} \\ &= f(x_t) - \frac{\lambda(x_t)}{\gamma} + \ln\left(1 + \frac{\lambda(x_t)}{\gamma + (\gamma - 1)\lambda(x_t)}\right) \end{aligned}$$

Einige weitere Umformungen ergeben dann:

$$\begin{aligned} f(x_t) - f(x_{t+1}) &\geq \frac{\lambda(x_t)}{\gamma} - \ln\Big(1 + \frac{\lambda(x_t)}{\gamma + (\gamma - 1)\lambda(x_t)}\Big) \\ &\geq \frac{\lambda(x_t)}{\gamma} - \frac{\lambda(x_t)}{\gamma + (\gamma - 1)\lambda(x_t)} \\ &= \lambda(x_t)\Big(\frac{1}{\gamma} - \frac{1}{\gamma + (\gamma - 1)\lambda(x_t)}\Big) \\ &> \frac{r}{1 + r}\Big(\frac{1}{\gamma} - \frac{1}{\gamma + \frac{(\gamma-1)r}{1+r}}\Big) \\ &> 0 \end{aligned}$$

wobei die zweite Ungleichung folgt wegen der Regel $\ln(1+x) \leq x$ für $x > -1$. Es sei angemerkt, dass die Regel, die in [1] (S. 36) benutzt wird, nicht korrekt ist. Setze $\nu = \frac{r}{1+r}\left(\frac{1}{\gamma} - \frac{1}{\gamma + \frac{(\gamma-1)r}{1+r}}\right)$. Dann sieht man, dass nach

$$\frac{f(x_1) - f(x^*)}{\nu}$$

Iterationsschritten, garantiert werden kann, dass man den Punkt x_t erreicht hat mit $x^* \in W_r(x_t)$.

Lemma 5.8. (siehe [1]) Sei f eine konvexe Funktion. Angenommen, es existiert eine konvexe Menge $\mathcal{K}$ und eine positiv semidefinite Matrix A, so dass für alle $x \in \mathcal{K}$ gilt: $\alpha A \preceq \nabla^2 f(x) \preceq \beta A$. Dann gilt folgende Ungleichung zwischen zwei vollen Gradientenschritten von Algorithmus 6:

$$\mathbb{E}[f(x_s) - f(x^*)] \leq \left(\frac{1}{\alpha\eta(1-2\eta\beta)S_1} + \frac{2\eta\beta}{(1-2\eta\beta)}\right)\mathbb{E}[f(x_{s-1}) - f(x^*)].$$

Beweis. Der Beweis verläuft ähnlich zu dem von SVRG in [5]. Die Änderungen ergeben sich durch die verallgemeinerte Norm. Dieser Beweis, der aus [1] kommt, enthält einige Schreibfehler. Diese wurden entsprechend korrigiert. Für beliebiges i betrachte

$$g_i(x) = f_i(x) - f_i(x^*) - \langle \nabla f_i(x^*), x - x^* \rangle.$$

Es gilt $g_i(x^*) = \min_x g_i(x)$ wegen $\nabla g_i(x^*) = 0$. Da f_i β-glatt sind folgt auch, dass g_i β-glatt ist. Dann gilt:

$$\begin{aligned} 0 = g_i(x^*) &\leq \min_\eta \left(g_i(x - \eta A^{-1}\nabla g_i(x))\right) \\ &\leq \min_\eta \left(g_i(x) - \eta||\nabla g_i(x)||^2_{A^{-1}} + \frac{1}{2}\beta\eta^2||\nabla g_i(x)||^2_{A^{-1}}\right) \\ &= g_i(x) - \frac{1}{2\beta}||\nabla g_i(x)||^2_{A^{-1}}, \end{aligned}$$

mit $\eta = \frac{1}{\beta}$. Weitere Umformungen ergeben: Summieren über $i = 1, ..., m$ und setzt man x^* als das Minimum von f, also $\nabla f(x^*) = 0$, ergibt:

$$\frac{\sum_{i=1}^m ||\nabla f_i(x) - \nabla f_i(x^*)||^2_{A^{-1}}}{2m\beta} \leq f(x) - f(x^*). \tag{5.1}$$

Definiere nun $v_t = \nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{\mu}$. Konditioniert auf x_{t-1} ergibt sich für den

Erwartungswert mit Bezug auf i_t:

$$\begin{aligned}\mathbb{E}[||v_t||^2_{A^{-1}}] &\leq 2\mathbb{E}[||\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(x^*)||^2_{A^{-1}}] + 2\mathbb{E}[||\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*) - \nabla f(\tilde{x})||^2_{A^{-1}}] \\ &\leq 2\mathbb{E}[||\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(x^*)||^2_{A^{-1}}] + 2\mathbb{E}[||\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)||^2_{A^{-1}}] \\ &\leq 4\beta\big(f(x_{t-1}) - f(x^*) + f(\tilde{x}) - f(x^*)\big). \qquad (5.2)\end{aligned}$$

Die Ungleichungen folgen aufgrund dieser drei Fakten:

- $||a+b||^2_{A^{-1}} \leq 2||a||^2_{A^{-1}} + 2||b||^2_{A^{-1}}$, für $a, b \in \mathbb{R}^d$.
- $\tilde{\mu} = \nabla f(\tilde{x})$.
- $\mathbb{E}[||X - \mathbb{E}[X]||^2_{A^{-1}}] \leq \mathbb{E}[||X||^2_{A^{-1}}]$, für $X \in \mathbb{R}^{dxd}$

Insbesondere folgt die dritte Ungleichung wegen (5.1). Konditionert auf x_{t-1} ergibt $\mathbb{E}[v_t] = \nabla f(x_{t-1})$. Dann folgt:

$$\begin{aligned}\mathbb{E}[||x_t - x^*||^2_A] &\leq ||x_{t-1} - \eta A^{-1} v_t - x^*||^2_A \\ &= ||x_{t-1} - x^*||^2_A - 2\eta\langle x_{t-1} - x^*, AA^{-1}\mathbb{E}[v_t]\rangle + \eta^2\mathbb{E}[||v_t||^2_{A^{-1}}] \\ &\leq ||x_{t-1} - x^*||^2_A - 2\eta\langle x_{t-1} - x^*, \nabla f(x_{t-1})\rangle + 4\beta\eta^2\big(f(x_{t-1}) - f(x^*) + f(\tilde{x}) - f(x^*)\big) \\ &\leq ||x_{t-1} - x^*||^2_A - 2\eta\big(f(x_{t-1}) - f(x^*)\big) + 4\beta\eta^2\big(f(x_{t-1}) - f(x^*) + f(\tilde{x}) - f(x^*)\big)\big) \\ &\leq ||x_{t-1} - x^*||^2_A - 2\eta(1 - 2\eta\beta)(f(x_{t-1}) - f(x^*)) + 4\beta\eta^2(f(\tilde{x}) - f(x^*)).\end{aligned}$$

