HAW HAMBURG

HAUSARBEIT

Optimierung der Hyperparameter eines Neuronalen Netzes durch einen evolutionären Algorithmus

vorgelegt am 02. August 2024

Jonas Metzger

Prüfer: Prof. Dr. Peer Stelldinger

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Department Informatik
Berliner Tor 7
20099 Hamburg

Inhaltsverzeichnis

1 Einführung

1.1 Einleitung

Neuronale Netze sind allgegenwärtig. Immer häufiger werden sie zum Lösen komplexer Problemstellungen benutzt, in welcher ein regelbasierter Ansatz nicht möglich oder zu komplex erscheint. In den letzten Jahren gab es dabei erhebliche Fortschritte in der Entwicklung, die mittlerweile auch fest in der Populärkultur angekommen sind. ChatGPT hat in den letzten Jahren einen breiten Diskurs in der Öffentlichkeit ausgelöst. (Wolfangel, 2023).

Neuronale Netze zu entwickeln ist ein sehr langwieriger Prozess der oftmals auch sehr viele Ressourcen verbraucht. Dabei ist die Netztopologie ein wichtigen Faktor, welcher maßgeblich die Performance bestimmt. Eine Optimierung der Hyperparameter ist dabei eine weitere Möglichkeit die Performance des Netzes zu steigern. Zur Optimierung der Parametern werden üblicherweise Methoden wie Grid Search oder die Bayes'sche Optimierung benutzt (Feurer & Hutter, 2019).

Eine weitere vielversprechende Möglichkeit zur Optimierung der Hyperparameter bieten die Evolutionären Algorithmen. Es wird eine Population mit Individuen erstellt welche dann nach Regeln der Selektion und Mutation eine neue Population bilden. Dabei werden vielversprechende Lösungen miteinander kombiniert, um neue, bessere Lösungen zu finden. Genetische Algorithmen werden in Optimierungsproblemen verwendet, die über einen großen Suchraum über mehrere Dimensionen verfügen. (Stelldinger, 2024)

1.2 Fragestellung

Meine Hausarbeit beschäftigt sich mit der Optimierung von Hyperparameter eines neuronalen Netzes. Mithilfe eines genetischen Algorithmus soll eine möglichst optimale Konfiguration von Hyperparametern gefunden werden. Dabei soll das unterliegende neuronale Netz als *Blackbox* behandelt werden. Fokus dieser Hausarbeit liegt auf der Optimierung der genetischen Operationen wie der Selektion, Rekombination als auch die Mutation. Zudem soll der implementierte Algorithmus konventionellen Methoden wie Grid Search gegenübergestellt werden.

2 Vorgehensweise

Im folgenden Kapitel beschäftige ich mich mit der Ausarbeitung theoretischer Grundlagen als auch mit der groben Planung des Algorithmus und deren Komponenten.

2.1 Überblick zum genetischen Algorithmus

Genetische Algorithmen versuchen die Prozesse der natürlichen Evolution nachzuahmen. Dabei wird initial eine Population von möglichst unterschiedlichen Individuen erstellt, hier eine Konfiguration aus Hyperparameter, die dann durch die Fitness-Funktion bewertet wird. Ein niedriger Wert bei der Fitnessfunktion bedeutet eine gute Lösung. Mittels einer Selektion werden schlechtere Konfigurationen entfernt, sodass durch eine Kombination mehrerer guter Lösungen wieder bessere entstehen können. Um eine große Diversität zu gewährleisten, werden die einzelnen Individuen mutiert, dabei werden einzelne Merkmale arithmetisch verschoben. Dieser Prozess wiederholt sich ständig, sodass weitere, womöglich bessere, Konfiguration entdeckt werden.

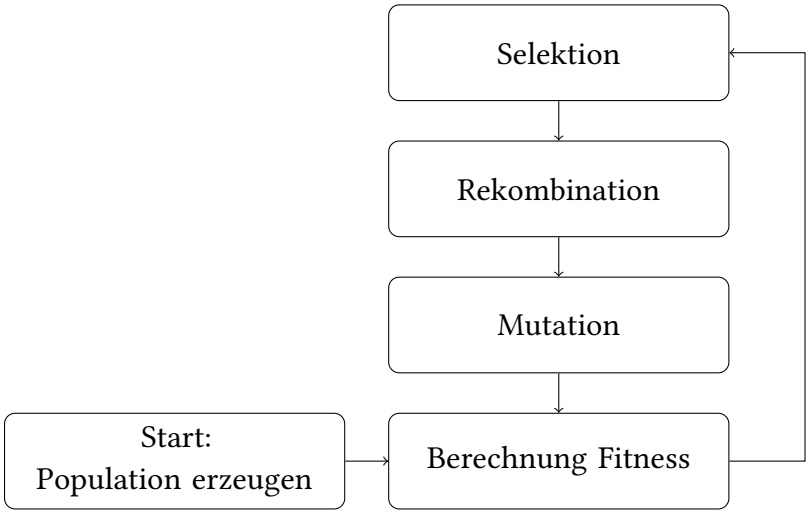

Abbildung 2.1: Grundschema Genetischer Algorithmus (Stelldinger, 2024, Seite 5)

