Elektronisches Zusatzmaterial (ESM)

Methoden des bestärkenden Lernens für die Produktionsablaufplanung

Autor: Sebastian Lang

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

A2C	–	Advantage Actor-Critic (bestärkendes Lernverfahren)
AC	–	Actor-Critic (Klasse von bestärkenden Lernverfahren)
BO	–	Bayes‘sche Optimierung (bestärkendes Lernverfahren)
DDPG	–	Deep Deterministic Policy Gradient
DQN	–	Deep Q-Networks (bestärkendes Lernverfahren)
EDD	–	Earliest Due Date (Prioritätsregel für die Reihenfolgebildung)
EPA	–	Entscheidungspolitik-approximierend (Klasse von bestärkenden Lernverfahren)
EXP	–	Experiment
F-ALLOK-KNN	–	Künstliches Neuronales Netz zur Allokation von Auftragsfamilien (Families) zu Ressourcen
GRU	–	Gated Recurrent Unit (rekurrentes Neuronenmodell)
ISBO	–	Integrated Simulation-Based Optimization (problemspezifische Heuristik)
J-ALLOK-KNN	–	Künstliches Neuronales Netz zur Allokation von Aufträgen (Jobs) zu Ressourcen
KNN	–	Künstliches Neuronales Netz
LST	–	Least Slack Time (Prioritätsregel für die Reihenfolgebildung)
LSTM	–	Long Short-Term Memory (rekurrentes Neuronenmodell)
MIN-WL	–	Minimum Workload (Allokationsregel für die Zuweisung von Aufträgen zu Ressourcen)
ML	–	Maschinelles Lernen
MLP	–	Multiple Layer Perceptron (mehrlagiges Perzeptron)
NEAT	–	NeuroEvolution of Augmenting Topologies (bestärkendes Lernverfahren)
PMP	–	Parallel-Maschinen-Problem (Ablaufplanungsproblem)
PPO	–	Proximal Policy Optimization (bestärkendes Lernverfahren)
PRE-SEQ-KNN	–	Künstliches Neuronales Netz zur Vorsequenzierung von Aufträgen
ReLU	–	Rectified Linear Unit (neuronale Aktivierungsfunktion)
RNN	–	Rekurrentes Neuronales Netz
SPT	–	Shortest Processing Time (Prioritätsregel für die Reihenfolgebildung)
Tanh	–	Tangens-Hyperbolicus
TD3	–	Twin Delayed Deep Deterministic Policy Gradient (bestärkendes Lernverfahren)
UML	–	Unified Modeling Language (vereinheitlichte Modellierungssprache)

MATHEMATISCHE NOTATION

Für Probleme der Produktionsablaufplanung

Indizes

i	Indexvariable für Ressourcen
j	Indexvariable für Aufträge
o	Indexvariable für Operationen
v	Indexvariable für Operationspläne

Auftragsbezogene Größen

$\|J\|$	Anzahl von Aufträgen
$\|B\|$	Anzahl von Auftragslosen
$\|b = 0\|$	Anzahl von Aufträgen im ersten Auftragslos zum Zeitpunkt 0

Ressourcenbezogene Größen

$\|M\|$	Anzahl von Ressourcen

Zeitbezogene Größen

$o_{i,j,v}$	Bearbeitungszeit von Operation o aus Operationsplan v von Auftrag j auf Ressource i
d_j	Fertigstellungsfrist (Due Date) von Auftrag j

Optimierungskriterien und Zielfunktionen

C_{max}	Gesamtdauer des Ablaufplans (Makespan)
T	Gesamtverspätung über alle Aufträge (Total Tardiness)

Für Künstliche Neuronale Netze (KNN)

Für mehrlagige Perzeptronen (MLP)

X	Datensatz von Merkmalen bzw. KNN-Eingaben
x	Eine beliebige Merkmalstichprobe bzw. KNN-Eingabe
x_i	Merkmal i aus Merkmalstichprobe / Eingabe für KNN-Eingabeneuron i
Y	Datensatz von Beobachtungen bzw. erwarteten KNN-Ausgaben (Trainingslabel)
y	Eine beliebige Beobachtung bzw. erwartete KNN-Ausgabe
l	Indexvariable für Neuronenschichten (Layer) eines KNN
L	Anzahl von Neuronenschichten bzw. Indexvariable für Ausgabeschicht eines KNN
θ	Menge der trainierbaren Parameter (synaptische Gewichte) eines KNN
$\theta_{i,j}^{(l)}$	Synaptisches Gewicht zwischen Neuron i in Schicht l und Neuron j in Schicht $l+1$
b bzw. $\theta_{0,j}^{(l)}$	Konstanter Bias-Term
v bzw. $v^{(l)}$ bzw. $v_i^{(l)}$	Gewichtete Eingabe bzw. gewichtete Eingabe für Schicht l (Vektor) bzw. für Neuron i in Schicht l (Skalar)
$\varphi(\cdot)$	Neuronale Aktivierungsfunktion
$a^{(l)}$ bzw. $a_i^{(l)}$	Ausgabe (Aktivierung) von Schicht l (Vektor) bzw. von Neuron i in Schicht l (Skalar)
$a^{(L)}$ bzw. $h_\theta(\cdot)$	KNN-Ausgabe (Hypothese)

$\delta^{(l)}$	Ausgabe- bzw. Prognosefehler von Schicht l

Für Rekurrente Neuronale Netze (RNN)

x_t	Eingabe für eine RNN-Zelle in Verarbeitungsschritt t
y_t	Ausgabe einer RNN-Zelle in Verarbeitungsschritt t
h_t	Versteckter Zustand einer RNN-Zelle in Verarbeitungsschritt t. Dient neben x_{t+1} als weitere Eingabe im nachfolgenden Verarbeitungsschritt $t+1$. Für alle in dieser Arbeit behandelten RNN-Modelle gilt: $h_t = y_t$
$W_{\cdot x}$ bzw. $W_{\cdot h}$	Trainierbarer Gewichte-Vektor von Gatter · für Eingabe x_t bzw. h_{t-1}
$b_{\cdot x}$ bzw. $b_{\cdot h}$	Konstanter Bias-Term von Gatter · für Eingabe x_t bzw. h_{t-1}
c_t	Zellzustand (Langzeitspeicher) einer LSTM-Zelle. Dient neben x_{t+1} als weitere Eingabe im nachfolgenden Verarbeitungsschritt $t+1$.
$\tilde{c}_t$	Zellgatter einer LSTM-Zelle
i_t	Eingangsgatter einer LSTM-Zelle
f_t	Forget-Gatter einer LSTM-Zelle
o_t	Ausgangsgatter einer LSTM-Zelle
r_t	Reset-Gatter einer GRU-Zelle
z_t	Aktualisierungsgatter einer GRU-Zelle
n_t	Kandidatengatter einer GRU-Zelle

Für Bayes'sche Optimierung (BO)

t	Indexvariable für Zufallsvariablen (Lösungskandidat)
T	Finite Menge aus mehrdimensionalen normalverteilten Zufallsvariablen
X, X', X_t	Eine ein- oder mehrdimensionale normalverteilte Zufallsvariable für Eingabewerte
Y, Y', Y_t	Eine eindimensionale normalverteilte Zufallsvariable für Ausgabewerte
x bzw. y	Ein konkreter Wert von X_t bzw. Y_t
f bzw. $f(x)$	Optimierungsproblem bzw. zu optimierende Fitnessfunktion
y^* bzw. $f(x^*)$	Fitness der bisher besten gefundene Lösung
$\mathcal{D}$	Menge aus historischen Beobachtungen von Eingangs- und Ausgangsdaten
$\mathcal{X}_D$ bzw. $\mathcal{Y}_\mathcal{D}$	Menge aus historischen Beobachtungen von (ein- oder mehrdimensionalen) Eingangs- bzw. (eindimensionalen) Ausgangsdaten
$\mathcal{X}_\mathcal{C}$	Menge aus Lösungskandidaten (ein- oder mehrdimensionale Eingangsdaten ohne Ausgangsdaten)
$\mathcal{GP}\big(\mu(x), \Sigma(x, x')\big)$	Gauß-Prozess
$\mu(x)$	Erwartungswertfunktion eines Gauß-Prozesses
$\Sigma(x, x')$	Kovarianzfunktion eines Gauß-Prozesses
$\|x - x'\|$	Euklidische Distanz zwischen x und x'
$u(x)$	Nutzenfunktion
$a_{EI}(x)$	Akquisitionsfunktion, welche die erwartete Verbesserung EI (Expected Improvement) berechnet
$\mathcal{N}_n\big(\mu(x), \Sigma(x, x)\big)$	Multivariate Normalverteilung mit n Dimensionen für Lösungskandidat x mit Erwartungswert $\mu(x)$ und Varianz $\Sigma(x, x)$
$\mathcal{N}_n\big(\cdot; \mu(x), \Sigma(x, x)\big)$	Wahrscheinlichkeitsdichtefunktion von $\mathcal{N}\big(\mu(x), \Sigma(x, x)\big)$

