Ruprecht-Karls-Universität Heidelberg

Fakultät für Mathematik und Informatik

Seminar: Ist künstliche Intelligenz gefährlich?

Evolutionsstrategien

als Alternative zu Bestärkendem Lernen

Author:
Daniela Schacherer
Dozent:
Prof. Dr. Köthe
Abgabedatum
10.08.2017

Inhaltsverzeichnis

1 Einleitung **1**

2 Evolutionäre Algorithmen **1**

3 Evolutionstrategien **2**
3.1 Historie . . . 2
3.2 Begriffsdefinitionen . . . 3
3.3 Grundkonzept . . . 3
3.4 Ablauf . . . 3
3.4.1 Schritt 1: Initialisierung . . . 4
3.4.2 Schritt 2: Bewertung der Lösungsalternativen . . . 4
3.4.3 Schritt 3: Reproduktion . . . 4
3.4.4 Schritt 4: Selektion . . . 5

4 Anwendung von Evolutionsstrategien im Bereich KI **6**
4.1 Evolutionsstrategien als Alternative zu Reinforcement Learning . . . 6
4.1.1 Reinforcement Learning . . . 6
4.1.2 Evolutionsstrategien . . . 7
4.2 Vergleich beider Methoden . . . 9

5 Diskussion **9**

6 Anhang **11**

7 Quellenverzeichnis **12**

1 Einleitung

Bereits 1927 wurde dem Kinopublikum der Stummfilm *Metropolis* präsentiert, ein Science-Fiction Film von Fritz Lang, in dem eine künstliche Intelligenz eine Revolte der niederen Arbeiterschicht gegen die Oberschicht verhindern soll. Auch in vielen weiteren Filmen gibt es sie längst, ob als Bedrohung empfunden oder als Errungenschaft gefeiert. Doch auch in der Realität hat sich das Forschungsfeld um die Künstliche Intelligenz (engl. artifical intelligence) stetig weiterentwickelt.
Das Seminar *Ist künstliche Intelligenz gefährlich?* betrachtet und erörtert die Thematik der künstlichen Intelligenz aus verschiedenen Perspektiven. Die folgende Seminararbeit beschäftigt sich dabei mit der Funktionsweise von KI's, genauer mit Evolutionären Algorithmen, die neben dem bestärkenden Lernen als eine Schlüsseltechnologie im Bereich der künstlichen Intelligenz angesehen werden.

2 Evolutionäre Algorithmen

Bei evolutionären Algorithmen (EA) handelt es sich um eine Klasse stochastischer Suchverfahren, die Mechanismen der biologischen Evolution, insbesondere Mutation, Rekombination und Selektion, abstrahieren, um für ein bestimmtes Problem den gegebenen Suchraum nach der besten Lösung abzusuchen (globales Optimierungsproblem). EA werden den heuristischen Methoden zugeordnet, was bedeutet, dass sie in begrenzter Zeit oftmals nicht die optimale Lösung finden, wohl jedoch eine hinreichend gute Lösung [Nis97].

Bereits in den frühen 1960er Jahren wurden erste Ansätze entwickelt, die Prinzipien des biologischen Evolutionsprozesses auf Optimierungsprobleme verschiedenster Fachdisziplinen anzuwenden. Ein bekanntes Beispiel ist die X-Band Antenne der drei Space Technology 5 Satelliten, die die NASA im Jahr 2006 im Rahmen des New Millenium Projektes ins Weltall entsandte. Die Form der Antenne wurde mithilfe eines evolutionären Algorithmus so berechnet, dass bei möglichst geringem Stromverbrauch eine möglichst hohe Datenübertragungsrate erreicht werden konnte [HGLL].

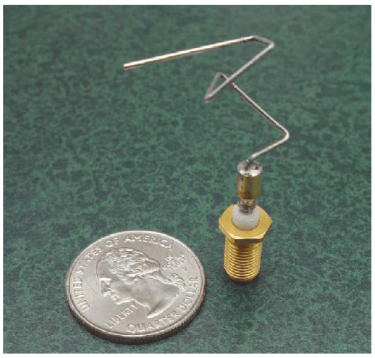

Abbildung 1: Antenne der Space Technology 5 Satelliten
Quelle: `http://www.osgi.org/Technology/WhatIsOSGi`

Im historischen Kontext werden vier Hauptströmungen evolutionärer Algorithmen unterschieden: genetische Algorithmen (GA), genetische Programmierung (GP), evolutionäre Programmierung (EP) und Evolutionsstrategien (ES), wobei letztere das Thema dieser Ausarbeitung sind [Nis97].

