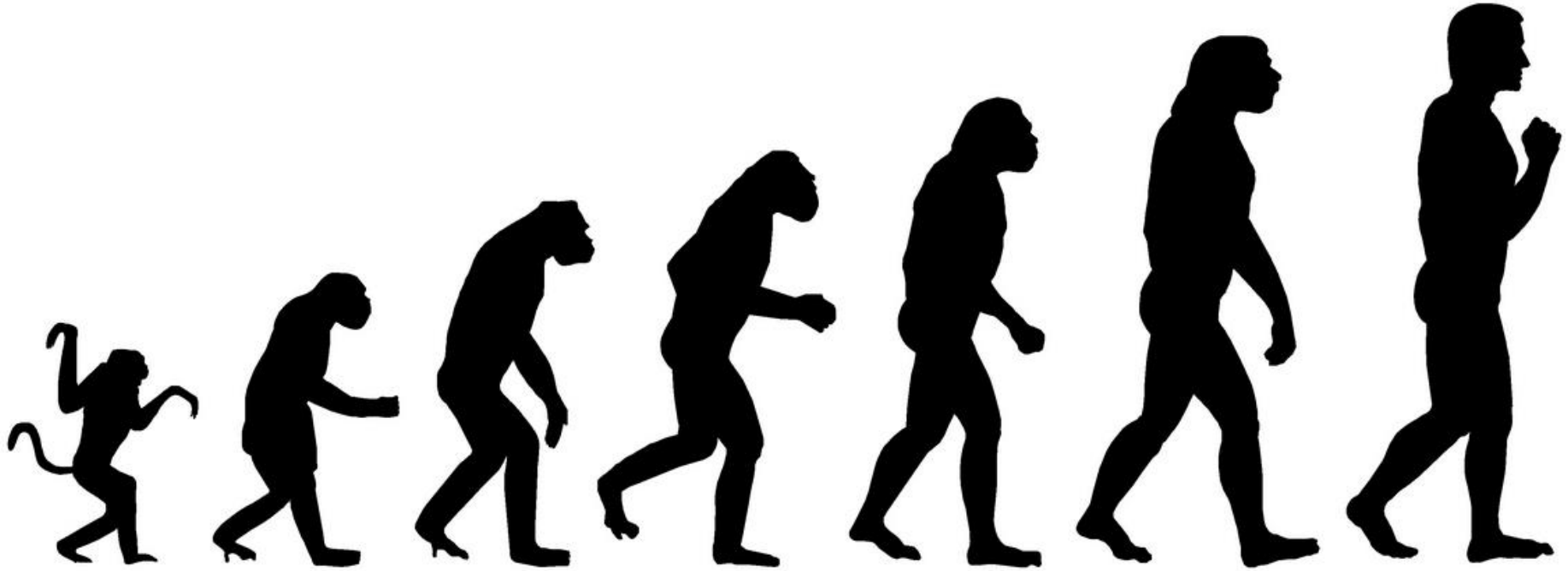


Evolution Strategies

Daniela Schacherer

Seminar: Ist künstliche Intelligenz gefährlich?

Sommersemester 2017

Inhalt

- Einleitung und Überblick
- Evolutionsstrategien – Grundkonzept
- Evolutionsstrategien als Alternative zu Bestärkendem Lernen
- Benchmarking
- Weitere Anwendungen
- Fazit

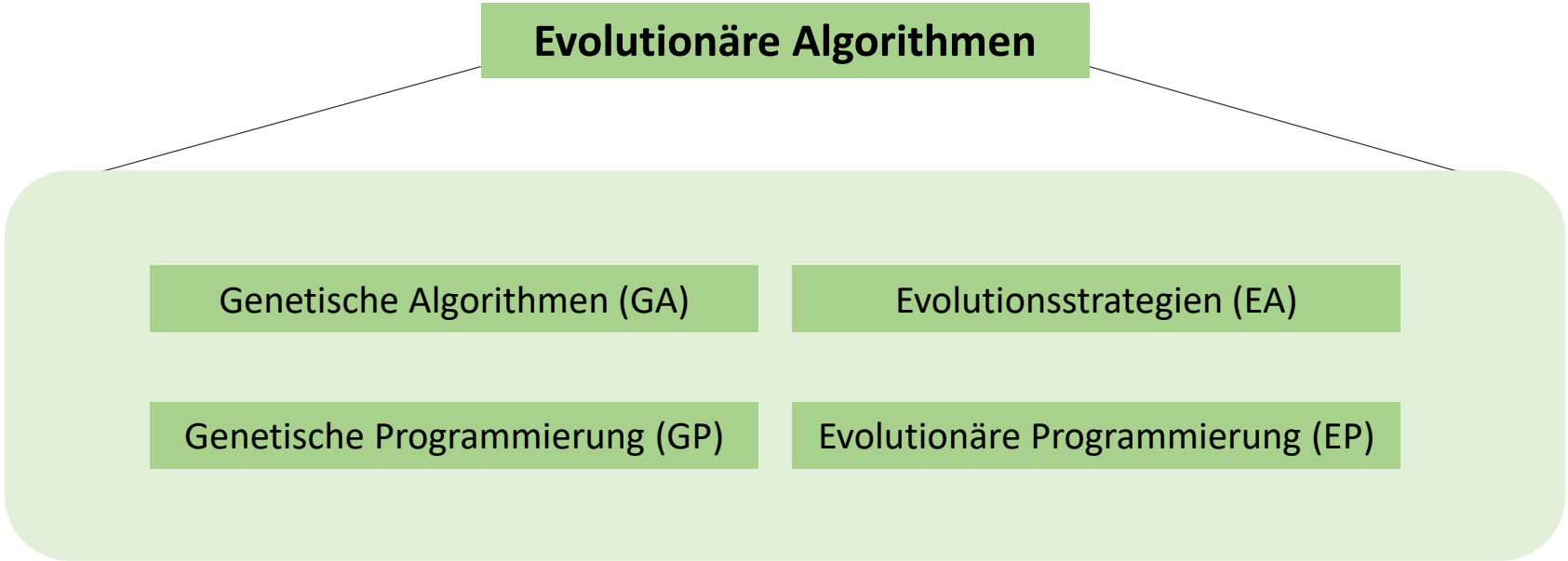
Einleitung und Überblick

Evolutionäre Algorithmen

- stochastische Suchverfahren (heuristisch)
- Abstraktion von Mechanismen der biologischen Evolution
  - Rekombination
  - Mutation
  - Selektion

