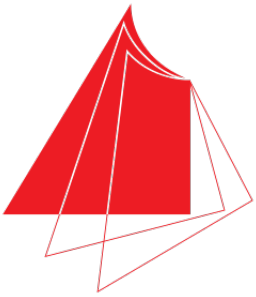
SE[SUBALG]IndentEndIndent *Indent *EndIndent



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Genetische Algorithmen zur Optimierung von Hyperparametern eines künstlichen neuronalen Netzes

Fakultät für Maschinenbau und Mechatronik
der Hochschule Karlsruhe
Technik und Wirtschaft

Bachelorarbeit

vom 01.03.2018 bis zum 31.08.2018
vorgelegt von

Christian Heinzmann
geboren am 18.02.1995 in Heilbronn
Martrikelnummer: 52550

Winter Semester 2019

Professor	Prof. Dr.-Ing. habil. Burghart
Co-Professor	Prof. Dr.-Ing. Olawsky
Betreuer FZI:	M. Sc. Kohout

Eigenständigkeitserklärung und Hinweis auf verwendete Hilfsmittel

Eigenständigkeitserklärung und Hinweis auf verwendete Hilfsmittel. Hiermit bestätige ich, dass ich den vorliegenden Praxissemesterbericht selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen des Berichts, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Datum: _____ Unterschrift: _____

Ausschreibung

BACHELORARBEIT

Genetische Algorithmen zur Optimierung von Hyperparametern eines künstlichen neuronalen Netzes

Zur Vermeidung der weiteren Ausbreitung von multiresistenten Keimen in Einrichtungen des Gesundheitswesens (insb. Krankenhäuser) werden am FZI Systeme und Methoden entwickelt, welche helfen sollen die Händehygiene-Compliance von Mitarbeitern dort zu erhöhen. Durch technische Unterstützung bei häufig wiederkehrenden Maßnahmen, soll mehr Zeit geschaffen und gleichzeitig auf die Wichtigkeit von Desinfektionsmaßnahmen aufmerksam gemacht werden. Dies soll mit Hilfe von Augmented-Reality umgesetzt werden. Dabei ist das Ziel, bekannte Prozesse, wie beispielsweise den Wechsel von postoperativen Wundverbänden, visuell zu unterstützen und automatisch zu dokumentieren.

AUFGABEN

Im Kontext der Detektion von Aktionen und Objekten, soll die Optimierung von Neuronalen Netzen mit Genetischen Algorithmen durchgeführt werden. Das Ziel hierbei ist es, ein Framework zu entwickeln und zu implementieren, in welchem künstliche neuronale Netze automatisiert trainiert werden. Unter anderem sollen die Hyperparameter mit Hilfe von Genetischen Algorithmen inteligent angepasst und das trainierte Netz anschließend ausgewertet werden. Diese Aufgaben sollen voll automatisiert ablaufen. Daraus ergeben sich folgende Aufgaben:

- Literaturrecherche über aktuelle Genetische Algorithmen und aktuelle Neuronale Netze
- Einarbeiten in vorhandene Frameworks für Genetische Algorithmen und Neuronale Netze
- Konzeptionierung und Implementierung des ausgewählten Ansatzes zur Optimierung von Hyperparameter des Neuronalen Netzes
- Evaluation und Auswertung speziell unter der Beachtung geringer Datenmengen
- Wissenschaftliche Aufbereitung und Dokumentation des Projekts

WIR BIETEN

- Aktuelle Softwaretools im täglichen wissenschaftlichen Einsatz
- eine angenehme Arbeitsatmosphäre
- konstruktive Zusammenarbeit

WIR ERWARTEN

- Grundkenntnisse in maschinellem Lernen
- Kenntnisse in folgenden Bereichen sind von Vorteil: Python, Tensorflow, C/C++
- selbständiges Denken und Arbeiten
- sehr gute Deutsch- oder Englischkenntnisse
- Motivation und Engagement

ERFORDERLICHE UNTERLAGEN

Wir freuen uns auf Ihre PDF-Bewerbung an Herrn Lukas Kohout, kohout@fzi.de, mit folgenden Unterlagen:

- aktueller Notenauszug
- tabellarischer Lebenslauf

WEITERE INFORMATIONEN

- Start: ab sofort

FZI Forschungszentrum Informatik | Embedded Systems and Sensors Engineering (ESS)
Lukas Kohout | kohout@fzi.de | Tel. +49 721 9654 - 167
Mehr Informationen unter www.fzi.de/karriere

Wird noch zu Abstact geändert

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

1 Einleitung

1.1 Motivation

Künstliche Neuronale Netze dominieren das Feld des Maschinellen Lernens, aber ihr training und ihr erfolg hängt immer noch von sensibel empiriisch ausgesuchten Hyperparametern. Zu diesen Hyperparametern können Modelachitekur, Verlustfunktion und optimierungs Algorithmen. [JDO+17]

Momentan werden diese meist nach groben ermessen des Entwicklers ausgewählt. Das Auswählen und Testen beansprucht sehr viel Zeit und Mühe. Es gibt ansätze wie Random Search und Grid Search welche

1.2 Aufgabenstellung

Ziel der Arbeit ist es, zunächst die Optimierung von Hyperparametern zu vereinfachen. Dazu ist eine automatiesierter Trainings- und Auswerte-vorgang nötig. Anschließend sollen die Hyperparameter mit Hilfe von Genetischen Algorithmen noch verbessert werden, um schneller bessere Ergebnisse zu erhalten. Diese Ergebnisse sollen dann einer klassichen Grid Search gegenübergestellt werden.

Um dies zu vereinfachen soll ein Konzept geschaffen werden, welches die Vorgänge automatisiert und Optimiert. Dabei geht es hauptsächlich um den Vorgang der Auswahl von Hyperparameter und die Auswahl der Dimension eines Künstlichen Neuronalen Netzes. Diese berechneten Werte sollen gespeichert und anschließend übersichtlich anzeigt werden, wodurch sich die idealen Parameter herausbilden. Diese Ergebnisse sollen dem momentanen Ansatz gegenübergestellt werden.

(Mit diesem Ansatz kann die Dimensionierung eines Netzes einfacher umgesetzt werden.) Ein weiter Anwendungsfall ist die Hyperparameterauswahl. Mit Hilfe dieses Werkzeugs soll eine einfachere und bessere Auswahl der Hyperparameter erfolgen. Diese berechneten Werte sollen gespeicher und anschließend übersichtlich und intuitiv anzeigt werden, wodurch sich die idealen Parameter herausbilden. Mit diesem Ansatz soll die Richtigkeit des Netzes erhöht werden, sodass es bessere Ergebnisse liefesert. Dieses Werkzeug soll Konzeptioniert und Implementiert werden. Anschließend soll eine Evaluation und Auswertung über die mögliche Verbessung durchgeführt werden.

1.3 Aufbau der Arbeit

Zunächst wird im zweiten Kaptiel auf die verwendeten Grundlagen eingegangen. Zunächst wird im zweiten Kaptiel auf die Grundlagen zu Genetischen Algorithmen und Künstlichen Neuronalen Netzen eingegangen.

Welche Algorythmen bei dieser Arbeit verwendet werden. Und mit welchen Künstlichen Neuronalen Netzen diese Optimierungs Algorithmen getestet werde. Außderdem wird in Abschnitt 3 auf den Momentanen Stand der Technik und Forschung eingegangen dort werden auch einige Anwendungbeispiele der Genetischen Algorithmen genannt. Nun folgt in Kapitel 4 die ausarbeitung des Konzeptes mit erklärungen der einzelnen Ideen. Darauf aufbauend kommt Implementierung in Kapitel 5 in welcher mit psyodocode erklärt wird wie die Arbeit umgesetzt wurde. Anschließend wird das Implementierte system Evaluiert und getestet. Zum Schluss in Kapitel 7 gibt es eine Zusammenfassung

2 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen, welche zum Verständnis der vorliegende Arbeit wichtig sind, beschrieben. Zu Beginn erfolgt ein Einstieg in künstlichen neuronalen Netze, anschließend werden die Hyperparameter und ihre Besonderheiten thematisiert. Dann folgt eine kurze Einführung in die Optimierungsverfahren. Abschließend werden die grundlegende Aspekte von Zufallssuche und Genetischen Algorithmen erläutert.

2.1 Künstliche Neuronale Netze

Im folgenden Kapitel wird zuerst der Aufbau eines Neurons erklärt. Anschließend wird der strukturelle Aufbau eines Künstlichen Neuronalen Netzes näher gebracht. Es werden wichtige Funktionen wie die Verlustfunktion und die Fehlerrückführung erklärt. Zum Schluss wird auf die Hyperparameter, welche für die Arbeit essenziell sind eingegangen.

