Vergleich von zwei unterschiedlichen Ansätzen eines künstlichen neuronalen Netzwerks und die Relevanz von Optimierungsalgorithmen sowie spezialisierter Hardware

Erstellt von: Reise Kato / Victor Kowalewski / Philip Erdmann

Studiengang: Informatik & Cybersecurity (B.Sc)

Semester: 3

Betreut von: Frau Doerthe Vieten

Datum: 10.01.2024

Verantwortlichkeiten

Philip Erdmann (10%)	**Reise Kato (50%)**	**Victor Kowalewski (40%)**
Projektbericht	Netzwerk R (0)	Netzwerk V (1)
Einleitung	Überarbeitung beider Netze	Überarbeitung beider Netze
Quellen und Zitieren	Erstellung und Weiterverarbeitung von Trainingsdaten	Erstellung von Trainingsdaten
Projektbericht - Literaturarbeit: 2.3, 2.4	Erstellung Topologien	Erstellung Topologien
	Projektorganisation	Projektorganisation
	Auswertung der Rechenzeit beider Netze auf verschiedene Parameter	Auswertung und Plotten der total Error beider Netze
	Auswertung beider Netze auf unbekannten Daten	Auswertung beider Netze auf unterschiedlichen learning rates
	Auswertung beider Netze auf unterschiedlichen Topologien	Projektbericht - Literaturarbeit: 2.5, 2.6
	Auswertung beider Netze auf benötigte Iterationen für bestimmten total Error Wert	Projektbericht - Methoden: 3.3.x

	Projektbericht - Inhaltsverzeichnis	Projektbericht - Ergebnisse: 4.1.x
	Projektbericht - Einleitung (Überarbeitung)	
	Projektbericht - Literaturarbeit: 2.1, 2.2, 2.7, 2.8, 2.9	
	Projektbericht - Methoden: 3.1, 3.2, 3.4, 3.5, 3.6	
	Projektbericht - Ergebnisse: 4.2, 4.3, 4.4	
	Projektbericht - Diskussion: 5.1, 5.2, 5.3	
	Projektbericht Struktur	

Inhaltsverzeichnis

1. Einleitung

Die dritte Phase der Projektarbeit ist zur selbstständigen Implementation eines beliebigen Forschungsthemas vorgesehen. Durch unsere Lage aus der ersten Phase der Projektarbeit bot sich der Vergleich der beiden neuronalen Netzwerke an, weshalb die Entscheidung, auf den Vergleich und die Optimierungsmöglichkeiten von künstlichen neuronalen Netzen fiel.
In 5 Arbeitswochen sollten wir, nachdem wir in der ersten Phase die Grundlagen der neuronalen Netze[1] und in der zweiten Phase die Optimierungsmöglichkeiten erarbeitet haben, unsere Vergleiche der beiden Netze, unter der Voraussetzung der Nutzung der Programmiersprache Java, anfertigen.
Als Vergleichsparameter fielen die Entscheidungen auf die Laufzeit und den durchschnittlichen Total Error. Mithilfe dieser Parameter lassen sich die Genauigkeit der Netze auf unbekannten Daten, so wie die benötigten Trainings-Iterationen für das Erreichen eines Ziel-Errors, als auch die einzelnen Laufzeiten vergleichen.
Mithilfe selbst erstellter Datensätze und Topologie-Konfigurationen sind die genannten Vergleiche unter unterschiedlichen Voraussetzungen möglich.
Mithilfe dieser Vergleiche sollen die Notwendigkeit von Optimierungsalgorithmen und leistungsstarker Hardware, wie Tensor Processing Units, veranschaulicht werden.

2. Literaturarbeit

In diesem Abschnitt werden verschiedene Konzepte der Optimierungsmöglichkeiten vorgestellt, sowie grundlegende Begriffe, wie “total Error”, eingeführt, um das Verständnis der folgenden Arbeit zu vereinfachen.

2.1 Constructing Neural Networks by Extending the Optimization Field [2]

Die Publikation „Constructing Neural Networks by Extending the Optimization Field“ von Wenyuan Zhang beschäftigt sich mit der Optimierung der Lernfähigkeit von sehr großen Netzwerktopologien. Das grundlegende Problem basiert auf der Beeinträchtigung der Performanz eines künstlichen neuronalen Netzwerks bei großen und tiefen Strukturen, wie VGG oder LeNet, die durch neue Hardware, wie TPU (Tensor Processing Unit) und GPU (Graphics Processing Unit), ermöglicht wurden. Mit zunehmender Komplexität steigen auch die Lernparameter solcher Netze, weshalb die Notwendigkeit der Modifizierung und Erweiterung des Netzwerks

um die Ableitung verschiedener Netze, die sich in unterschiedlichen Regionen (Optimierungsfelder), welche sich nicht überschneiden, optimieren sollen, eingeführt wird. Zum Trainieren des Netzes werden diese zusammengefügt, da ein globales Optimierungsfeld immer besser ist, als ein lokales Optimierungsfeld.

2.2 Asynchronous Parallel Stochastic Gradient Descent [3]

Der Artikel „Asynchronous Parallel Stochastic Gradient Descent“, von Janis Keuper und Franz-Josef Pfreudt, publiziert durch das Fraunhofer ITWM, beschäftigt sich mit einer Modifikation des SGDs (Stochastic Gradient Descent). SGD hat sich bereits in großen Topologien als eine gute Methode erwiesen, um akkurate Ergebnisse und Annäherungen zu berechnen. Das Ziel dieser Arbeit ist die Präsentation einer neuen, parallel rechnenden Methode des SGDs für die Optimierung von großen Algorithmen des maschinellen Lernens in Kluster Umgebungen, wie Bildsegmentierungen oder Mustererkennungen. Die asynchrone Kommunikation zwischen den Prozessen ermöglicht ein „lock-free“ Modell mit „work-queues“, wodurch die Prozesse immer belastet sind. Als Vorteile erwiesen sich eine schnellere Konvergenz und stabilere Genauigkeiten nach bereits wenigen Iterationen in einem hoch ausgestatteten Laborumfeld.

2.3 Optimierungs Algorithmen in Modernen künstlichen neuronalen Netzwerken [4]

KNN (künstliche neuronale Netzwerke) sind die gängigste Kategorie im Bereich des Machine Learning. Ein großes Problem der Netze ist es, in kürzester Zeit eine akkurate Ausgabe zu erhalten. Einer der ersten Versuche war es, durch SGD (Stochastic Gradient Descent) durch jede Iteration erneut zu überprüfen, wie genau das Netzwerk ist. SGD ist eine Optimierung erster Ordnung, es gibt Algorithmen in der zweiten Ordnung, die die Krümmung der Zielfunktion mithilfe der Hesse-Matrix berücksichtigen, wie zum Beispiel die Newton-Optimierungsmethode. Aufgrund der Komplexität der Berechnungen werden jedoch quasi-Newton-Optimierungsmethoden verwendet, die die invertierte Hesse-Matrix approximieren.