2.2 Überblick Neuronale Netze

Neuronale Netze sind eine Form des überwachten Lernens, das heißt sie lernen anhand von bereits bekannten Zielwerten. Im Beispiel der Klassifikation von Tierbildern ist beim überwachten Lernen bereits für jedes Bild bekannt, um welches Tier es sich dabei handelt. Das neuronale Netz erkennt selbstständig Muster in den Tierbildern, welche dann zur Generalisierung unbekannter Daten verwendet werden. Ziel hierbei ist die Extraktion der wichtigen Merkmale, nicht das stupide Auswendiglernen der Daten („*Overfitting*"). Das neuronale Netz wird dabei mit Trainingsdaten gefüttert, anhand dessen die Muster gelernt werden. Üblicherweise wird die Effektivität oder Genauigkeit des Modells an Daten überprüft, die nicht zum Training gehören (Sonnet, 2022).

2.2.1 Hyperparameter in neuronalen Netzen

Hyperparameter sind Parameter, welche den Lernprozess betreffen. Im Kontrast zu den Modellparametern, die während des Trainings aufgebaut werden, sind Hyperparameter beim Training bereits bekannt. Es gibt viele neuronale Netze, die Erfolg mit dynamischen Parametern haben (Schaul et al., 2013). Andere typische Hyperparameter eines neuronalen Netzes beinhalten die Batch-Größe, Anzahl von Epochen sowie auch die Anzahl von Neuronen in der verdeckten Schicht. In dieser Hausarbeit beschränke ich mich auf die genannten Parameter.

2.3 Aufbau des Genetischen Algorithmus

Im nachfolgenden Kapitel beschäftige ich mich mit der Planung des genetischen Algorithmus, dabei wird auf die einzelnen Operationen eingegangen.

2.3.1 Chromosom

Ein Chromosome wird üblicherweise in binärer Darstellung kodiert. Dies hat den Vorteil, dass bitweise Operationen sehr effizient und schnell umsetzbar sind. Allerdings kann dies auch zu Problemen führen. Bei der Mutation oder Rekombination ist es daher zu beachten, dass die höherwertigen Bits (MSB) einer Zahl, die Zahl stärker verändern können als es die Least Significant Bits. Um diese Nicht-Linearität zu vermeiden, müssen die Operationen besser angepasst werden (Herrera et al., 1998).

Ein Chromosom besteht aus vier Parametern: Batch-Größe, Neuronen in der verdeckten Schicht, Anzahl von Epochen und der Lernrate. Da die geplante Rekombination mit Operationen auf reellen und natürlichen Zahlen durchführt werden soll, habe ich mich gegen eine Repräsentation in Binärform entschieden. Bis auf die Lernrate werden alle Parameter als natürliche Zahl kodiert.

2.3.2 Population

Initial wird eine Startpopulation erzeugt, dabei wird aus einen vorher definierten Bereich zufällig eine Zahl gewählt.

- **Lern Rate**: Es wird eine logarithmische Verteilung im Wertebereich von $[0.1, 0.00001]$ vorgenommen. Eine nicht-logarithmische Verteilung ist hierbei nicht gewünscht, da die Lernrate ansonsten in aller Regel sehr nah an *0.1* bleibt. Eine logarithmische Verteilung garantiert hier eine ausgeglichene Verteilung.
- **Batch-Größe**: Stetige Gleichverteilung im natürlichen Intervall $[8, 512]$.
- **Epochen**: Stetige Gleichverteilung im natürlichen Intervall $[5, 20]$.
- **Neuronen in der verdeckten Schicht**: Stetige Gleichverteilung im natürlichen Intervall $[10, 1000]$.

Durch erste Tests konnte eine maximale Konfiguration mit einer Batch-Größe von 512 und 1000 Neuronen in der verdeckten Schicht ermittelt werden. Eine größere Konfiguration ist mit der aktuell verwendeten Hardware nicht möglich, da der Grafikspeicher nicht ausreichend groß ist.