$\Phi_n\left(\cdot; \mu(x), \Sigma(x,x)\right)$	Verteilungsfunktion von $\mathcal{N}\left(\mu(x), \Sigma(x,x)\right)$

Statistische Kenngrößen

μ	Mittelwert
σ	Standardabweichung

ANHANG A – EXKURS: KÜNSTLICHE NEURONALE NETZE

Ein künstliches neuronales Netz (KNN) ist ein Modell des maschinellen Lernens (ML), welches dem menschlichen Gehirn nachempfunden ist (Alpaydin 2019, S. 275). Eine weitverbreitete Definition stammt von Haykin (1999, S. 24), der ein KNN als einen verteilten, hochparallelisiert arbeitenden Prozessor betrachtet, der aus einfachen Prozessoreinheiten, sogenannten »Neuronen«, besteht und die Fähigkeit zur Informationsspeicherung und -bereitstellung besitzt. Diese Definition mag schwer greifbar erscheinen, da Haykin ein KNN als eine adaptive Maschine betrachtet, die entweder physischer oder digitaler Natur ist. Im Rahmen dieser Arbeit bezieht sich der Begriff »künstliches neuronales Netz« auf ein algorithmisch-mathematisches Modell, das als Graph, bestehend aus Knoten und Kanten, versinnbildlicht werden kann. Ein Knoten wird als »Neuron« bezeichnet. Die Verbindungen zwischen den Knoten sind mit Gewichten versehen und werden als »Synapsen« bezeichnet.

Abbildung A-1 veranschaulicht den Aufbau und die Funktionsweise von KNN. Bildabschnitt (a) und (b) zeigen zwei beispielhafte KNN-Architekturen für ein Regressionsmodell (a) bzw. ein Klassifikationsmodell (b). Der Unterschied zwischen beiden Modellen liegt in der Interpretation des Ausgangssignals y. Bei einem Regressor-KNN repräsentiert das Ausgangssignal das prognostizierte Ergebnis für die Eingabe x. Die Anzahl der Ausgabeneuronen entspricht der Anzahl der zu berechnenden Größen (im Beispiel von Abbildung A-1 (a) wird lediglich eine Größe prognostiziert). Hingegen entspricht bei einem Klassifikator-KNN jedes Ausgabeneuron einer Klasse, zu welcher eine Eingabe x zugeordnet werden kann. In diesem Fall werden die Ausgangssignale als Wahrscheinlichkeiten interpretiert, zu welcher Klasse eine Eingabe x zuzuordnen ist (das Beispiel von Abbildung A-1 (b) berücksichtigt vier Klassen).

Des Weiteren ist in Abbildung A-1 zu sehen, dass Neuronen in Schichten organisiert sind. Ein KNN besteht aus mindestens einer Schicht. Die meisten KNN-Architekturen bestehen jedoch aus drei oder mehr Schichten, wobei mindestens eine Eingabeschicht die Eingangsdaten verarbeitet und mindestens eine Ausgabeschicht die Ergebnisdaten ausgibt. Die weiteren Schichten zwischen Eingabe- und Ausgabeschicht dienen der Identifikation von Mustern, Zusammenhängen und Abhängigkeiten in den Eingangsdaten. Bei beiden KNN in Abbildung A-1 (a) und (b) handelt es sich um sogenannte mehrlagige Perzeptronen (MLP). Hauptmerkmal von MLP-Modellen ist, dass die Neuronen einer Schicht (l) ausschließlich zu allen Neuronen der darauffolgenden Schicht $(l+1)$Verbindungen pflegen (Kruse et al. 2015, S. 43).

Der Aufbau und die Berechnungsvorschrift eines einzelnen Neurons wird in Abbildung A-1 (c) am Beispiel des dritten Neurons in der zweiten Schicht demonstriert. Es handelt sich hierbei im Wesentlichen um das von Rosenblatt (1958) entwickelte »Perzeptron«, welches als standardmäßiges Neuronenmodell des MLP gilt. Zunächst werden die empfangenen Signale $a_{1,\dots,4}^{<1>}$ gemeinsam mit den synaptischen Gewichten $\theta_{i,3}^{(1)}\ \forall i = (1,\dots,4)$, wobei i die Indizes der sendenden Neuronen repräsentiert, über eine Eingabefunktion zu einem Skalar v zusammengefasst. Bei einem Perzeptron wird die Eingabefunktion stets durch eine gewichtete Summe repräsentiert. Zudem wird ein sogenannter Bias-Term aufaddiert. Hintergrund der Verwendung des Bias-Terms ist, dass viele Neuronenmodelle analog zu ihrem biologischen Vorbild mit Schwellenwerten arbeiten. Die in das Neuron eingehenden Werte müssen gemeinsam den Schwellenwert überschreiten, sodass das Neuron aktiviert wird und eine Ausgabe ungleich null produziert. Die schwellenwertbasierte Aktivierung entspricht einer diskreten, nicht differenzierbaren Sprungfunktion.