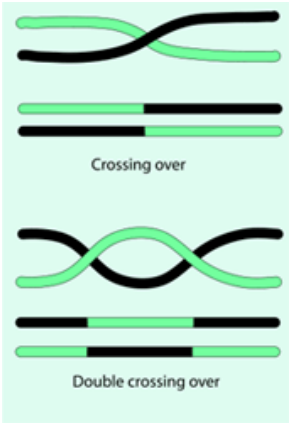
Evolutionäre Algorithmen

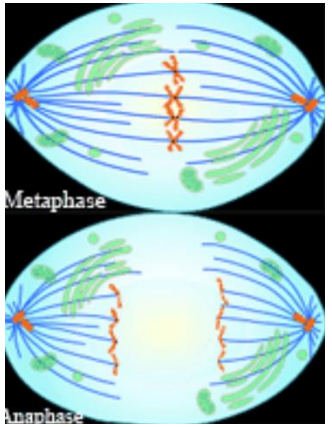
- stochastische Suchverfahren (heuristisch)
- Abstraktion von Mechanismen der biologischen Evolution
  - Rekombination
  - Mutation
  - Selektion

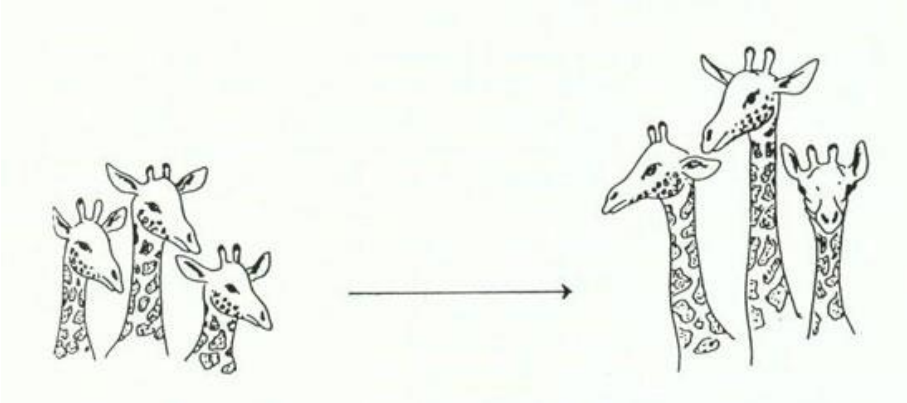


Evolution (biologisch)

- Rekombination: Neuanordnung von vorhandenem genetischen Material (DNA, RNA)
- Punktmutation: Austausch einer einzelnen Nukleotidbase
- Selektion: natürliche Auslese basierend auf der Fitness







… AGCTGCGCGCAAA …

↓

… AGCTGCCCGCAAA …

Evolutionsstrategien

- Unterklasse der **Evolutionären Algorithmen**
- 1960er: Ingo Rechenberg und Hans-Paul-Schwefel

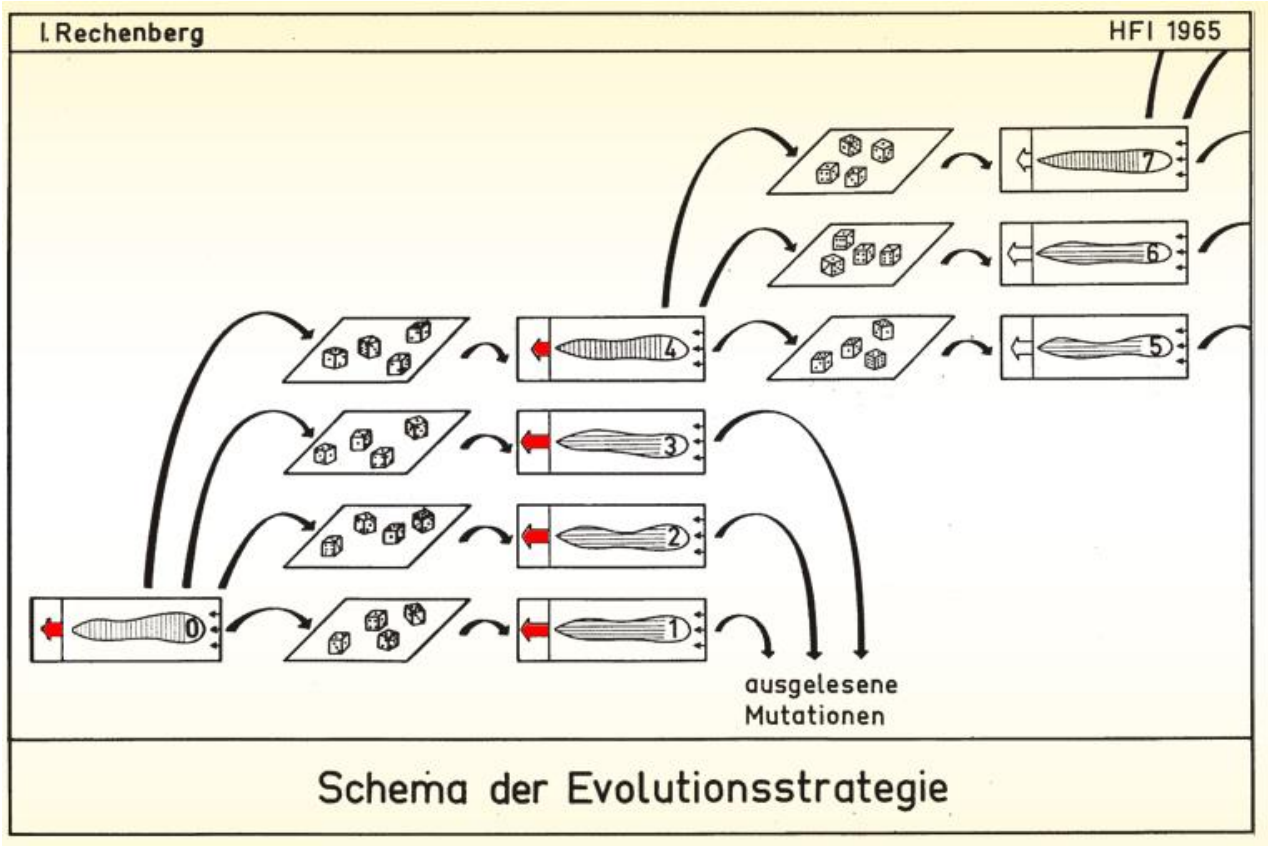
„Der Dumme, der einfach losgeht, kommt weiter als der Schlaue, der sitzen bleibt und sich vor lauter Nachdenken nicht entscheiden kann.“

Prof. Hans-Paul Schwefel

Prof. Ingo Rechenberg

Schema der Evolutionsstrategien

- **gegeben:**
  - Objekt beschrieben durch einen Satz an Variablen
  - Startwerte für die Variablen
- **gesucht:**
  - Werte aller Variablen, sodass das Objekt bzgl. eines bestimmten Zielzustandes optimal ist.
- Zwei Regeln:
  1) zufällige Veränderung aller Variablen ausgehend von den Startwerten
  2) Auswahl des Sets an Variablen, das das beste Resultat liefert und setzen dieser als neue Startwerte.