2.4 Optimierte künstliche neuronale Netzwerke mit Datenanalyse in der Industrie [5]

In diesem Artikel werden vier Qualitätsvorhersagen durchgeführt, wobei versucht

wird, die richtige Anzahl an Layers und Neuronen pro Layer (2 Hyperparameter) zu finden. Es werden vier verschiedene Regressionsmetriken verwendet: Pearson-Korrelationskoeffizient (R), Quadratischer Korrelationskoeffizient (R2), Absoluter relativer Fehler (RAE), Wurzelmittler quadratischer Fehler (RMSE) und Mittlerer absoluter Fehler (MAE). Aufgrund der Herausforderungen bei der manuellen Einstellung von Hyperparametern, wie Lernrate und Aktivierungsfunktion, wird in diesem Versuch Hill Climbing mit Random Restart (zur Vermeidung von Lernplateaus) und einer Tabu-Liste eingesetzt. Die Suche konzentriert sich auf einen kleinen Datensatz und 1-2 Layers mit weniger als 20 Neuronen, wodurch eine effiziente Suche nach dem lokalen und globalen Optimum ermöglicht wird, wobei die Tabu-Liste durch Markieren bereits besuchter Lösungen die Geschwindigkeit erhöht. Im Vergleich zur vorigen Arbeit der gleichen Autoren mit den gleichen Versuchsparametern erweist sich diese Methode, unterstützt durch einen zusätzlichen Algorithmus zur Optimierung von Hyperparametern, als fehlerfreier (um 80% bei MAE, RMSE und %RAE).

2.5 Optimizing connection weights in neural networks using the whale optimization algorithm [6]

Der Artikel "Optimizing connection weights in neural networks using the whale optimization algorithm" behandelt die Anwendung des Whale Optimization Algorithm (WOA) als Trainer für Neuronale Netze mit FeedForward-Architektur. Die Autoren vergleichen den WOA mit sieben anderen Optimierungsalgorithmen, darunter Backpropagation, auf zwanzig verschiedenen Datensätzen. Die Ergebnisse zeigen, dass der WOA in Bezug auf Genauigkeit, Vermeidung lokaler Optima und Konvergenzgeschwindigkeit die anderen Algorithmen, einschließlich Backpropagation, übertreffen kann. Die Autoren betonen die Effizienz und Wettbewerbsfähigkeit des WOA im Vergleich zu anderen Trainings Algorithmen wie dem genetischen Algorithmus (GA).

2.6 Global optimization for neural network training [7]

Der Artikel "Global optimization for neural network training" beschreibt das NOVEL-Minimierungsverfahren für neuronales Netz Training. NOVEL kombiniert globale und lokale Suche, um kleine oder weniger fehleranfällige Netzwerke zu verbessern. Das Ziel ist, gute lokale Minima zu finden, da Gradienten-basierte Methoden in vielen lokalen Minima feststecken können. NOVEL verwendet eine

deterministische Methode, um aus lokalen Minima herauszukommen. Die Methode besteht aus globaler Suche zur Identifizierung von Minima-haltigen Regionen und lokaler Suche zur genauen Minima-Lokalisierung. NOVEL spart Rechenzeit, indem es effiziente Trainingsstartpunkte ermittelt. Im Vergleich mit anderen globalen Optimierungsmethoden war NOVEL beim Training besser oder äquivalent zur TN-MS-Methode, zeigte jedoch teilweise schlechtere Testergebnisse. Die Autoren betonen Fortschritte, sehen aber weiteren Forschungsbedarf für die Anwendung auf nichtlineare Optimierungsprobleme.

2.7 Tabelle - Relevanz der Literaturarbeit im Projekt

Titel	Erscheinungsjahr	Relevanz für das Projekt
Extending the Optimization Field	2020	- als Lösungsansatz für Minimierung der Rechenzeit in Ergebnisse vorgeschlagen
Asynchronous Parallel Stochastic Gradient Descent	2015	- als Lösungsansatz für Minimierung der Rechenzeit in Ergebnisse vorgeschlagen
Survey of Optimization Algorithms in Modern Neural Net	2023	- Grundidee SGD als Lösungsansatz
Optimized neural networks in industrial data analysis	2020	- nicht genügend Kapazitäten für die Umsetzung
Optimizing connection weights in neural networks using the whale optimization algorithm	2016	-Implementation des wale algorithm möglich, hat jedoch keine Relevanz zum Projekt
Global optimization for neural network training	1996	- lokale Minima betrachtbar in Auswertungen, jedoch nicht als Lösungsansatz in Ergebnisse besprochen

2.8 total Error

Hinter dem Begriff “total Error” verbirgt sich eine Dezimalzahl, welche die Abweichung des berechneten Ergebnisses zum erwarteten Ergebnis eines Datensatzes darstellt. Berechnet wird dieser mittels folgender Formel:

$E(total) = \sum_{i=1}^{n} \frac{1}{2}(target - output)^2$ n: Anzahl der Output-Neuronen

Das “total Error” wird für jeden Datensatz neu berechnet. Gibt es zehn verschiedene Outputs im Datensatz, so wird für jedes ein eigener “total Error” berechnet. Das Ziel des Trainierens eines neuronalen Netzwerks liegt darin, das Netz so lange zu trainieren, bis ein gewünschter “total Error” erreicht ist. Gibt es kein bestimmtes Ziel, so ist das Ziel, dieses “total Error” so gering wie möglich zu halten. Eine Frage wie “Was ist ein gutes “total Error”?”, ist somit situationsabhängig und kann nicht generell beantwortet werden.

2.9 Laufzeit

Die Laufzeit eines Programms wird durch die benötigte Zeit beschrieben, die ein Programm braucht, um alle Anweisungen abzulaufen. Dabei ist die Komplexität eines Programms für die Laufzeit ausschlaggebend. Mit erhöhter Komplexität, wie einer hohen Anzahl an verketteten Schleifen, steigt die Laufzeit eines Programms.