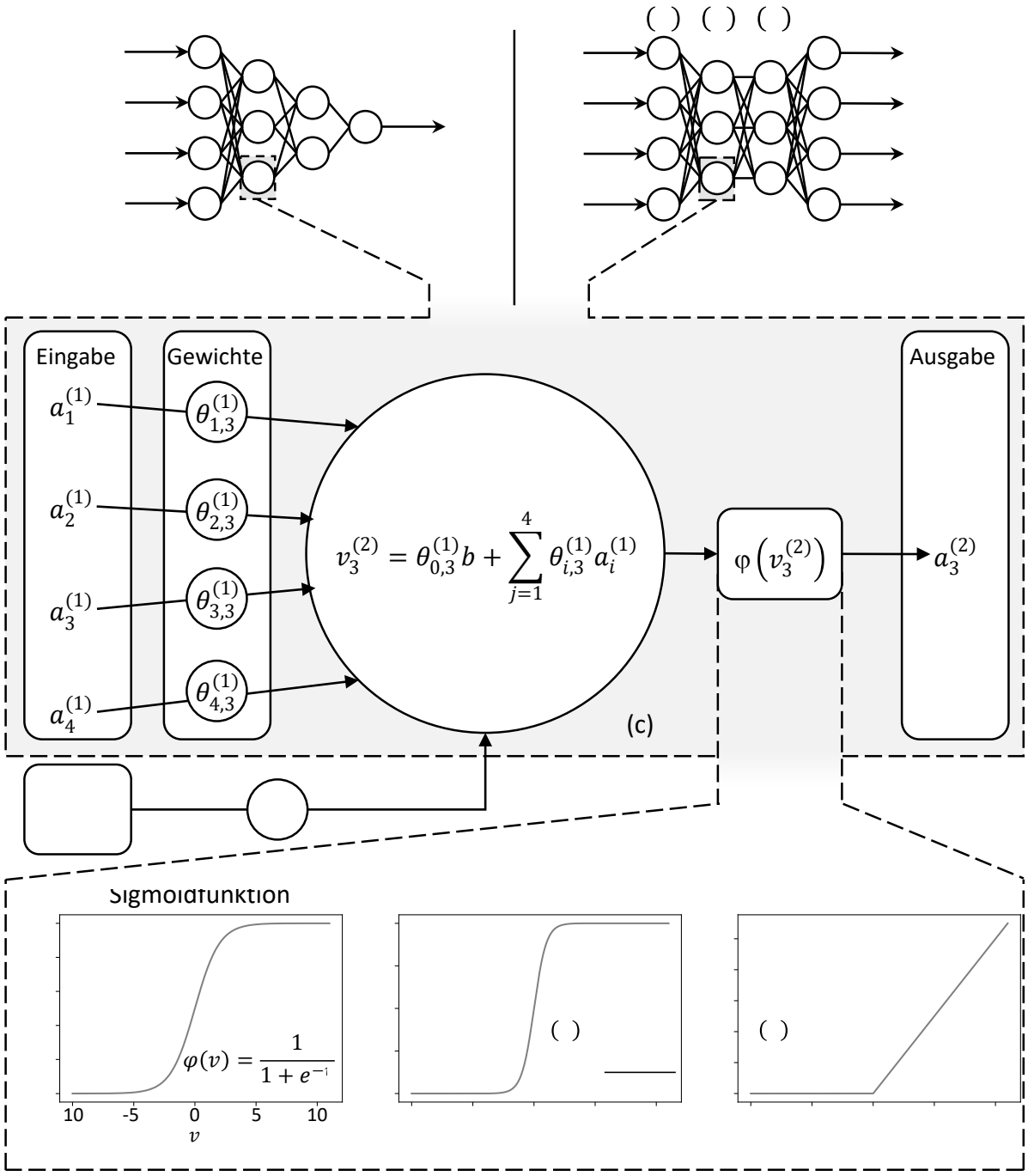


Abbildung A-1. Aufbau und Funktionsweise von künstlichen neuronalen Netzen: Beispielhafte KNN-Architekturen für (a) Regressions- bzw. (b) Klassifikationsprobleme, (c) Aufbau und Funktionsweise eines künstlichen Neurons, (d) Beispiele für neuronale Aktivierungsfunktionen

Die weitverbreiteten gradientenabhängigen Lernverfahren können aus diesem Grund nicht eingesetzt werden, da sie darauf angewiesen sind, dass ein ML-Modell vollständig differenzierbar ist (Kruse et al. 2015, S. 32). Statt eines Schwellenwerts wird zusätzlich ein Bias-Neuron eingeführt, das die Eingabe abschwächt oder auch verstärkt – je nachdem ob die verbindende Synapse einen positiven oder negativen Wert besitzt (vgl. Kriesel 2007, S. 46 f.). Aus mathematischer Sicht verschiebt der Bias-Term die Netzeingabe und somit die darauffolgende Aktivierungsfunktion nach oben oder unten (Haykin 1999, S. 33). Der Bias-Term entspricht somit dem Ordinatenabschnitt n einer linearen Gleichung in ihrer Normalform $y = f(x) = mx + n$. Darauffolgend geht die aggregierte Netzeingabe v als Parameter in die

Aktivierungsfunktion ein, welche die finale Ausgabe des Neurons berechnet. Bei der Aktivierungsfunktion handelt es sich um eine nichtlineare Funktion, wodurch das KNN in der Lage ist, ebenfalls nichtlineare Zusammenhänge zu erlernen. Abbildung A-1 (d) enthält drei Beispiele für häufig verwendete Aktivierungsfunktionen: Die Sigmoidfunktion bzw. logistische Funktion (links), die Tangens-Hyperbolicus-Funktion (mitte) sowie die Funktion »Rectified Linear Unit« (rechts).

Wie bereits angedeutet, sind gradientenabhängige Verfahren für das Anlernen von KNN am weitesten verbreitet. Die geläufigen Verfahren bauen auf dem Backpropagation-Algorithmus auf, dessen Grundprinzipien von Kelley (1960) für Steuerungsprobleme hergeleitet und erstmals von Werbos (1982) für das Anlernen von KNN adaptiert wurden. Die Idee des Backpropagation-Verfahrens ist durch Rückwärtsrechnung (Rückpropagierung) den Einfluss jedes Neurons auf den Fehler zwischen der KNN-Ausgabe und der erwarteten Ausgabe anteilsmäßig zu bestimmen. Hieraus können die Gradienten der KNN-Parameter abgeleitet werden, auf deren Basis eine schrittweise Parameteroptimierung erfolgt.

Abbildung A-2 demonstriert das Backpropagation-Verfahren anhand eines MLP-Modells mit drei Schichten. In der Literatur finden sich diverse Beschreibungen zur Implementierung des Backpropagation-Algorithmus, die sich im Detail voneinander unterscheiden. Das Beispiel in Abbildung A-2 wurde anhand der Implementierungsvorschrift von Ng (2012) entwickelt. Zum Zweck der Übersichtlichkeit sind alle Berechnungen in vektorisierter Form dargestellt. Ausgangsbasis ist eine Eingabe x, für welche das KNN eine Ausgabe prognostiziert (Vorwärtspropagierung). Im Beispiel von Abbildung A-2 handelt es sich um die Ausgabe der Aktivierungsfunktion der letzten Neuronenschicht $a^{(3)}$. Mithilfe einer Fehlerfunktion (z. B. absoluter oder quadratischer Fehler) wird die Differenz zwischen der prognostizierten und erwarteten Ausgabe quantifiziert.

Nun beginnt das eigentliche Backpropagation-Verfahren. Zunächst wird der anteilige Fehler jeder Neuronenschicht $\delta^{(l)}$ (ausgenommen der Eingabeschicht) berechnet. Der Fehler jeder versteckten Schicht $\delta^{(2,\dots,L-1)}$ resultiert aus der gewichteten Summe des Fehlers der nachfolgenden Schicht $\theta^{(l)}\delta^{(l+1)}$, elementweise multipliziert mit der ersten Ableitung der Aktivierungsfunktion φ' in Abhängigkeit von der gewichteten Schichteingabe $v^{(l)}$ (ermittelt durch Vorwärtspropagierung).

$$\delta^{(l)} = \theta^{(l)}\delta^{(l+1)} .* \varphi'\left(v^{(l)}\right) \tag{A.1}$$

Die Gradienten der Modellparameter $\nabla\theta^{(l)}$, respektive der synaptischen Gewichte, ergeben sich aus der Aktivierungsausgabe der Neuronenschicht $a^{(l)}$ (ermittelt durch Vorwärtspropagierung) multipliziert mit dem anteiligen transponierten Fehler der nachfolgenden Neuronenschicht $\left(\delta^{(l)}\right)^T$

$$\nabla\theta^{(l)} = a^{(l)}\left(\delta^{(l+1)}\right)^T \tag{A.2}$$

Die Gradienten können nun verwendet werden, um mit dem Gradientenabstiegsverfahren oder einem anderen gradientenabhängigen Optimierungsalgorithmus die KNN-Parameter schrittweise zu justieren.