Künstliche Inteligenz (engl. Artificial Intelligence) sorgte in den letzten Jahren für weitreichende Schlagzeilen. Im Detail versteckt sich hinter dieser künstlichen Intelligenz meist Maschinelles Lernen im speziellen tiefe künstliche neuronale Netze (engl. Deep Artificial neuronal Networks oder Deep Learning). Diese werden für Anwendungen mit großen Datenmengen eingesetzt bspw. in Bereichen der Bildverarbeitung, Chatbots, Fahrassistenz Systeme. Beim Deep Learning werden die Funktionen durch ein Modell erlernt, welches neuronale Netze genannt wird. Dieser Begriff stammt ursprünglich aus der Neurobiologie. Bei den Künstlichen Neuronalen Netzen handelt es sich aber nicht um Nachbildungen des Gehirns, das natürliche Vorbild dient nur als Inspiration. In Abbildung 2 ist so ein Künstliches Neuronales Netz zu sehen, jede Schicht ist aus einzelnen Neuronen aufgebaut. Diese können dann Schichtweise zusammengefügt werden, wodurch ein Netz entsteht. Diese Verbindungen repräsentieren die Gewichte, über welche einem Netz, lineare als auch nicht lineare Zusammenhänge an trainiert werden. [SSF+16, p. 11]

2.1.1 Aufbau eines Neurons

Abbildung 1 zeigt ein Neuron, welches aus dem schematischen Aufbau besteht: Eingänge, Gewichte, Schwellwert, Aktivierungsfunktion und einem Ausgang. In den nachfolgenden Unterkapiteln werden diese Ausführlich erklärt, Informationen stammen wenn nicht anderweitig angegeben aus der Arbeit [SSF+16, p. 12].

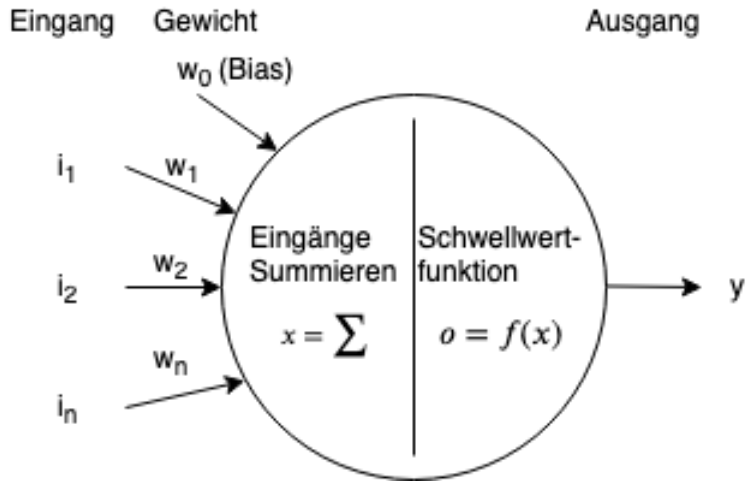


Abbildung 1: Aufbau eines Neurons

Eingang Bei den Eingangswerten $i_1, ..., i_n$ handelt es sich um relle Zahlen, diese werden mit den Gewichten $w_1, ..., w_n$ verrechnet. Ein Neuron hat meist mehrere Eingangsgrößen, welche alle zusammen mit den Gewichten und dem Schwellwert aufsummiert werden. Die Gewichte werden zufällig initialisiert und per Training verbessert, somit handelt es sich um einen angelernten Werte, welche durch die Fehlerrückführung (engl. Backproagation) verbessert werden. Auf dieses aufsummierten Ergebnisse wird anschließend ein Schwellwert (engl. Bias) w_0 gerechnet, dieser führt zu einem besseren Verhalten beim Trainieren. Beim Schwellwert handelt es sich auch um einen angelernten Wert. Dieser hilft die Flexibilität des Netzes zu erhöhen. Der Schwellwert sorgt dafür das ein Ausganswert erzeugt wird auch wenn kein Eingangswert anliegt. Daraus ergibt sich folgende Übertragungsfunktion:

$$x = \sum_{k=1}^{n} i_k * w_k + w_0 \tag{1}$$

Aktivierungsfunktion Die Aktivierungsfunktion kann man sich als Schwellwertfunktion vorstellen. Sie stellt den Zusammenhang zwischen Eingang und Ausgang her. Die entstandene Summe x wird Aktivierung genannt und anschließend über eine Aktivierungsfunktion transformiert:

$$o = f(x) \tag{2}$$

Die Aktivierungsfunktion kann dabei verschiedene Formen haben. Für einfache Aufgaben kann Beispielsweise eine Sprungfunktion verwendet werden:

$$\sigma(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0 \end{cases} \tag{3}$$

Für das approximieren von wertkontinuierlichen Funktionen wird die Sigmoid Funktion verwendet.

$$sig(t) = \frac{1}{1 + e^{-t}} \tag{4}$$

Bei der Klassifikation werden, ReLU-Layer (5) oder Leaky-ReLU Layer (6) benutzt, diese verhindern das Explodieren bzw. Verschwinden des Grandienten beim Training:

$$R(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases} \tag{5}$$

$$R(z) = \begin{cases} 1, & z \geq 0 \\ \alpha z, & z < 0 \end{cases} \tag{6}$$

Ausgang Die Schwellwertfunktion übergibt einen Wert an den Ausgang. Dieser Ausgangswert kann entweder an andere Neuronen weitergeben oder als finales Ergebnis verwendet werden.

2.1.2 Struktureller Aufbau eines Künstlichen Neuronalen Netzes

Aus schlauem zusammenschließen solcher Neuronen entsteht ein Künstliches Neuronales Netz welches auch Multi-Layer-Perzeptron(MLP) genannt wird. Dabei sind grundsätzliche jegliche strukturelle Anordnung der Neuronen möglich. Besonders verbreitet ist das sogenannte Feedforward Netz(FFW). Bei den FFW Netzen sind die Verbindungen nur in eine Richtung gerichtet. Heißt die Informationen werden von Vorne(Eingang) nach Hinten(Ausgang) weiter gegeben. Dies wird beispielhaft in Abbildung 2 gezeigt. Hier ist gut zu sehen, dass ein künstliches neuronales Netz in drei Schichten unterteilt werden kann. [SSF+16, p. 21]

- **Eingangsschicht** Die Neuronen der Eingangsschicht sind nicht wie die in 2.1.1 beschriebenen Neuronen aufgebaut. Die einzige Aufgabe der Eingangsneuronen ist das verteilen der Eingagsinformationen an die versteckte Schicht, weshalb es immer genau soviele Eingangsneuronen wie Eingangssignale geben muss.

- **Versteckte Schicht** Die versteckte Schicht (engl. hidden Layer) besteht aus den in 2.1.1 erläuterten Neuronen. Sie kann aus einer beliebigen Anzahl von Neuronen aufgebaut sein. Es kann auch beliebig viele dieser Schichten geben. In der Literatur werden Neuronale Netze mit mehr als einer versteckten Schicht als Deep Neural Networks bezeichnet.

- **Ausgangsschicht** Die Ausgangsschicht beinhaltet genau soviele Neuronen wie Ausgangsgrößen gewünscht. Aus dieser können dann die Klassifizierungswerte entnommen werden.

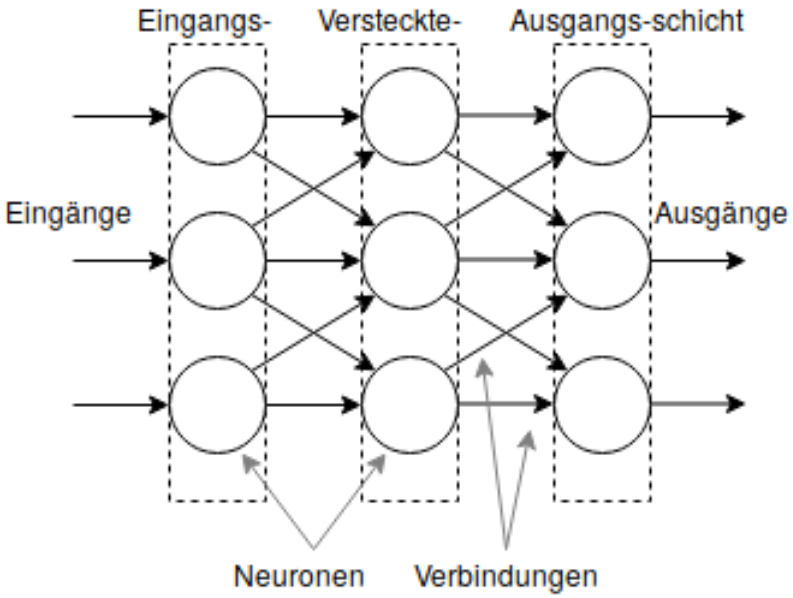


Abbildung 2: Künstliches Neuronales Netz mit drei Schichten je drei Neuronen

2.1.3 Verlustfunktion

Die Verlustfunktion(engl. Lossfunction) stellt ein ausgesuchtes Maß der Diskrepanz zwischen den beobachteten und den vorhergesagten Daten dar. Sie bestimmt somit die Leistungsfähigkeit des Künstlichen Neuronalen Netzes während des Trainings. Beim Rückgabewert wird von dem Fehler (engl. loss) gesprochen.