3 Evolutionstrategien

3.1 Historie

Die Entwicklung der Evolutionsstrategien geht auf Ingo Rechenberg und Hans-Paul Schwefel, Studenten der Flugtechnik an der Technischen Universität Berlin, Mitte der 60er Jahre zurück. Um die Form von minimalen Widerstandskörpern im Bereich von Flugobjekten automatisch anzupassen, entwickelten sie einen auf nur zwei Regeln basierenden Algorithmus, der die das Objekt beschreibenden Variablen optimieren sollte:

(1) Verändere alle Variablen zur selben Zeit, wobei die Veränderungen zufällig sind und kleinere Veränderungen wahrscheinlicher sind als große.

(2) Wähle aus den so erhaltenen Sets an Variablen dasjenige aus, welches die Performance des gesamten Objektes am meisten verbessert und gehe zu (1). Stoppe, falls das aktuelle Set dem Optimalzustand ausreichend nahe kommt.

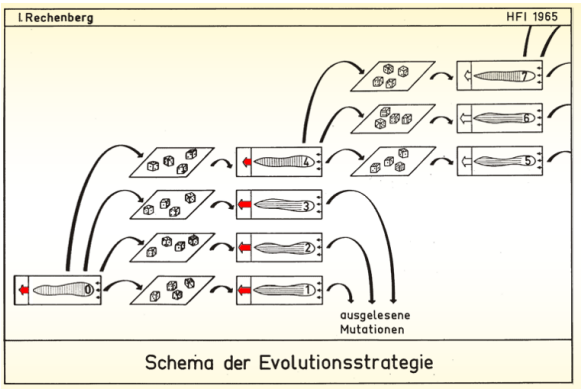

Abbildung 2: Schema der Evolutionsstrategien
Quelle: `http://www.bionik.tu-berlin.de/institut/skript/Fo15ES1.htm`

In den darauffolgenden Jahren kam es basierend auf diesem Ansatz zu zahlreichen an Komplexität zunehmenden Weiterentwicklungen, welche zum Teil aufeinander aufbauende, zum Teil jedoch auch in sich abgeschlossene Teilbereiche darstellen [BS02].

3.2 Begriffsdefinitionen

Begriff	Bedeutung im Kontext von ES
Individuum	Punkt im Suchraum, meist ein Vektor von n Elementen, der eine mögliche Lösung des Optimierungsproblems darstellt
Population	Menge gleichartig strukturierter Individuen
Fitness(wert)	Kennwert für die Güte einer Lösung im Rahmen des gegebenen Optimierungsproblems
Generation	eine Generation entspricht einer Iteration im Verfahren
Rekombination	Kombination von Elementen verschiedener Individuen zu einem neuen Individuum
Mutation	Operator, der die Komponenten eines Individuums auf zufälliger Basis verändert
Selektion	Operator, der Individuen auf Basis eines bestimmten Kennwerts (meist des Fitnesswertes) auswählt

Tabelle 1: Biologische Begriffe und deren Bedeutung im Kontext von Evolutionsstrategien

3.3 Grundkonzept

Ausgehend von einer oder mehreren möglichen Lösungen des Optimierungsproblems wird stochastisch eine größere Menge (Population) von Lösungsalternativen (Individuen) gebildet. Für jedes Individuum wird anschließend mittels einer Zielfunktion ein Fitnesswert bestimmt und diejenigen Individuen mit den besten Fitnesswerten selektiert. Dieses Vorgehen wird iterativ so lange fortgesetzt bis eine im Voraus definierte Abbruchbedingung erfüllt ist. Es handelt sich hierbei um ein populationsbasiertes Suchverfahren, da man in jeder Verfahrensiteration von mehreren anstatt von nur einer Lösungsalternative ausgehend nach der optimalen Lösung sucht. Das Wechselspiel zwischen zufälliger Veränderung und gezielter Selektion führt dabei über mehrere Generationen hinweg zu sukzessiv besseren Lösungsvorschlägen, die immer näher an der optimalen Lösung liegen. Daher lassen sich Evolutionsstrategien auch als eine Form des unüberwachten maschinellen Lernens betrachten.