Die in Abbildung A-1 und Abbildung A-2 dargestellten MLP-Modelle sind in ihren Analysefähigkeiten insoweit beschränkt, als dass sie nicht in der Lage sind, sequenzielle Zusammenhänge in Eingangsdatenströmen zu identifizieren. Bezugnehmend auf die Produktionsablaufplanung erscheint es insbesondere für die Reihenfolgeplanung von Aufträgen vorteilhaft, sequenzielle Zusammenhänge in den Eingangsdaten für die Entscheidungsfindung zu berücksichtigen. Als Beispiel sei ein KNN-Regressor

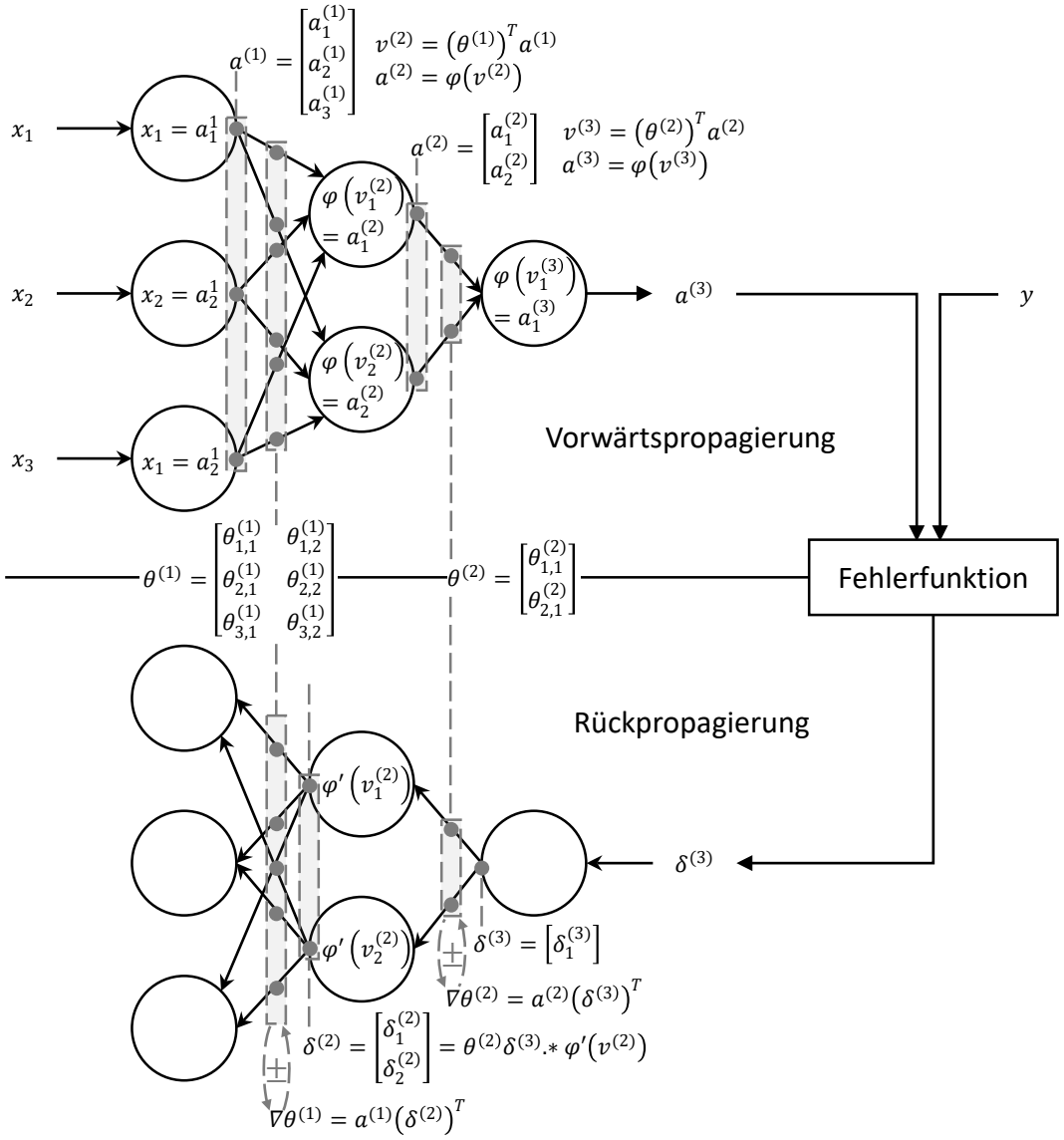


Abbildung A-2. Funktionsweise und Berechnungsvorschriften des Backpropagation-Algorithmus am Beispiel eines dreilagigen Perzeptrons

(Abbildung A-1 (a)) angenommen, der über sein Ausgabeneuron Aufträge in einer Warteschlange priorisiert. Nachdem alle Aufträge priorisiert wurden, wird derjenige Auftrag mit der höchsten Priorität für die Bearbeitung auf der Produktionsressource ausgewählt. Intuitiv erscheint es sinnvoll, dass der Agent bei der Priorisierung eines spezifischen Auftrags ebenfalls die Attribute der anderen auswählbaren Aufträge mitberücksichtigen sollte, um auf diese Weise Aufträge untereinander vergleichen zu können. Sogenannte rekurrente neuronale Netze (RNN) sind hierzu in der Lage, indem sie nicht nur vorwärtsgerichtete Verbindungen zu nachfolgenden Neuronen, sondern ebenfalls rückwärtsgerichtete (rekurrente) Verbindungen zu sich selbst oder zu vorgelagerten Neuronen pflegen.

Für die Konstruktion eines RNN eignet sich das Neuronenmodell »Perzeptron« nur bedingt. Hochreiter (1991) hat festgestellt, dass bei der Anwendung des Backpropagation-Verfahrens in vielschichtigen RNN, welche aus Perzeptronen bestehen, die Gradienten derjenigen versteckten Schichten, die näher zur Eingabe- als zur Ausgabeschicht liegen, entweder zu groß werden können oder gegen null tendieren. Wie in Abbildung A-2 ersichtlich, wird der Fehler bei der Rückpropagierung mit den synaptischen Gewichten zwischen den neuronalen Schichten multipliziert. Durch Gewichte, die größer als eins sind, tendiert der

Fehler dazu, in Korrelation mit der Anzahl der Schichten zu wachsen, durch welche der Fehler zurückgeführt wird. Neuronenschichten, die sich nahe der Eingabeschicht befinden, neigen hierdurch zu übermäßig starken Änderungen der synaptischen Gewichte. Sind synaptische Gewichte wiederum kleiner als eins, kann es dazu kommen, dass der Fehler in Korrelation mit der Rückführung durch die Schichten schrumpft. Die synaptischen Gewichte von Neuronenschichten, die sich nahe an der Eingabeschicht befinden, erfahren hierdurch nur geringe oder gar keine Veränderungen während des Trainings.

Moderne rekurrente Neuronenmodelle, die für die Modellierung von KNN mit mehreren versteckten Schichten (Deep Learning) eingesetzt werden, sind in ihrem Aufbau und ihrer Funktionsweise wesentlich komplexer als das klassische Perzeptron. Sie besitzen verschiedene Eingänge und Ausgänge – sogenannte Gatter –, die den Zufluss und Abfluss von Informationen steuern, um so die Gradienten bei Anwendung des Backpropagation-Verfahrens relativ konstant zu halten. Vor diesem Hintergrund werden moderne rekurrente Neuronenmodelle oftmals als Zellen bezeichnet. Abbildung A-3 illustriert ein allgemeines Modell einer RNN-Zelle. Im Unterschied zum Perzeptron verarbeitet eine RNN-Zelle nicht einen einzelnen Eingabevektor x, sondern eine zusammenhängende Sequenz von Eingabevektoren $x_t \forall t = (1, \dots, n)$, um eine entsprechende Sequenz von Ausgabevektoren $y_t \forall t = (1, \dots, n)$ zu berechnen. Die Berechnung ist deshalb rekurrent, weil in Verarbeitungsschritt t nicht nur die Eingabe x_t berücksichtigt wird, sondern ebenfalls Informationen aus den vorherigen Verarbeitungsschritten, die im versteckten Zustand h_t subsumiert werden. Abbildung A-3 (a) veranschaulicht die Architektur einer RNN-Zelle. Abbildung A-3 (b) verdeutlicht das Prinzip der Datenverarbeitung, indem die RNN-Zelle entlang der Verarbeitungsschritte $t-1$, t und $t+1$ entfaltet dargestellt wird. Zwei populäre rekurrente Neuronenmodelle sind die LSTM-Zelle und die GRU-Zelle, die im Folgenden kurz vorgestellt werden sollen.