Die erste Ungleichung ergibt sich nach Definition des Algorithmus. Die zweite Ungleichung folgt wegen (5.2) und die dritte benutzt die Konvexität von $f(x)$. Betrachte die fixierte Variable s, so dass $\tilde{x} = \tilde{x}_{s-1}$ und $\tilde{x}_s$ ausgewählt werden, nachdem alle Updates berechnet wurden. Summieren über $t = 1, ..., S_1$ und der Erwartungswert ergeben:

$$\begin{aligned}\mathbb{E}[||x_{S_1} - x^*||^2_A] + 2\eta(1 - 2\eta\beta)S_1\mathbb{E}[f(x_s) - f(x^*)] &\leq \mathbb{E}[||x_0 - x^*||^2_A] + 4\beta S_1\eta^2\mathbb{E}[f(\tilde{x}) - f(x^*)] \\ &= ||x_0 - x^*||^2_A + 4\beta S_1\eta^2\mathbb{E}[f(\tilde{x}) - f(x^*)] \\ &\leq 2\frac{\mathbb{E}[f(\tilde{x}) - f(x^*)]}{\alpha} + 4\beta S_1\eta^2\mathbb{E}[f(\tilde{x}) - f(x^*)] \\ &= 2\left(\frac{1}{\alpha} + 2\beta S_1\eta^2\right)\mathbb{E}[f(\tilde{x}) - f(x^*)].\end{aligned}$$

Die zweite Ungleichung ergibt sich dadurch, dass f α-streng konvex ist. Damit gilt dann schließlich

$$\mathbb{E}[f(x_s) - f(x^*)] \leq \left(\frac{1}{\alpha\eta(1 - 2\eta\beta)n} + \frac{2\eta\beta}{(1 - 2\eta\beta)}\right)\mathbb{E}[f(x_{s-1}) - f(x^*)].$$

Schließlich kann das Theorem zur linearen Konvergenz des Algorithmus ellipsoidCover formuliert werden.

Theorem 5.9. (siehe [1]) Sei $0 < r < 1$ und $\gamma \geq 1$. Sei ν eine Konstante, die von γ, r abhängt. Setze $\eta = 10(1-r)^2, S_1 = c_r = \frac{50}{(1-r)^4}$,wobei c_r Konstante ist, die von r abhängt. Sei die Anzahl der Iterationsschritte $T = \frac{f(x_1)-f(x^*)}{\nu}$. Dann gilt nach $t > T$ Iterationsschritten die folgende lineare Konvergenz zwischen zwei vollen Gradientenschritten für Algorithmus 5:

$$\mathbb{E}[f(w_s) - f(w^*)] \leq \frac{1}{2}\mathbb{E}[f(w_{s-1} - f(w^*)].$$

Außerdem ist der Rechenaufwand von jedem vollen Gradientenschritt $O(md + c_r d^2)$, wobei c_r eine Konstante ist, die von r abhängt (und unabhängig von der Konditionszahl ist).

Beweis. Für $t = \frac{f(x_1)-f(x^*)}{\nu}$ kann Lemma 5.7 benutzt werden. Damit folgt, dass der Minimierer im Dikin Ellipsoid mit Radius r ist, das heißt $x^* \in W_r(x_t)$, wobei ν eine Konstante ist, die von γ, r abhängt. Mit Lemma 5.6 ist dann folgende Darstellung gegeben für alle $x \in W_r(x_t$:

$$(1-r)^2\nabla^2 f(x_T) \preceq \nabla^2 f(x+h) \preceq (1-r)^{-2}\nabla^2 f(x_T)$$

Mit Lemma 5.8 und der Wahl der Parameter folgt die Behauptung.

KAPITEL 6

Simulation

Dieses Kapitel behandelt experimentelle Anwendungen der zuvor erarbeiteten theoretischen Ergebnisse. Während im experimentellen Teil von [1] mit vier verschiedenen Datensätzen gearbeitet wird (siehe [1], Tabelle 2), beziehen sich die Ergebnisse dieses Kapitels auf den MNIST4-9 Datensatz. Der zugehörige Python code, mit Hilfe dessen ein Großteil der Abbildungen dieses Kapitels erstellt wurden, befindet sich im Anhang. Insbesondere wurde der LiSSA Algorithmus aus dem theoretischen Teil dieser Arbeit in Python implementiert.

Der MNIST (*Modified National Institute of Standards and Technology*) Datensatz umfasst 60.000 Beispiele handgeschriebener Ziffern und ist als experimenteller Datensatz für Klassifikationsprobleme weit verbreitet. Jedes Beispiel bildet eine 28×28 dimensionale Pixel-Matrix mit Graustufen Einträgen (Werte im Bereich [0, 255], skaliert auf [0, 1]), die in einen 784-dimensionalen Vektor entfalten wurden. Zudem liegen Labels vor, welche jedem Beispiel die entsprechende Ziffer aus $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ klar zuordnen. Üblicherweise werden mit Hilfe des MNIST Datensatzes multiple oder binäre Klassifikationsprobleme untersucht, indem man versucht, von den transformierten Bilddaten im 784-dimensionalen Vektor zum korrekten Label und somit auf die Korrekte Ziffer des jeweiligen Beispiels zu gelangen. Werden Beispiele von allen oder mehr als zwei der Ziffern aus $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ berücksichtigt, so handelt es sich um ein multiples Klassifikationsproblem, da mehrere Klassen vorliegen. Wird ein reduzierter Datensatz mit lediglich Beispielen zweier Klassen gewählt, so handelt es sich um ein binäres Klassifikationsproblem.

Analog zum experimentellen Vorgehen in [1] wird in den Ergebnissen dieses Kapitels mit einem reduzierten MNIST Datensatz bestehend aus zwei Klassen gearbeitet. Der Datensatz wurde von `http://deeplearning.net/data/mnist/` geladen und liegt in drei Teilen vor: einem Trainnigsteil mit 50.000 Beispielen (75% der Daten) und einem Validations- und Testteil mit jeweils 10.000 Beispielen (jeweils 12,5% der Daten). Für die Zwecke der Simulationsstudien in diesem Kapitel wird lediglich der Trainingsteil verwendet. Zudem wird sich auf das binäre Klassifikationsproblem beschränkt, in dem nur Beispiele zweier Ziffern berücksichtigt werden. Aus den 50.000 Beispielen werden Beispiele der Ziffern '4' und '9' gewählt. Nach der Wahl dieser Ziffern enthält der reduzierte Datensatz insgesamt 9.847

Beispiele (4859 Beispiele der Ziffern '4' und 4988 Beispiele der Ziffern '9'). Abbildung 6.1 zeigt zur Veranschaulichung 40 Beispiele, die zufällig aus den 9.847 gewählt wurden. Nulleinträge in den Elementen der 28×2 dimensionalen Matrix eines Beispiels werden im Sinne der Graustufen als weiß interpretiert, während Einträge näher zu 1 als schwarz interpretiert werden.