2.3.3 Fitness

Die Fitness bewertet eine mögliche Konfiguration an Hyperparameter innerhalb des neuronalen Netzes. Hierfür wird das neuronale Netz mit Daten trainiert und anschließend mit ungesehenen Daten validiert. Die Fehlerfunktion („Loss“), auf Basis der Validierungsdaten, ist dabei ein wertvoller Indikator für ein gut trainiertes Netz, denn anders als die Genauigkeit, welche nur ein metrischer Wert ist, reflektiert die Fehlerfunktion auch bei kleinen Änderungen im Netz einen niedrigeren Wert. Daher gilt es diese Funktion zu minimieren (Goodfellow et al., 2016, Kapitel 4.3). Insgesamt ist die Berechnung der Fitness ein sehr aufwändiges Unterfangen, da hierzu das komplette Netz trainiert und anschließend mit Validierungsdaten überprüft werden muss. Gemittelt dauert eine Berechnung eine Minute.

2.3.4 Rekombination

Die Rekombination erfolgt über zwei Elternteile. Hierfür wird im Wertebereich des Parameters eine Normalverteilung zwischen den zwei Elternteilen formiert. So ist ein Wert innerhalb der beiden Elternteile wahrscheinlich, nicht aber zwangsläufig verpflichtend. Durch einen Parameter kann die Standardabweichung beeinflusst werden. So kann ein Fokus entweder auf Exploitation oder Exploration gelegt werden. Diese Art von Rekombination ist inspiriert durch die Gauss-Mutation (Kruse & Held, 2013, Seite 8).

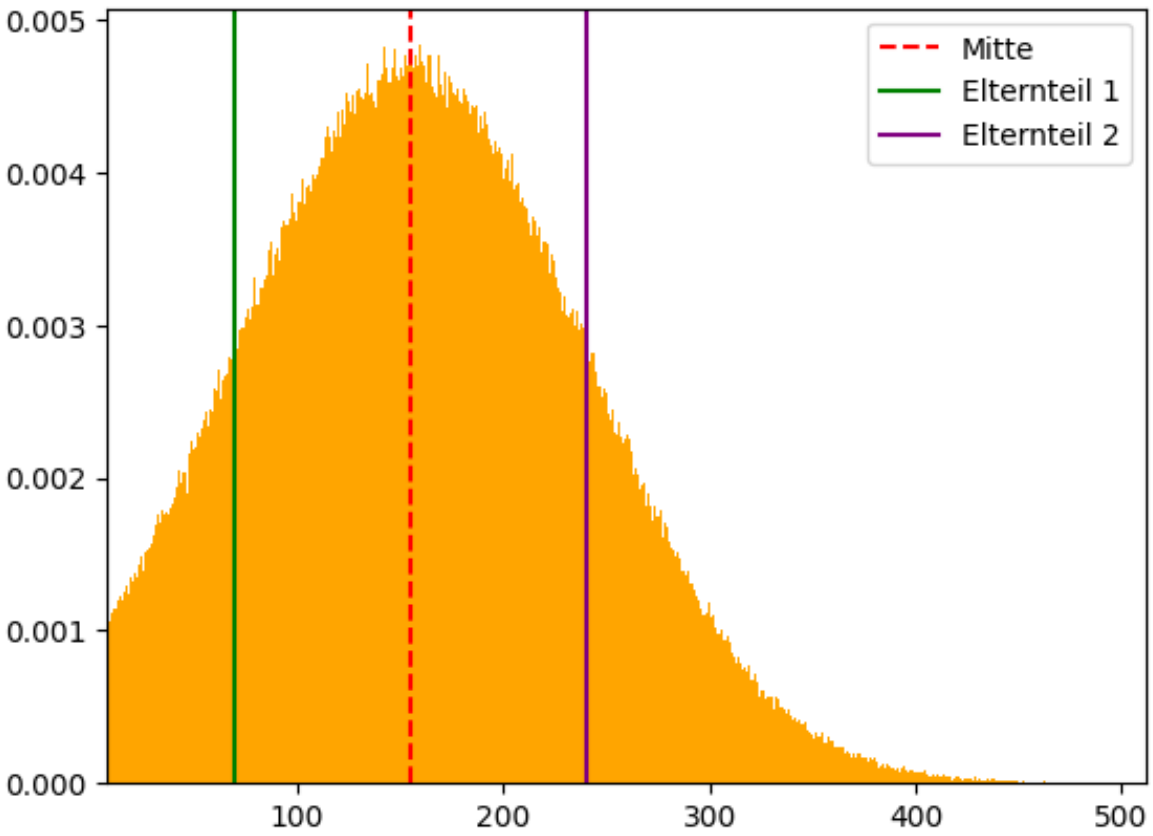

Abbildung 2.2: Normalverteilung bei der Rekombination für Batch-Größe

2.3.5 Selektion

Die Selektion reduziert die Population nach der gewählten Strategie. So wird Platz für besser angepasste Individuen. Um bereits gefundene lokale Optimas nicht zu verlieren, werden pro Generation jeweils zwei Elitisten bestimmt, diese werden unverändert in die nächste Generation übernommen. Zusätzlich findet eine Turnierselektion zwischen zwei Chromosomen statt. Die Gewinner kommen in die nächste Generation, ein erneutes Auswählen findet nicht statt. So können auch schlechtere Chromosomen in die nächste Generation kommen, sodass lokale Optimas besser umgangen werden können. Durch die Turnierselektion wird die Population um 50% reduziert.