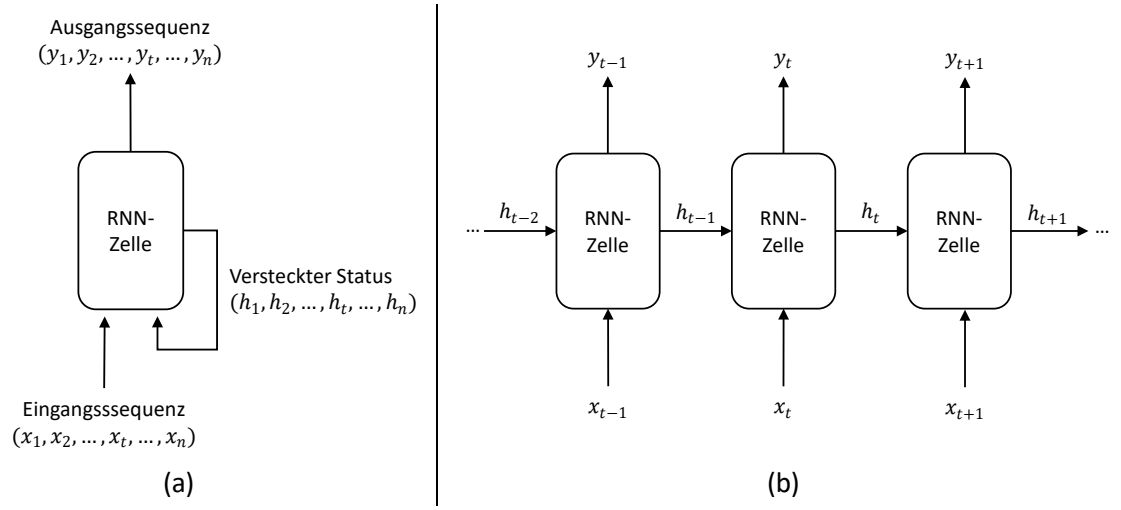
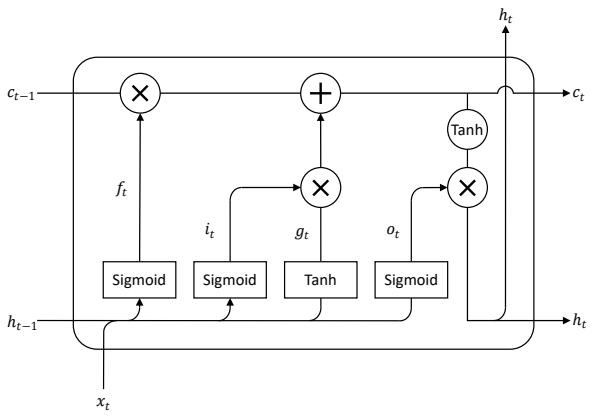


Abbildung A-3. Allgemeines Modell einer RNN-Zelle (a) und Veranschaulichung des Datenflusses einer RNN-Zelle entlang von drei Verarbeitungsschritten (b)

Die LSTM (Long Short-Term Memory) -Zelle wurde von Hochreiter und Schmidhuber (1997) entwickelt. Es handelt sich um den Urtyp der in Abbildung A-3 dargestellten RNN-Zelle. Abbildung A-4 veranschaulicht den Aufbau einer LSTM-Zelle (a) sowie ihre Berechnungsvorschriften zur Verarbeitung von Daten (b). Bei der abgebildeten Variante handelt es sich um die von Gers et al. (2000) weiterentwickelte LSTM-Zelle, welche ebenfalls in den ML-Bibliotheken PyTorch und TensorFlow implementiert ist. Sie besitzt ein zusätzliches Gatter, das sogenannte »Forget«-Gatter f_t, welches die Speicherung bzw. Überschreibung von Informationen aus vergangenen Verarbeitungsschritten regelt (Formel (A.3)). Ferner besitzt die LSTM-Zelle ein Eingangsgatter i_t, ein Zellgatter $\tilde{c}_t$ und ein Ausgangsgatter o_t. Die vier Gatter werden durch drei



(a)

$$f_t = Sigm(W_{fx}x_t + b_{fx} + W_{fh}h_{t-1} + b_{fh}) \quad \text{(A.3)}$$

$$i_t = Sigm(W_{ix}x_t + b_{ix} + W_{ih}h_{t-1} + b_{ih}) \quad \text{(A.4)}$$

$$\tilde{c}_t = Tanh(W_{gx}x_t + b_{gx} + W_{gh}h_{t-1} + b_{gh}) \quad \text{(A.5)}$$

$$o_t = Sigm(W_{ox}x_t + b_{ox} + W_{oh}h_{t-1} + b_{oh}) \quad \text{(A.6)}$$

$$c_t = f_t .* c_{t-1} + i_t .* \tilde{c}_t \quad \text{(A.7)}$$

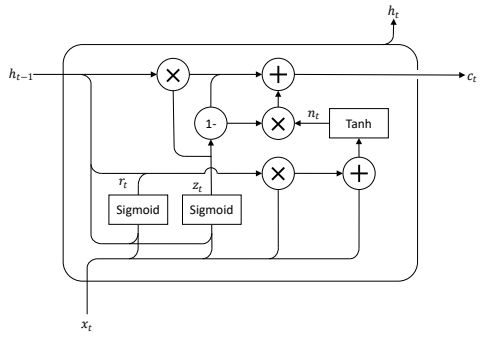
$$h_t = y_t = o_t .* Tanh(c_t) \quad \text{(A.8)}$$

(b)

Abbildung A-4. Aufbau (a) und Berechnungsvorschriften (b) einer LSTM-Zelle

Datenquellen gespeist, nämlich den Eingabevektor x_t, den versteckten Zustand h_{t-1} und den Zellzustand c_{t-1}. Die LSTM-Zelle gibt Ergebnisse über drei Ausgänge aus, nämlich den Ausgabevektor y_t, den versteckten Zustand h_t sowie den Zellzustand c_t. Bei der LSTM-Zelle sind y_t und h_t identisch. Die Unterscheidung in Abbildung A-4 (a) dient zum einen der Veranschaulichung, welche Ergebnisse auf welche Weise weiterverarbeitet werden, und zum anderen der Konformität mit dem in Abbildung A-3 (a) dargestellten verallgemeinerten Modell einer RNN-Zelle. In Abbildung A-4 (a) repräsentieren rechteckige Blöcke Schichten, die Eingaben vektorisiert verarbeiten, während kreisrunde Symbole auf eine elementweise Ausführung von Operationen hinweisen. Jedes Gatter · ist durch einen trainierbaren Gewichte-Vektor für Eingaben $W_{\cdot,x}$ und für versteckte Zustände $W_{\cdot,y}$ sowie durch konstante Bias-Terme $b_{\cdot,x}$ bzw. $b_{\cdot,y}$ definiert. Analog zum Perzeptron werden die Gewichte-Vektoren innerhalb der Zellen mittels eines Gradientenabstiegsverfahrens trainiert. Die Gradienten werden durch das Backpropagation-Verfahren ermittelt. Dies ist möglich, da alle Berechnungsschritte innerhalb einer LSTM-Zelle vollständig differenzierbar sind. In jedem Verarbeitungsschritt erhält die LSTM-Zelle die aktuelle Eingabe x_t sowie den Zellzustand und die Ausgabe des vorherigen Verarbeitungsschritts c_{t-1} bzw. h_{t-1}. Für den ersten Verarbeitungsschritt werden c_{t-1} und h_{t-1} mit null initialisiert. Das Forget-Gatter, das Eingangsgatter und das Ausgangsgatter steuern wie viele Informationen vergessen, eingelassen bzw. ausgegeben werden. Alle drei Gatter transformieren ihre jeweilige Eingabe mittels Sigmoid-Funktion in einen Wertebereich zwischen null und eins. Das Zellgatter $\tilde{c}_t$ steuert, inwiefern die Eingangsdaten x_t und h_{t-1} im Zellzustand berücksichtigt werden. Während der versteckte Zustand h_{t-1} lediglich die Ausgabe aus dem vorherigen Verarbeitungsschritt darstellt, repräsentiert der Zellzustand c_{t-1} den Langzeitspeicher der LSTM-Zelle. Mit der Tanh-Funktion wird die Linearkombination der Eingangsdaten x_t und h_{t-1} in einen Wertebereich zwischen minus eins und eins transformiert. Die Formeln (A.3–6) dienen lediglich der Datenvorverarbeitung. Die Hauptberechnungsschritte innerhalb der LSTM-Zelle finden in den Formeln (A.7) und (A.8) statt. Formel (A.7) beschreibt die Aktualisierung des Zellzustands. Der erste Term drückt aus, welche Informationen aus dem Zellzustand des vorherigen Verarbeitungsschritts vergessen bzw. beibehalten werden. Zu diesem Zweck wird die Ausgabe des Forget-Gatters f_t elementweise mit dem alten Zellzustand c_{t-1} multipliziert. Der zweite Term beschreibt, welche Informationen aus der aktuellen Eingabe im neuen Zellzustand gespeichert werden sollen. Dies wird über die elementweise Multiplikation der Ergebnisse des Eingangs- und Zellgatters realisiert. Formel (A.8) berechnet die Zellenausgabe bzw. den versteckten Zustand, der in den darauffolgenden Verarbeitungsschritt miteinbezogen wird. Konkret wird das Ergebnis des Ausgabegatters mit dem Zellzustand gewichtet. Abschließend wird jedes Element des Zellzustands mittels Tanh-Funktion in einen Wertebereich zwischen minus eins und eins transformiert.