2.10 aussagenlogische Formeln

Zur Vereinfachung dieses komplexen Themas wird in diesem Teil nur die Grundlage eingeführt, die für den weiteren Verlauf ausreichend ist.
Eine aussagenlogische Formel kann aus verschiedenen Junktoren bestehen. Hier wird sich nur auf “und” und “oder” begrenzt. Mittels einer Wahrheitstabelle oder einer Min-Max-Funktion lassen sich die Wahrheitswerte einer Formel berechnen. So kann für folgende Formel: a oder b mit a = 0 und b = 0 der Wahrheitswert nicht 1 sein.
Zur Erstellung von eigenen Datensätzen wurden von aussagenlogischen Formeln Gebrauch gemacht.

3. Methoden

Beide neuronalen Netze wurden für dieses Projekt in einen Projektordner zusammengefügt, um den Zugriff zu vereinfachen und zu strukturieren. Für die Methoden der Auswertungen wurde eine Main-Klasse verwendet, worin alle Methoden vorzufinden sind. Nach dem Sortieren der Datensätze und Überprüfung auf Vollständigkeit wurden für die Auswertungen aus Abschnitt 3.4 bis 3.6 über alle Datensätze mit zwei verschiedenen Topologie-Konfigurationen iteriert, wobei

Datensätze mit mehr als 2000 verschiedenen Möglichkeiten nur zur Auswertung der Laufzeit behandelt wurden, um “java.lang.OutOfMemoryError” vorzubeugen. Zur Laufzeitbestimmung wurde aus demselben Grund für große Datensätze und wegen der Proportionalität der Iterationen gegenüber der Laufzeit eine Iterationsanzahl von 100 gewählt. Die Auswertung der Laufzeiten geschieht über den Mittelwert von 20 Läufen. Alle Ergebnisse aus 3.4 bis 3.6 sind unter dem Ordner “results” gespeichert. Zudem arbeiten beide Netze mit Zufälligen Startwerten für Gewichts -und Neigungswerte (zwischen -1, 1). Des Weiteren wurden konstante Lernraten in den Auswertungen verwendet.

3.1 Erstellung der Trainings Datensätze

Im Rahmen des Projektes sind unterschiedliche Datensätze erforderlich. Hierfür wurden aussagenlogische Formeln verwendet. Die genaue Erstellung eines Datensatzes wird im Folgenden anhand eines Beispiels erläutert:

```
R1tData.csv:
p,q,r,s,t,w
(p or q) and r and (s or t and w)
-->     (p or q) = P            (expected column 1)
        r = Q                   (expected column 2)
        (s or t and w) = R      (expected column 3)
```

In diesem Beispieldatensatz erhält das Netz sechs Input Werte (p, q, r, s, t, w) und drei Output Werte ((P = p or q), (Q = r), (R = s or t and w)). Mithilfe einer Excel-Tabelle wurden die Datensätze als csv-Datei gespeichert.

Alle weiteren Datensätze wurden nach demselben Verfahren mit unterschiedlichen logischen Formeln erstellt. Um die Netze auf möglichst unterschiedlichen Datensätzen vergleichen zu können, wurden hier zwischen drei Input Werten und elf Input Werten variiert. Arten von Trainings Datensätzen sind folgende:

- **noise tData**: zufällige Abweichung von 0 und 1 zwischen 1E-5 und 0.1 (erstellt mittels Methode, welche clean tData einliest und zufällig verändert)
- **clean tData**: nur 0 und 1
- **whole tData**: vereinigte Menge von “clean” und “noise tData”
- **partial/known tData**: Teilmenge aus “whole tData”
- **unknown tData**: “whole tData” ohne “partial tData“

3.2 Erstellung der Topologien

Beide Netze benötigen eine Layer-Konfiguration, um die Daten erfolgreich aufzunehmen und verarbeiten zu können. Auf Grundlage der Flexibilität von

neuronalen Netzen wurden die Topologie-Konfigurationen frei gewählt. Nur die Anzahl der Input- und Output-Werte sind von dem zugehörigen Datensatz festgelegt.

```
layers;6;7;5;3
```
Konfiguration zum obigen Datensatz mit insgesamt 4 Layern

3.3.1 Auswertung Total Error

Zum Vergleich des Total Errors wurden die Trainingsdaten V1 - V9 verwendet. In der Methode Main.computeTotalError() wurden beide Netze auf den Trainingsdaten initialisiert und trainiert. Die Anzahl der Iterationen betrug 100000, als Lernrate wurde die Konstante 0.05f gewählt.

Für jeden Datensatz wurde der Total Error je Iteration über 100.000 Iterationen berechnet und in einer csv-Datei gespeichert. Diese Dateien wurden im Anschluss innerhalb des Verzeichnis sortiert gespeichert, so dass sie mit der “csv” Bibliothek eingelesen werden konnten.

Je Datei ergaben sich zwei Listen für beide Netze, bei denen jeder Index eine Liste mit Fehlern enthält, welche einen schnellen Zugriff auf den Inhalt und eine einfache grafische Darstellung der Werte ermöglichten.

Der durchschnittliche Fehler pro Iteration wurde durch List slicing und Division durch die Summe berechnet. Aus den Berechnungen ergab sich für jedes Netz eine 2D-Liste, mit dem jeweiligen durchschnittlichen Total Error für die Trainingsdaten, die anschließend als Graph dargestellt wurden.

3.3.2 Auswertung Total Error - mit verschieden Lernraten

Bei der dieser Auswertung wurden folgende Lernraten angewendet:

float[] LRs = {2.5f, 1.5f, 1.0f, 0.1f, 0.001f, 0.00001f};

Die folgende Auswertung basiert auf der Layer Konfiguration “V9_layerConfig.csv” und den Trainingsdaten aus der Datei “V9_tdata_known.csv”.

In der Methode Main.learnrates() wurden für jede Lernrate beide Netze mit 10000 Iterationen trainiert und der Total Error anschließend in eine CSV-Datei geschrieben. Die Auswertung erfolgte, dann sowie in 3.3.1 bereits erläutert.

Da bei der Lernrate 0.00001f, nicht erkennbar war, an welchen Wert sich die Netze annähern, wurde die Anzahl der Iteration für speziell diese Lernrate auf 100000 gesetzt.

3.3.3 Auswertung Total Error - Performance mit unbekannten Daten

Die Netze wurden mit der Layer Konfiguration “V2_layerConfig.csv” initialisiert und mit den Trainingsdaten von “V2_tData_known.csv”, den Unterschiedlichen LRs siehe 3.3.2 und mit 10000 Iterationen trainiert.

Im Anschluss durchliefen beide Netze für jede Lernrate jeweils einmal die Trainingsdaten aus der Datei “V2_tData_unknown.csv”, die Ergebnisse wurden für jeden Datensatz bzw. Für jede LR (Lernrate) in eine CSV-Datei geschrieben.