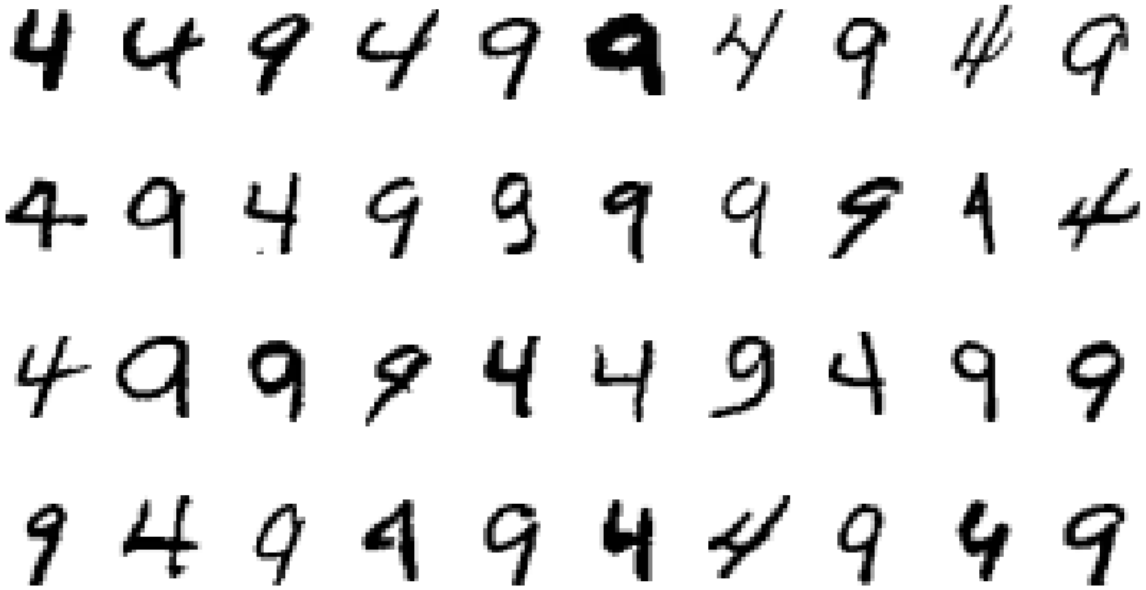


Abbildung 6.1.: Veranschaulichung von 40 zufällig gewählten Bildbeispielen aus dem reduzierten MNIST Datensatz.

Abhängig vom experimentellen Versuchsaufbau, kann im Fall des binären Klassifikationsproblems das Label y transformiert werden um Werte in $\{-1, 1\}$ oder $\{0, 1\}$ anzunehmen. Analog zu [1] wird in diesem Kapitel mit y Werten in $\{-1, 1\}$ gearbeitet indem die zugehörigen Labels der Ziffern '4' zu 1 und der Ziffern '9' zu -1 umgewandelt werden. Prinzipiell, wird in der Literatur und Praxis oft die Verlustfunktion der logistischen Regression in folgender Schreibweise verwendet (hier ohne Regularisierungsterm dargestellt):

$$\mathcal{L}(x) = -\frac{1}{m}\Big(\sum_{k=1}^{m} y_k \log \sigma(x^\top v_k) + (1 - y_k)\log(1 - \sigma(x^\top v_k))\Big) \tag{6.1}$$

$$\text{mit } \sigma(x^\top v_k) := \frac{1}{1 + \exp\{-x^\top v_k\}}, \quad y_k \in \{0, 1\},\ v_k, x \in \mathbb{R}^d \tag{6.2}$$

Analog zum Vorgehen in [1] wird jedoch in diesem Kapitel die folgende logistische Regression mit l_2 Regularisierungsterm verwendet:

$$f(x) = \frac{1}{m}\sum_{k=1}^{m} \log\left(1 + \exp\{-y_k x^\top v_k\}\right) + \lambda||x||^2, \quad y_k \in \{\pm 1\},\ v_k, x \in \mathbb{R}^d \tag{6.3}$$

Wobei $(v_k, y_k),\ k = 0, 1, \dots m-1$ ($m = 9.847$) Datentupel darstellen, mit Daten $v_k \in \mathbb{R}^d$, zugehörigen Labels $y_k \in \{-1, 1\}$, und Regularisierungsfaktor λ. Die Parameter der logistischen Regression $x \in \mathbb{R}^d$ resultieren somit aus dem folgenden Optimierungsproblem:

$$\min_{x \in \mathbb{R}^d} f(x) = \min_{x \in \mathbb{R}^d} \left\{ \frac{1}{m} \sum_{k=1}^{m} \log\left(1 + \exp\{-y_k x^\top v_k\}\right) + \lambda ||x||^2 \right\} \tag{6.4}$$

Für den Gradienten $\nabla f(x)$ ergibt sich:

$$\nabla f(x) = \frac{1}{m} \sum_{k=1}^{m} \left(- y_k \frac{\exp\{-y_k x^\top v_k\}}{1 + \exp\{-y_k x^\top v_k\}} v_k \right) + 2\lambda x, \tag{6.5}$$

und für die Hesse Matrix $\nabla^2 f(x)$ ergibt sich:

$$\nabla^2 f(x) = \frac{1}{m} \sum_{k=1}^{m} \left(y_k^2 \frac{\exp\{-y_k x^\top v_k\}}{(1 + \exp\{-y_k x^\top v_k\})^2} v_k v_k^\top \right) + 2\lambda I. \tag{6.6}$$

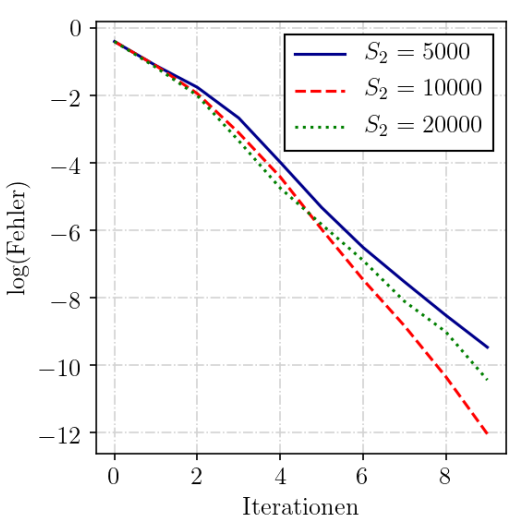
Aus der Theorie zum LiSSA Algorithmus ergibt sich $S_2 \geq 2\hat{\kappa}_l \ln(4\hat{\kappa}_l) \sim \hat{\kappa}_l \ln(\hat{\kappa}_l)$, mit der Definition der Konditionszahl $\hat{\kappa}_l$ aus Kapitel 2. In den Experimenten der Authoren in [1] ergibt sich für S_2 der Wert m (siehe [1], S. 31). In der Anwendung des LiSSA Algorithmus folgen wir zudem [1] und setzen $S_1 = 1$. Abbildung 6.2 zeigt die Konvergenzraten des LiSSA Algorithmus in Abhängigkeit verschiedener Werte für S_2.