Bei der GRU (Gated Recurrent Unit) -Zelle handelt es sich im Wesentlichen um eine Vereinfachung der LSTM-Zelle, indem sie auf die Verwaltung eines Zellzustands verzichtet. Die GRU-Zelle wurde von Cho et al. (2014) entwickelt. Abbildung A-5 illustriert den Aufbau und die Berechnungsvorschriften der GRU-Zelle.



(a)

$$r_t = Sigm(W_{rx}x_t + b_{rx} + W_{rh}h_{t-1} + b_{rh}) \quad \text{(A.10)}$$

$$z_t = Sigm(W_{zx}x_t + b_{zx} + W_{zh}h_{t-1} + b_{zh}) \quad \text{(A.11)}$$

$$n_t = Tanh(W_{nx}x_t + b_{nx} + r_t .* (W_{nh}h_{t-1} + b_{nh})) \quad \text{(A.12)}$$

$$h_t = y_t = (1 - z_t) .* n_t + z_t .* h_{t-1} \quad \text{(A.13)}$$

(b)

Abbildung A-5. Aufbau (a) und Berechnungsvorschriften (b) einer GRU-Zelle

Analog zu Abbildung A-4 (a) weisen in Abbildung A-5 (a) rechteckige Blöcke auf Schichten hin, die Eingaben vektorisiert verarbeiten, während kreisrunde Symbole auf eine elementweise Ausführung von arithmetischen Operationen hinweisen. In die GRU-Zelle fließen zwei Eingaben, nämlich der Eingabevektor x_t und der versteckte Zustand bzw. der Ausgabevektor aus dem vorherigen Verarbeitungsschritt h_{t-1}. Die GRU-Zelle produziert lediglich eine Ausgabe, die sowohl den Ausgabevektor y_t als auch den versteckten Zustand h_t der Zelle repräsentiert. Die Datenvorverarbeitung erfolgt über drei Gatter, die jeweils analog zur LSTM-Zelle anlernbare Gewichte-Vektoren enthalten. Die Grundidee der GRU-Zelle ist, dass auf Basis des Eingabevektors x_t und der vergangenen Ausgabe h_{t-1} eine Kandidatenausgabe über das sogenannte Kandidatengatter n_t berechnet wird (Formel (A.12)). Zuvor wird über das Reset-Gatter r_t bestimmt, zu welchem Grad die Ausgabe aus dem vergangenen Verarbeitungsschritt h_{t-1} in der neuen Kandidatenausgabe berücksichtigt werden soll (Formel (A.10)). Das Ergebnis des Reset-Gatters geht als Term in das Kandidaten-Gatter mit ein. Es wird verwendet, um mittels elementweiser Multiplikation die vergangene Ausgabe h_{t-1} zusätzlich zu gewichten, bevor diese in die aktuelle Eingabe x_t miteinbezogen wird. Der Lösungskandidat wird schließlich mittels Tanh-Funktion in einen Wertebereich zwischen minus eins und eins transformiert. Die eigentliche Ausgabe h_t wird über Polyak-Mittelung der Kandidatenausgabe n_t und der vergangenen Ausgabe h_{t-1} berechnet (Formel (A.13)). Der Vektor, der als Gewichtungskoeffizient für die Polyak-Mittelung dient, wird durch das Aktualisierungsgatter z_t berechnet (Formel (A.11)).

Abschließend sollen zwei unterschiedliche Arten der Dateneingabe für RNN-Zellen erklärt werden. Standardmäßig wird eine Sequenz von Eingabevektoren unidirektional von $t = 1$ bis T eingelesen. Diese Art der Dateneingabe ist sinnvoll, sofern die Eingabesequenz eine Zeitreihe beschreibt. Im Beispiel der Reihenfolgeplanung unterliegt die Anordnung von Aufträgen in einem Puffer keiner spezifischen Ordnung, sodass ein RNN bei einer unidirektionalen Verarbeitung der Auftragsdaten womöglich falsche Schlüsse zieht. Ein konkretes Problem, das vor diesem Hintergrund auftritt, ist, dass bei der Priorisierung des ersten Auftrags in einem Puffer keine Informationen zu weiteren Prioritäten vorliegen, während bei der Priorisierung des letzten Auftrags das RNN alle zuvor vergebenen Prioritäten in die Berechnung miteinbeziehen kann. Um alle Aufträge gleichermaßen zu berücksichtigen, kann eine Sequenz von Eingabevektoren ebenfalls bidirektional eingelesen werden. Hierbei wird die Sequenz erst von vorne nach hinten ($t = 1, \dots, T$) und darauffolgend von hinten nach vorn ($t = T, \dots, 1$) eingelesen. Es resultiert ein Ausgabevektor, der doppelt so groß ist wie die eingelesene Sequenz. Dieser kann in einer daran anschließenden versteckten

Schicht weiterverarbeitet werden, die entsprechend der Größe des Ausgabevektors doppelt so groß sein muss wie die vorgelagerte RNN-Schicht. Alternativ ist es möglich einen Mittelwert über die Ausgaben der vorwärts- und rückwärts eingelesenen Sequenz zu bilden und den resultierenden Tensor vorwärts durch die darauffolgenden Schichten zu propagieren. In diesem Fall besitzt die unmittelbar nachfolgende versteckte Schicht lediglich genauso viele Neuronen wie die vorgelagerte RNN-Schicht.

ANHANG B – EXKURS: A2C-ALGORITHMUS

Der A2C (Advantage Actor-Critic) -Algorithmus gehört der Klasse der Actor-Critic (AC) -Verfahren an, deren Grundlagen bereits in Abschnitt 3.3.5 im Hauptwerk beschrieben wurden. Wie jedes AC-Verfahren kombiniert der Agent ein Actor-Modell, das die Entscheidungspolitik repräsentiert, mit einem Critic-Modell, welches eine Nutzenfunktion approximiert und die erwartete Gesamtbelohnung zurückgibt. Konkret handelt es sich beim A2C-Algorithmus um ein stochastisches entscheidungspolitik-approximierendes (EPA) -Verfahren, d. h. das Actor-Modell lernt eine stochastische Entscheidungspolitik $\pi(a|s;\theta_A)$, mit der zufällige Aktionen stichprobenartig ermittelt werden, während das Critic-Modell die Zustandsnutzenfunktion $v_\pi(s;\theta_C)$ erlernt, welche die erwartete kumulierte Belohnung ausgehend vom aktuellen Zustand ausgibt. Abbildung B-1 illustriert die Architektur und das Funktionsprinzip des A2C-Algorithmus.