3.4 Ablauf

Eine Grundform von Evolutionsstrategien soll nun in Bezug auf eine der häufigsten Anwendungen von ES, die Optimierung einer Funktion F von n kontinuierlichen Entscheidungsvariablen, im Detail erläutert werden [Nis97]. F bildet dabei einen Vektor mit n Variablen auf einen skalaren Wert ab und wir nehmen ohne Beschränkung der Allgemeinheit an, dass dieser Wert maximiert werden soll:

$$F : \mathbb{R}^n \to \mathbb{R}$$

Ein solcher Vektor stellt in der Analogie zur Biologie ein Individuum dar und enthält zum einen die Werte aller Entscheidungsvariablen $x_j \in \mathbb{R}$, zum anderen n_σ (i.d.R. $n_\sigma = n$) Standardabweichungen. Ein Individuum lässt sich also schreiben als $\vec{a} = (\vec{x}, \vec{\sigma})$.

σ_i kontrollieren als Strategieparameter statistische Eigenschaften der genetischen Operatoren, insbesondere des Mutationsoperators. Sie werden während des Verfahrens selbstadaptiv eingestellt und bilden somit die Grundlage des Lernprozesses.

3.4.1 Schritt 1: Initialisierung

Man startet mit einer Population $P(t = 0)$ mit μ Individuen $a_i, i = 1, 2, ..., \mu$. Dabei sollte darauf geachtet werden, dass die Individuen möglichst gleichmäßig über den Suchraum verteilt sind, weshalb es sich anbietet, die Ausgangspopulation stochastisch zu generieren.

3.4.2 Schritt 2: Bewertung der Lösungsalternativen

Jedes Individuum stellt eine Lösungsmöglichkeit zur Optimierung der Zielfunktion F dar. Es wird daher für alle Individuen $F(\vec{a_i})$ berechnet und das Ergebnis als Fitnesswert des Individuums bezeichnet.

3.4.3 Schritt 3: Reproduktion

Im nun folgenden Schritt können ein oder mehrere Evolutionsfaktoren zur Erzeugung von λ Nachkommen aus den μ Eltern verwendet werden. In der hier dargestellten Variante werden durch Rekombination (einschließlich Replikation) die verschiedenen Eigenschaften der beiden Eltern an ihren Nachkomme weitervererbt. Durch anschließende Mutation werden weitere Veränderungen in die Erbinformation des Nachkommen eingeführt. Diese Veränderungen können sowohl zu einer Verbesserung - in unserem Fall einer Erhöhung des Zielfunktionswertes $F(\vec{a}_{Nachkomme})$ - führen, wie auch zu einer Verschlechterung - also einer Verringerung von $F(\vec{a}_{Nachkomme})$.

1. **Auswahl der Eltern**
   Die Auswahl des Elternpaares erfolgt stochastisch, wobei jedes Individuum der aktuellen Population zu jedem Zeitpunkt die gleiche Wahrscheinlichkeit $1/\mu$ hat, ausgewählt zu werden (Ziehen mit Zurücklegen).

2. **Replikation und Rekombination**
   Die einzelnen Komponenten x_j und σ_j beider Elternvektoren werden kombiniert. Dies kann nach verschiedenen Schemata erfolgen. Im einfachsten Fall wählt man für jede Komponente stochastisch entweder den Wert des einen oder den des anderen Elternteils aus (diskrete Rekombination). Eine weitere Möglichkeit besteht darin, den Mittelwert der Komponenten beider Eltern zu bilden (intermediäre Rekombination). Oftmals bietet es sich an, für die Entscheidungsvariablen x_j und die Strategieparameter σ_j verschiedene

Rekombinationsformen anzuwenden, auch um die selbstadaptive Einstellung letzterer zu gewährleisten.