Mit dem Ziel, die Abbildung 5 auf S. 32 in [1] zu replizieren, wurden die drei Werte $S_2 \in \{5.000, 10.000, 20.000\}$ gewählt und der LiSSA Algorithmus für $T = 10$ Iterationen angewandt. Hierzu wurde x_1 zunächst mit Hilfe des Gradientenverfahrens errechnet. Weitere verwendete Parameter zur Anwendung des LiSSA Algorihmus sind $m = 9.847, \lambda = 0,0001$ (entspricht ungefähr $\frac{1}{m}$) und $d = 784$. Gemäß [1] wird der Fehler des Algorithmus in Abbildungen 6.2 a) und b) als $\log(f(x_t) - f(x_t^*))$ angegeben, wobei $f(x_t^*)$ der (optimale) Funktionswert der Verlustfunktion nach Anwendung von 500 Iterationen des LiSSA Algorithmus ist. Abbildung 6.2 a) zeigt den erreichten Fehler nach $T = 10$ Iterationen und Abbildung 6.2 b) zeigt die benötigte Laufzeit.

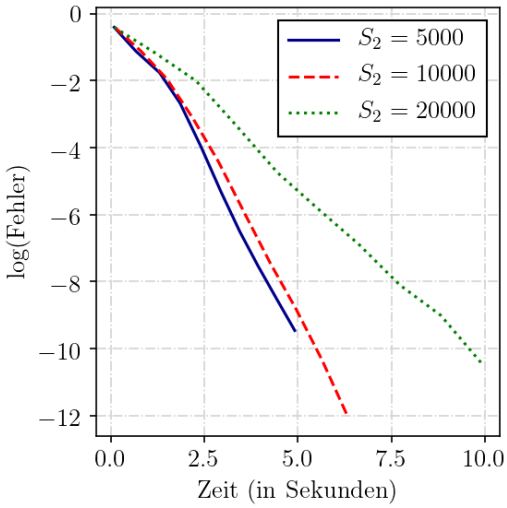
Die Ergebnisse bestätigen dass für den verwendeten reduzierten MNIST Datensatz, $S_2 = 10.000$ eine optimale Wahl darstellt. Im Gegenzug zu den experimentellen Simulationen in [1], wo $S_1 = 1$ gesetzt wird und die Auswahl für S_2 sich lediglich auf die Auswahl $S_2 \in \{5.000, 10.000, 20.000\}$ begrenzt, untersucht dieses Kapitel als weiterführende Analyse eine größere Variation beider Parameter. Für eine ausführlichere Simulation, wurde der LiSSA Algorithmus auf dem reduzierten MNIST Datensatz für $T = 10$ mit den oben beschriebenen Parametern und der Auswahl $S_1 \in \{1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$ und $S_2 \in \{1000, 5000, 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000\}$ angewandt. Die berücksichtigten Werte für S_1 und S_2 ergeben insgesamt $11 \times 11 = 121$ Simulationens-

läufe.

Abbildung 6.2.: Konvergenzraten des LiSSA Algorithmus in Abhängigkeit von S_2



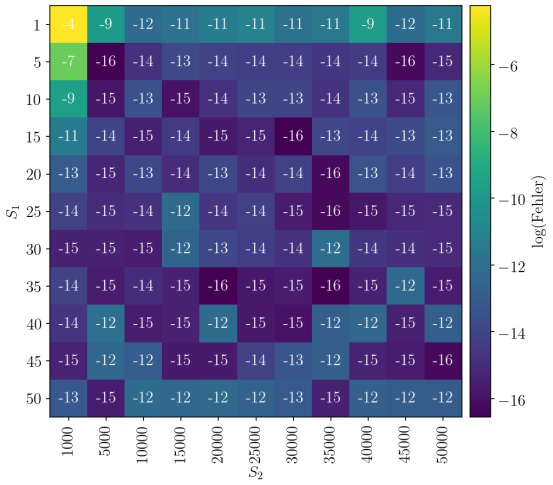
a) Erreichter Fehler nach $T = 10$ Iterationen für Werte von S_2 in $\{5000, 10000, 20000\}$.



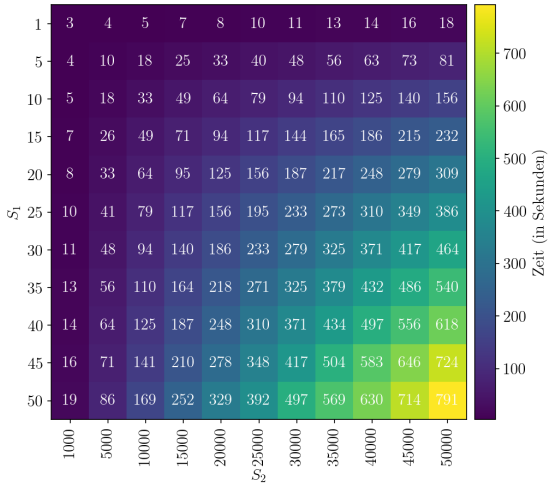
b) Benötigte Laufzeit und erreichter Fehler nach $T = 10$ Iterationen für Werte von S_2 in $\{5000, 10000, 20000\}$.

Abbildungen 6.4 a) und b) veranschaulichen die Ergebnisse der erweiterten Simulationsstudie. Aus Abbildung 6.4 a) ergibt sich nach wie vor, dass für S_1 der optimale S_2 Wert bei 10.000 liegt, da hier nach $T = 10$ Iterationen der geringste Fehler (unfegähr -12) vorliegt.

Abbildung 6.4.: Fehler und Laufzeit des LiSSA Algorithmus in Abhängigkeit von S_1 und S_2



a) Erreichter Fehler nach $T = 10$ Iterationen für verschiedene Werte S_1 und S_2. Zur Veranschaulichung wurden die Fehlerwerte auf ganze Zahlen gerundet.



b) Benötigte Laufzeit nach $T = 10$ Iterationen für verschiedene Werte S_1 und S_2. Zur Veranschaulichung wurden die Laufzeiten auf ganze Sekunden gerundet.

Abbildung 6.4 b) zeigt, dass diese Anwendung eine Laufzeit von ungefähr 5 Sekunden hat. Zwar liegt mit $S_1 = 1$ mit $S_2 = 45.000$ ein ähnlich niedriger Fehler vor (ebenfalls ungefähr -12), dieser wird jedoch erst mit einer über drei mal längeren Laufzeit von 16 Sekunden erreicht (siehe Abbildung 6.4 b). Insgesamt zeigt Abbildung 6.4 b) die zu erwartende Laufzeit bei verschiedenen S_1 und S_2 Werten mit einer Laufzeit von 791 Sekunden oder über 13 Minuten für die rechenaufwändigste der gewählten Kombination ($S_1 = 50, S_2 = 50.000$). Die komplette Simulation benötigte ungefähr 12 Stunden. In [1] wurden weitere Simulationen gemacht, bei denen LiSSA mit verschiedenen Algorithmen verglichen wurde, darunter erste Ordnung Algorithmen (siehe [1] Figure 4), beschleunigte erste Ordnung Algorithmen (siehe [1], Figure 2) und zweite Ordnung Algorithmen (siehe [1] Figure 3).