Evolutionsstrategien

Grundkonzept

Begrifflichkeiten

- **Individuum**: Punkt im Suchraum, meist Vektor von n Elementen, der eine mögliche Lösung des Optimierungsproblems darstellt
- **Population**: Menge gleichartig strukturierter Individuen
- **Generation**: 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 Kennwertes (**Fitnesswert**) auswählt.

Ablauf

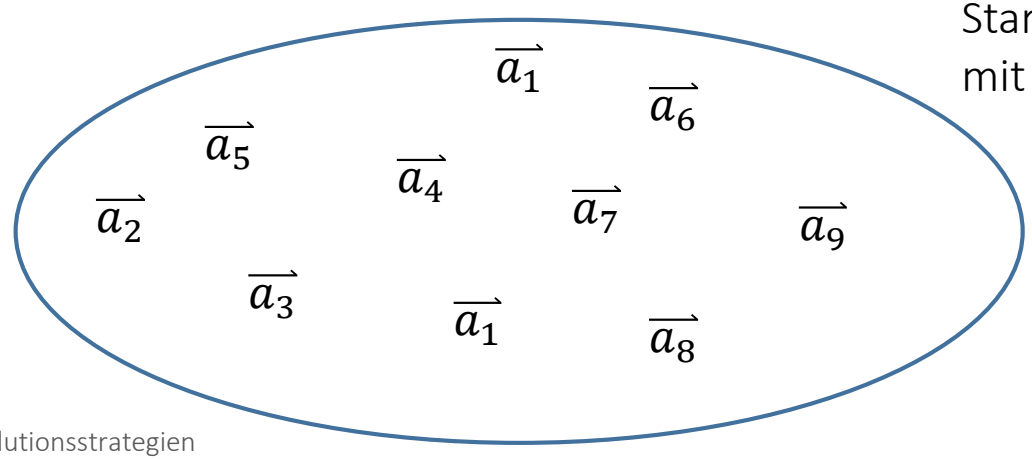
Entscheidungsvariablen $\vec{x} = \begin{pmatrix} x \\ \vdots \\ x_n \end{pmatrix}$

Strategieparameter $\vec{\sigma} = \begin{pmatrix} \sigma \\ \vdots \\ \sigma_n \end{pmatrix}$

Individuum $\vec{a} = (\vec{x}, \vec{\sigma})$

Schritt 1: Initialisierung

Startpopulation mit μ Individuen



Fitnessfunktion / Zielfunktion

$\mathrm{F}(\vec{a}) \rightarrow max$

Ablauf /2

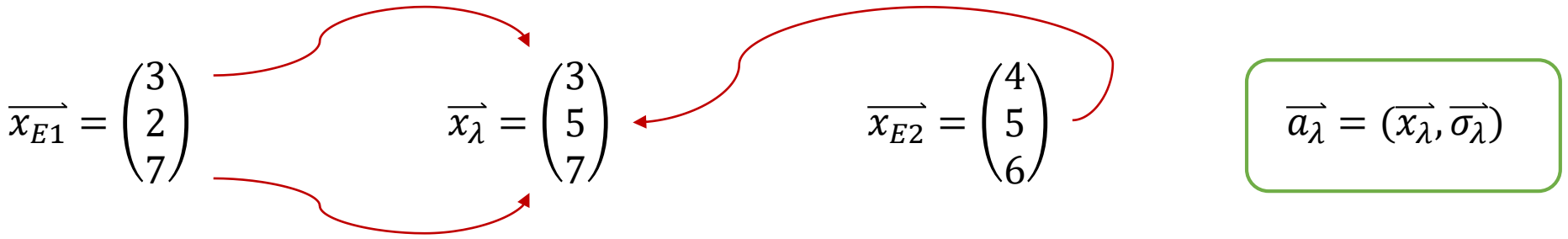
Schritt 2: Bewertung der Individuen → berechne $\mathrm{F}(\vec{a})$ für jedes Individuum

Schritt 3: Reproduktion → erzeuge λ Nachkommen aus den μ Eltern, $\lambda > \mu$

Schritt 3-1: zufällige Auswahl der Eltern → a_{E1} und a_{E2}

Schritt 3-2: Rekombination für Entscheidungsvariablen und Strategieparameter

Diskrete Rekombination



Ablauf /2

Schritt 2: Bewertung der Individuen → berechne $\mathrm{F}(\vec{a})$ für jedes Individuum

Schritt 3: Reproduktion → erzeuge λ Nachkommen aus den μ Eltern, $\lambda > \mu$

Schritt 3-1: zufällige Auswahl der Eltern → a_{E1} und a_{E2}

Schritt 3-2: Rekombination für Entscheidungsvariablen und Strategieparameter

Intermediäre Rekombination

$$Mittelwert\ (\overrightarrow{x_{E1}}, \overrightarrow{x_{E2}}) \rightarrow \overrightarrow{x_\lambda} = \begin{pmatrix} 3.5 \\ 3.5 \\ 6.5 \end{pmatrix}$$

$$\overrightarrow{a_\lambda} = (\overrightarrow{x_\lambda}, \overrightarrow{\sigma_\lambda})$$