2.1.4 Fehlerrückführung

Fehlerrückführung (engl. Backpropagation) ist ein Optimierungsalgorithmus welcher auf dem Gradientenabstieg bassiert. Dies ist nur möglich da ein KNN aus verkettenen differzierbaren Gewichten aufgebaut ist, somit kann die Kettenregel angewendet werden. Ziel ist es den Fehler der Verlustfunktion zu minimieren. Hierfür wird ein Muster am Eingang des Netzes angelegt. Der Ausgangswert wird mit dem Soll-Ergebnis verglichen. Der aufsummierte, Fehler wird Schicht-weise von der Ausgansschicht zur Eingangsschicht zurück propagiert. Währenddessen werden die Gewichte so geändert, dass der Fehler minimiert wird. Es gibt verschiedene Ansätze der Fehlerrückführung, welche auch Optimier genannt werden. Diese bestimmen die Regeln, wie die Verlustfunktion zur Aktualisierung der Parameter verwendet wird. [Cho18, p. 83]

2.1.5 Hyperparameter

Als Hyperparameter werden, in Bezug auf künstliche neuronale Netze, meist die Anfangsparameter bezeichnet, welche zum Traininieren eines Netzes Essenziell sind. Für diese gelten keine universellen Werte, sondern müssen je nach Daten und Model speziell angepasst und verändert werden. Infolge dessen sind nur einige Regeln und grobe Abschätzungen bekannt, in welchen Grenzen sich die Hyperparameter befinden. Zu den Hyperparameter welche in dieser Arbeit verwendet werden gehören folgende:

- **Lernrate** Die Lernrate (engl. Learning rate) ist das Maß für die Schrittgröße beim Gradientenabstieg. Eine zu kleine Lernrate kann zu einem sehr langsamen Training führen. Eine zu große Lernrate kann dazu führen, dass der Wert um ein Minimum oszilliert. Dies ist grafisch in Abbildung 3 dargestellt. [Fra17, p. 49]

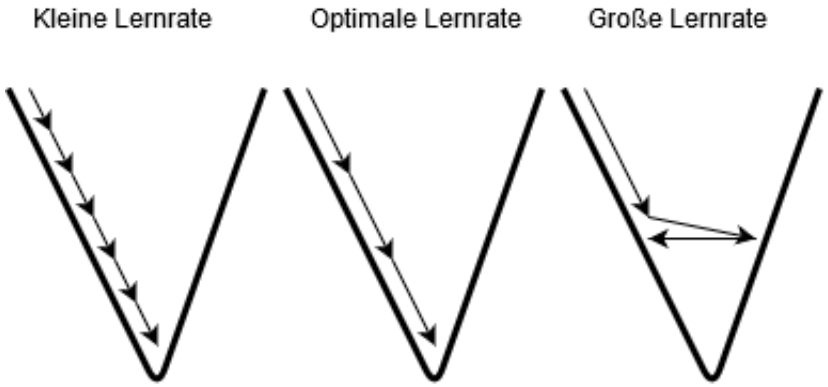


Abbildung 3: Links zu kleine Lernrate, mitte optimale Lernrate, rechts zu große Lernrate

- **Der Dropout** gibt an, wie viel Prozent einer Schicht während des Trainings deaktiviert wird. deaktiviert heißt sie werden ignoriert und für diese Iteration ausgesetzt. Symbolisch zu sehen in Abbildung 4. Durch das deaktivieren werden zufällig Neuronen "Null"gesetzt, dadurch muss das Netz diese deaktivierten Neuronen durch andere Neuronen ersetzen, wodurch die Informationen, welche in den Neuronen gespeichert ist, besser verteilt werden. Dropout wird nur an den versteckten Schichten durchgeführt. Dies kann zu einem verbesserten und robusteren Ergebnis führen.[Fra17, p. 109]

- **Die Durchläufe**(engl. Epochen) geben an, wie viele Iterationen aller Trainingsdaten durchgeführt werden sollen. Zu langes Training kann zu einem Auswendig lernen der Daten führen dementsprechend wird nicht mehr richtig generalisiert. Bei einer zu geringen Epochen Anzahl kann das Netz nicht genügend Informationen lernen, das Ergebnis ist dann entsprechend schlecht. [Fra17, p. 109]

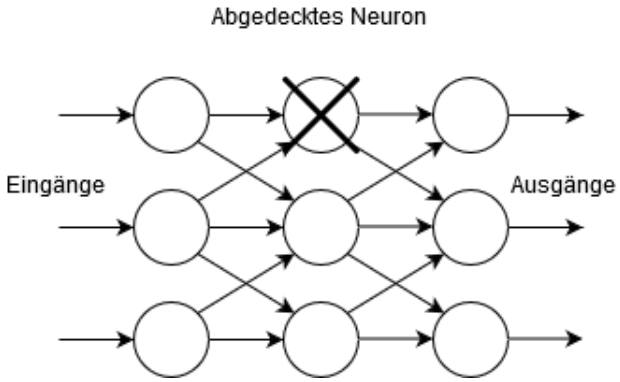


Abbildung 4: Abdeckungsgrad entspricht einem Drittel

- **Die Stapelgröße**(engl. Batchsize) gibt an, in welchen Paketen die Daten dem künstlichen neuronalen Netz zugeführt werden. Künstliche Neuronale Netze verarbeiten die Daten nicht auf einmal, sondern teilen sie in sogenannte Stapel ein. [Fra17, p. 109]

- **Der Optimierer** (engl. Optimizer) gibt vor, wie sich der Fehler auf die Aktualisierung der Parameter auswirkt. So gibt es Beispielsweise einen stochastischer Gradientenabstieg mit Momentum Optimierer (engl. Stochastic Gradient Descent with momentum) der die Trägheit des Gradientenabstiegsverfahren berücksichtige und leichter lokale Minima über geht[Fra17, p. 109]. Die Erklärung aller Optimierer ist nicht teil der Arbeit, es wird daher auf die Arbeit [Rud16] verwiesen, in welcher die wichtigsten Optimierer ausführlich erklärt werden.

- **Die Modell-Achitektur** definiert wie ein Netz aufgebaut ist. Hier werden die Anzahl der Schichten und Neuronen festgelegt. Ein zu kleines Netz kann nicht alle Informationen lernen die in den Daten vorhanden sind. Dies zeigt sich schnell in schlechten Ergebnissen. Ein zu großes Netz mit zu vielen Schichten und Neuronen kann dazu führen, dass die Genauigkeit auf den Trainingsdaten erhöht aber auf den Testdaten sich verschlechtert. Zudem ist die Dauer des Trainings wesentlich länger[Fra17, p. 70].

2.2 Optimierung

Als Optimierung wird im allgemeinen die Maximierung oder Minimierung einer Zielfunktion bezeichnet, wobei eine oder mehrere Restriktionen eingehalten werden sollen. Hierbei existiert ein Lösungsraum, in dem sich sämtliche mögliche Lösungen befinden. Das Optimierungsverfahren soll einerseits den Lösungsraum definieren und diesen Lösungsraum nach der Lösung mit dem optimalen Zielfunktionswert absuchen [GLL11, p. 15]. Bei

der Hyperparameter-Optimierung handelt es sich um ein Mehrzieloptimierungsverfahren. Wodurch sich ein Mehrdimensionaler Suchraum ergibt. Eine exakte Berechnung des Optimums ist, wegen des großen Suchraums meist nicht möglich. Aus diesem Grund ist für diese Aufgabe nur eine heuristische Optimierung möglich. Bei heuristischen Verfahren wird der Lösungsraum nach einem festgelegten Muster durchsucht und bewertet. Die gefundenen Lösungen sind nicht die besten Lösungen, sind aber das Mittel/Kompromiss aus vertretbarem Rechenaufwand und hinreichend gut eingeschätzter Lösung. Trotz allem kann nicht garantiert werden, dass es sich bei der gefunden Lösung um die optimale Lösung handelt[BNR06]. Für diese heuristischen Verfahren zur Hyperparameter Optimierung werden in dieser Arbeit Zufallssuche und Genetischen Algortihmen gegenübergestellt und anhand ihrer Lösungen verglichen und bewertet.

Vereinfachtes Beispiel für Mehrzieloptimerung Angenommen es soll ein Künstliches Neuronales Netz mit k Schichten und j Neuronen zur Klassifizierung von einfachen handgeschriebenen Zahlen erstellt werden. Der Entwickler entscheidet sich für ein 3 Schichten Netz mit jeweils 3 Neuronen. Nach dem Training hat es die Genauigkeit von 85 Prozent. Nun kann der Programmier/Entwickler nicht sicher sagen, ob für k = 3 und j = 3 die optimale Lösung gefunden wurde. Um dies beurteilen zu können, müssen viele Experimente durchgeführt werden. Die Frage ist, wie die besten Werte für k und j zu finden sind um die Klassifizierungsgenauigkeit zu maximieren? Dieser Vorgang, speziell im Zusammenhang mit künstlichen Neuronalen Netzen, wird als Hyperparameter-Optimierung bezeichnet. Bei der Optimierung wird mit einem Initialwert gestartet. Dieser ist in den seltensten Fällen die exakte Lösung. Dieser Initialwert muss einige Male verändert werden, um auf ein Optimum zu gelangen. In diese Arbeit ist dafür die Zufallsuche bzw. der Genetische Algorithmus zuständig[Gad18, p. 130].