KAPITEL 7

Fazit und Ausblick

In dieser Arbeit wurden zur Lösung von Optimierungsproblemen Algorithmen zweiter Ordnung behandelt. Die Motivation hinter LiSSA war es, mit Hilfe einer Approximation des Newton Verfahrens von einer quadratischen zu einer linearen Konvergenz überzugehen um schnellere Laufzeiten bei gleichzeitiger Reduktion des benötigten Rechenaufwandes zu erzielen. Dabei wurde ein stochastischer Schätzer definiert um die Berechnung der Hesse-Matrix und dessen Inverse zu vermeiden. In Kapitel 3 wurde gezeigt, dass dieser Schätzer in linearer Zeit berechnet werden kann. Für den Parameter S_1 von LiSSA wurde in der Theorie die Schranke $O(\hat{\kappa}_l^{\max})$ gezeigt. In der Simulation wurde jedoch beobachtet, dass eine kleine Konstante (zum Beispiel S_1) ausreichend ist. Ähnliche Überlegungen wurden in [1] gemacht. Dort wird die Annahme getroffen, dass S_1 zu $O(1)$ verbessert werden kann, was für zukünftige Arbeiten interessant wäre.

Als nächstes wurde LiSSA-Sample eingeführt. Der Idee von LiSSA-Sample lag zu Grunde, dass man mithilfe des Newton Verfahrens von einem konvexen Optimierungsproblem zur Lösung von quadratischen Unterproblemen übergehen kann. LiSSA-Sample verbindet dabei Methoden erster Ordnung mit Techniken, bei denen die Bestandteile einer Matrix in eine Gruppe aufgeteilt werden und aus dieser Gruppe einer Stichprobe gezogen wird um die ursprüngliche Matrix zu approximieren (sogenannte Matrix Sampling Methoden). In Kapitel 4 wurden die Laufzeiten zu LiSSA-Sample gezeigt. Für den Fall, in dem die Anzahl der Trainingsdaten sehr viel höher ist als die Dimension des Optimierungsproblems, ist LiSSA-Sample nach Aussage von [1] der theoretisch schnellste Algorithmus.

Anschließend behandelte Kapitel 5 selbst-konkordanten Funktionen, welche eine Untermenge konvexer Funktionen darstellen. Auch hier wurde ein Algorithmus ellipsoidCover eingeführt, welcher lineare Konvergenz aufweist und dessen Laufzeiten unabhängig von der Konditionszahl der Funktion ist.

In Kapitel 6 wurde der Algorithmus LiSSA implementiert und die theoretischen Ergebnisse überprüft und bestätigt. Für zukünftige Arbeiten wäre es interessant die Algorithmen

LiSSA-Sample und ellipsoidCover zu implementieren, die theoretischen Ergebnisse zu überprüfen und einen Vergleich zu herkömmlichen Algorithmen zu machen.

Abbildungsverzeichnis

ANHANG A

Literaturverzeichnis

[1] Agarwal, N., Bullins, B., and Hazan, E. (2017). Second-order stochastic optimization for machine learning in linear time. The Journal of Machine Learning Research, 18(1), 4148-4187.

[2] Hongzhou Lin, Julien Mairal, and Zaid Harchaoui. A universal catalyst for first-order optimization. In Advances in Neural Information Processing Systems, pages 3384– 3392, 2015.

[3] Michael B. Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. In Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, pages 181–190. ACM, 2015.

[4] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul):2121–2159, 2011.

[5] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In Advances in Neural Information Processing Systems, pages 315– 323, 2013.

[6] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In Advances in Neural Information Processing Systems, pages 1646–1654, 2014.

[7] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. Journal of Machine Learning Research, 14(Feb):567– 599, 2013.

[8] EE227C: Convex Optimization and Approximation, taught at UC Berkeley in Spring, 2018., Link: https://ee227c.github.io/notes/ee227c-notes.pdf, Zuletzt aufgerufen am 19.08.2019.

[9] Yurii Nesterov. Introductory Lectures on Convex Optimization: A Basic Course. Springer Science and Business Media, 2013.

[10] Joel A Tropp. User-friendly tail bounds for sums of random matrices. Foundations of Computational Mathematics, 12(4):389–434, 2012.

[11] https://resources.mpi-inf.mpg.de/departments/d1/teaching/ss10/MFI2/kap50.pdf, Zuletzt aufgerufen am 19.08.2019.

[12] Stewart, Gilbert (1998). Matrix Algorithms: Basic decompositions. SIAM. p. 55.

[13] Convex Optimization: Algorithms and Complexity, Sebastien Bubeck

[14] Arkadi Nemirovski. Interior point polynomial time methods in convex programming. Lecture notes, 2004.

[15] Lijun Zhang, Mehrdad Mahdavi, and Rong Jin. Linear convergence with condition number independent access of full gradients. In Advances in Neural Information Processing Systems, pages 980–988, 2013.

[16] Nicolas L. Roux, Mark Schmidt, and Francis R. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In Advances in Neural Information Processing Systems, pages 2663–2671, 2012.

[17] Shai Shalev-Shwartz and Tong Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. Mathematical Programming, 155(1-2):105– 145, 2016.

[18] Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. arXiv preprint arXiv:1603.05953, 2016.

ANHANG B

Quellcode

B.0.1. Anmerkungen

Implementierung: Alle Ergebnisse wurden mit Hilfe von Python (Version 3.7) auf einem Microsoft Windows® 8 Rechner mit 8GB Arbeitsspeicher und einem Intel®Core™ i5-3210M mit 2.50GHz Taktfrequenz erzeugt.

Datensatz: Im experimentellen Teil der Masterarbeit wird der MNIST Datensatz verwendet. Dieser besteht aus 70.000 Beispielen handgeschriebener Ziffern. Jedes Beispiel repraesentiert ein Bild mit einer Aufloesung von 28×28 Pixeln. Die Graustufen Werte dieser Pixel (Eintraege im Bereich [0, 255], skaliert auf [0, 1]) liegen im MNIST Datensatz in Form von 784-dimensionalen Vektoren vor. Zu jedem Beispiel liegt zudem ein Label vor, welches dem Beispiel eine Ziffer aus $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ klar zuordnet. Der MNIST Datensatz wurde von der folgenden Website heruntergerladen: `http://deeplearning.net/data/mnist/`

Code Darstellung: Der verwendete Python code wird in den nachfolgenden Unterkapiteln in Segmenten dargestellt und ausfuerlich kommentiert. Zeilenumbrueche im code werden durch das Symbol ↪ dargestellt.