2.3.6 Mutation

Die Mutation wird nach der Rekombination für jedes Konfiguration durchgeführt, die kein Elitist ist. Sie sorgt dafür, dass sich schlechtere wie auch bessere Konfigurationen verbessern oder verschlechtern können und sorgt für eine Diversität in der Population. Im genetischen Algorithmus wird jedes Gen mit einer Wahrscheinlichkeit von 20% komplett neu generiert. So wird die Exploration verstärkt.

2.4 Bemerkungen zum Ansatz

Der hier gewählte Ansatz hat seine Tücken. So ist die Wahl der Standardabweichung bei der Rekombination sehr wichtig, um eine Balance zwischen Exploration und Exploitation zu finden. In der Rekombination wird eher Wert auf die Exploitation gesetzt, während in der Mutation der Fokus auf die Exploration des Suchraums gesetzt wird.

3 Methodik

3.1 Versuchsaufbau

3.1.1 Neuronales Netz

Nachfolgend ist die serielle Architektur des neuronalen Netzes aufgezeigt, zur Minimalisierung der Fehlerfunktion wird Adam eingesetzt.

1. Verdeckte Schicht: Vollständig verbundene Schicht mit durch genetischen Algorithmus optimierten Neuronenzahl, benutzt die ReLu Funktion zur Aktivierung.
2. Ausgabeschicht: Vollständig verbundene Schicht mit 10 Neuronen, welche durch Softmax aktiviert werden

Bei der Wahl der Architektur wurde drauf geachtet, ein nicht allzu kompliziertes Netz zu bauen. Ein flaches Netzwerk hat viel weniger Hyperparameter als ein tiefes Netz mit Convolutional- und Pooling-Schichten. Damit wurde der Suchraum für den genetischen Algorithmus drastisch reduziert.

3.1.2 Trainingsdaten

Das Modell wird auf dem populären Mnist Datensatz laufen. Dieser umfasst 60.000 Daten zum Training und 10.000 zur Validierung. Ein Datensatz enthält eine handgeschriebene Ziffer von 0 bis 9, mit 28x28 Pixel in Graustufe („MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges“, n. d.).

3.2 Implementierung

Die Implementierung erfolgt in Python. Das neuronale Netz wird mit Keras/Tensorflow ausgeführt und der genetische Algorithmus wird selbst implementiert. Hierfür lagen noch einzelne Codestücke in Java vor. Da das Training länger dauern kann, soll insbesondere ein Fokus auf Stabilität, als auch auf die Persistenz. Einzelne Generationen werden gesondert zu in einem „State“ gespeichert und von diesem kann der Algorithmus auch wieder fortfahren.

3.3 Durchführung

Der genetische Algorithmus wird in der „ICC Cloud“ der HAW Hamburg ausgeführt. Der Zugriff erfolgt über JupyterHub. Durch die Persistenz kann der Algorithmus jederzeit gestoppt und fortgefahren werden. Dabei haben sich die Jupyter Notebooks als nicht praktikabel erwiesen. Der Output war teilweise schwammig und nicht sichtbar, sodass der Verlauf des Algorithmus nicht gut beobachtet werden. Deshalb wird das Programm als Standalone Python Script innerhalb von JupyterHub ausgeführt. Das Training lief auf einer Tesla V100 GPU, was die Trainingszeiten erheblich verbesserte.

Die Durchführung erfolge mit zwei Konfigurationen, einmal mit einer höheren Standardabweichung bei der Rekombination als auch mit einer niedrigeren. Somit kann bei allen nachfolgenden Generationen entweder der Fokus auf Exploration oder auf die Exploitation gelegt werden.

3.4 Herausforderungen

3.4.1 Memory Leak in Tensorflow

Das mehrfache Ausführen in Tensorflow verursachte leider konstant einen Speicherleck, sodass eine zuverlässige Ausführung des Algorithmus unmöglich wurde. Das Speicherleck trat sowohl bei unterschiedlichen Tensorflow/Keras Versionen auf, als auch auf verschiedenen Systemen. Daher wurde das Programm in zwei unterschiedliche Programme geteilt, eines welches den genetischen Algorithmus ausführte und eines, das die Ausführung des neuronalen Netzes durchführte. Dieses wurde dann vom genetischen Algorithmus pro Fitness-Berechnung ausgeführt und die Resultate als Datei übergeben.