2.3 Rastersuche und Zufallssuche

Rastersuche(engl. Grid Search) ist einer der am weiterverbreiteten Optimierungalgorithmen. Der Suchraum wird durch ein Raster aufgeteilt, je höher die Abtastrate desto feiner ist das Raster. Die Überschneidungen des Gitters sind im Endeffekt die einzelnen Punkte welche getestet werden. Während des Suchvorgangs wird das Raster nicht verändert. Somit kann es passieren, dass die optimalsten Parameter nicht gefunden werden, zudem ist die Suche sehr langsam bzw. rechenaufwendig. Um eine schnellere Berechnung von Hyperparametern zu garantieren wurde von James Bergtra und Yoshua Bengio die Zufallssuche untersucht. Sie konnten statistisch feststellen, dass die Zufallssuche schnellere zu bessere Ergebnisse liefert. [BB12] Wie in Abbildung 5 zu sehen ist die Abdeckung des Suchraums durch die Zufallssuche, rechten Seite, wesentlich besser als die Rastersuche auf der linken Seite. Dies ist durch ein ungleiches zufälliges Raster möglich.

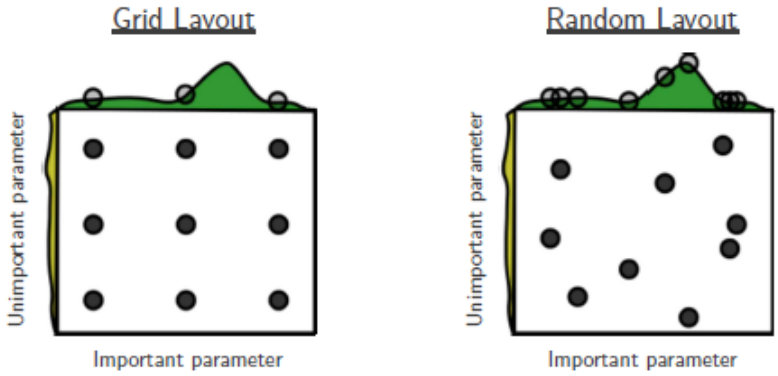


Abbildung 5: Linke Seite: veraltete "Grid Search", Rechte Seite: "Random Search"[BB12]

2.4 Genetische Algorithmen

Im folgenden Kapitel werden auf die 5 Schritte des Genetischen Algorithmus 1 näher eingegangen. Zuerst wird auf Aufbau und die Initialisierung einer Population eingegangen. Anschließend folgt eine Zusammenfassung der Fitnessfunktion. Darauf folgend werden verschiedene Möglichkeiten für die Eltern Selektion dargelegt. Anschließend wird die Vermehrung durch Crossover und Mutation erklärt. Zum Schluss wird auf den Austausch der Populationen eingegangen.

Genetische Algorithmen sind heuristische Suchansätze. Im Wesentlichen zeichnet sie eine probabilistische Eltern Selektion als primären Suchoperator aus. Als weitern Opterator wird auf die Mutation zurückgegriffen. Dieser garantiert eine Erreichbarkeit aller Punkte im Suchraum und erhält die Grunddiversität in der Population. Es gibt zwei verschiedene Algorithmen, der Standard-GA tauscht nach einer Generation die komplette Elternpopulation durch die Kinderpopulation aus. Dieser bestehen in der Regel immer aus den gleichen fünf gleichen Schritten: Schritt 1, Initialisieren der Anfangspopulation. Schritt 2, Fitness berechnen mit Hilfe der Fitnessfunktion. Schritt 3, Selektieren der Eltern. Schritt 4, Vermehren durch Crossover und Mutation. Schritt 5, Austausch der Populationen. Diese Schritte sind in Abbildung 6 noch einmal zum leichterem verständnisse als Flussdiagram dargestellt. Im Gegensatz dazu steht der Steady-State-GA, welcher sich durch eine überlappende Population auszeichnet. Das heißt pro Generation wird nur das schlechteste Individuum durch ein neues Individuum ausgetauscht. Der Steady-State-GA wird in der Arbeit nicht verwendet und wird deswegen nicht weiter erklärt[Wei15, p. 128] [Kra17, p. 11].

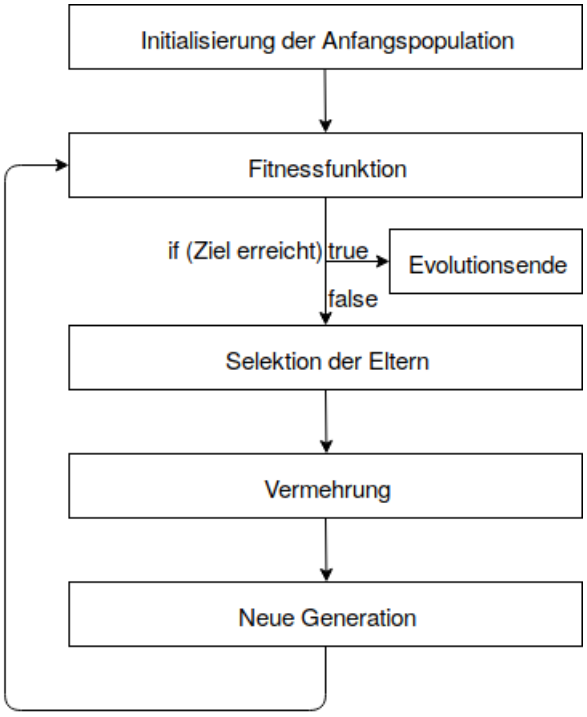


Abbildung 6: Ablaufdiagramm eines genetischen Algorithmus mit 5 Schritten

Algorithm 1 Genetischer Algorithmus

Initialisieren der Anfangspopulation

while $Fitness \leq Abbruchbedingung$ **do**

Fitness aller Individuen berechnen

Selektieren der Eltern

Vermehren durch Crossover und Mutation

Austausch der Populationen

2.4.1 Aufbau und Initialisierung der Population

Der klassische genetische Algorithmus basiert auf einer Reihe von Kandidatenlösungen. Die Größe der Population ist somit auch die Anzahl der Lösungen. Jede Lösung kann als einzelnes Individuum gesehen werden und wird durch einen Chromosomenstrang repräsentiert. Ein Chromosom besteht wiederum aus beliebig vielen Genen, wobei im vorliegenden Problem jedes Gen einen Hyperparameter repräsentiert. Der Aufbau ist grafisch

in Abbildung 7 dargestellt. Um die Grundlagen nahe des später folgenden Konzepts zuhalten, wird der Ablauf per Dezimal-Genen erklärt[Gad18, p. 134].

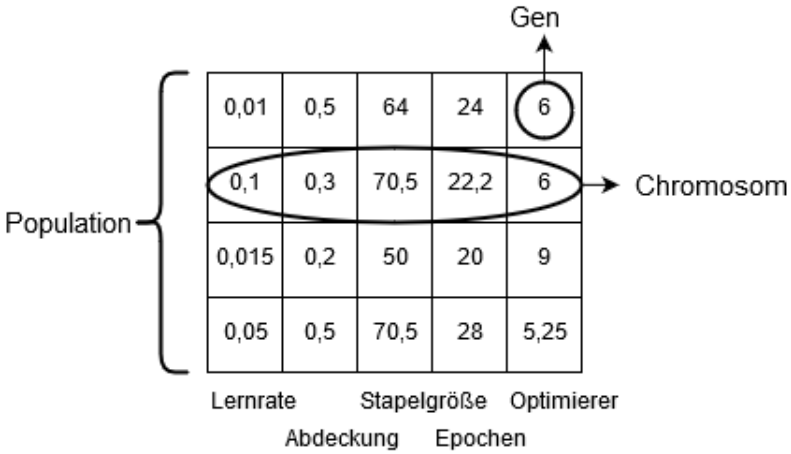


Abbildung 7: Beispiel einer Population mit 4 Induviduen(Chromsomen) mit 5 dezimalen Genen

Diese Anfangspopulation (Generation 0) wird zufällig initialisiert, um die größtmögliche Abdeckung des Suchraums zu gewähren. Die erste Generation besitzt eine sehr geringe Fitness, welche sich im verlauf des Trainings stetig steigert bis sie das Maximum erreicht.

2.4.2 Fitnessfunktion

Die Fitnessfunktion (engl. Fitnessfunction) bewertet das Individuum anhand seiner Funktionstauglichkeit, bezogen auf die vorhandene Aufgabe. Dabei wird das Chromosom (Idividuum) als ganzes Bewertet, somit wird keine Aussage über einzelne Gene getroffen. Die Fitnessfunktion muss für jede Anwendung speziell geschrieben werden. Der Rückgabewert der Fitnessfunktion entspricht einer reellen Zahl. Ein höherer Fitnesswert steht stehst für eine höher Qualität des Individuums, sprich für eine bessere Lösung[Gad18, p. 135].