Ablauf /3

Schritt 3-3: Mutation

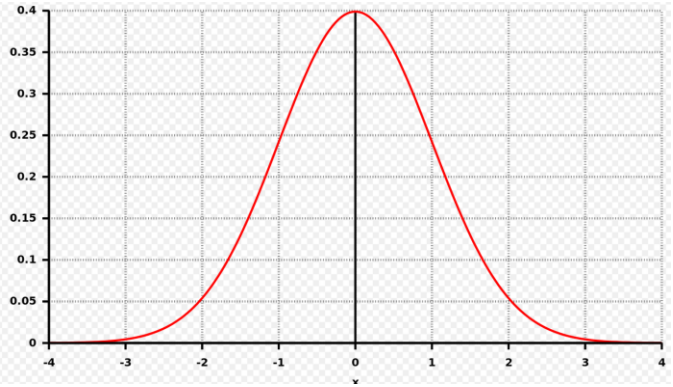
$$\overrightarrow{a_\lambda} = (\overrightarrow{x_\lambda}, \overrightarrow{\sigma_\lambda})$$

$$= \left(\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \begin{pmatrix} \sigma_1 \\ \vdots \\ \sigma_n \end{pmatrix} \right)$$

$$\sigma_j^{mut} = \sigma_j * e^{\tau_1 * N(0,1)} * e^{\tau_2 * N_j(0,1)}$$

für alle σ_j gleich

wird für alle σ_j neu bestimmt → individuelle Anpassung der Strategieparameter



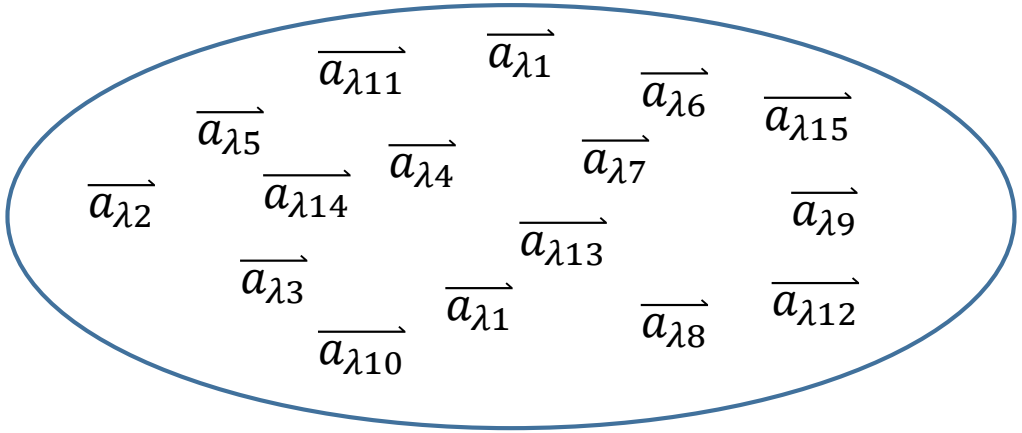
$$x_j^{mut} = x_j + \sigma_j^{mut} * N_j(0,1)$$

Schritt 3-4: Bewertung der Nachkommen → berechne $\mathrm{F}(\vec{a})$ für jedes der λ Nachkommen