3.4.2 Hyperparameterbegrenzung durch Hauptspeicherlimitierung

Eine zu hohe Konfiguration der Hyperparameter führte innerhalb der Maschine zum Abbruch, da eine hohe Batch-Größe in Kombination mit einer großen Neuronenzahl zu einer hohen Speichernutzung führt. Dies kann soweit führen, dass die Ausführung nicht mehr möglich ist. Hierfür muss eine Limitierung der Parameter gefunden werden. Dies ist bei ein oder zwei Parametern trivial, kann aber bei tieferen Netzen zum Problem werden.

4 Ergebnisse

4.1 Überblick

Der genetische Algorithmus wurde mit einer Populationsgröße von 50 über 30 Generationen hinweg optimiert. Dabei konnte sowohl die Genauigkeit minimal verbessert werden als auch die Fehler reduziert. Der genetische Algorithmus konvergiert sehr schnell zum Ergebnis. Die Ausführung dauerte 14 Stunden. Dabei handelte es sich um die Konfiguration mit einer einer geringeren Standardabweichung bei der Rekombination, das bedeutet, dieser Lauf fokussierte sich mehr auf die Exploitation. Im Anhang finden sich die Resultate zum Lauf der Exploration.

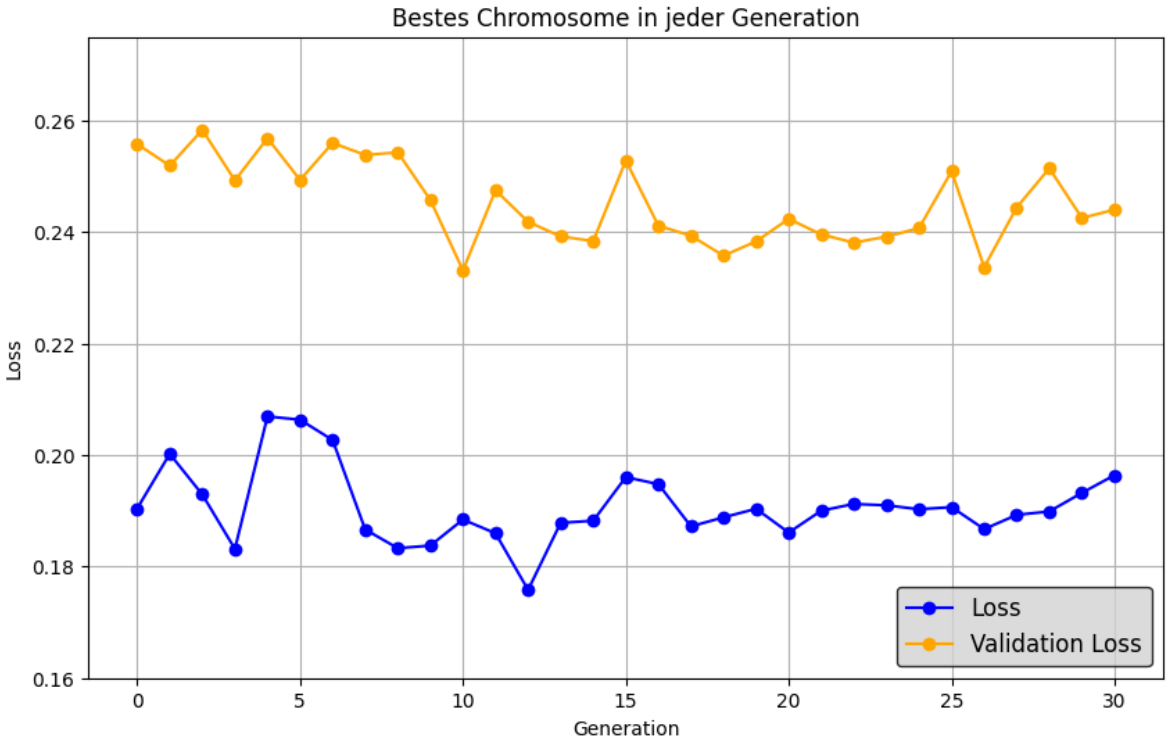

Abbildung 4.1: Geringster Fehler pro Generation

4.2 Interpretation der Ergebnisse

Die Testergebnisse sind nicht wirklich aussagekräftig, denn die Fitness-Berechnung ist leider nicht deterministisch. So kann es passieren, dass ein Elitist, der gerade noch einen Fehler von 0.24 hatte, in der nächsten Generation einen Fehler von 0.23 ermittelt. Die Startkonfiguration der Neuronen sind zufällig und so fließt leider auch eine gewisse nicht-deterministische Komponente in die Fitness-Funktion ein. Dies wurde deutlicher, wenn probiert wurde, das beste Individuum zu reproduzieren: hierbei wurde oft ein schlechterer Wert erreicht. Das gute Ergebnis kam daher durch einen optimaleren Bias im Neuron.