2.4.3 Selektion der Eltern

Bei dem Schritt Selektion (engl. Select Parents) geht es darum einen Elternpool zu generieren, aus welchem die neue Generation erstellt wird. Deshalb ist es wichtig, nur die

qualitativ hochwertigsten Individuen auszuwählen. Es gibt verschiedene Ansätze bei der Selektion, die elementar wichtigsten werden genannt und erläutert. Informationen zu den Selektionsmethoden wurden aus der Arbeit [SPM15] [ES+03, p. 80]

- **Auswahl proportional zur Fitness (engl. Fitness Proportionate Selection(FPS))** Die Eltern werden proportional nach ihrer Fitness ausgewählt und zum Elternpool hinzugefügt. $f(a_i)$ ist die Fitness des Individuums a_i in der Population. $ps(a_i)$ ist die Wahrscheinlichkeit des Individuums a_i als Elternteil ausgewählt zu werden.

$$ps(a_i) = \frac{f(a_i)}{\sum_{j=1}^{n} f(a_j)}; i \in 1, 2, ..., n \tag{7}$$

  wobei n die Anzahl der Individuen einer Population ist. Diese Wahrscheinlichkeit ps wird als Anteil auf einem Rouletterad, wie Abblidung 8 zeigt, umgesetzt. Auf dem zufällig die Eltern aus den Individuen $a1, .., an$ „ausgedreht "werden. Dieser Ansatz hat leider das Problem, dass Individuen welche sich Anfang sich als gut beweisen, schnell die ganze Population übernehmmen. Das kann dazu führen, dass eine mögliche bessere Lösung durch den Algorithmus im Suchraum nicht gefunden wird.

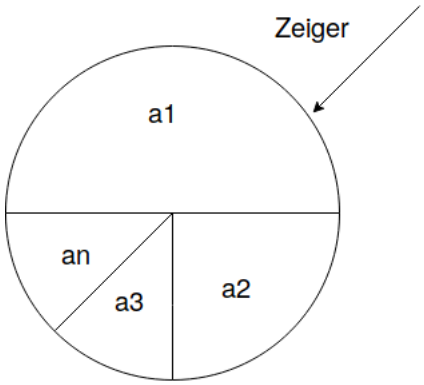


Abbildung 8: Rouletterad mit proportinalem Anteil der Individuen anhand ihrere Fitness

- **Ranking Selektion** Diese Selektion wurde von Backer als Verbesserung der Fitness Proportonal Selection entwickelt [Bak85]. Dabei werden die Eltern nicht direkt nach ihrer Fitness ausgewählt. Die Fitness dient nur zum Einteilen in eine Rangliste. Anhand dieser Rangliste wird nun die Verteilung auf dem Roulettrad berechnet. Dazu gibt es zwei Ansätze:

  Das lineare Ranking:

$$p_i = \frac{1}{N}(n^- + (n^+ - n^-)\frac{i-1}{N-1}; i \in 1, ..., N \tag{8}$$

  Wobei p_i die Wahrscheinlichkeit des Individuums ist selektiert zu werden. $\frac{n^-}{N}$ ist

die Wahrscheinlichkeit des schlechtesten Individuums selektiert zu werden und $\frac{n^+}{N}$ ist die Wahrscheinlichkeit des besten Individuums selektiert zu werden.

Das expontienelle Ranking:

$$p_i = \frac{c^{N-i}}{\sum_{j=1}^{N} c^{N-j}}; i \in 1, ..., N \tag{9}$$

die Summe $\sum_{j=1}^{N} c^{N-j}$ normalisiert die Wahrscheilichkeit um Sicherzustellen, dass $\sum_{i=1}^{N} p_i = 1$. Wobei die Berechnungen 8 und 9 nur den Anteil eines Individuums auf dem Rouletterad verändern.

- **Turnier Selektion** In diesem Verfahren, welches in Abbildung 9 zusehen ist, werden zufällig k Individuen der Population ausgewählt. Diese k Individuen treten wie in einem Turnier gegeneinander an. Der Gewinner ist das Individuum mit dem besten Fitnesswert, dieser wird dann auch als Elternteil ausgewählt. Hierbei wird auf den Elternpool verzichtet und direkt ein Kind aus zwei Gewinnern erstellt. Eingesetzt wird dies bei kleineren Populationen mit weniger als 20 Individuen.

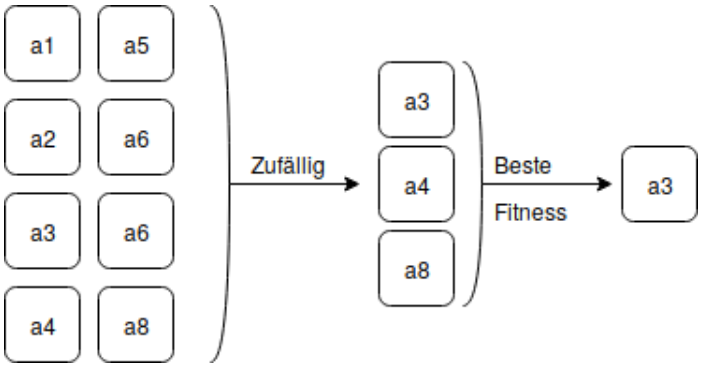


Abbildung 9: Turnier Selektion mit k = 3 Individuen und dem Gewinner Individuum 3

2.4.4 Vermehrung

Aus dem Elternpool werden nun Nachkommen (neue Individuen) geschaffen. Alleine durch die Paarung(engl. Crossover) von qualitativ hochwertigen Individuen wird erwartet, dass die Nachkommen eine bessere Qualität besitzen als die ihrer Eltern. Als zweite Verbesserung wird noch die Mutation einzelner Gene angewendet. In diesem Abschnitt werden die verschiedenen Ansätze für Crossover und Mutation näher erklärt.

Crossover Hierbei werden aus zwei Eltern Chromosomen, die Chromosomen der Kinder Zusammengesetzt. Information zu Crossover wurden aus der Arbeit [US15] entnommen. Die bedeutendsten Cossovervarianten sind:

- **One-Point-Crossover** Bei Ein-Punkt-Crossover wird zufällig ein Punkt im Chromsomenstrang festgelegt wird. Ab diesem Punkt wird der Chromosomenstrang dann aufgeteilt und anschließend mit dem Crossover des anderen Elternteils wieder zusammengesetzt. Ein einfaches Beispiel ist im oberen Teil der Abbildung 10 zu sehen.

- **k-Point-Crossover** Eine Abwandlung des Ein-Punkt-Crossover ist das Zwei-Punkt-Crossover oder k-Punkt-Crossover. Hier wird der Chromsomenstrang an k Punkten aufgeteilt und anschließend mit dem Anteil des zweiten Elternteil wieder zusammengesetzt. In mittleren Teil der Abbildung 10 ist ein k = 2 Crossover oder auch zwei-punkt-crossover (engl. two-point-crossover) zu sehen.

- **Uniform-Crossover** Eine weitere grundlegende Operation beim Crossover ist die Uniform-crossover [Sys89]. Hier wird für jedes Gen zufällig entschieden aus welchem Elternteil das Gen entnommen wird. Dies ist im unteren Teil der Abbildung 10 veranschaulicht.

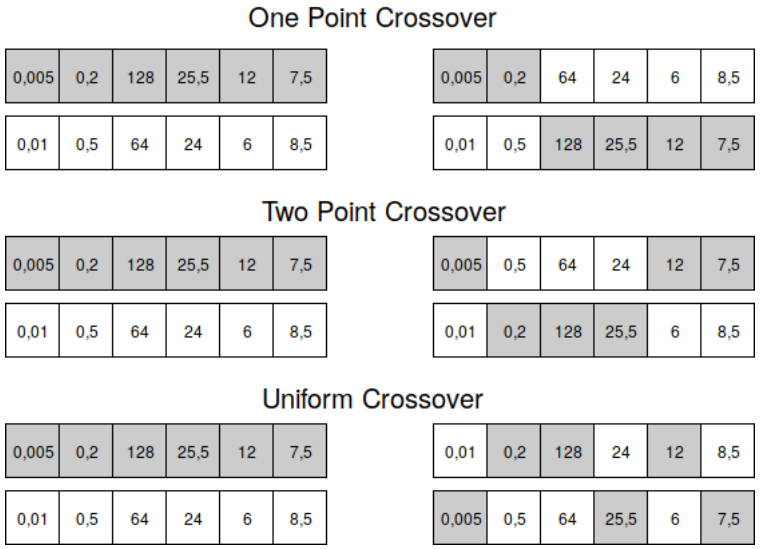


Abbildung 10: Die Wichtigsten drei Crossover Operationen. Oben die one-point Crossover, mitte two-point Crossover, unten Uniform Crossover. Auf der linken Seite sind Eltern abgebildet und auf der Rechtenseite die neu erzeugten Kinder

Mutation Hierbei wird jedes Gen des Individuums zufällig mit einer zufälligen Mutation versehen. Zum einen gibt es eine einstellbare Wahrscheinlichkeit ob ein Gen verändert werden soll und zum anderen wie groß die Mutation sein soll. Durch diese Mutation wird eine höhere Diversität in die nachfolgende Generation übergeben. Diese Mutation macht es möglich einen größeren Suchraum abzudecken und somit die Werte genauer anzupassen, um so auf die optimale Lösung zu kommen. Es wird ein Zufallswert aus der Normal-

oder Gauß-verteilung zum Gen hinzu addiert. Durch diese Wahrscheinlichkeitsverteilung erhalten viele Gene nur kleine Veränderungen, aber auch größere Muationen sind möglich. Um die vorher mit Crossover neu bestimmten Individuen, welche schon eine hohe Grundfitness besitzen, nicht zu stark zu verändern, wird ein Gen nur um wenige Prozent verändert. Dies ist in Abbildung 11 zu sehen[Wei15, p. 60] [GLL11, p. 140].