B.0.2. Generation des Datensatzes zum binaeren Klassifikationsproblem

Datei: `Datensatz_Management.py`

```
1 import gzip
2 import pickle
3 import numpy as np
4 import pandas as pd
5 from sklearn.preprocessing import normalize
6
```

```
7  def Generiere_Datensatz():
8
9  # Lade MNIST Datensatz. Der gespeicherte Datensatz ist in 3
       ↪ Teile
10 # getrennt (Trainingsteil: 75% der Daten, Validationsteil: 12,5%
       ↪  der
11 # Daten, Testteil: 12,5% der Daten). Diese Aufteilung wird
12 # typischerweise in der Praxis des machinellen Lernens verwendet
       ↪ .
13 # Zum Zwecke der Simulationen in dieser Masterarbeit bilden wir
       ↪ unseren
14 # Datensatz aus dem ersten Tripel des MNIST Datensatzes. Dies
       ↪ ergibt
15 # insgesamt 50.000 Beispiele handgeschriebener Ziffern.
16 f = gzip.open('mnist.pkl.gz', 'rb')
17 Trainingsdaten, Validierungsdaten, _ = pickle.load(f, encoding='
       ↪ latin1')
18 f.close()
19 Daten = pd.DataFrame(Trainingsdaten[0])
20 Labels = pd.Series(Trainingsdaten[1])
21
22 # Analog zum Paper, Kapitel 7.1, begrenzen wir uns auf
       ↪ Bildbeispiele der
23 # handgeschriebenen Ziffern 4 und 9. Somit wird das multiple
24 # Klassifikationsproblem zu einem binaeren
       ↪ Klassifikationsproblem
25 # umgewandelt. Es bleiben 9.847 Beispiele der Ziffern 4 und 9.
26 Daten['Label'] = Labels
27 Daten = Daten[(Daten['Label']==4) | (Daten['Label']==9)]
28
29 # Um der Notation in Version 3 des Papers zu folgen, werden die
       ↪ Labels der
30 # Beispiele {4, 9} zu {-1, 1} geaendert.
31 Daten['Label'] = Daten['Label'].replace([4, 9], [1, -1])
32 Daten = [Daten[Daten.columns.values[0:-1]].values, Daten['Label'
       ↪ ].values]
33
34 # Um sicher zu gehen, dass die Norm der Hesse Matrix der
       ↪ Logistischen
35 # Regression abgeschlossen ist, werden alle Eintraege im
       ↪ Datensatz
36 # zur Einheitsnorm skaliert.
37 Daten_normiert = [normalize(Daten[0], axis=1, norm='l2'), Daten
       ↪ [1]]
38
39 # Resultat: m = 9.847 Beispiele von 784-dimensionalen Vektoren
       ↪ und binaere
```

```
# Labels y bilden den Datensatz: (v_k, y_k), k=0,1,...,m-1
v, y = Daten_normiert
return v, y
```

Quellcode B.1: .

B.0.3. Hilfsfunktionen zum LiSSA Algorithmus

Datei: `Hilfsfunktionen.py`

```
import numpy as np
import Datensatz_Management

v,y = Datensatz_Management.Generiere_Datensatz()

# f_k : Logistische Regression fuer ein einzelnes Datentupel
def f_k(x, k):
z = (-1) * y[k] * np.dot(x, v[k])
return np.log(1 + np.exp(z))

# f : Logistische Regression fuer den kompletten Datensatz mit
    ↪ Groesse m
def f(x, m, Lambda):
f = 0
for k in range(m):
f = f + f_k(x, k)
f = f / m
f = f + Lambda*np.dot(x, x)
return f

# Grad_f_k: Gradient der Logistischen Regression fuer ein
    ↪ Datentupel
def Grad_f_k(x, k):
z = (-1) * y[k] * np.dot(x, v[k])
return (-1) * y[k] * v[k] * (np.exp(z)/(1 + np.exp(z)))

# Grad_f: Gradient der Logistischen Regression fuer den
    ↪ kompletten Datensatz
# mit Groesse m
def Grad_f(x, m, Lambda, d):
Grad_f = np.zeros(d)
for k in range(m):
Grad_f = Grad_f + Grad_f_k(x, k)
Grad_f = Grad_f / m
Grad_f = Grad_f + 2 * Lambda * x
```

```
return Grad_f

# Die Hesse Matrix von f kann auch als v_k*(v_k)^T geschrieben
    ↪ werden
def v_k(x, k):
z = (-1) * y[k] * np.dot(x, v[k])
v_k = y[k] * (np.sqrt(np.exp(z))/(1 + np.exp(z))) * v[k]
return v_k

# Hess_f: Hesse Matrix von f, gibt v*vT + (2*lambda*I) wieder
def Hess_f(x, m, Lambda, d):
Hess_f = np.zeros(d, d)
for k in range(m):
Hess_f = Hess_f + np.outer(v_k(x, k), v_k(x, k))
Hess_f = Hess_f / m
Hess_f = Hess_f + 2 * Lambda * np.identity(d)
return Hess_f

# Da die Funktion f aus der Klasse der verallgemeinerten
    ↪ linearen Modelle
# ist, wird das Matrix-Vektor Produkt zum Vektor-Vektor Produkt
    ↪ und wir
# erhalten Konvergenz in O(d) Zeit.
def Hess_f_k_Vektor_Produkt(x, k, vektor, Lambda, d):
Hess_f_k_Vektor_Produkt = np.zeros(d)
v = v_k(x, k)
Hess_f_k_Vektor_Produkt = np.dot(v, vektor) * v
Hess_f_k_Vektor_Produkt = Hess_f_k_Vektor_Produkt + 2 * Lambda *
    ↪  vektor
return Hess_f_k_Vektor_Produkt

# Bestimmung von x_1 mit Hilfe des Gradientenverfahren
def Gradientenverfahren(T_1, x_0, alpha, m, Lambda, d):
x_1 = x_0
for i in range(T_1):
Grad_f_1 = Grad_f(x_1, m, Lambda, d)
x_1 = x_1 - alpha * Grad_f_1
return x_1
```

Quellcode B.2: .

B.0.4. LiSSA Algorithmus

Datei: `LiSSA_Algorithmus.py`

```
import numpy as np
import time
from Hilfsfunktionen import f, Grad_f, Hess_f_k_Vektor_Produkt

# Implementierung von Algorithmus 1 des behandelten Papers. Zur
# Verstaendlichkeit wurde dabei die Symbolnotation des
    ↪ Algorihmus in die
# verwendeten Variablennamen im Code uerbernommen.
def LiSSA(x_1, T, S_1, S_2, m, Lambda, d):
x_t = x_1
Fehler = np.zeros(T)
Zeit = np.zeros(T)
Startzeit = time.time()
for t in range(T):
Fehler[t] = f(x_t, m, Lambda)
Zeit[t] = time.time() - Startzeit
Grad_f_1 = Grad_f(x_t, m, Lambda, d)
X_t = Grad_f_1
X_t_vektor = []
X_t_vektor_sum = np.zeros(d)
for i in range(S_1):
for j in range(S_2):
k = int(np.random.uniform(m))
Hess_f_k_Vektor_Produkt_1 = Hess_f_k_Vektor_Produkt(x_t, k, X_t,
    ↪  Lambda, d)
X_t = Grad_f_1 + X_t - Hess_f_k_Vektor_Produkt_1
X_t_vektor.append(X_t)
X_t_vektor_sum += X_t_vektor[i]
X_t = (1 / S_1) * X_t_vektor_sum
x_t = x_t - X_t
print('LiSSA Algorithmus abgeschlossen.')
return x_t, Fehler, Zeit
```

Quellcode B.3: .