3. **Mutation**
   Zunächst werden die Strategieparameter mit einer log-normalverteilten Zufallsvariablen multipliziert. Der erhaltene Wert stellt die neue Mutationsschrittweite dar und wird anschließend mit einer weiteren Zufallsgröße multipliziert und zum Wert der Entscheidungsvariablen addiert. Man erhält also die mutierten Komponenten des Nachkommen zu:

$$\sigma_j^{mut} = \sigma_j \cdot exp(\tau_1 \cdot N(0,1) + \tau_2 \cdot N_j(0,1)) \qquad \tau_1, \tau_2 = konst. \tag{1}$$

   und daraus

$$x_j^{mut} = x_j + \sigma_j^{mut} \cdot N_j(0,1) \tag{2}$$

   Die logarithmisch normalverteilte Zufallsgröße, die in Gleichung (1) den Strategieparameter anpasst, setzt sich aus zwei Termen zusammen. Der erste Term enthält die vom Zähler j unabhängige standard-normalverteilte Zufallsvariable $N(0,1)$, wohingegen der zweite Term für jeden Wert von j, also für jede Vektorkomponente, eine neue Zufallsgröße $N_j(0,1)$ generiert und somit eine individuelle Änderung der einzelnen Strategieparameter ermöglicht. Durch dieses Vorgehen passen sich die Strategieparameter ebenso wie die Entscheidungsvariablen über mehrere Iterationen hinweg selbstständig an, was nochmal dafür spricht, Evolutionsstrategien dem Bereich des unüberwachten maschinellen Lernens zuzuordnen.

4. **Bewertung der Nachkommen**
   Für jeden Nachkommen wird abschließend ebenfalls der Fitnesswert bestimmt.

3.4.4 Schritt 4: Selektion

Durch Anwendung eines deterministischen Selektionsoperators (auch als (μ, λ)- oder Komma-Selektion bezeichnet) werden aus den λ Nachkommen diejenigen μ Individuen mit den besten (in unserem Fall höchsten) Fitnesswerten ausgewählt und in die nächste Generation übernommen. Wird hingegen aus dem gemeinsamen Pool von μ Eltern und λ Nachkommen selektiert, wird dies als $(\mu + \lambda)$- oder Plus-Selektion bezeichnet.

Schritt 3 und 4 werden so lange wiederholt, bis eine definierte Abbruchbedingung greift. Dies kann z.B. die Bedingung sein, dass die Differenz der Fitnesswerte des besten und schlechtesten Individuums einen bestimmten Schwellenwert unterschreitet.

4 Anwendung von Evolutionsstrategien im Bereich KI

Ziel der Forschung im Bereich Künstliche Intelligenz ist es, Methoden zu finden, welche es Computern ermöglichen, intelligente Leistungen zu vollbringen. Teilgebiete umfassen dabei die Entwicklung wissensbasierter Systeme, die Mustererkennung, - klassifikation und -vorhersage und in diesem Kontext auch die Entwicklung von Lernstrategien. Viele dieser Herausforderungen lassen sich als Suchproblem formulieren, welches - gesetzt den Fall, man zielt darauf ab die beste Lösung zum gegebenen Problem zu finden - zu einem Optimierungsproblem wird.

Evolutionäre Algorithmen im Allgemeinen und Evolutionsstrategien im Speziellen erweisen sich aus dem Grund als interessanter Ansatz, dass sie einen *intelligenten Suchprozess* realisieren. Einerseits werden die Suchaktivitäten durch die fitnessbasierte Selektion stärker auf die erfolgversprechenden Bereiche gelenkt, andererseits wird durch die stochastischen Elemente sicher gestellt, dass immer auch neue Bereiche des Suchraums bewertet werden.

4.1 Evolutionsstrategien als Alternative zu Reinforcement Learning

Eine aktuelle Studie [SHCS] konnte zeigen, dass Evolutionsstrategien in vielen Fällen nicht nur mit den Leistungen von modernen Reinforcement Learning Techniken mithalten können sondern dabei außerdem viele Schwierigkeiten der bestehenden Methoden umgehen.