Ablauf /4

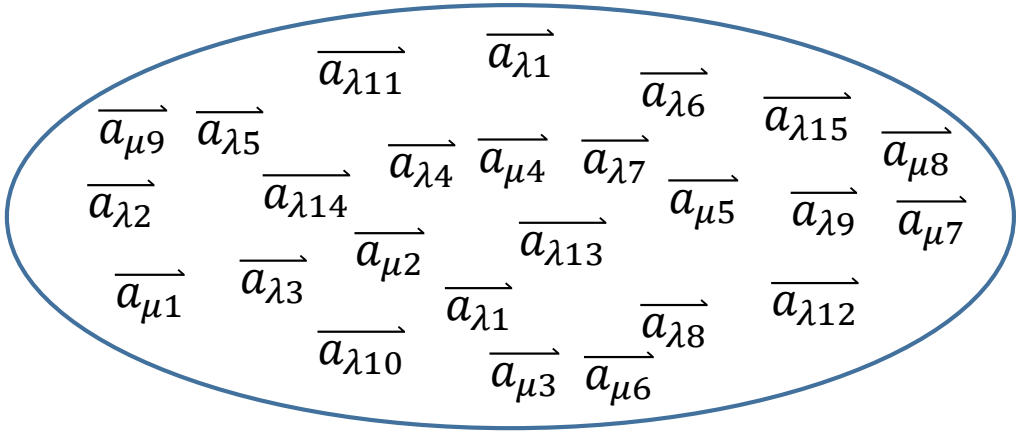
Schritt 4: Selektion

(μ, λ)-Selektion



→ Wähle die besten μ Individuen aus λ Nachkommen

$(\mu + \lambda)$-Selektion

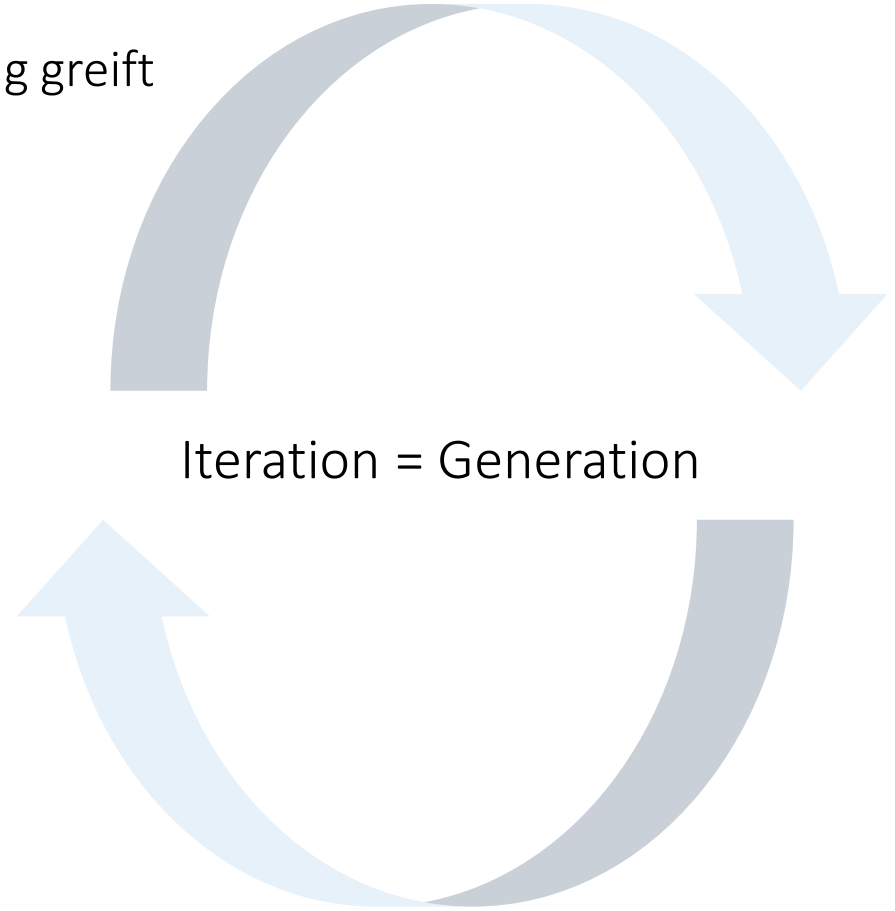


→ Wähle die besten μ Individuen aus $\mu + \lambda$ Nachkommen

Ablauf /5

Schritt 5: gehe zu **Schritt 3** solange, bis eine Abbruchbedingung greift

unüberwachtes maschinelles Lernverfahren



- Populationsbasierte Suche
- Intelligenter Suchprozess
- Selbstanpassung der Strategieparameter
- Mutation, Rekombination bedeutend
- Selektionsoperator arbeiten auf Phänotyp
- Deterministische Überlebensselektion

Optimierung einer Kugelbahn

Beispielprogramm

Evolutionsstrategien

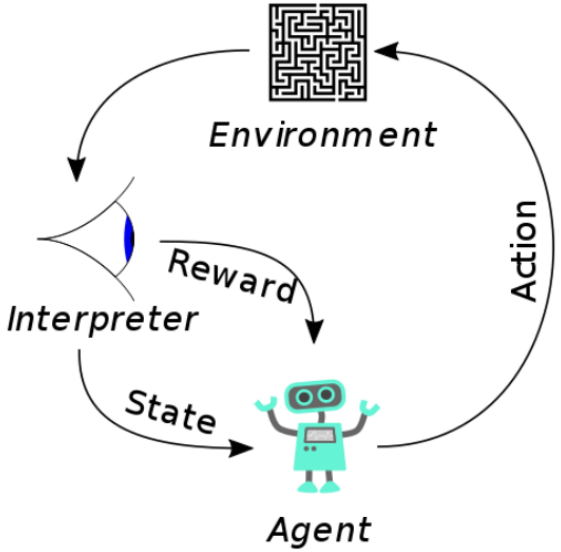
als Alternative zum bestärkenden Lernen

[https://blog.openai.com/evolution-strategies/]

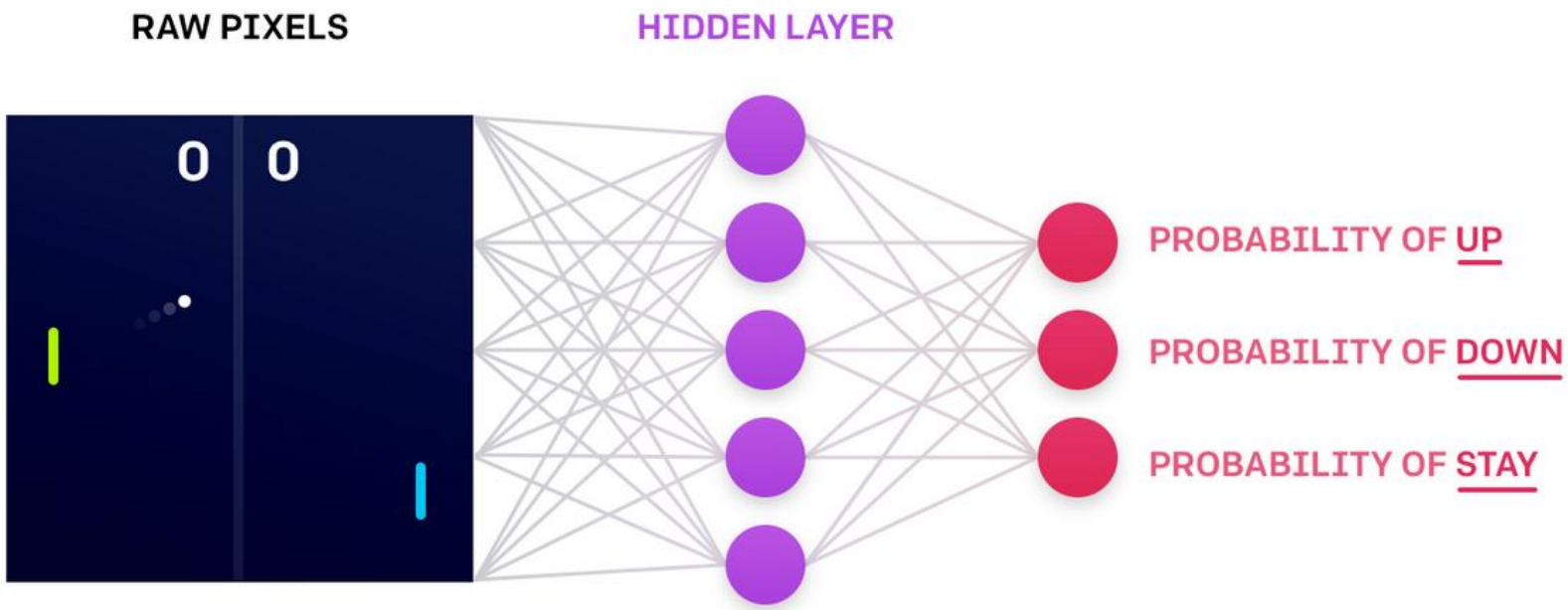
Bestärkendes Lernen

- engl.: reinforcement learning
- **Prinzip:** Ein Agent erlernt eine Zuordnung („mapping“) von Zuständen zu Aktionen durch try-and-error Interaktionen mit der Umwelt

- Hauptbestandteile eines Reinforcement Learning Problems:
  - **Umgebung**
    - z.B. ein Spiel
  - **Policy Funktion**
    - Zuordnung von Zuständen zu Aktionen
    - „Anleitung“ für den Agenten, wie er sich zu verhalten hat (anfangs stochastisch)
    - implementiert als neuronales Netz
  - **Reward Funktion / Reinforcement Funktion**
    - Zuordnung von Zuständen+Aktionen zu „Belohnungen“ (engl. Reward) → $\sum reward$ soll m
  - **Value Funktion**
    - Zuordnung von Zuständen+Aktionen zu dem Reward, den man langfristig erwarten kann, ausgehend vom gegebenen Zustand