Damit die Daten dokumentiert werden konnten, wurden sie in einem Python-Skript “UnknownLR.py” eingelesen. Es wurde schließlich für jedes Netz pro LR eine Abbildung erstellt, in der sich die vorhergesagten Werte und die tatsächlichen Werte von den unbekannten Trainingsdaten befinden.

3.4 Auswertung der Laufzeit

Die Laufzeiten beider Netze wurden auf vier Aspekte geprüft: Dauer der Initialisierung, Einlesen der Daten, Trainingsdauer und die gesamte Laufzeit. Zur Ermittlung der Trainingsdauer wurden die Netze zu jedem Datensatz genau 100 Iterationen mit einer learning rate von 0.05 trainiert. Um Abweichungen festzustellen, wurde die Ermittlung der Laufzeit zu jedem Datensatz 20 mal wiederholt. Die Laufzeit wurde mittels System.nanoTime() ermittelt, wobei die Startzeit von der Endzeit abgezogen wurde. Ist die Summe der drei Unterkategorien nicht gleich der gesamten Laufzeit, so wird in allen Kategorien der maximale long-Wert als Laufzeit eingetragen.

Die Ergebnisse sind im Unterordner “runtime” des Ordners “results” gespeichert.

3.5 Auswertung auf unbekannten Datensätzen

Zur Auswertung des gesamten durchschnittlichen total Errors auf unbekannten Datensätzen wurden die Netze erst nur mit den “partial tData” mit 100.000 Trainings-Iterationen und einer learning rate von 0.05 trainiert. Die “unknown tData” wurden zur Auswertung daraufhin dem Netz übergeben. Das daraus berechnete durchschnittliche total Error ist das Ergebnis der Genauigkeit der Netze nach 100.000 Trainings-Iterationen, falls dem Netz nach dem Trainieren nur unbekannte Daten übergeben werden. Als Referenzwert wurde dieser Schritt mit dem gesamten Datensatz wiederholt, wobei die unbekannten Daten bereits beim Trainieren des Netzes verwendet wurden und somit als bekannt gelten.

Die Ergebnisse sind im Unterordner “behaving_on_unknown_data” des Ordners “results” gespeichert.

3.6 Auswertung der benötigten Iterationen für ein Ziel-total-Error

Zur Überprüfung, wie viele Iterationen die jeweiligen Netze benötigen, um ein gewisses Ziel-total-Error zu erreichen, wurden die Netze solange trainiert, bis das durchschnittliche total Error des jeweiligen Netzes kleiner als der Zielwert ist. Während dieses Prozesses bleibt die learning rate bei genau 0.05. Als Abbruchkriterium wurde eine Grenze von 100.000 Iterationen entschieden. Dadurch wird eine Anzahl von 999.999 Iterationen ausgegeben, falls dieses Kriterium erreicht wird. Als Ergebnis wird die Anzahl der benötigten Trainings-Iterationen zurückgegeben.
Die Ergebnisse sind im Unterordner “target_total_error” des Ordners “results” gespeichert.

3.7 Hardware und Software

CPU: Intel Core i5 12600K
RAM: 32 GB DDR4 3200MHz, davon 16GB für JVM
JDK: 21

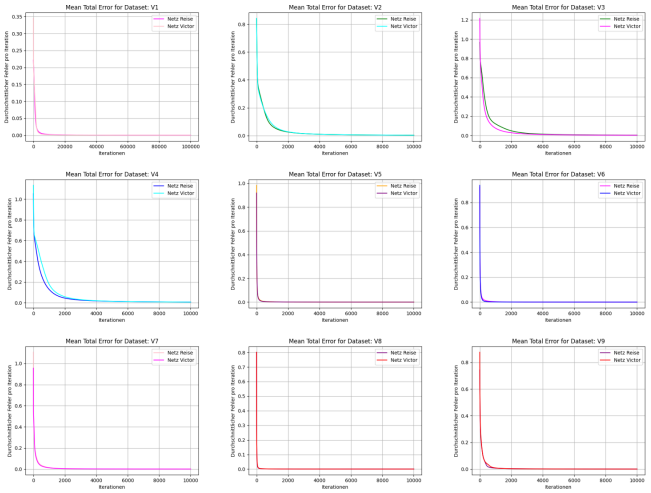
4. Ergebnisse

In diesem Abschnitt werden die Ergebnisse der eingeführten Methoden näher erläutert.

4.1.1 Ergebnisse Total Error

Abbildung 1 zeigt neun Graphen, die X-Achse entspricht der Anzahl an Iterationen und die Y-Achse entspricht dem durchschnittlichen Total Error. Insgesamt zeigt sich, dass sich der durchschnittliche Gesamtfehler von beiden Netzen null annähert. Am Anfang sind Größenunterschiede zu erkennen, z. B. bei V3, da die kNNs zufällige Startkonfigurationen für die Gewichte und Neigungen haben. Dies hat zur Folge, dass manche Startkonfigurationen weniger oder mehr Berechnungen brauchen, um das globale Minimum zu erreichen, als andere. Ab 4000 Iterationen nähert sich der durchschnittliche Total Error sich wieder an und die Graphen verlaufen wieder parallel, das lässt sich auf die gleiche Berechnung zurückschließen.

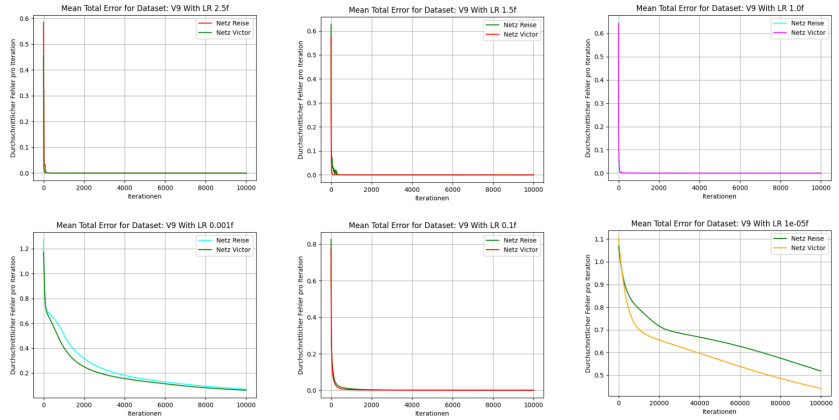
Abb.1:



Quelle: eigene Darstellung

4.1.2 Ergebnisse Total Error mit versch. LRs

Abb.2:



Quelle: eigene Darstellung

Abbildung 2 visualisiert die Minimierung des Total Errors anhand von verschiedenen Lernraten. Die X-Achse zeigt die Anzahl an Iterationen und die Y-Achse beschreibt den durchschnittlichen Total Error.