B.0.5. LiSSA Simulation

Datei: LiSSA_Training.py

```
import numpy as np
import matplotlib.pyplot as plt
import sys
sys.path.append("..")
from Hilfsfunktionen import Gradientenverfahren
import Datensatz_Management
```

```
from LiSSA_Algorithmus import LiSSA

# matplotlib Optionen zur Anpassung der Schrift zum verwendeten TeX Stil
plt.rc('font', **{'family':'serif','serif':['Computer Modern'],'size':13})
plt.rc('text', usetex=True)

# Generation des Datensatzes
_, y = Datensatz_Management.Generiere_Datensatz()
```

Quellcode B.4: .

Kapitel 7, Abb. 1

Datei: `MNIST_Beispielplot.py`

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import sys
sys.path.append("..")
import Datensatz_Management

# matplotlib Optionen zur Anpassung der Schrift zum verwendeten TeX Stil
plt.rc('font', **{'family': 'serif', 'serif': ['Computer Modern'], 'size':13})
plt.rc('text', usetex=True)

v, y = Datensatz_Management.Generiere_Datensatz()

plt.figure(figsize=(7,4), dpi=150)
# set the spacing between axes.
plt_cols = 10; plt_rows = 4
gs1 = gridspec.GridSpec(plt_rows, plt_cols)
gs1.update(wspace=0.025, hspace=0.025)
for i in range(plt_rows * plt_cols):
ax1 = plt.subplot(gs1[i])
Beispiel = np.int(np.floor(np.random.uniform(low=0, high=len(v) -1)))
ax1.imshow(v[Beispiel].reshape(28, 28), cmap='binary')
ax1.axis('off')
ax1.set_xticklabels([]); ax1.set_yticklabels([])
ax1.set_xticks(()); ax1.set_yticks(())
ax1.set_aspect('equal')
plt.savefig('Abb71.png', dpi=150)
```

Quellcode B.5: .

Kapitel 7, Abb. 2a) und Abb. 2b)

```
# Initialisierung von Parametern
m = len(y)          # Anzahl der MNIST Beispiele
Lambda = 1e-4       # circa 1/m
d = 784             # Dimension von x
S_2 = 10000         # circa m
S_1 = 1             # innere Schleifen Obergrenze
T = 10              # Anzahl der Iterationsschritte

T_1 = 5
x_0 = np.zeros(d)
alpha = 5

x_1 = Gradientenverfahren(T_1, x_0, alpha, m, Lambda, d)

# waehle S_2 Werte in {5000, 10000, 20000}
xt_1, Fehler_1, Zeit_1 = LiSSA(x_1, T, S_1, S_2=5000, m, Lambda,
    ↪  d)
xt_2, Fehler_2, Zeit_2 = LiSSA(x_1, T, S_1, S_2=10000, m, Lambda
    ↪ , d)
xt_3, Fehler_3, Zeit_3 = LiSSA(x_1, T, S_1, S_2=20000, m, Lambda
    ↪ , d)

# Optimalwert = Fehler nach Anwendung von 500 Iterationen von
    ↪ LiSSA
Optimalwert = np.load('LiSSA_Optimalwert.npy')

# Fehler = log(Fehler) = log(derzeitiger Wert - Optimalwert)
Fehler1 = np.zeros(T); Fehler2 = np.zeros(T); Fehler3 = np.zeros
    ↪ (T)
for t in range(T):
Fehler1[t] = np.log10(Fehler_1[t] - Optimalwert)
Fehler2[t] = np.log10(Fehler_2[t] - Optimalwert)
Fehler3[t] = np.log10(Fehler_3[t] - Optimalwert)
Iterationen = np.arange(0, T, 1)

# Kapitel 7, Abb. 1a)
plt.figure(figsize=(4, 4), dpi=150)
plt.grid(True, color='lightgray', linestyle='-.', zorder=2)
plt.plot(Iterationen, Fehler1, color='darkblue', linestyle='-',
    ↪ label='$S_2 = 5000$', zorder=3)
plt.plot(Iterationen, Fehler2, color='red', linestyle='--',
```

```
    ↪ label='$S_2 = 10000$', zorder=4)
plt.plot(Iterationen, Fehler3, color='green', linestyle=':',
    ↪ label='$S_2 = 20000$', zorder=5)
plt.xlabel('Iterationen')
plt.ylabel('log(Fehler)')
plt.legend(loc='upper right', frameon=True, fancybox=False,
    ↪ edgecolor='black',facecolor='white', framealpha =1.0)
plt.tight_layout()
plt.savefig('Kapitel_7_Abb_1a.png', dpi=150)

# Kapitel 7, Abb. 1b)
plt.figure(figsize=(4, 4), dpi=150)
plt.grid(True, color='lightgray', linestyle='-.', zorder=2)
plt.plot(Zeit_1, Fehler1, color='darkblue', linestyle='-', label
    ↪ ='$S_2 = 5000$', zorder=3)
plt.plot(Zeit_2, Fehler2, color='red', linestyle='--', label='
    ↪ $S_2 = 10000$', zorder=4)
plt.plot(Zeit_3, Fehler3, color='green', linestyle=':', label='
    ↪ $S_2 = 20000$', zorder=5)
plt.xlabel('Zeit (in Sekunden)')
plt.ylabel('log(Fehler)')
plt.legend(loc='upper right', frameon=True, fancybox=False,
    ↪ edgecolor='black',facecolor='white', framealpha =1.0)
plt.tight_layout()
plt.savefig('Kapitel_7_Abb_1b.png', dpi=150)
```

Quellcode B.6: .

Kapitel 7, Abb. 3a) und Abb. 2b)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys
sys.path.append("..")
from Hilfsfunktionen import Gradientenverfahren
import Datensatz_Management
from LiSSA_Algorithmus import LiSSA

# matplotlib Optionen zur Anpassung der Schrift zum verwendeten
    ↪ TeX Stil
plt.rc('font', **{'family': 'serif', 'serif': ['Computer Modern'
    ↪ ], 'size':13})
plt.rc('text', usetex=True)

_, y = Datensatz_Management.Generiere_Datensatz()