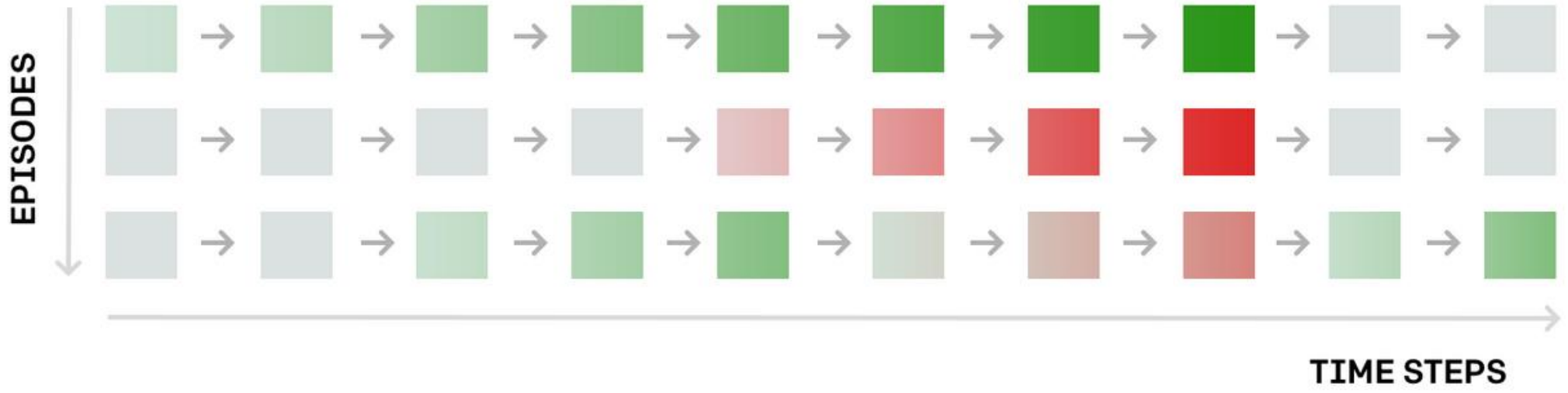
Bestärkendes Lernen



Ping Pong: Die Policy ist implementiert als neuronales Netz, das für jeden Pixel (jede mögliche Situation) die Wahrscheinlichkeit für Herauf-, Herunter oder Nicht-Bewegen des Schlägers berechnet.

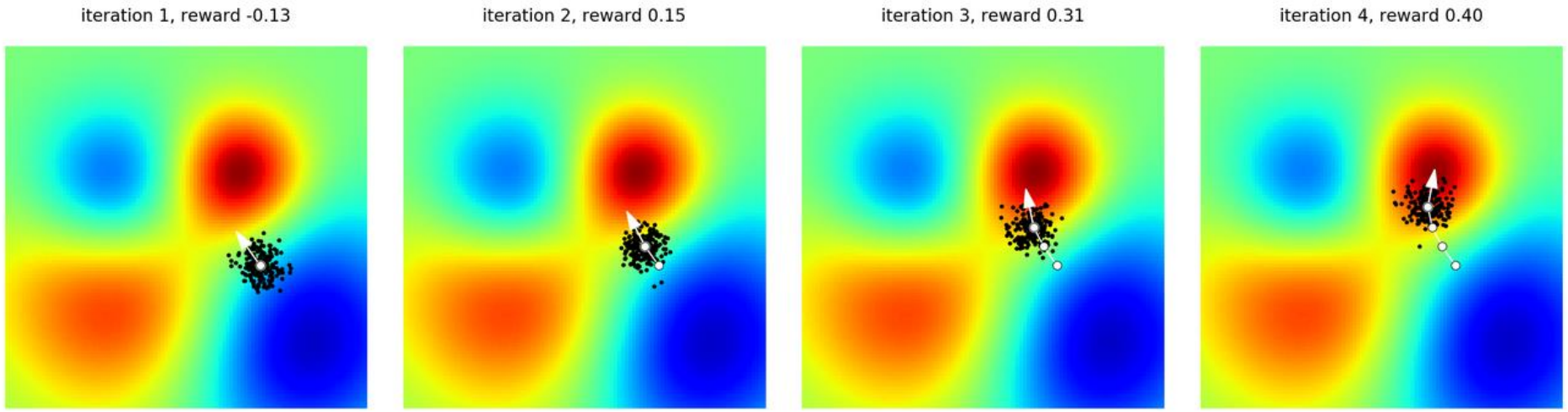
Bestärkendes Lernen

- Trainingsprozess (iterativ)
  - Wähle zufälligen Startpunkt, d.h. zufällige Werte für die Parameter des neuronalen Netzes
  - Sammle Interaktionserfahrung und werte sie aus
  - Aktualisiere die Parameter des neuronalen Netzes mittels Backpropagation



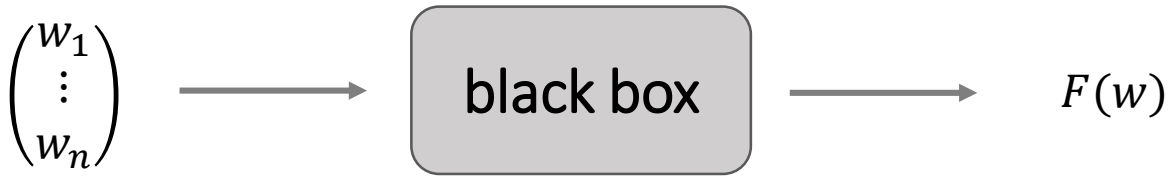
Evolutionsstrategien

- Ablauf (iterativ)
  - Wähle zufällige Werte für die Parameter des neuronalen Netzes → Parametervektor $\vec{w}$
  - Gauß‘sches Rauschen → generiere Population von Parametervektoren und werte jeden Vektor seperat aus
  - Aktualisiere $\vec{w}$ entsprechend.



Evolutionsstrategien

- Wie funktioniert die Auswertung?
- Black-box-Optimierung



1) Erstelle neuronales Netz mit Gewichten w_i
2) Wende das Netz eine Weile auf die Umgebung an
3) Summiere alle erhaltenen Belohnungen auf und gib den Gesamt-Reward $F(w)$ aus.