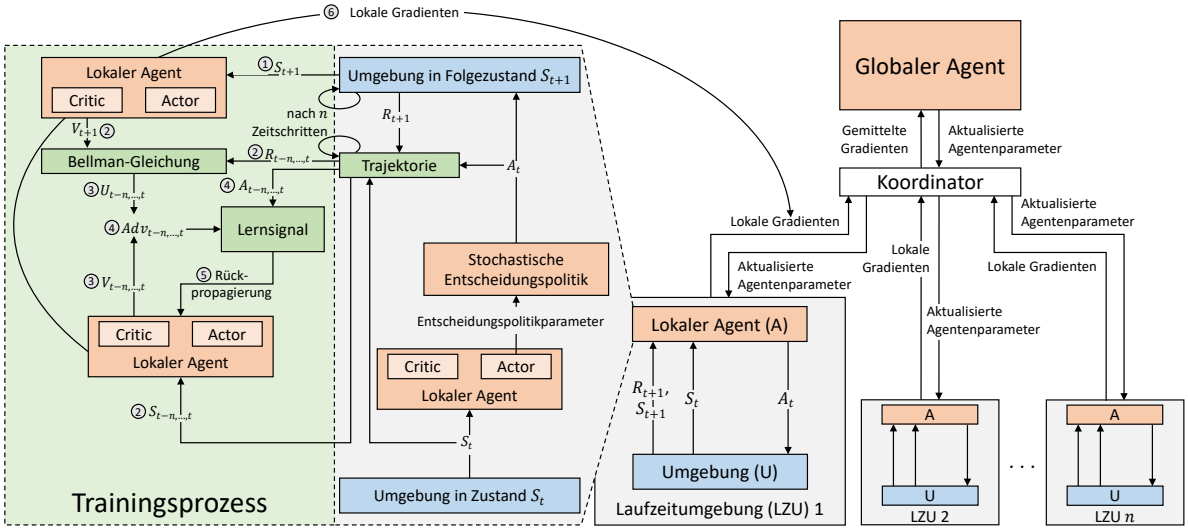


Abbildung B-1. A2C-Architektur und algorithmischer Ablauf

Der A2C-Algorithmus wird klassischerweise parallelisiert auf mehreren Prozessorkernen ausgeführt. Der Grund ist, dass Lernsignale mithilfe des N-Step-Bootstrapping-Verfahrens (siehe Abschnitt 3.3.5 im Hauptwerk) erzeugt werden und durch Parallelisierung mehrere Trajektorien (und somit mehr Lerndaten) gleichzeitig in unterschiedlichen Laufzeitumgebungen generierbar sind. In jeder Laufzeitumgebung agiert ein lokaler Agent mit einer Instanz der Umgebung gemäß dem Prozessmodell für das gradientenabhängige bestärkende Lernen (siehe Abbildung 3-5 in Abschnitt 3.3.1 im Hauptwerk). Der Aufbau und Trainingsprozess eines lokalen Agenten ist am Beispiel der ersten Laufzeitumgebung im linken Teil von Abbildung B-1 dargestellt. In jedem Zustand S_t prognostiziert das Actor-Modell des Agenten die Parameter (z. B. Mittelwert und Standardabweichung bei kontinuierlichen Aktionsraum) der stochastischen Entscheidungspolitik (z. B. Normalverteilung bei kontinuierlichen Aktionsraum). Auf Basis der Parameter wird eine zufällige Aktion A_t als Stichprobe gezogen und angewandt. Die Umgebung geht in den Folgezustand S_{t+1} über. Damit einhergehend wird die Belohnung R_{t+1} zurückgegeben. Der 3-Tupel (S_t, A_t, R_{t+1}) wird der Trajektorie hinzugefügt. Sobald der Trajektorie n (S_t, A_t, R_{t+1})-Tupel hinzugefügt wurden (wobei n benutzerdefinierbar ist), wird der Trainingsprozess initiiert.

In Schritt (1) des Trainingsprozesses wird zunächst der Folgezustand S_{t+1} vorwärts durch das Critic-Modell propagiert, um den Zustandsnutzen V_{t+1} zu ermitteln. Ausgehend von V_{t+1} werden unter Verwendung der Belohnungen aus dem Trainingslos $R_{t-n,\dots,t}$ die Ziel-Aktionsnutzen $U_{t-n,\dots,t}$ mittels N-Step-Bootstrapping ermittelt (2). Die exakte Berechnungsvorschrift zur Ermittlung der Ziel-Aktionsnutzen ist in

Formel (3.20) in Abschnitt 3.3.5 des Hauptwerks definiert. Des Weiteren werden in Ablaufschritt (2) die Zustandsnutzen $V_{t-n,\dots,t}$ für die Zustände $S_{t-n,\dots,t}$ der Trajektorie prognostiziert. In Schritt (3) werden die prognostizierten Zustandsnutzen $V_{t-n,\dots,t}$ von den Ziel-Aktionsnutzen $U_{t-n,\dots,t}$ subtrahiert. Das Ergebnis sind die Advantage-Werte $Adv_{t-n\dots t}$ für die Zeitschritte $t-n, \dots, t$.

$$Adv_{t-n\dots t} = U_{t-n,\dots,t} - V_{t-n,\dots,t} \tag{B.1}$$

Der Advantage repräsentiert somit eine Prognose, wie stark sich die erwartete Gesamtbelohnung durch die Auswahl einer Aktion ändert verglichen zu der erwarteten Gesamtbelohnung in demjenigen Zustand, in welchem die Aktion gewählt wird. Auf Basis der Advantage-Werte werden die Lernsignale für das Actor- und Critic-Modell generiert. Das Lernsignal des Critic-Modells berechnet sich wie folgt.

$$\delta_C = \frac{1}{n}\sum_{k=t-n}^{t} Adv_k^2 \tag{B.2}$$

Zur Generierung des Lernsignals für das Actor-Modell wird das Policy Gradient Theorem (siehe Abschnitt 3.3.4 im Hauptwerk) herangezogen. Die logarithmische Wahrscheinlichkeit für den Eintritt von Aktion A_t unter Entscheidungspolitik $\pi(a|s;\theta_A)$ wird anstelle des Aktionsnutzen mit dem jeweiligen Advantage Adv_t multipliziert. Unter Verwendung des Aktionsvektors $A_{t-n,\dots,t}$ der Trajektorie ergibt sich die folgende Berechnungsvorschrift für das Lernsignal.

$$\delta_A = \frac{1}{n}\sum_{k=t-n}^{t}(Adv_k * \log \pi(A_k|S_k;\theta_A)) \tag{B.3}$$

Die Lernsignale δ_A und δ_C werden entweder individuell rückwärts durch das jeweilige Actor- bzw. Critic-Modell propagiert, sofern Actor und Critic als separate Modelle implementiert sind. Alternativ teilen sich das Actor- und das Critic-Modell dieselbe Eingabeschicht und bilden somit ein gemeinsames Modell. In diesem Fall wird ein gemitteltes Lernsignal aus δ_A und δ_C gebildet, das rückwärts durch das gemeinsame Modell propagiert wird. Das Ergebnis sind die Gradienten des lokalen Agenten der jeweiligen Laufzeitumgebung.

Der beschriebene Trainingsprozess läuft in mehreren Laufzeitumgebungen parallel ab. Im A2C-Algorithmus wird die Interaktion zwischen Agent und Umgebung nach jeweils n Zeitschritten über alle Laufzeitumgebungen synchronisiert. Ein Koordinator-Modul sorgt dafür, dass nach n Zeitschritten der Prozess einer Laufzeitumgebung unterbrochen wird, bis in allen weiteren Laufzeitumgebungen dieselbe Anzahl von Zeitschritten absolviert und die Gradienten des jeweiligen lokalen Agenten berechnet wurden. Das Koordinator-Modul sammelt die lokalen Gradienten der jeweiligen Laufzeitumgebungen und berechnet deren Mittelwert. Auf Basis der gemittelten Gradienten werden die Parameter eines globalen Agenten aktualisiert. Danach werden die Parameter aller lokalen Agenten mit den aktualisierten Parametern des globalen Agenten überschrieben. Der Prozess wiederholt sich alle n Zeitschritte, bis alle Umgebungen in einem finalen Zustand S_T terminieren.