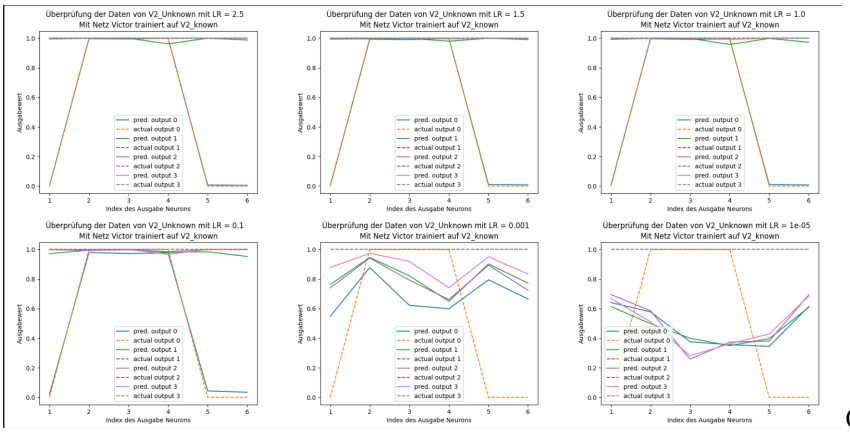
Aus der Abbildung geht hervor, dass bei einer sehr kleinen LR, beispielsweise 10^-5, mehr Iterationen benötigt werden, um den Fehler zu minimieren. Dies sieht man vor allem daran, dass sich der Fehler bei einer LR von 2.5 schon ab 2.000 Iterationen nicht mehr verändert. Bei der LR 10^-5 hat der durchschnittliche Fehler von beiden Netzen nicht annähernd 0.2 erreicht, genauso bei Erhöhung der Iterationen auf 100.000. Da die LR angibt, wie viel Gewichtsänderung während des Trainings erfolgt, resultiert daraus ein langsamer Fortschritt bei der Fehler Minimierung, wenn die LR kleiner ist als 0.001f.

Grundsätzlich lässt sich feststellen, dass beide Netze den Fehler auf ein gleiches Niveau reduzieren und es sich anhand dieser dargestellten LR, keine LR identifizieren lässt, bei der sich das Verhalten der durchschnittlichen Fehler stark unterscheidet. Durch Abbildung 2 wird deutlich, dass es sich empfiehlt eine LR zwischen 2.5f und 0.1f zu wählen um bei den ausgewählten Trainingsdaten schnelle Fehler Minimierung zu erreichen.

4.1.3 Ergebnisse Total Error mit unknown data

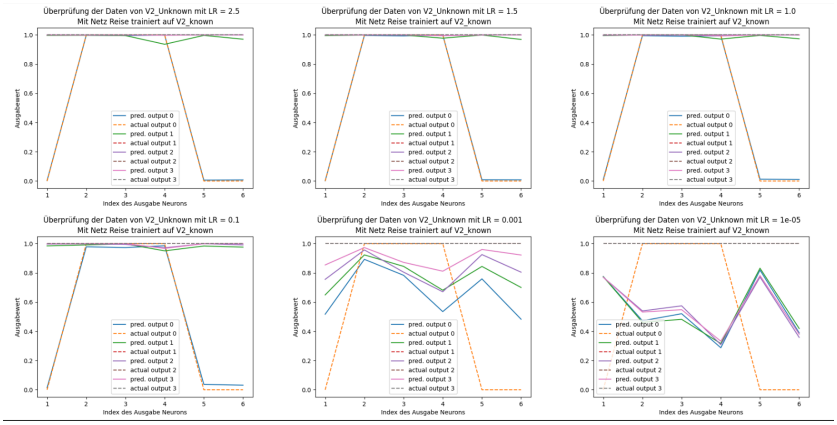
Die beiden Grafiken (Abb. 3 und 4) zeigen jeweils 6 Plots zu den LR aus 3.3.2, da die Netzwerke die Layer Konfiguration “V2_layerConfig.csv” verwenden und deswegen sechs Output Neuronen haben. Die X-Achse zeigt die jeweiligen Output Neuronen des Netzes. Die Ergebniswerte der jeweiligen Neuronen werden auf der Y-Achse dargestellt. In jedem Plot gibt es jeweils für einen Datensatz aus der “V2_tData_unknown.csv” zwei Geraden, einmal mit den vom Netz vorhergesagten Werten und einmal von den tatsächlichen Werten (gestrichelt). Wenn man die beiden Abbildungen 3 und 4 vergleicht, so fällt auf, dass die Netze ähnlich abschneiden, dies sieht man zum Beispiel für die LR >= 0.1f, dort sind die Berechnungen der Netze am präzisesten und der “actual output…” stimmt am meisten mit dem “predicted output…” überein. Bei einer LR < 0.1f, lässt sich feststellen, dass die vorhergesagten Werte der einzelnen Neuronen, noch stark von den tatsächlichen Werten abweichen, dies könnte auf die Anzahl an Iterationen zurückgeführt werden, da die Netze durch kleine Lernraten den Fehler langsamer minimieren.

Abb.3:



Quelle: eigene Darstellung

Abb.4:



Quelle: eigene Darstellung

4.2 Ergebnisse der Laufzeiten

Die Auswertungen der Laufzeiten werden beispielhaft anhand von zwei Datensätzen je unterschiedlicher Topologien verglichen.

layer: Initialisierung des Netzes; data: Lesen der Daten; train: Training

runtime: gesamte Laufzeit

1. Training Datensatz V3

```
Topologien:     3;6;7                                   3;6;15;24;17;9;7
                layer;data;train;runtime                layer;data;train;runtime
OOP             96164;1970385;973455;3040005            118694;1390094;6154755;7663544  (Nanosekunden)
Array           17315;168545;2831509;3017370            46784;213164;19578714;19838664  (Nanosekunden)

OOP/Array       5.554;11.691;0.344;1.008                2.537;6.521;0.314;0.386
(wenn x < 1, dann Array langsamer um x, sonst Array schneller um x)
```

2. Training Datensatz R5

```
Topologien:     8;10;6;4;2                              8;15;20;25;17;10;4;2
                layer;data;train;runtime                layer;data;train;runtime
OOP:            97260;317069695;33321260;350488215      167195;318321600;237957165;556445960    (Nanosekunden)
Array:          27125;518625;125978755;126524505        44040;529185;666981230;667554455        (Nanosekunden)

OOP/Array:      3.586;611.366;0.264;2.77                3.796;601.532;0.357;0.834
(wenn x < 1, dann Array langsamer um x, sonst Array schneller um x)
```