Der Algorithmus

Algorithm 1 Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
    Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

Python Code: https://gist.github.com/karpathy/77fbb6a8dac5395f1b73e7a89300318d

Der Algorithmus /2

```
"""
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
```

Der Algorithmus /3

```
# 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)
```

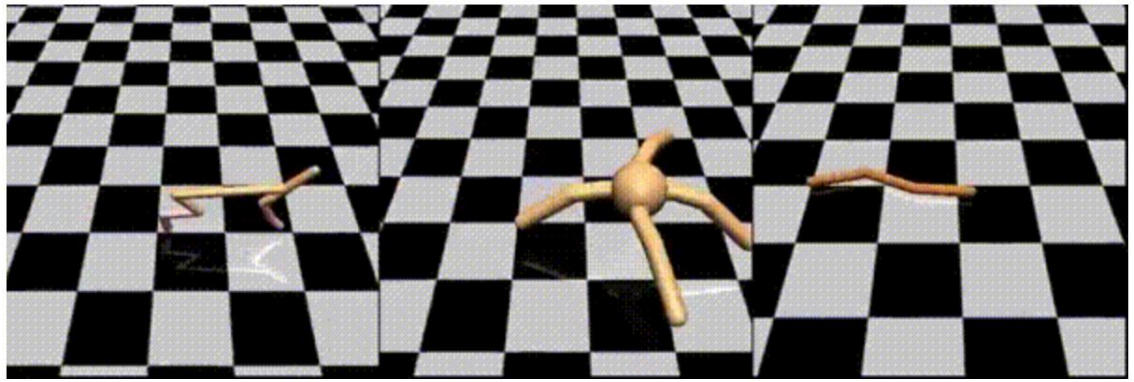
Betrachtungen der „Zufälligkeit“

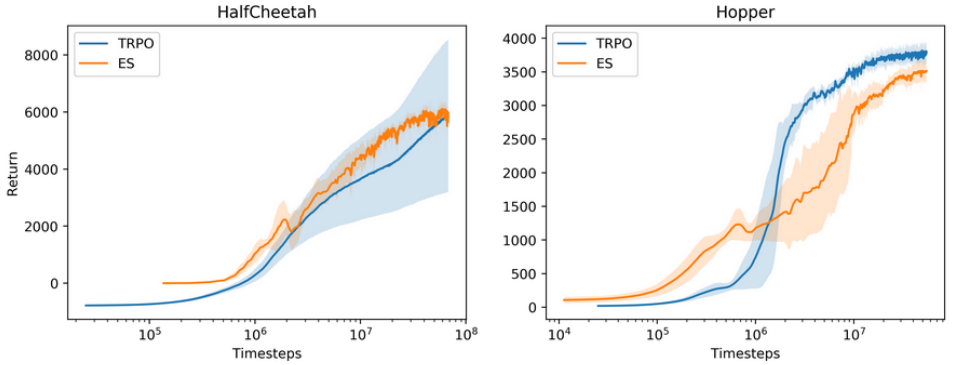
- Zielfunktion ist identisch → in beiden Fällen optimieren wir die erwartete „Belohnung“
- Die Einbringung von Varianz bzw. Rauschen unterscheidet sich
  - Bestärkendes Lernen: in den Aktionsraum → guess and check on actions
  - Evolutionsstrategien: in den Parameterraum → guess and check on parameters

Benchmarking

Benchmarking

- MuJoCo physics simulator
  - **Input:** Positionen aller Gelenke
  - **Output:** die jeweils nötige Drehmomente, um die Figur vorwärts zu bewegen



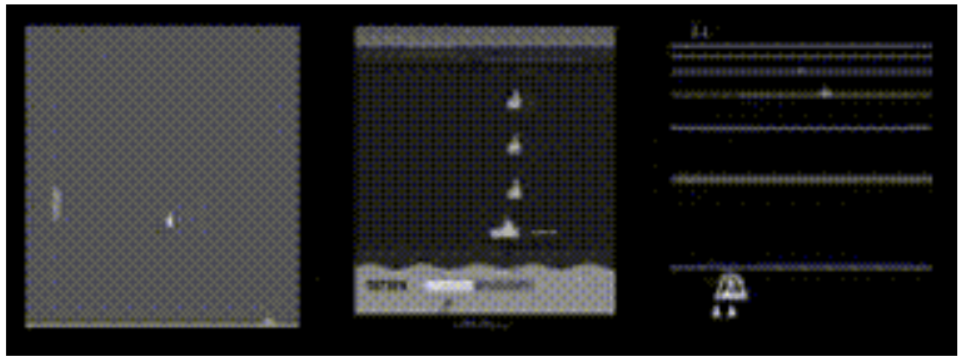


https://blog.openai.com/evolution-strategies/

→ vergleichbare Performance, aber nicht in allen Fällen

Benchmarking /2

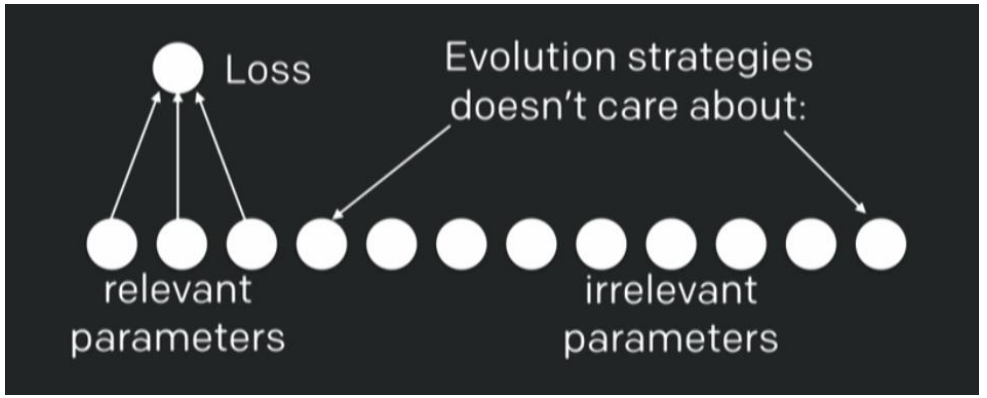
- Atari Games mit Pixelinput



- Schwierigkeit:
  - Hinzufügen von Rauschen führt nicht immer dazu, dass ein Gradientensignal erhalten wird
- Lösung:
  - Virtual batch normalization