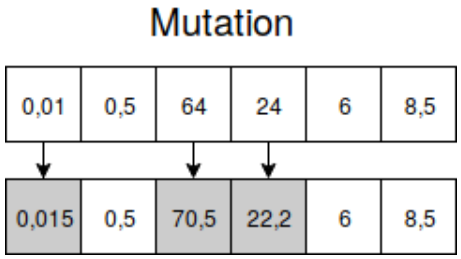


Abbildung 11: Beispielhafte Mutation. Die Mutationen wurden zufällig gewählt.

2.4.5 Neue Generation

Der letzte Schritt des Genetischen Algorithmus besteht aus dem Austausch der Generationen. Die neue Kind Generation tauscht nun die alte Eltern Generation aus. Anschließend folgen die gleichen 4 Schritte so lange bis der gewünschte Fitnesswert erreicht ist. Nach dem erreichen der Abbruchbedingung, kann aus der letzten Generation das qualitativ hochwertigste Individuum ausgesucht und als Lösung verwendet werden.

2.5 Zusammenfassung

3 Stand der Technik und Forschung

Durch die gestiegene Rechenleistung der heutigen Computer ist auch die Anwendungshäufigkeit von Genetischen Algorithmen deutlich gestiegen. Es werden nicht nur neue Ansätze erforscht, sondern auch erste Anwendungen entwickelt. Diese finden sich nicht nur im IT-Bereich. Der erste Abschnitt befasst sich mit Firmen und ihren Forschungsergebnissen im Bereich der Genetischen Algorithmen. Im zweiten Abschnitt wird auf die Anwendungen zu Genetischen Algorithmen eingegangen.

3.1 Stand der Forschung

3.1.1 Deep Mind

Den größten Fortschritt der letzen Jahre verzeichnete Googles Deepmind. Deepmind setzt dabei auch auf einen Populations basierenden Algorithmus, welcher den Genetischen Algorithmen ähnelt. Das PopulationBasedTraining [JDO+17] wird vor allem im Bereich der Künstlichen Intelligenz für Brett- und Computerspiele erforscht. So konnten sie bei verschiedene Künstlichen Neuronalen Netzen wie AlphaGo(Go) [SHM+16] und AlphaStar(Starcraft2) [ACT19] deutliche Verbesserungen erreichen. Das PBT wird aber nicht nur für Spiele genutzt, sondern auch für Anwendungen im Bereich des autonomen Fahrens. Dort wurde in Zusammenarbeit mit der Google-Tochter Waymo eine Verbesserung der Zuverlässigkeit ihrer Künstlichen Neuronalen Netze ausgearbeitet. Sie konnten eine Reduzierung um 24% der falsch positiven Ergebnisse gegenüber den von hand-modifizierten Netzen erreichen. Mithilfe der neu berechneten Hyperparametern, wurden die alten KNN von Waymo deutlich übertroffen und dies mit nur der Hälfte an Trainings-zeit und Ressourcen. Im allgemeinen verwendet der PBT-Ansatz nur die Hälfte an Rechenleistung im Vergleich zu Random-Parallel-Search. Somit hat Google Deepmind in mehreren Fällen gezeigt das ihre PBT ein deutliche Verbesserung in der Optimierung von Hyperparametern ist. [hC19]

3.1.2 PopulationBasedTraining

Populations basiertes Training (PBT) ist eine Kombination aus Random-Search 2.3 und Genetischen Algorithmen 2.4. Der zu Grunde liegende Suchalgorithmus ist wie beim GA ein heuristische Suchansatz mit einer Kandidatenlösungsmenge. Wie beim Genetischen Algorithmus und der Zufallssuche wird mit zufällig initialisierten Hyperparametern gestartet. Es werden Methoden wie SSurvival of the Fittestünd Mutation aus dem GA übernommen. Darüber hinaus wurde ein online Anpassung der Hyperparameter während des Trainings implementiert. Das heißt schlechte Individuen (bei PBT als worker

bezeichnet) werden durch Individuen mit besserem Fitnesswert ersetzt(exploit). Sie werden nicht einfach ersetzt sondern erhalten kleine Mutationen (explore). Dieses Individum muss nicht von beginn neu trainiert werden, sondern kann auf die Gewichtungen der Eltern zurückgreifen. Dies ist gut in Abbildung 12 zusehen. Die online Anpassung ist die größte Verbesserung, es kann mit einem Pool beim Mutliprocessing verglichen werden. (Nach beenden der Berechnung der Fitness wird ein neues Individuum berechnet, ohne das dabei auf andere Individuen gewartet werden muss.) [JDO+17]

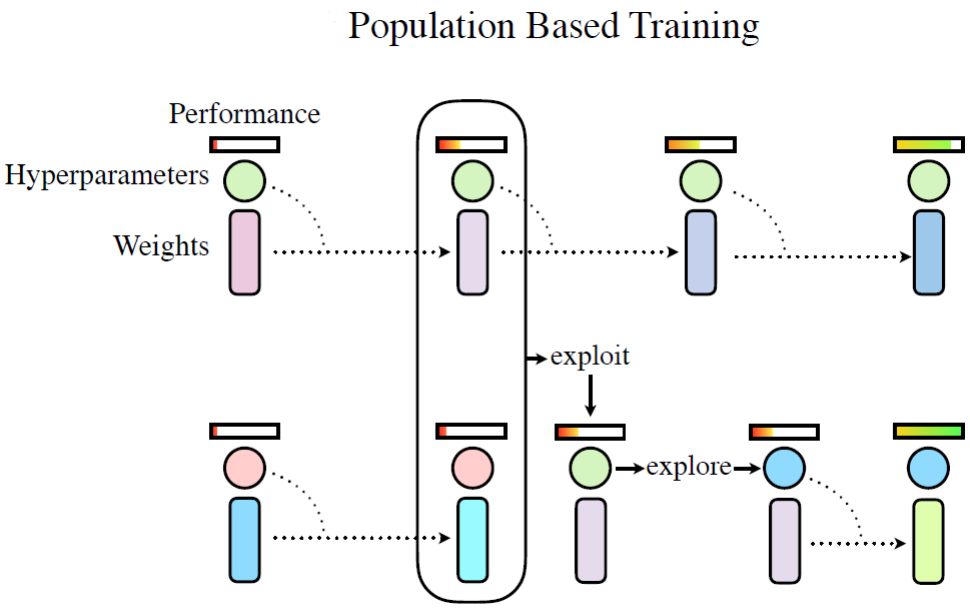


Abbildung 12: Population Based Training [JDO+17]

3.1.3 Evolving Neural Networks through Augmenting Topologies (NEAT)

Neat ist eine Optimierungsverfahren für Modelachitekturen von KNN‘s. Dabei wird auf die Topologie der Netze und Gewichte verbessert. Sie konnten mit ihrem Algorithmus zeigen das es möglich ist, Netze zu Optimieren und gleichzeitig Komplexer zu gestalten. Der Algorithmus baut auf dem Genetischen Algorithmen auf. Das Netz wird so einfach wie möglich initialisiert und dann über Generationen komplexer und qualitativ verbessert. In Abbildung 13 ist so ein Chromsomenstrang zu sehen mit den Verbindungs-Gene (engl. Connection Genes) aus welchen die Neruronen-Gene (engl. Node Genes) erstellt werden können. Unterhalb ist das künstliche neuronale Netz zu sehen. Über die Mutation können einzelne Verbindungen(Gewichte) hinzugefügt werden, zusehen in Abbildung 14. Somit wird das Netz Komplexer. Durch hinzufügen von Gewichten wächst die Län-

ge des Chomosomestrangs. Es werden zufällig auch einzelne Verbindungen ausgeschaltet um ihre Nützlichkeit zu überprüfen, sie können zu späteren Zeitpunkten zufällig wieder aktiviert werden. Dies ist in Abbildung 14 im Chromosomstrang zusehen und als DISAB maktieret. Für die Eltern-Auswahl werden die Netze mit ähnlicher Struktur in sogenannte Spezies eingeteilt. Nun wird von jeder Spezies die schlechte Hälfte gelöscht. Damit ist die Eltern-Selektion nicht direkt von der Fitness abhängig und gibt schlechteren Individuen eine Überlebenschance und somit auch die Möglichkeit sich weiterzuentwickeln. Crossover funktioniert bei Neat ähnlich wie beim klassischen Genetischen Algorithmus, der unterschied liegt darin das die Gene addiert werden. Heißt Verbindungen die nur in einem Individuum vorkommen werden an das Nachkommen übergeben. Wenn ein Gen Ausschalten ist wird es im Nachkommen mit hoher Wahrscheinlichkeit auch Ausgeschaltet. Da nun die nachkommen gebildet wurden kann aus diesen eine neue Generation gebildet werden und der Optimierungsprozess beginnt von neuem. [SM02]