Anhand dieser zwei unterschiedlichen Datensätze ist klar zu erkennen, dass die Nutzung von Objekten im Vergleich zu Arrays in Hinsicht der Laufzeit eine große Auswirkung hat. So ist das Netz, welches für jedes Neuron und Layer ein eigenes Objekt benötigt, deutlich langsamer bei der Initialisierung als ein Netz basierend auf Arrays. Das Einlesen der Daten dauert ebenfalls deutlich länger, da jedes Objekt einzeln gerufen und die Daten neu überarbeitet und übergeben werden müssen. Die Nutzung des Datentyps “float” zeigt sich bei der Berechnung des Netzes stark. So ist die Trainingszeit im Schnitt fast dreimal so schnell wie ein Netz basierend auf double Werten. Sind die Topologien klein, so muss das neuronale Netz nicht besonders viel

rechnen, weshalb das OOP-Netz in solchen Fällen insgesamt schlechter abschneidet. Werden größere Topologien verwendet, so ist die Rechengeschwindigkeit von “float”-Werten deutlich schneller als die der “double”-Werte. Faktisch schnell genug, um die langsame Initialisierungszeit, welche ebenfalls mit einer wachsenden Topologie steigt, zurückzugewinnen. Beide Netze wurden mit je 100 Trainings-Iterationen trainiert. Mit höherer Trainings-Iteration vergrößert sich, wie im folgenden Beispiel zu sehen, auch der Unterschied zwischen den beiden Netzen (training Datensatz R1):

```
Topologien:     6;7;5;3                                 6;7;5;3
Iterationen:    100                                     100.000
                train;runtime                           train;runtime
OOP:            5463680;27845870                        5324385810;5353005980   (Nanosekunden)
Array:          22401785;22707185                       27885499740;27885911175 (Nanosekunden)

OOP/Array:      0.244;1.226                             0.191;0.0.192
(wenn x < 1, dann Array langsamer um x, sonst Array schneller um x)
```

Interessant sind für diesen Vergleich nur die Trainings- und die gesamten Laufzeiten, da die Initialisierung und das Einlesen der Daten nicht von der Iterationsanzahl abhängig sind. Während das Array-Netz bei 100 Iterationen 1.2 mal so schnell ist, ist das OOP-Netz bei 100.000 Iterationen knapp über fünfmal so schnell wie das Array-Netz. Der Grund hierfür ist, dass die Trainingszeit mehr als 99% der gesamten Laufzeit einnimmt und somit der einzige entscheidende Faktor ist.
Die Laufzeit wurde unter allen Arten von Trainings Datensätzen durchgeführt, ein relevanter Unterschied zwischen ihnen ließ sich jedoch nicht ableiten, weshalb hier stets die “whole tData” verglichen werden.

4.3 Ergebnisse auf unbekannten Daten

Auf Grundlage eines Datensatzes lässt sich eine Gemeinsamkeit gut ablesen: Ist die Netzwerktopologie unangemessen groß, so wird das “total Error” bei unbekannten Datensätzen größer. Dieses Phänomen sticht vor allem bei dem OOP-Netzwerk stark hervor. Hier könnte die Nutzung von “float”-Werten der Grund sein. Je mehr Layer und Neuronen es gibt, desto höher ist die Wahrscheinlichkeit einer größeren Auswirkung von “floating-point-error”. Beide Netzwerke schneiden durchschnittlich im gesamten Testverlauf bei allen Trainings Datensätzen mit einem “total Error” von unter 0.1 ab. Wird der gesamte Datensatz zum Trainieren genutzt, sind die durchschnittlichen “total Error” beider Netze bei den “unbekannten Daten”, welche nun bekannt sind, in allen Fällen sehr ähnlich.

```
layers: [3, 6, 7] partial_dataset
Training Iteration: 100000
Learning Rate: 0.05

OOP:    Mean of total Error: 3.8343843592230774E-4
Array:  Mean of total Error: 0.00135988014554011
```
Datensatz V3

```
layers: [3, 6, 15, 24, 17, 9, 7] partial_dataset_layer_alternative
Training Iteration: 100000
Learning Rate: 0.05

OOP:    Mean of total Error: 0.07347006241910106
Array:  Mean of total Error: 0.002787177692838134
```
V3

```
layers: [3, 6, 7] whole_dataset
Training Iteration: 100000
Learning Rate: 0.05

OOP:    Mean of total Error: 1.1471071429353207E-4
Array:  Mean of total Error: 1.3987903722647465E-4
```
V3

Ausnahmen, wie ein “total Error” von über 0.1 im Array-Netz im Vergleich zu deutlich unter diesem Wert im OOP-Netz, weisen jedoch auf Fehler oder Inkonsistenz im Array-Netz selbst hin.

```
layers: [6, 7, 5, 3] partial_dataset
Training Iteration: 100000
Learning Rate: 0.05

OOP:    Mean of total Error: 6.305868298188402E-6
Array:  Mean of total Error: 0.1571462649720795
```
Datensatz R1

4.4 Ergebnisse zu benötigten Iterationen für ein Ziel-total-Error

Zwischen allen Ergebnissen ist ein Merkmal deutlich aufgefallen. Mit erhöhter Anzahl an Layer und Neuronen ist die nötige Anzahl an Iterationen für das Erreichen des Ziel-total-Errors gesunken. Während das Array Netzwerk mit manchen Topologie Konfigurationen über 100.000 Trainings Iterationen nötig hatte, konnte es mit einer anderen Topologie das Ziel in unter 100 Iterationen erreichen. Dies könnte sowohl ein Fehler im Netz sein, als auch an der Topologie liegen. In den meisten Fällen, wie mit dem Trainingsdatensatz V5 zu sehen, benötigt das OOP-Netz jedoch weniger Iterationen als das Array-Netz, um das Ziel eines “total Errors” von unter 0.05 zu erreichen.

```
Training Data: V5_tData.csv
LayerConfig: [9, 9, 10]
Learning Rate: 0.05
Target Total Error: 0.05

OOP total Error, Iteration and Time needed: 0.04933793871459171  -  93.0  -  3.8650224299E10 (time)
Array total Error, Iteration and Time needed: 0.04994482552846286  -  182.0  -  1.910311E8 (time)
Comparison; OOP MINUS Array: -6.06886813871152E-4  -  -89.0  -  3.8459193199E10 (time)
```

```
Training Data: V5_tData.csv
LayerConfig: [9, 15, 20, 17, 10]
Learning Rate: 0.05
Target Total Error: 0.05

OOP total Error, Iteration and Time needed: 0.04921164849336841  -  77.0  -  2.7092352001E10 (time)
Array total Error, Iteration and Time needed: 0.01931037059418554  -  212.0  -  1.1614973E9 (time)
Comparison; OOP MINUS Array: 0.02990127789918287  -  -135.0  -  2.5930854701E10 (time)
```