```

```
# Parameter Initialisierung
m = len(y)          # Anzahl der MNIST Beispiele
Lambda = 1e-4       # circa 1/m
d = 784             # Dimension von x
T = 10              # Anzahl der Iterationsschritte

T_1 = 5
x_0 = np.zeros(d)
alpha = 5

x_1 = Gradientenverfahren(T_1, x_0, alpha, m, Lambda, d)

# Optimalwert = Fehler nach Anwendung von 500 Iterationen des LiSSA Algorithmus
Optimalwert = np.load('LiSSA_Optimalwert.npy')

# Beruecksichtigte Parameterwerte fuer S_1 und S_2
S_1 = np.arange(0,50+1,1)[0::1*5]
S_1[0] = 1
S_2 = np.arange(0,50000+1,1000)[0::1*5]
S_2[0] = 1000

Iterationen = np.arange(0, T, 1)

container_Fehler_1 = {}
container_Zeit_1 = {}
container_berechneter_Fehler = {}

# Achtung: Laufzeit der unteren Code Schleife betraegt ca. 12 Stunden
for k1 in np.arange(0, len(S_1)):
for k2 in np.arange(0, len(S_2)):
print('Simulation nun bei S_1 = ',S_1[k1],', S_2 = ',S_2[k2])
xt_1, Fehler_1, Zeit_1 = LiSSA(x_1=x_1, T=T, S_1=S_1[k1], S_2= S_2[k2], m=m ,Lambda=Lambda, d=d)
Berechneter_Fehler = np.zeros(T)
for t in range(T):
# Fehler = log(Fehler) = log(derzeitiger Wert - Optimalwert)
Berechneter_Fehler[t] = np.log10(Fehler_1[t] - Optimalwert)
container_berechneter_Fehler.update({ str(S_1[k1])+'_'+str(S_2[ k2]) : Berechneter_Fehler })
container_Fehler_1.update({ str(S_1[k1])+'_'+str(S_2[k2]) : Fehler_1 })
container_Zeit_1.update({ str(S_1[k1])+'_'+str(S_2[k2]) : Zeit_1 })

dt_berechneter_Fehler = pd.DataFrame.from_dict(
```

```
    ↪ container_berechneter_Fehler)
dt_Fehler_1 = pd.DataFrame.from_dict(container_Fehler_1)
dt_Zeit_1 = pd.DataFrame.from_dict(container_Zeit_1)

# speichern der produzierten Ergebnisse
dt_berechneter_Fehler.to_pickle('temp_dt_berechneter_Fehler.pkl'
    ↪ )
dt_Fehler_1.to_pickle('temp_dt_Fehler_1.pkl')
dt_Zeit_1.to_pickle('temp_dt_Zeit_1.pkl')

plt.rc('font', **{'family': 'serif', 'serif': ['Computer Modern'
    ↪ ], 'size':14})
plt.rc('text', usetex=True)

# Kapitel 7, Abb. 2a)
temp = []
for c in dt_berechneter_Fehler.columns:
if dt_berechneter_Fehler.iloc[len(dt_berechneter_Fehler)-1][c] <
    ↪  0:
temp.append(dt_berechneter_Fehler.iloc[len(dt_berechneter_Fehler
    ↪ )-1][c])
else:
if dt_berechneter_Fehler.iloc[len(dt_berechneter_Fehler)-2][c] <
    ↪  0:
temp.append(dt_berechneter_Fehler.iloc[len(dt_berechneter_Fehler
    ↪ )-2][c])
else:
if dt_berechneter_Fehler.iloc[len(dt_berechneter_Fehler)-3][c] <
    ↪  0:
temp.append(dt_berechneter_Fehler.iloc[len(dt_berechneter_Fehler
    ↪ )-3][c])
else:
temp.append(dt_berechneter_Fehler.iloc[len(dt_berechneter_Fehler
    ↪ )-4][c])
temp = np.array(temp)
temp = temp.reshape((len(S_1), len(S_2)))
dt_berechneter_Fehler = pd.DataFrame(temp)
dt_berechneter_Fehler.columns = S_2
dt_berechneter_Fehler.index = S_1
# Einige Eintraege werden zu -Inf oder NaN
# in diesem Fall werden sie durch interpolation der zwei
    ↪ naechsten verfuegbaren Werte gefuellt
data = dt_berechneter_Fehler
data = data.replace([np.inf, -np.inf], np.nan)
data = data.interpolate(method='linear', limit_direction='
    ↪ forward', axis=0)
plt.figure(figsize=(7,7), dpi=150)
```

```
ax = plt.gca()
plt.imshow(data)
for i in range(len(data)):
for j in range(len(data)):
plt.text(j, i, np.int(data.iloc[i, j]), ha='center', va='center'
    ↪ , color='white')
plt.xticks(np.arange(0,len(data))[0::1*1], S_2[0::1*1], rotation
    ↪ ='vertical')
plt.yticks(np.arange(0,len(data))[0::1*1], S_1[0::1*1])
plt.xlabel('$S_2$'); plt.ylabel('$S_1$')
divider = make_axes_locatable(ax)
cax = divider.append_axes('right', size="5%", pad=0.10)
cbar = plt.colorbar(cax=cax)
cbar.set_label('log(Fehler)', rotation=90)
plt.savefig('Abb73a.png', dpi=150)

# Kapitel 7, Abb. 2b)
temp = []
for c in dt_Zeit_1.columns:
if dt_Zeit_1.iloc[len(dt_Zeit_1)-1][c] < 0:
temp.append(dt_Zeit_1.iloc[len(dt_Zeit_1)-1][c])
else:
if dt_Zeit_1.iloc[len(dt_Zeit_1)-2][c] < 0:
temp.append(dt_Zeit_1.iloc[len(dt_Zeit_1)-2][c])
else:
if dt_Zeit_1.iloc[len(dt_Zeit_1)-3][c] < 0:
temp.append(dt_Zeit_1.iloc[len(dt_Zeit_1)-3][c])
else:
temp.append(dt_Zeit_1.iloc[len(dt_Zeit_1)-4][c])
temp = np.array(temp)
temp = temp.reshape((len(S_1), len(S_2)))
dt_Zeit_1 = pd.DataFrame(temp)
dt_Zeit_1.columns = S_2
dt_Zeit_1.index = S_1
data = dt_Zeit_1
plt.figure(figsize=(7,7), dpi=150)
ax = plt.gca()
plt.imshow(data)
for i in range(len(data)):
for j in range(len(data)):
plt.text(j, i, np.int(data.iloc[i, j]), ha='center', va='center'
    ↪ , color='white')
plt.xticks(np.arange(0,len(data))[0::1*1], S_2[0::1*1], rotation
    ↪ ='vertical')
plt.yticks(np.arange(0,len(data))[0::1*1], S_1[0::1*1])
plt.xlabel('$S_2$'); plt.ylabel('$S_1$')
divider = make_axes_locatable(ax)
```

```
135 cax = divider.append_axes('right', size="5%", pad=0.10)
136 cbar = plt.colorbar(cax=cax)
137 cbar.set_label('Zeit (in Sekunden)', rotation=90)
138 plt.savefig('Abb73b.png', dpi=150)
```

Quellcode B.7: .

Eidesstattliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit mit dem Titel *Untersuchung stochastischer Optimierung zweiter Ordnung für maschinelles Lernen* selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommenen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind

Münster, 21. August 2019 _____

(Alexander Betz)

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

Münster, 21. August 2019 _____

(Alexander Betz)