4.1.1 Reinforcement Learning

Reinforcement Learning oder auch bestärkendes Lernen umfasst eine Reihe von Methoden des Maschinellen Lernens, die im Bereich der KI eingesetzt werden, um einen Agenten mittels eines Systems aus Belohnung und Bestrafung für eine bestimmte Aufgabe zu trainieren. Die grobe Vorgehensweise ist dabei wie folgt: es wird zuerst eine Umgebung definiert, welche beispielsweise ein Ping Pong Spiel sein kann (siehe Bild). Der Agent soll trainiert werden, dieses Spiel zu gewinnen. Er kann dazu in jeder Situation aus verschiedenen Aktionen wählen, welche beim Ping Pong Spiel das Herauf-, Herab- oder Nicht-Bewegen des Schlägers sind.
Eine Reward Funktion ordnet jedem Zustand-Aktions-Paar einen skalaren Wert als Belohnung zu, je nachdem wie positiv oder negativ der so erreichte Zustand im Kontext der Umgebung zu bewerten ist.
Um das Verhalten des Agenten zu beschreiben, soll eine Policy definiert werden, die genau angibt, in welcher Situation der Agent wie handeln soll - sie ist also eine Abbildung von Zuständen auf Aktionen. In der Praxis ist die Policy oftmals als neuronales Netz implementiert, das die aktuelle Situation als Input erhält und daraufhin ausgibt, welche der möglichen Aktionen vermutlich die Beste wäre.

Ziel des Reinforcement Learning ist es nun, die in der Policy Funktion enthaltenen Parameter zu finden. Dazu lässt man den Agenten viele Folgen von Aktionen ausführen und untersucht im Anschluss, ob diese Aktion in der gegebenen Situation sinnvoll war. Für jede Aktion im Kontext

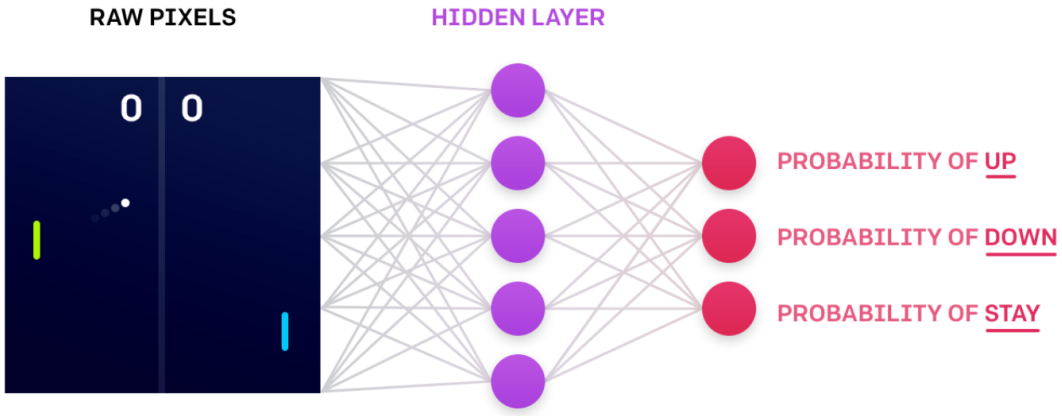

Abbildung 3: Ping Pong Spiel: hierbei entspricht die Policy einem neuronalen Netz, das für jeden Bildpixel bzw. jede Situation die Wahrscheinlichkeit für ein Herauf-, Herab- oder Nicht-Bewegen des Schlägers berechnet.
Quelle: `https://blog.openai.com/evolution-strategies/`

des vorherigen Zustandes gibt es dabei eine durch die Reward Funktion definierte Belohnung, abhängig davon, wie sehr diese Aktion zum Erfolg beigetragen hat. Mittels Backpropagation durch das neuronale Netz können nun die einzelnen Parameter verbessert werden. Durch viele solcher Testläufe lernt der Agent bzw. das neuronale Netz, wie die entsprechende Aufgabe zu lösen ist [SHCS].

4.1.2 Evolutionsstrategien

Der Ansatz hierbei ist, die gegebene Situation zu vereinfachen, indem alle Parameter des neuronalen Netzes als ein Vektor $\vec{w}$ zusammengefasst werden. Eine Funktion $f(\vec{w})$ nimmt diesen Vektor als Input und gibt ein Ergebnis (den Gesamtreward) zurück. Mathematisch gesehen versucht man nun also die Funktion f bezüglich des Parametervektors $\vec{w}$ zu optimieren. Dabei werden vorerst keine Annahmen über die genaue Formulierung der Funktion f gemacht, weshalb dieses Vorgehen auch als black-box optimization bezeichnet wird [SHCS].