Vergleich

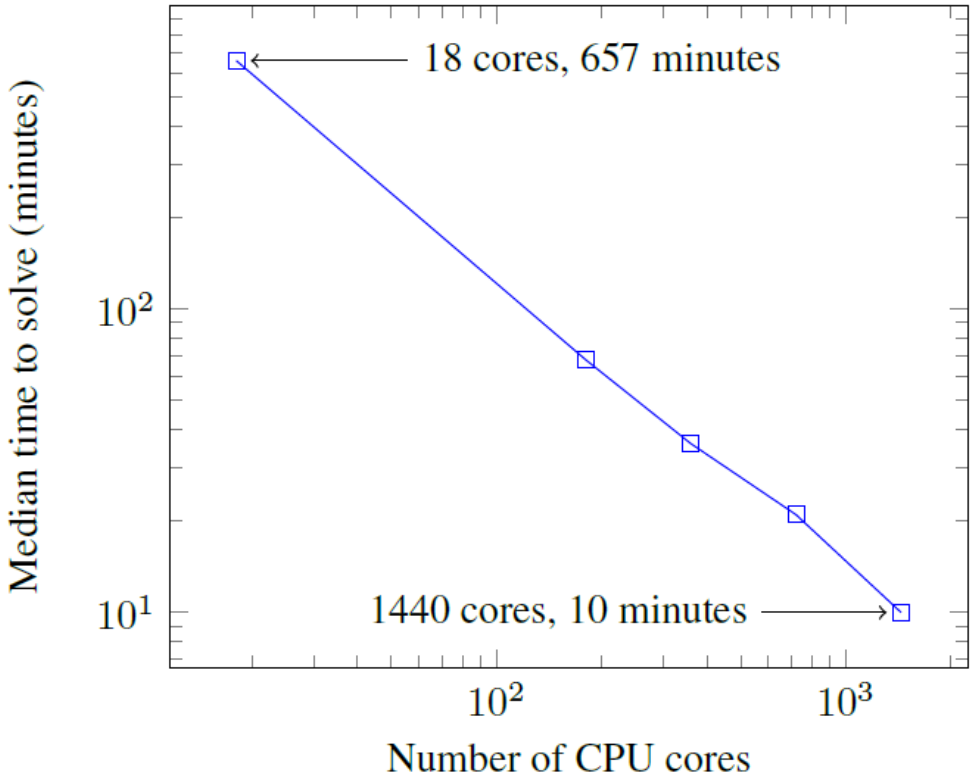
- Vorteile Evolutionsstrategien
  - Einfach zu implementieren
  - Keine Approximation der Value Funktion
  - Keine exakte Berechnung von Gradienten
  - Keine Backpropagation
  - Weniger Hyperparameter
  - Robustheit gegenüber Aktionsfrequenz
  - Erkennt intrinsische Dimensionalität



Vergleich

- Vorteile Evolutionsstrategien
  - Einfach zu implementieren
  - Keine Approximation der Value Funktion
  - Keine Approximation von Gradienten
  - Keine Backpropagation
  - Weniger Hyperparameter
  - Robustheit gegenüber Aktionsfrequenz
  - Erkennt intrinsische Dimensionalität

  - **Parallelisierbarkeit**



Vergleich /2

- Vorteile Evolutionsstrategien
  - Einfach zu implementieren
  - Keine Approximation der Value Funktion
  - Keine Approximation von Gradienten
  - Keine Backpropagation
  - Weniger Hyperparameter
  - Robustheit gegenüber Aktionsfrequenz
  - Erkennt intrinsische Dimensionalität
  - **Parallelisierbarkeit**
- Nachteile Evolutionsstrategien
  - Man erhält nicht immer ein Gradientensignal
  - Nicht kompetitiv mit RL Methoden, die den Gradienten mittels Backpropagation exakt berechnen können (supervised learning).

Vergleich /3

- Wann sollte man also ES gegenüber RL bevorzugen?
  - Große Anzahl an Schritten
  - Aktionen haben langfristige Effekte
  - Es gibt keine gute Schätzung für die Value Funktion

Weitere Anwendungen?

Anwendungen

- Training neuronaler Netze
- Optimierung von Bauelementen
  - Brücken
  - Strömungskrümmer
  - Überschalldüsen für die Raumfahrt (Wirkungsgrad bei der Stromerzeugung optimieren)
  - Optische Linsen
- Optimierung von Algorithmen der Videosignalverarbeitung
  - → Minimierung des Fehlers der Filterfunktion
- Tourenplanung (kombinatorische Optimierung)

Take home

- Stochastische Suchverfahren (heuristisch)
- Basierend auf Rekombination, Mutation und Selektion
- Einfaches Grundprinzip, viele Varianten
- Hauptvorteil gegenüber klassischen RL-Verfahren: Parallelisierbarkeit
- Zahlreiche weitere Anwendungen in den Ingenieur- und Naturwissenschaften

Quellen

- Bildquellen:
  - https://de.wikipedia.org/wiki/Space_Technology_5#/media/File:St_5-xband-antenna.jpg
  - http://www.bionik.tu-berlin.de/institut/skript/Fo15ES1.htm
  - http://www.biologie-schule.de/evolutionsfaktor-rekombination.php
  - https://blog.openai.com/evolution-strategies/
  - https://en.wikipedia.org/wiki/Reinforcement_learning#/media/File:Reinforcement_learning_diagram.svg
  - http://learningradiology.com/lectures/chestlectures/collagenvasculardz/Collagen%20Vascular%20Disease/data/images/img41.png
  - https://www.technologyreview.com/s/603916/a-new-direction-for-artificial-intelligence/

Quellen /2

- Literaturquellen:
  - Salimans, Tim ; Ho, Jonathan ; Chen, Xi ; Sutskever, Ilya: Evolution Strategies as a Scalable Alternative to Reinforcement Learning
  - Beyer, Hans-Georg ; Schwefel, Hans-Paul: Evolution strategies: A comprehensive introduction. In: Natural Computing 1 (2002), Nr. 1, S. 3-52.
  - Hornby, Gregory ; Globus, Al ; Linden, Derek ; Lohn, Jason: Automated
  - Antenna Design with Evolutionary Algorithms. In: Space 2006
  - Nissen, Volker: Einf□uhrung in Evolution□are Algorithmen: Optimierung nach dem Vorbild der Evolution. Wiesbaden : Vieweg+Teubner Verlag, 1997 (Computational Intelligence).
  - https://www.technologyreview.com/s/603916/a-new-direction-for-artificial-intelligence/