Insgesamt konnte das Netz nicht merklich verbessert werden, dies könnte an der geringen Auswahl der Hyperparameter liegen. Wahrscheinlich wäre es besser ein tieferes Netz mit mehr Schichten zu nehmen, in welcher dann auch die Neuronenanzahl optimiert werden, dies würde aber auch den Suchraum extrem erweitern. Karlupia et. al. hatten Erfolge bei der Benutzung eines genetischen Algorithmus zur Optimierung eines CNN für die Gesichtserkennung. CNN's haben durch die Faltungs- und Pooling-Schichten deutlich mehr Hyperparameter zum Optimieren. (Karlupia et al., 2023).

Das rein zufällige Erstellen der Generationen nach „Random Grid Search", führt daher zu einen ähnlichen Ergebnis. Es liegt also nahe, dass die Optimierung der vier Hyperparameter schlicht weg zu wenig war. Die Devianz der einzelne Läufe waren dafür zu hoch.

4.3 Verbesserungsmöglichkeiten

Eine Erhöhung der Gene im Chromosom führt zu einem erweiterten Suchraum und mehr Dimensionen, so würde der Unterschied zu Grid Search, welches öfters Probleme mit einer hohen Anzahl von Dimensionen hat, größer werden. Daher erscheint es sinnvoll mehr Hyperparameter aufzunehmen.

4.4 Vergleich zu Grid Search

Um einen Vergleich zu einer anderen Hyperparameter-Optimierungen bekommen wurde das gleiche neuronale Netz mittels Grid Search optimiert. Hierfür wurden die einzelnen Parameter im gleichen Intervall rasterförmig verstreut und ausprobiert. Dabei wurden um die 700 Kombinationen der Parameter ausgeführt und deren Genauigkeit/Fehler verglichen. Dabei wurde beobachtet, dass diese Brute Force Herangehensweise bei einem solch kleinen Netz zu ähnlich guter Konfiguration führt.

4.5 Fazit

Genetische Algorithmen erweisen gute Dienste bei dem Optimierungsproblem, allerdings sollte drauf geachtet werden den Suchraum nicht zu klein zu lassen. Wenn es die Problemstellung zulässt, dann

sollte am besten der Suchraum erst ausreichend groß skaliert werden. Wenn dann in gegebener Zeit keine gute Lösung gefunden wird, dann kann der Suchraum auch verkleinert werden.

Literatur

Feurer, M., & Hutter, F. (2019). Hyperparameter Optimization [Series Title: The Springer Series on Challenges in Machine Learning]. In F. Hutter, L. Kotthoff & J. Vanschoren (Hrsg.), *Automated Machine Learning* (S. 3–33). Springer International Publishing. https://doi.org/10.1007/978-3-030-05318-5_1
and it was also established early that different hyperparameter configurations tend to work best for different datasets" (Feurer and Hutter, 2019, p. 2).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press.
Applied math and machine learning basics. Linear algebra – Probability and information theory – Numerical computation – Machine learning basics – Deep networks: modern practices. Deep feedforward networks – Regularization for deep learning – Optimization for training deep models – Convolutional networks – Sequence modeling: recurrent and recursive nets – Practical methodology – Applications – Deep learning research. Linear factor models – Autoencoders – Representation learning – Structured probabilistic models for deep learning – Monte Carlo methods – Confronting the partition function – Approximate inference – Deep generative models.

Herrera, F., Lozano, M., & Verdegay, J. L. (1998). Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis.

Karlupia, N., Mahajan, P., Abrol, P., & Lehana, P. K. (2023). A genetic algorithm based optimized convolutional neural network for face recognition. *International Journal of Applied Mathematics and Computer Science*, *33*(1). https://doi.org/10.34768/amcs-2023-0002

Kruse, R., & Held, P. (2013). Evolutionäre Algorithmen Variation und genetische Operatoren. http://fuzzy.cs.ovgu.de/ci/ea/ea2013_v04_operatoren.pdf

MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. (n. d.). Verfügbar 29. Juli 2024 unter https://yann.lecun.com/exdb/mnist/

Schaul, T., Zhang, S., & LeCun, Y. (2013, Februar). No More Pesky Learning Rates [arXiv:1206.1106 [cs, stat]]. Verfügbar 27. Juli 2024 unter http://arxiv.org/abs/1206.1106

Sonnet, D. (2022). *Neuronale Netze kompakt: Vom Perceptron zum Deep Learning*. Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-29081-8

Stelldinger, P. D. P. (2024). Naturanaloge Optimierungsverfahren – Genetische Algorithmen. *Künstliche Intelligenz*.