Salimans *et al.* verwendeten eine Variante der Evolutionsstrategien, die genannten *Natural Evolution Strategies*. Ausgehend von einer Version des Parametervektors $\vec{w}$ mit zufälligen Werten wird wiederholt eine größere Population an untereinander leicht variierenden Vektoren gebildet, indem den Elementen von $\vec{w}$ gauß'sches Rauschen hinzuaddiert wird. Dann wird für jeden der Vektoren der Gesamtreward bestimmt, indem das neuronale Netz mit den entsprechenden Parametern für eine Weile in der Umgebung angewendet wird. Der Gesamtreward entspricht der Summe aller Belohnungen oder Bestrafungen für die Einzelaktionen in diesem Durchlauf. Anschließend wird ein neuer Vektor $\vec{w}$ als mit den entsprechenden Gesamtrewards gewichtete Summe aller Vektoren bestimmt.
Mathematisch entspricht dieses Vorgehen auch dem Abschätzen des Gradienten der Zielfunktion im Parameterraum mittels der Finite Differenzen Methode.

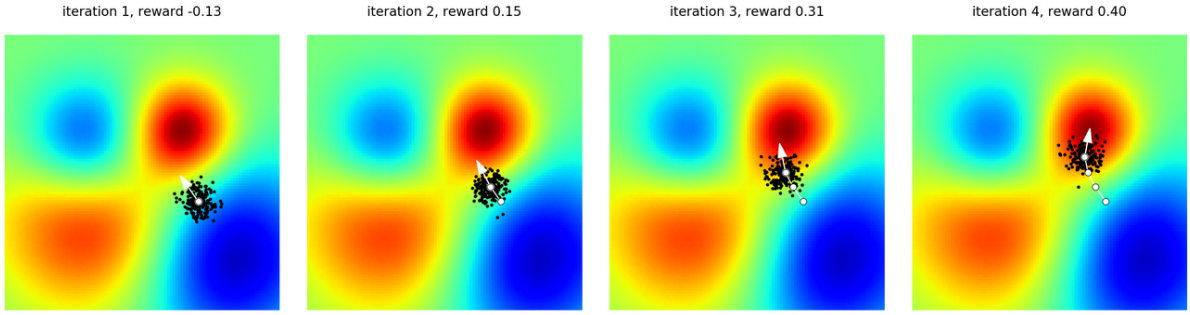

Abbildung 4: Darstellung von 4 Evolutionsstrategie-Iterationen. Die Farbgebung des Hintergrundes gibt die Werte der Reward Funktion an (rot = hohe Werte, blau = niedrige Werte). Der aktueller Parametervektor ist als weißer Punkt dargestellt, die Population in schwarz, der weiße Pfeil stellt den aktuell geschätzten Gradienten dar. Es ist gut erkennbar, dass die Population an Parametervektoren sich mit jeder Iteration der optimalen Lösung (dem höchsten Wert der Reward Funktion) annähert. Die Population ist dabei immer gleichmäßig um den aktuellen Parametervektor verteilt, wodurch eine Vorstellung der lokalen Struktur der Reward Funktion erhalten wird, mittels der sich der Gradient schätzen lässt.
Quelle: `https://blog.openai.com/evolution-strategies/`

Der Algorithmus kann in Pseudocode folgendermaßen notiert werden. Eine detailliertere Version in Pyhton, findet sich im Anhang.

```
1  Input: Lernrate α, Standardabweichung des Rauschens σ, Parametervektor w_0
       , Iterationszahl N, Populationsgroesse k
2
3  for t in range(1,N):
4    Sampling ε_1,...ε_k ~ N(0,1)
5
6    for i in range(1,k):
7      Berechnung des Rewards F_i = F(w_t + σε_i)
8
9    Setze w_{t+1} = w_t + α 1/(kσ) Σ_{i=1}^{k} F_i ε_i
```

Algorithmus 1: Algorithmus Evolutionsstrategien: Für die vorgegebene Anzahl an Iterationen werden Stichproben aus der verwendeten Verteilung entnommen und der Fitnesswert des ursprünglichen Parametervektors plus die Standardabweichung multipliziert mit dem Sample berechnet. Da die Standardabweichung immer den gleichen Wert hat, werden alle Vektorkomponenten gleich stark verändert. Der Fitnesswert berechnet sich als Gesamtreward, also die Summe aller Belohnungen bzw. Bestrafungen, die während der Anwendung des neuronalen Netzes erhalten werden. Der neue Parametervektor wird als mit den entsprechenden Fitnesswerten gewichtete Summe aller Vektoren bestimmt.
Quelle: `https://blog.openai.com/evolution-strategies/`