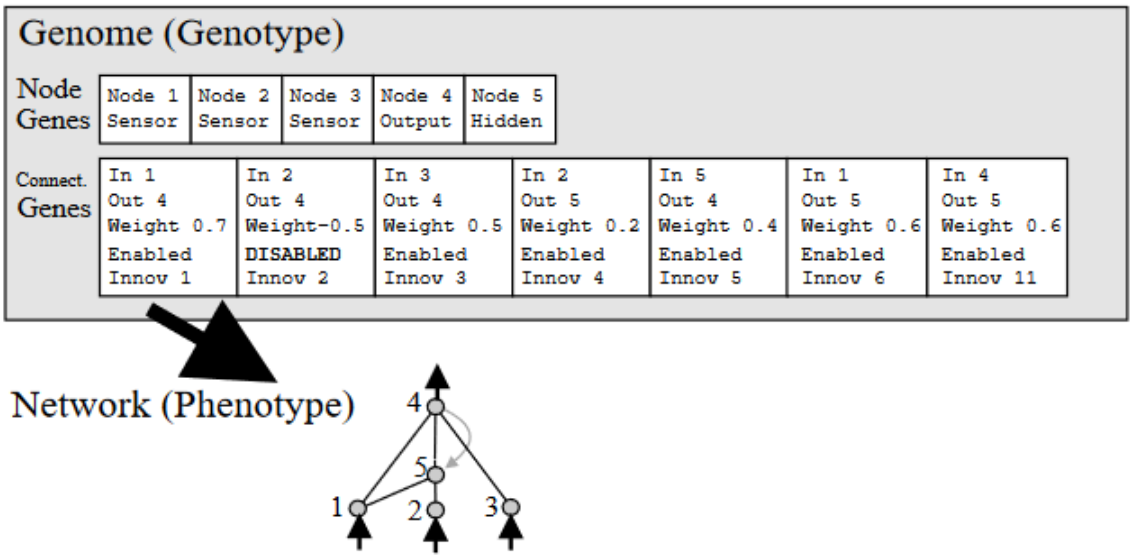


Abbildung 13: Aufbau eines Chromosomes bei Neat [SM02]

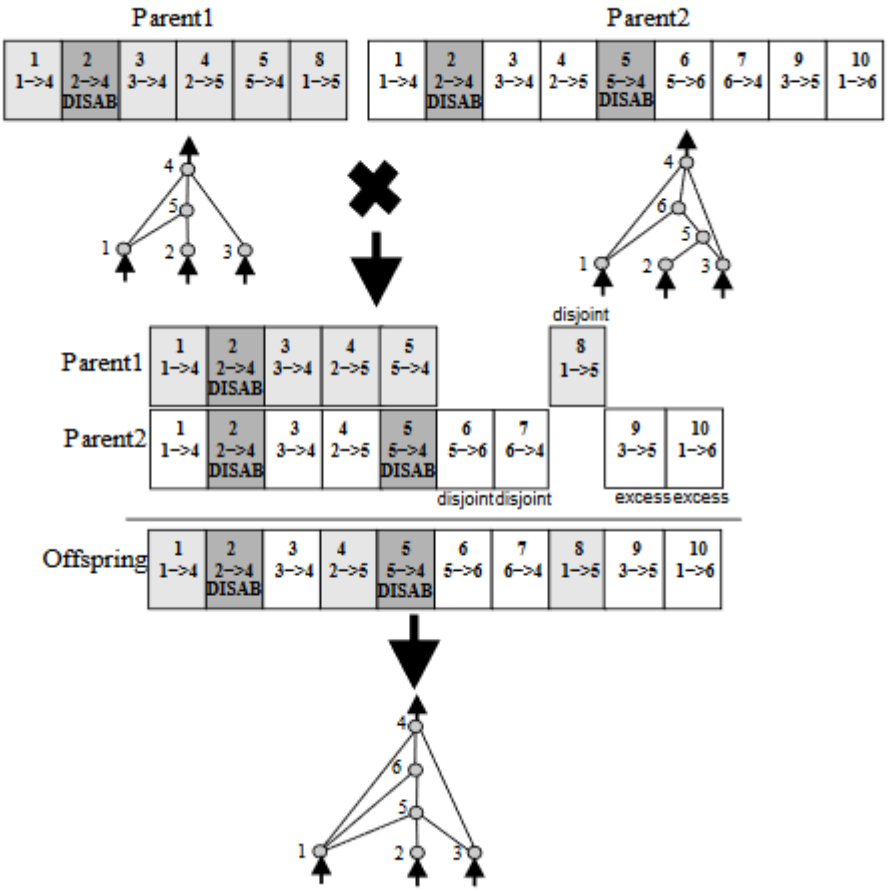


Abbildung 14: Crossover bei Neat [SM02]

3.2 Stand der Technik

3.2.1 Travelling Salesman Problem

Einer der bekanntesten Anwendungen für GA ist das Travelling Salesman Problem, in welchem die kürzeste Route beim Austragen von Briefen für einen Postboten berechnet werden soll. Es ist ein gute Anwendung für GA, da durch die zurückgelegte Steckte eine leichte eine Fitnessfunktion erstellt werden kann. Des weiteren wird der Suchbereich mit jedem auszutragenden Brief um einen Exponenten größer. Und ist somit für gängige Algorithmen zu komplex.

3.2.2 Software Testing mit GA

Die Softwarebewertung spielt eine entscheidende Rolle im Lebenszyklus eines Software-Produktionssystems. Die Erzeugung geeigneter Daten zum Testen des Verhaltens der Software ist Gegenstand vieler Forschungen im Software-Engineering. In diesem Beitrag wird die Qualitätskontrolle mit Kriterien zur Abdeckung von Anwendungspfad betrachtet und ein neues Verfahren auf der Grundlage eines genetischen Algorithmus zur Erzeugung optimaler Testdaten entwickelt. [KJ11]

3.2.3 Temperatur Schätzungen

Es gibt Forschungen zur Berechnung von Temperaturverläufen. Als Eingangsdaten werden verschiedene Daten benutzt wie beispielsweise: Sonnenaktivität, Vulkanausbrüche und Greenhousegasanteil in der Atmosphäre. Als Ausgabe erhalten wir dann eine Abschätzung der Temperatur für die nächsten Jahre. Vom gleichen Autor gibt es auch eine Abschätzungen von Wärmeflusses zwischen Atmosphäre und Meereis in Polarregionen [SKV15]. All diese Vorhersagen wurden mithilfe von Genetischen Programmierung berechnet. Als Trainingstag wurden Aufzeichnungen von Temperaturverläufen und andren Eingangsdaten der letzten Jahrhunderten verwendet.

3.2.4 Generatives Design

Heute gibt es in viele rechnerunterstütztes Konstruieren Programme (eng. Computer-aided design - CAD) Implementierungen von Generativen Design Werkzeugen. Diese berechnen über Iterationen neue mögliche Designs, die Berechnungen Basieren auf der Genetischen Algorithmen. Mit ihnen ist es möglich Bionische Strukturen zu designen. So kann aus einem Bauteil, ein wesentlich Leichtes und Material spaarenderes Model entwickelt werden. In Abbildung 15 sind verschiedene Schritte des Generativen Design zusehen. Die Fertigung ist meist nur mit additiven Fertigungsmethoden möglich, da die berechneten Strukturen keinen Fertigungsregeln unterliegen. [Cal08]

Abbildung 15: Additives Design Beispiel von Siemens [sie19]

NASA - Antenne Die Nasa hat 2006 eine Weltraumantenne mit Hilfe evolutionären Algorithmen entwickelt. In Abbildung 16 sind zwei Prototypen der Weltraumantenne. Da die Entwicklung der Antenne von Hand zu zeitaufwändig und arbeitsintensiv wäre. Zusätzlich braucht es großes Wissen in der entsprechenden Domain. Weshalb die Forscher einen Population bezogenen Suchansatz nutzten, um Umgebungsstrukturen und elektromagnetische Auswirkungen mit einzubeziehen. Die entwickelte Antenne wurde produziert und auch auf Space Technology 5 mission genutzt. [HGLL]

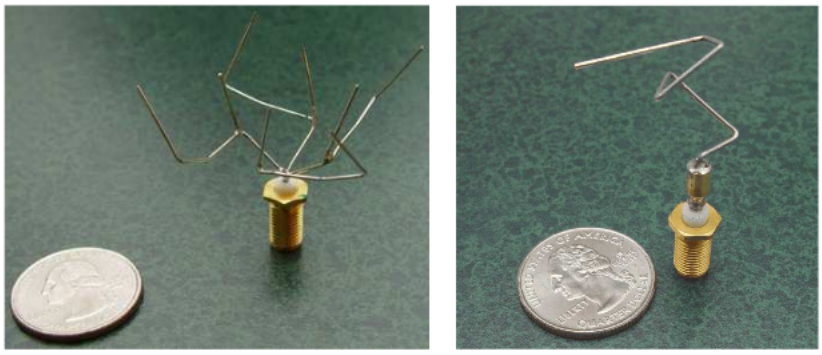


Abbildung 16: Foto der Prototypen [HGLL]

3.3 Zusammenfassung

4 Konzept

Im folgenden Kapitel wird auf die Ideen und Konzepte zu Optimierung mittels künstlicher neuronaler Netze eingegangen. Im Speziellen wir auf das Konzept des Genetischen Algorithmusses eingegangen.

Dazu wird zuerst das grobe Konzept des Genetischen Algorithmus erklärt. Anschließend wird auf die verwendeten Methoden des Genetische Algorithmus eingegangen. Anschließend wir auf die Optimierungskonzepte für verschiedene Netze eingegangen.

4.1 Anforderungsanalyse

Keine Ahnung was hier stehen soll???