Wolfangel, E. (2023). ChatGPT: Wie nah sind wir an der Superintelligenz? *Die Zeit*. Verfügbar 25. Juli 2024 unter https://www.zeit.de/digital/2023-04/chatgpt-kuenstliche-intelligenz-forschung

Anhang

Der Sourcecode steht auf Github zur Verfügung:
https://github.com/jonasmetzger2000/KI-STG-GI-Neural-Network-Optimizer

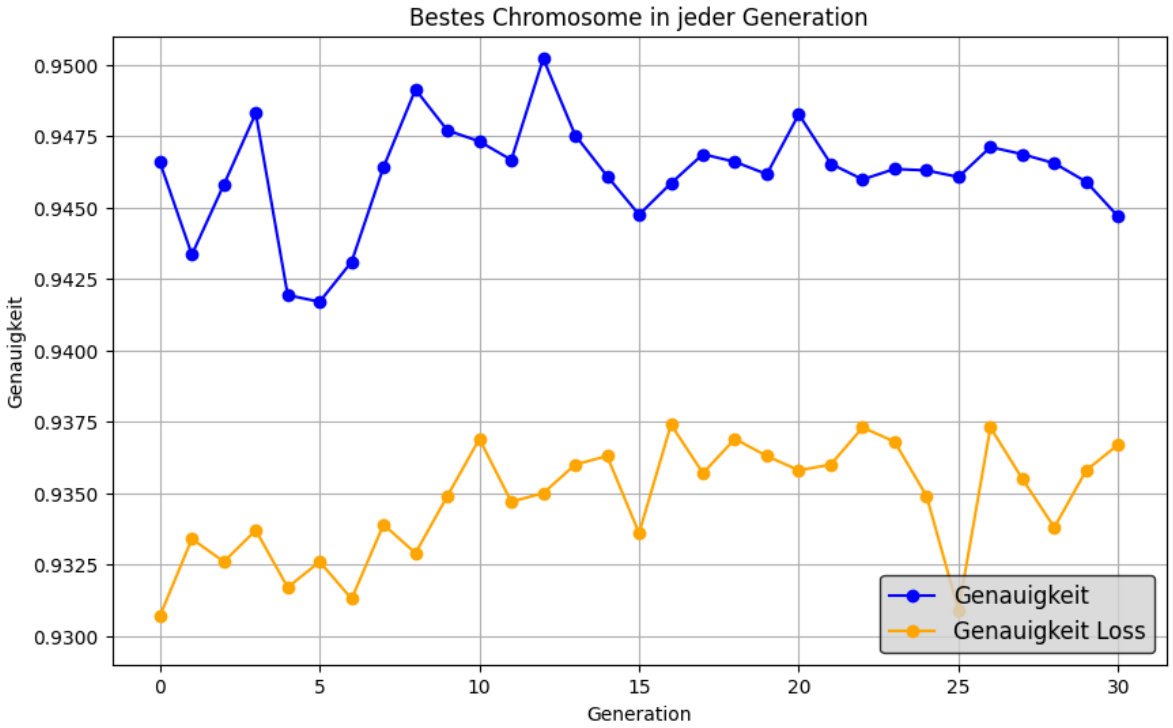

Abbildung 1: Lauf mit Priorisierung der Exploitation: Höchste Genauigkeit pro Generation