Es gibt aber auch Ausnahmen. Wie durch Trainingsdatensatz R2 veranschaulicht, kann das Array-Netz mit weniger Iterationen (5 fach) als das OOP-Netz unter einer bestimmten Topologie deutlich schneller das Ziel erreichen, wobei es in einer anderen Topologie deutlich langsamer (10 fach) ist.

```
Training Data: R2_tData.csv
LayerConfig: [9, 12, 15, 10, 5, 4]
Learning Rate: 0.05
Target Total Error: 0.05

OOP total Error, Iteration and Time needed: 0.04933601280922854  -  141.0  -  5.29911004E10 (time)
Array total Error, Iteration and Time needed: 0.0498098125668482  -  1690.0  -  8.419667E9 (time)
Comparison; OOP MINUS Array: -4.7379975761966536E-4  -  -1549.0  -  4.45714334E10 (time)
```

```
Training Data: R2_tData.csv
LayerConfig: [9, 14, 20, 15, 10, 7, 4]
Learning Rate: 0.05
Target Total Error: 0.05

OOP total Error, Iteration and Time needed: 0.04987417388815973  -  227.0  -  5.4601237E10 (time)
Array total Error, Iteration and Time needed: 0.04826591371302965  -  39.0  -  2.045887E8 (time)
Comparison; OOP MINUS Array: 0.0016082601751300774  -  188.0  -  5.43966483E10 (time)
```

5. Diskussionen und Fazit

Dieser Abschnitt erläutert die Probleme, auf die wir während unseres Projektes gestoßen sind, und gibt Lösungsansätze zu diesen an.

5.1 Hardware-Limitierungen

Durch die hohe Anzahl an möglichen Inputs mancher Trainings Datensätze, waren 16GB RAM, welche dem JVM zur Verfügung gestellt wurden, nicht ausreichend. Da Java den Benutzer keine endgültige Entscheidung über seinen RAM Speicher dem Benutzer zulässt, ist die Möglichkeit einer RAM Optimierung stark limitiert. Auch die Rechenzeiten solcher großen Datensätze unter größeren Topologien erfordern viel Geduld. So wurde Datensatz R3 mit folgender Topologie: 11;12;10;7;4 über die Nacht laufen gelassen, um das Netz mit 100.000 Iterationen zu trainieren. Es stellte sich heraus, dass noch kein Ergebnis zur Verfügung steht. R1 hingegen mit 6;7;5;3 ließ sich nach etwa fünf Sekunden mit 100.000 Iterationen trainieren. Mit steigender Neuronen und Layeranzahl wächst auch die Rechenzeit. Somit lässt sich zeigen,

dass Optimierungsverfahren, wie die oben vorgestellten, von hoher Relevanz sein können, um die Rechenzeit zu minimieren. Eine Verlagerung der Programme auf eine stärkere Hardware, wie eine Grafikkarte, würde dieses Problem für das Erste ebenfalls umgehen, doch auch diese Lösung ist nur befriedigend, solange die Netzarchitekturen nicht noch komplexer werden. Die Parallelisierung der Prozesse wäre ebenfalls ein Ansatz zur Optimierung der Laufzeit eines neuronalen Netzes, welches anfangs über das Multi-Threading erprobt werden kann. Dazu wird die Thread-Klasse aus der Java Standard Bibliothek benötigt. Um dem Speicherproblem entgegenzuwirken, bietet sich eine Ableitung des Netzes in Subnetze an, welche in ihren lokalen Optimierung Feldern trainiert werden, um am Ende zusammengefügt zu werden.

5.2 Inkonsistenz

Bei der Auswertung der Daten ist eine Inkonsistenz der total Error Werte im Array-Netz aufgefallen, welche manche Daten unbrauchbar machen. In manchen Fällen funktioniert sie einwandfrei, während sie am nächsten Tag wieder über 100.000 Iterationen benötigt. So ist die Diskrepanz der benötigten Iterationen zwischen diesen beiden Netzen, beispielsweise in V5_required_iteration_whole_dataset.csv und layer_alternative, sehr groß.
In anderen Fällen trainiert das Array-Netz nicht sehr gut, sodass das “total Error” bei beispielsweise 0.426 (R1_Unknown_NNV_whole_dataset.csv) feststeckt und als minimalen Wert verwechselt wird. Somit ist die Korrektheit und Konsistenzprüfung der Vergleiche auf die Genauigkeit nicht gestattet. Während manche Datensätze mit beiden Netzen harmonieren, lösen andere Verwirrung aus. Solche Daten wurden jedoch bei der Auswertung der Ergebnisse nicht berücksichtigt.

5.3 Fazit

Als Fazit lässt sich ziehen, dass die Verwendung von float Werten zu einer erheblichen Leistungsförderung beiträgt. Die Verwendung von OOP für alle möglichen Parameter jedoch stellte sich, wie erwartet, nicht als effizienteste Lösung, um ein neuronales Netz anzulegen. Insgesamt scheint der kürzere 32-bit floating point precision keine Einflüsse auf die Genauigkeiten unserer Netze zu haben, weshalb die Verwendung dessen nur zu einer schnelleren Laufzeit führt

verwendete Literatur (letzter Zugriff: 9.1.2024):

[1] R.Kato, V.Kowalewski, P.Erdmann, “Projektarbeit Seminar zum “Lernen in natürlichen und künstlichen neuronalen Netzen” (KNNs) an der H-BRS zum 26.10.23”, Hochschule Bonn-Rhein-Sieg, DE, Informatik & Cybersecurity, 26.10.2023

[2] Wenyuan Zhang. 2020. Constructing Neural Networks by Extending the Optimization Field. In Proceedings of the 2020 European Symposium on Software Engineering (ESSE '20). Association for Computing Machinery, New York, NY, USA, 175–180. https://doi.org/10.1145/3393822.3432313

[3] Janis Keuper and Franz-Josef Pfreundt. 2015. Asynchronous parallel stochastic gradient descent: a numeric core for scalable distributed machine learning algorithms. In Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments (MLHPC '15). Association for Computing Machinery, New York, NY, USA, Article 1, 1–11. https://doi.org/10.1145/2834892.2834893

[4] Survey of Optimization Algorithms in Modern Neural Net – By Ruslan Abdulkadirov, Pavel Lyakhov, Nikolay Nagornov – 2023 - https://www.mdpi.com/2227-7390/11/11/2466

[5] Optimized neural networks in industrial data analysis – By Liesle Caballero, Mario Jojoa, Winston S. Percybrooks – 2020 -https://link.springer.com/article/10.1007/s42452-020-2060-5#Sec2

[6] I. Aljarah, H. Faris, and S. Mirjalili, “Optimizing connection weights in neural networks using the whale optimization algorithm,” *Soft Computing*, vol. 22, no. 1, pp. 1–15, Nov. 2016, doi: https://doi.org/10.1007/s00500-016-2442-1.