4.2 Vergleich beider Methoden

Der zentrale Unterschied der beiden Methoden liegt im Einbringen des Rauschens. Im Fall der oben beschriebenen RL Methode wird das Rauschen in den Aktionsraum eingebracht, indem man das neuronale Netz verschiedene Episoden in der Umgebung durchlaufen und jeweils die Wahrscheinlichkeiten für eine bestimmte Aktion berechnen lässt. Dabei wird sich der Agent mehrmals in der gleichen Situation wiederfinden, aber aufgrund der stochastischen Policy unterschiedliche Aktionen durchführen und aus dem jeweiligen Resultat lernen.
Bei den Evolutionsstrategien hingegen wird das Rauschen direkt im Parameterraum erzeugt und davon ausgehend versucht, eine optimale Kombination der Parameter für die Policy zu finden.

Salimans *et al.* haben ihre Evolutionsstrategie an zwei Standardlernproblemen getestet: dem MuJoCo Physics Simulator und verschiedenen Atari Games. Ersteres ist eine Funktionseinheit, die physikalische Figuren simuliert. Das neuronale Netz erhält dabei die Positionen aller Gelenke der Figur als Input und soll als Ausgabe die Drehmomente für jedes Gelenk so zurückgeben, dass die Figur in der Lage ist, sich fortzubewegen. Das Ergebnis der Studie zeigt, dass diese Aufgabe mit Evolutionsstrategien ähnlich gut bewältigt werden kann, wie mit den etablierten RL Methoden.
Auch die Atari Games, wie beispielsweise Ping Pong, lassen sich gut erlernen. Bei einigen Spielen trat jedoch das Problem eines fehlenden Gradientensignals auf. Es zeigte sich, dass das Hinzufügen von Gauß'schem Rauschen nicht ausreichte, den Parameterraum ausreichend gut abzutasten und ein verwertbares Gradientensignal zu erhalten. Als einen möglichen Ansatz zur Lösung dieses Problems nennen Salimans *et al.* die Verwendung einer virtuellen Batchnormalisierung, eines statistischen Verfahrens, das die Policy zu Anfang des Trainings empfindlicher gegenüber kleinen Veränderungen im Parameterraum macht [SHCS].

5 Diskussion

Evolutionsstrategien bieten einige Vorteile gegenüber den klassischen Reinforcement Learning Verfahren. Sie sind einfach zu implementieren und während bei RL Verfahren meist die Value Funktion oder der Gradient aufwändig approximiert werden müssen, ist dies für die Evolutionsstrategien nicht nötig. Zudem benötigt man keine Backpropagation und damit auch keine Speicherung der einzelnen Lernepisoden, wodurch der Code wesentlich kürzer und schneller wird. Für Evolutionsstrategien wird außerdem eine geringere Anzahl an Hyperparametern benötigt. Die Aktionsfrequenz bezeichnet die Geschwindigkeit, in der die Aktionen des Agenten aufeinander folgen. Während des Trainings hat diese Taktfrequenz, auch frame skip genannt, einen großen Einfluss auf die Qualität des Endergebnisses. Ist die Taktfrequenz in der Trainingsphase zu niedrig, kann der Agent in der realen Umgebung nicht ausreichend schnell reagieren. Bei einer zu hohen Taktfrequenz hingegen erschwert sich das Training mittels RL Methoden. Es ist leicht erkennbar, dass Evolutionsstrategien aufgrund ihres Charakters als Black-Box-Optimierung un-