4.2 Genetischer Algorithmus

Die Idee ist es als Hyperparamter als Gene einzusetzen. Heißt in Individuum steht für ein Netz mit verschiedenen Hyperparamter. Dies ist in Abbildung 7 zusehen. Diese sollen anschließend durch mehrere Generationen/Iterationen verbessert werden. Somit sollten die besten Hyperparameter gefunden werden. Es soll auch geprüft werden ob der Genetische Algorithmus zur Model-Achitektur eingesetzt werden kann.

In diesem Kapitel wird auf die Konzepte der einzelnen Schritte des Genetischen Algorithmus eingegangen.

4.2.1 Initialisierung der Anfangspopulation

Da es für die Hyperparameter schon grobe vorgaben gibt. Wird die Anfangspopulation nicht mit Zufälligen Hyperparameter Initialisiert, sondern zufälig aus einem für jeden Hyperparameter speziellen Rahmen ausgewählt. Dies verkleiner zwar den Suchraum, die Grenzen werden dennoch groß genug gewählt, dass alle sinnvollen Lösungen gefunden werden können. Dieses Initialisieren von Hyperparameter in einem bestimmten Rahmen reduziert den Lösungsraum und damit auch die Berechnungszeit. Die Bestimmung der Anzahl an Individuen und Generationen wurde anhand Experimenten durchgeführt.

4.2.2 Fitnessfunktion

Die Fitnessfunktion ist der Genauigkeit der Klassifikation gleich zusetzen. Das heißt es soll versucht werden die Genauigkeit zu Maximieren, wobei das Maximum der Genauigkeit bei 1 liegt. Dazu werden mit den Gene eines Individuums ein künstliches neuronales Netz initialisiert. Es wird mit diesen Hyperparamtern Trainiert und anschließend Evaluiert. Die Genauigkeit auf dem Testset fungiert als Rückgabewert, also der Fitness. Im spätern Kapitel **??** (Konzept für Speziele Fitnessfunktion) Soll versuche gestartet werden in welchen die Fitnessfuktion nicht nur von der klassifizierungsgenauigkeit abhängt. Und somit Model-Architekturen für spezielle Anwendungen wie, schnelle Performance oder schnelles Training, gefunden werden können.

4.2.3 Eltern Selektion

Bei der Eltern Selektion wird die Fitness proportional Selektion gegenüber der Ranking Selektion getestet werden.

4.2.4 Vermehrung

Für **Crossover** wird Zwei-Punkt-Crossover gegen über der Uniform-Crossover verglichen. Als **Muation** soll die Gaus-Mutation mit experimentell bestimmten Sigma verwendet werden. Damit sollte ein ausreichende Mutation vorhanden sein damit neue Lösungen gefunden werden können, diese aber nicht zu sehr verändert werden das die Ergebnisse sich verschlechtern.

4.2.5 Neue Generation

Zur neuen Generation werden die besten 2 Individuen der altern Generation ohne Mutation übernommen, somit werden die besten Lösungen nicht gelöscht. Die Restlichen 48 werden über die genannten Methoden ausgewählt. Die neue Generation durchläuft nun die gleichen Schritte.

4.3 Evaluierung des Genetische Algorithmus

Um den Genetischen Algorithmus zu evaluieren werden die Ergebnisse mit den Ergebnissen der Zufalls-suche gegenüber gestellt. Zur Findung von Methoden des Genetischen Al-

gorithmus werden Experimente mit einem kleinen Datensatzes und einem kleinen künstlichen Neuronalen Netz für Bildklassifizierung durchgeführt. Die Zielführenden Methoden werden dann zur Optimierung von verschieden neuronalen Netzen eingesetzt.

4.3.1 Optimierung verschiedener künstlicher neuronaler Netzen

In dieser Arbeit geht es um die Optimerung von künstlichen neuronalen Netzen. Um eine Aussagekräfiges Ergebniss zuerhalten wird der Genetische Algortihmus zur optimierung unterschiedlichen Netzen mit unterschiedlichen Aufgaben eingesetzt. Folgende Optimierungen sollen stattfinden:

Hyperparamet Optimierung - Klassifizieren von Bilder Es soll die Hyperparamter für ein Fully-Connectet-Netzwerk zur Klassifizierung von einfachen Bildern optimiert werden. Die Daten entsprechen den gleichen Bilddaten wie zur Evaluierung der Methoden des Genetischen Algorithmus. Doch der Datensatz ist größer(mehr Daten an sich) und die Model-Architektur ist auch größer(mehr Layer und Neuronen).

Hyperparameter Optimierung - Reichweitenberechnung Des weiteren sollen die Hyperparamter für ein Fully-Connectet-Netzwerk zur Reichweitenbestimmung bei Elektroautos optimiert werden. Wodurch gezeigt werden kann, das die Optimierung auch mit anderen Daten und Netzen funktioniert.

Model-Architekut Optimerung - Klassifizieren von Bilder Die Model-Architektur wurde bei diesem von einer anderen Arbeit übernommen. Abschließend soll die Model-Architektur des Klassifizierungsnetzes verbessert werden dafür werden die Neuronen pro Schicht als Gene umgesetzt. Wodurch eine optimale Model-Achitekur gefunden werden soll. Es werden die Hyperparameter des ersten Versuchs verwendet.

Hyperparamter Optimierung - kleinem Datenset/ wenig Trainigsdaten Diese optimierte Model-Achitektur wird dann für eine weiter Optimeriung verwendet. Hierbei ist das Ziel herauszufinden ob es möglich ist bei geringen Trainingsdaten durch Hyperparametertuning das Ergebnisse zu verbessern. Da in der Realität oft nur kleine Datensätze vorliegen.

4.4 Zusammenfassung

5 Implementierung

In diesem Kapitel werden die Implementierungen der Verfahren dieser Arbeit erläutert. Es wird der Genetische Algorithmus mit Pseudocode veranschaulicht.

5.1 Systemaufbau

5.2 Zusammenfassung

6 Evaluation und Tests

6.1 Einleitung

6.2 Testszenarien

6.3 Evaluation

6.4 Ergebniss und Interpretation

6.5 Zusammenfassung

7 Zusammenfassung und Ausblick

7.1 Einleitung

7.2 Zusammenfassung

7.3 Bedeutung der Arbeit

7.4 Ausblick

Literatur

[ACT19] Kai Arulkumaran, Antoine Cully, and Julian Togelius. Alphastar: An evolutionary computation perspective. *CoRR*, abs/1902.01724, 2019.

[Bak85] James Edward Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and their applications*, pages 101–111. Hillsdale, New Jersey, 1985.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[BNR06] Nicola Beume, Boris Naujoks, and Günter Rudolph. Mehrkriterielle optimierung durch evolutionäre algorithmen mit s-metrik-selektion. *Universität Dortmund, Chair for Algorithm Engineering*, 2006.

[Cal08] Luisa Caldas. Generation of energy-efficient architecture solutions applying gene_arch: An evolution-based generative design system. *Advanced Engineering Informatics*, 22(1):59–70, 2008.

[Cho18] Francois Chollet. *Deep Learning mit Python und Keras - Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek.* MITP-Verlags GmbH Co. KG, Heidelberg, 1. aufl. edition, 2018.

[ES+03] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[Fra17] Chollet Francois. Deep learning with python, 2017.

[Gad18] Ahmed Fawzy Gad. *Practical Computer Vision Applications Using Deep Learning with CNNs - With Detailed Examples in Python Using TensorFlow and Kivy.* Apress, New York, 1. aufl. edition, 2018.

[GLL11] Matthias Gerdts, Frank Lempio, and Frank Lempio. *Mathematische Optimierungsverfahren des Operations Research* -. Walter de Gruyter, Berlin, 1. aufl. edition, 2011.

[hC19] Yu hsin Chen. How evolutionary selection can train more capable self-driving cars, 2019.

[HGLL] Gregory Hornby, Al Globus, Derek Linden, and Jason Lohn. *Automated Antenna Design with Evolutionary Algorithms.*

[JDO+17] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017.

[KJ11] Sina Keshavarz and Reza Javidan. Software quality control based on genetic algorithm. *International Journal of Computer Theory and Engineering*, pages 579–584, 01 2011.

[Kra17] Oliver Kramer. *Genetic algorithm essentials*, volume 679. Springer, 2017.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[SHM+16] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[sie19] siemens. Generative design, 2019.

[SKV15] Karolina Stanislawska, Krzysztof Krawiec, and Timo Vihma. Genetic programming for estimation of heat flux between the atmosphere and sea ice in polar regions. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 1279–1286, New York, NY, USA, 2015. ACM.

[SM02] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[SPM15] Anupriya Shukla, Hari Pandey, and Deepti Mehrotra. Comparative review of selection techniques in genetic algorithm. 02 2015.

[SSF+16] Ivan Nunes da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, and Silas Franco dos Reis Alves. *Artificial Neural Networks - A Practical Course*. Springer, Berlin, Heidelberg, 1st ed. 2017 edition, 2016.

[Sys89] Gilbert Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[US15] Dr. Anantkumar Umbarkar and P Sheth. Crossover operators in genetic algorithms: A review. *ICTACT Journal on Soft Computing (Volume: 6 , Issue: 1)*, 6, 10 2015.

[Wei15] Karsten Weicker. *Evolutionäre algorithmen*. Springer-Verlag, 2015.