[7] Yi Shang and B. W. Wah, "Global optimization for neural network training," in Computer, vol. 29, no. 3, pp. 45-54, March 1996, doi: 10.1109/2.485892

Unsere Gruppen Dokumentation:

GitHub: https://github.com/ReiseKato/HBRS-neural-network

Kanban Board:
https://docs.google.com/spreadsheets/d/12npSSoFtenZHdrKOWXbEh6nSvsBu1qaA0eKvqemcT7g/edit?usp=sharing

Projektplan

Aufgabe	**Bearbeitungszeitraum**	**Verantwortung**
Projektplan & Pflichtenheft verfassen	30.11.23 – 6.12.23	Reise
Optimierungsmöglichkeiten in bisherigen Netzen ausarbeiten	30.11.23 – 6.12.23	Reise, Victor, Philip
Constructor beider Netze	5.12.23	Reise, Victor
Erstellen von Datensätzen	30.11.23 – 7.12.23	Reise, Victor, Philip
Zusammenfassen in ein Projektordner	6.12.23	Reise, Victor
Auswertung Rechenzeit beider kNN	8.12.23 – 20.12.23	Reise
Auswertung Ergebnisse auf unbekannten Daten (Vergleich)	8.12.23 – 20.12.23	Reise
Auswertung auf unterschiedlichen Topologien	9.12.23 – 25.12.23	Reise
Trainingszeit	8.12.23 – 20.12.23	Reise
Plotten der totalError Daten	8.12.23 – 20.12.23	Victor
Verhalten unter unterschiedlichen learning rates	8.12.23 – 20.12.23	Victor
Genauigkeit der einzelnen Ergebnisse	8.12.23 – 20.12.23	Reise

Projektbericht	21.12.23 – 09.01.24	Reise, Victor, Philip

Nach Absprache mit Frau D. Vieten wurde festgelegt, dass Philip keine Implementationen durchführt. Da diese Absprache erst nach dem Verfassen des Pflichtenheftes stattgefunden hat, ist das Pflichtenheft nicht aktuell.

Pflichtenheft

1. Einleitung

Das vorliegende Pflichtenheft enthält die an das Projekt gestellten funktionalen sowie nicht-funktionalen Anforderungen. Im Rahmen der 3. Phase des Projekt-Seminars „Lernen in natürlichen und künstlichen neuronalen Netzen“, betreut von Frau D. Vieten, beschäftigt sich das Team von Reise, Victor und Philip mit dem Vergleich von zwei unterschiedlich aufgebauten neuronalen Netzwerken. Die spezifischen Anforderungen sowie die Aufgabenstellung wurden von dem Team eigenständig erarbeitet. Am Ende dieses Projekts soll der Unterschied zwischen den beiden Netzwerken in unterschiedlichen Bereichen festgestellt werden.

2. Auftrag

Unsere besondere Situation aus der 1. Projektphase, die Implementation von zwei unterschiedlichen Netzwerken in Java, ermöglicht uns den Vergleich dieser beiden Netze in Hinsicht auf die Komplexität und Rechenzeit des Trainingsalgorithmus und der Netze, die Anzahl und Sinnhaftigkeit von Speicherzuweisungen, das Verhalten der Netze unter unbekannten Daten, unterschiedlichen Topologien und verschiedenen Learning Rates sowie die Genauigkeit der Ergebnisse. Damit beide Netze auf die oben genannten Merkmale untersucht werden können, werden die Netze auf verzichtbare Schnittstellen untersucht und auf einen neuen Standard gehoben, worauf die Vergleiche basieren. Optional ist die Optimierung der Netze nach einem beliebigen Optimierungsverfahren. Für die Vergleiche der Rechenzeit werden beide Netze unter den gleichen Hardware- und Softwarebedingungen laufen, um eine externe Beeinflussung zu eliminieren.

3. Bereits bestehende Systeme

Beide Netze stammen bereits aus vorigen Arbeiten innerhalb des Projekt-Seminars, wodurch uns die Nutzung und Erweiterung dieser im vollen Umfang zusteht.

4. Teams und Schnittstellen

Das Team besteht aus Victor Kowalewski, Philip Erdmann und Reise Kato. Unsere Kommunikation läuft überwiegend über Präsenztreffen und Nachrichtenkanäle unterschiedlicher Anbieter. Zur Organisation unserer Aufgabenstellungen dient ein Google Docs Dokument, worauf jedes Teammitglied zugreifen und bearbeiten kann. Die tatsächlichen Programme sind in einem gemeinsamen GitHub Repository hinterlegt.

5. Rahmenbedingungen

Als Bearbeitungszeitraum sind drei Wochen vorgesehen, wobei jeder 4 bis 6 Stunden pro Woche am Projekt arbeitet. Die gesamte Bearbeitungszeit liegt somit bei 12 bis 18 Stunden pro Person.

6. Technische Anforderungen

Dieses Projekt basiert auf einer stabilen Hardware und Softwareausstattung, so dass unsere Vergleiche stets konstant sind. Voraussetzungen für das Ausführen des Programms sind nicht hoch, so dass jeder moderne Rechner mit einem geeigneten Betriebssystem beide Netze mit Erfolg berechnen kann. Zumal beide Netze mit der Programmiersprache Java verfasst sind, wird eine entsprechende Java Version vorausgesetzt. Als Software dient eine beliebige IDE, welche Java Programme kompilieren und ausführen kann. Für die Plots wird die Programmiersprache Python und eine passende DIE benötigt.

7. Problemanalyse

Die größte Hürde wird die Speicherkapazität sein, die wir JVM zur Verfügung stellen können. Bei der Erfassung von kleinen floating point Werten laufen wir Gefahr, in ungenaue Dezimalstellen zu laufen. Hier ist eine Überlegung einer Einführung eines Schwellwertes sinnvoll, so dass die Genauigkeit bei einer bestimmten Abweichung als bereits 100% gilt.

8. Qualität

Die Qualität unserer Auswertungen wird mit sinnvollen Unittests abgesichert, so dass eine schnelle Überprüfung und Kontrolle der Kernaussagen ermöglicht wird.