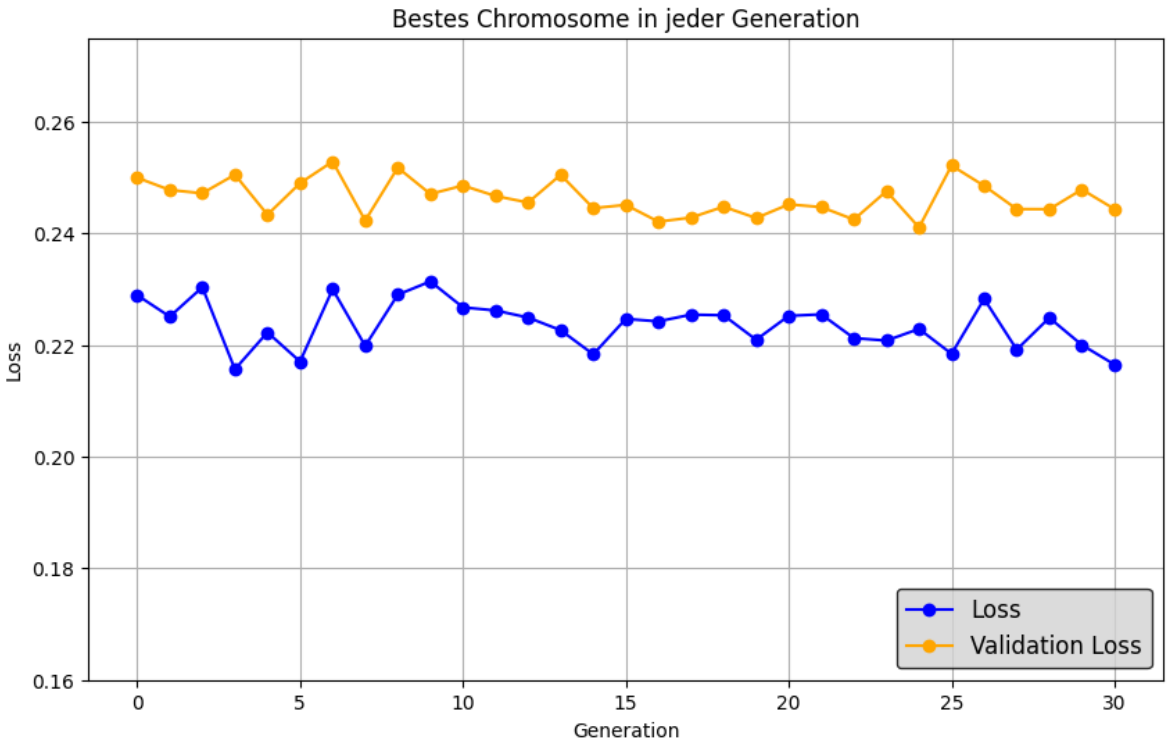

Abbildung 2: Lauf mit Priorisierung der Exploration: Geringster Fehler pro Generation

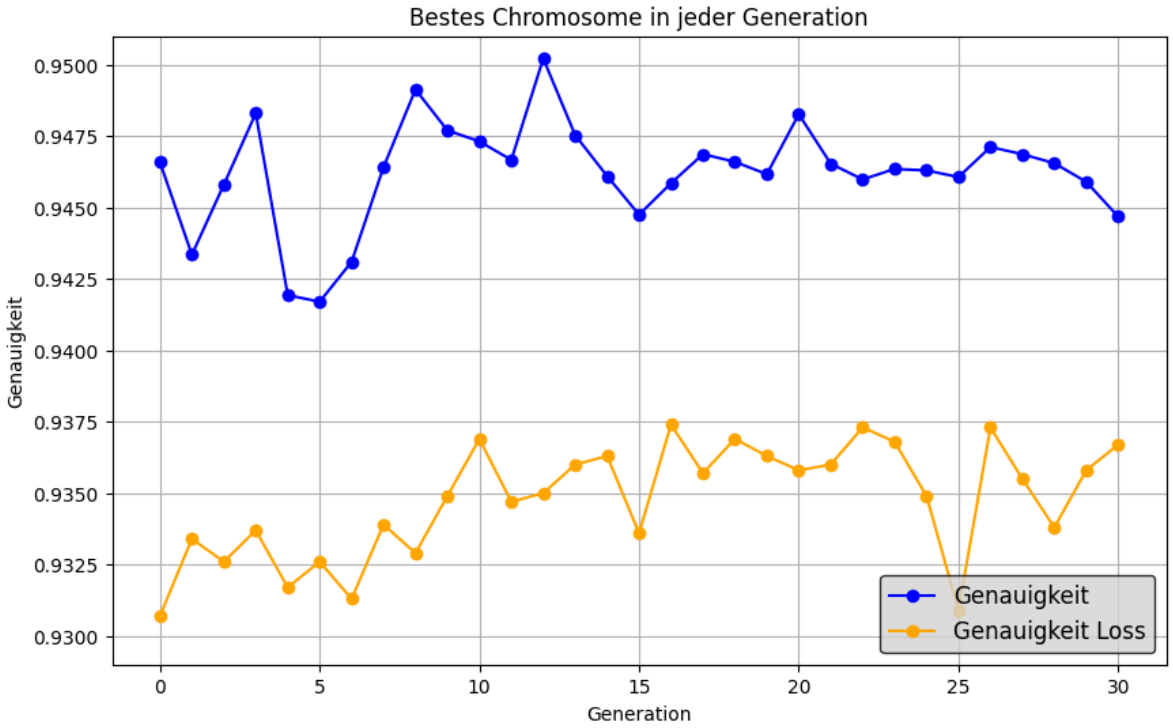

Abbildung 3: Lauf mit Priorisierung der Exploration: Höchste Genauigkeit pro Generation

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel

Optimierung der Hyperparameter eines Neuronalen Netzes durch einen evolutionären Algorithmus

selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Hamburg, 2. August 2024