abhängig von der Aktionsfrequenz sind.
Evolutionsstrategien sind zudem in der Lage, die intrinsische Dimensionalität eines Problems zu erkennen. Angenommen die Policy Funktion enthält 100 Parameter, so können Evolutionsstrategien relevante von irrelevanten Parametern zu unterscheiden und letztere automatisch vernachlässigen. Dadurch arbeiten Evolutionsstrategien auch bei hochdimensionalen Problemen noch sehr schnell. Der Hauptvorteil der Evolutionsstrategien liegt jedoch in ihrer Parallelisierbarkeit. Das bedeutet, es können parallel mehrere CPUs (auch worker genannt) eingesetzt werden, weil lediglich der Gesamtreward zwischen diesen übermittelt werden muss. Salimans *et al.* gelang es aufgrund dieser Tatsache, eine der schwierigsten MuJoCo Aufgaben unter Verwendung von 1440 über 80 Maschinen verteilte CPUs in nur 10 Minuten zu lösen. Die übliche Zeitdauer bei 32 CPUs auf einer Maschine liegt bei 10 Stunden. Die Steigerung der Rechengeschwindigkeit skaliert nach Salimans *et al.* linear mit der Anzahl der verwendeten CPUs.
Mit überwachten Lernverfahren, also RL Methoden, bei denen der Gradient mittels Backpropagation exakt berechnet werden kann, können Evolutionsstrategien jedoch nicht mithalten, da sie selbst nur eine Approximation des Gradienten verwenden. Letztendlich gilt es also, die Vor- und Nachteile von Evolutionsstrategien gegenüber den RL Verfahren für das jeweilige Problem abzuwägen und danach zu entscheiden, welche Methode eingesetzt werden soll.

6 Anhang

```
"""
A bare bones examples of optimizing a black-box function (f) using
Natural Evolution Strategies (NES), where the parameter distribution is a
gaussian of fixed standard deviation.
"""

import numpy as np
np.random.seed(0)

# the function we want to optimize
def f(w):
  # here we would normally:
  # ... 1) create a neural network with weights w
  # ... 2) run the neural network on the environment for some time
  # ... 3) sum up and return the total reward

  # but for the purposes of an example, lets try to minimize
  # the L2 distance to a specific solution vector. So the highest reward
  # we can achieve is 0, when the vector w is exactly equal to solution
  reward = -np.sum(np.square(solution - w))
  return reward

# hyperparameters
npop = 50 # population size
sigma = 0.1 # noise standard deviation
alpha = 0.001 # learning rate

# start the optimization
solution = np.array([0.5, 0.1, -0.3])
w = np.random.randn(3) # our initial guess is random
for i in range(300):

  # print current fitness of the most likely parameter setting
  if i % 20 == 0:
    print('iter %d. w: %s, solution: %s, reward: %f' %
          (i, str(w), str(solution), f(w)))

  # initialize memory for a population of w's, and their rewards
  N = np.random.randn(npop, 3) # samples from a normal distribution N(0,1)
  R = np.zeros(npop)
  for j in range(npop):
    w_try = w + sigma*N[j] # jitter w using gaussian of sigma 0.1
    R[j] = f(w_try) # evaluate the jittered version

  # standardize the rewards to have a gaussian distribution
  A = (R - np.mean(R)) / np.std(R)
  # perform the parameter update. The matrix multiply below
  # is just an efficient way to sum up all the rows of the noise matrix N,
  # where each row N[j] is weighted by A[j]
  w = w + alpha/(npop*sigma) * np.dot(N.T, A)
```

Abbildung 5: Python Beispielcode zu Evolutionsstrategien
Quelle: `https://blog.openai.com/evolution-strategies/`

7 Quellenverzeichnis

[BS02] Beyer, Hans-Georg ; Schwefel, Hans-Paul: Evolution strategies – A comprehensive introduction. In: *Natural Computing* 1 (2002), Nr. 1, S. 3–52. http://dx.doi.org/10.1023/A:1015059928466. – DOI 10.1023/A:1015059928466. – ISSN 15677818

[HGLL] Hornby, Gregory ; Globus, Al ; Linden, Derek ; Lohn, Jason: Automated Antenna Design with Evolutionary Algorithms. In: *Space 2006*

[Nis97] Nissen, Volker: *Einführung in Evolutionäre Algorithmen: Optimierung nach dem Vorbild der Evolution.* Wiesbaden : Vieweg+Teubner Verlag, 1997 (Computational Intelligence). http://dx.doi.org/10.1007/978-3-322-93861-9. http://dx.doi.org/10.1007/978-3-322-93861-9. – ISBN 3–528–05499–9

[SHCS] Salimans, Tim ; Ho, Jonathan ; Chen, Xi ; Sutskever, Ilya: *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*