ANHANG C – EXKURS: PPO-ALGORITHMUS

Der von Schulman et al. (2017) entwickelte PPO (Proximal Policy Optimization) -Algorithmus ähnelt zu großen Teilen dem in Anhang B beschriebenen A2C-Algorithmus. Analog zum A2C-Algorithmus handelt es sich um ein AC-Verfahren, in dem das Actor-Modell eine stochastische Entscheidungspolitik approximiert. Gleichermaßen verwendet der PPO-Algorithmus das N-Step-Bootstrapping-Verfahren (siehe Formel (3.20) in Abschnitt 3.3.5 im Hauptwerk) für die Ermittlung von Ziel-Aktionsnutzen und berechnet Advantage-Werte gemäß Formel (B.1) zur Generierung von Lernsignalen. Des Weiteren wird im PPO-Algorithmus das Critic-Modell nach derselben Berechnungsvorschrift wie im A2C-Algorithmus aktualisiert (Formel (B.2)). Bis auf einige wenige Unterschiede, die im Folgenden erläutert werden, gelten die in Abbildung B-1 gezeigte Architektur und der algorithmische Ablauf des A2C-Algorithmus ebenfalls für den PPO-Algorithmus.

Der PPO-Algorithmus unterscheidet sich vom A2C-Algorithmus insbesondere hinsichtlich des Lernsignals des Actor-Modells (d. h. Formel (B.3) findet keine Anwendung). Dem PPO-Algorithmus liegt die Überlegung zugrunde, dass bereits kleine Veränderungen in den Parametern des Actor-Modells zu großen Veränderungen in der resultierenden Entscheidungspolitik führen können. Infolgedessen aktualisiert der PPO-Algorithmus die Actor-Modellparameter über den sogenannten »Surrogate-Advantage« und begrenzt das Ausmaß der Aktualisierung über ein Clipping-Verfahren. Der Surrogate-Advantage $SurAdv_t$ resultiert aus dem Advantage Adv_t gewichtet mit dem Verhältnis zwischen der logarithmischen Auswahlwahrscheinlichkeit einer Aktion A_t unter der aktuellen Entscheidungspolitik $\pi(A_t|S_t;\theta_A)$ und unter der alten Entscheidungspolitik $\pi(A_t|S_t;\theta_A^-)$. Analog zum A2C-Algorithmus wird je Trainingsschritt eine vollständige Trajektorie verarbeitet. Aus diesem Grund zeigt die folgende Formel (C.1) die Berechnung des Surrogate-Advantage in vektorisierter Form für eine Trajektorie mit n Transitionen

$$SurAdv_{t-n,\dots,t} = \frac{\log\pi(A_{t-n,\dots,t}|S_{t-n,\dots,t};\theta_A)}{\log\pi(A_{t-n,\dots,t}|S_{t-n,\dots,t};\theta_A^-)} * Adv_{t-n,\dots,t} \tag{C.1}$$

Der PPO-Algorithmus verwendet den Surrogate-Advantage für die Aktualisierung des Actor-Modells als Lernsignal. Die Aktualisierungsvorschrift unterscheidet sich vom A2C-Algorithmus (Formel B.3) dadurch, dass die logarithmische Auswahlwahrscheinlichkeit für eine Aktion A_t unter der aktuellen Entscheidungspolitik $\pi(a|s;\theta_A)$ zusätzlich mit der logarithmischen Auswahlwahrscheinlichkeit für dieselbe Aktion A_t unter der vergangenen Entscheidungspolitik $\pi(a|s;\theta_A^-)$ ins Verhältnis gesetzt wird. Wie bereits beschrieben, wird im PPO-Algorithmus das Ausmaß der Parameteraktualisierung darüber hinaus durch ein Clipping-Verfahren beschränkt. Genau genommen begrenzt das Clipping-Verfahren das Verhältnis zwischen den Auswahlwahrscheinlichkeiten $\log\pi(A_t|S_t;\theta_A)$ und $\log\pi(A_t|S_t;\theta_A^-)$ auf ein Intervall $[1-\epsilon,\dots,1+\epsilon]$, wobei ϵ ein benutzerdefinierbarer Hyperparameter ist. Anders ausgedrückt steuert das Clipping-Verfahren den maximal tolerierten Unterschied zwischen der aktuellen und alten Entscheidungspolitik. Der Surrogate-Advantage (Formel (C.1)) und das Clipping-Verfahren werden zu der folgenden Berechnungsvorschrift kombiniert, um das Lernsignal δ_A für das Actor-Modell zu ermitteln.

$$\delta_A = \frac{1}{n}\sum_{k=t-n}^{t} \min\left(SurAdv_k\ ,\ \left(\text{clip}\left(\frac{\log\pi(A_k|S_k;\theta_A)}{\log\pi(A_k|S_k;\theta_A^-)}, 1-\epsilon, 1+\epsilon\right) * Adv_k\right)\right) \tag{C.2}$$

Die Intuition von Formel (C.2) ist, dass für jede Transition k der Trajektorie $t-n,\dots,t$ nur dann der wahre Surrogate-Advantage im Lernsignal berücksichtigt wird, wenn dieser kleiner als der beschränkte Surrogate-Advantage ist. Andernfalls geht für Transition k der beschränkte Surrogate-Advantage in das Lernsignal mit ein. Formel (C.2) berechnet somit eine untere Schranke des Surrogate-Advantages. Die

Wahrscheinlichkeit einer gewählten Aktion A_k, welche die erwartete Gesamtbelohnung erhöht ($Adv_k > 0$), wird in Abhängigkeit von maximal $(1 + \epsilon) * Adv_k$ vergrößert. Hingegen wird die Wahrscheinlichkeit einer Aktion gewählten A_k, welche die erwartete Gesamtbelohnung reduziert ($Adv_k < 0$), in Abhängigkeit von maximal $(1 - \epsilon) * Adv_k$ verringert.

Bezugnehmend auf die Architektur und den algorithmischen Ablauf des A2C-Algorithmus in Abbildung B-1 erfordert die Veränderung des Actor-Lernsignals, dass der PPO-Algorithmus für jede Transition zusätzlich die logarithmische Wahrscheinlichkeit für die Auswahl einer Aktion speichert. Demzufolge enthält der Trajektorienspeicher im PPO-Algorithmus keine 3-Tupel-Transitionen (S_t, A_t, R_{t+1}), sondern 4-Tupel-Transitionen $(S_t, A_t, \log \pi(A_t|S_t, \theta_A), R_{t+1})$. Ein weiterer Unterschied zum A2C-Algorithmus ist, dass die Trajektorie während eines Trainingsschritts nicht einmal, sondern über eine benutzerdefinierte Anzahl von Epochen für die Berechnung des Lernsignals verwendet wird. Hierbei sind in der ersten Epoche eines Trainingsschritts die Entscheidungspolitiken $\pi(a|s; \theta_A)$ und $\pi(a|s; \theta_A^-)$ identisch. Erst in den darauffolgenden Epochen wird für $\pi(a|s; \theta_A)$ stets die resultierende Entscheidungspolitik der jeweils vorausgegangenen Epoche verwendet, während $\pi(a|s; \theta_A^-)$ über alle Epochen innerhalb desselben Trainingsschritts konstant gehalten wird.

ANHANG D – EXKURS: DDPG-ALGORITHMUS

Der DDPG (Deep Deterministic Policy Gradient) -Algorithmus wurde von Lillicrap et al. (2015) entwickelt. Es handelt sich um ein AC-Verfahren (siehe Abschnitt 3.3.5 im Hauptwerk), bei dem das Actor-Modell eine deterministische Entscheidungspolitik $\pi(s;\theta_A)$ und das Critic-Modell die Aktionsnutzenfunktion $q_\pi(s,a;\theta_C)$ approximiert. Hierdurch unterscheidet sich DDPG von den meisten anderen AC-Verfahren, die eine stochastische Entscheidungspolitik $\pi(a|s;\theta_A)$ erlernen. Eine weitere Besonderheit von DDPG ist, dass der Algorithmus ausschließlich für Probleme mit kontinuierlichen Aktionsraum anwendbar ist. Vor diesem Hintergrund kann DDPG als Pendant zum DQN-Algorithmus verstanden werden, der sich lediglich für Probleme mit diskreten Aktionsräumen eignet. Abbildung D-1 illustriert die Architektur und den Ablauf des DDPG-Algorithmus.

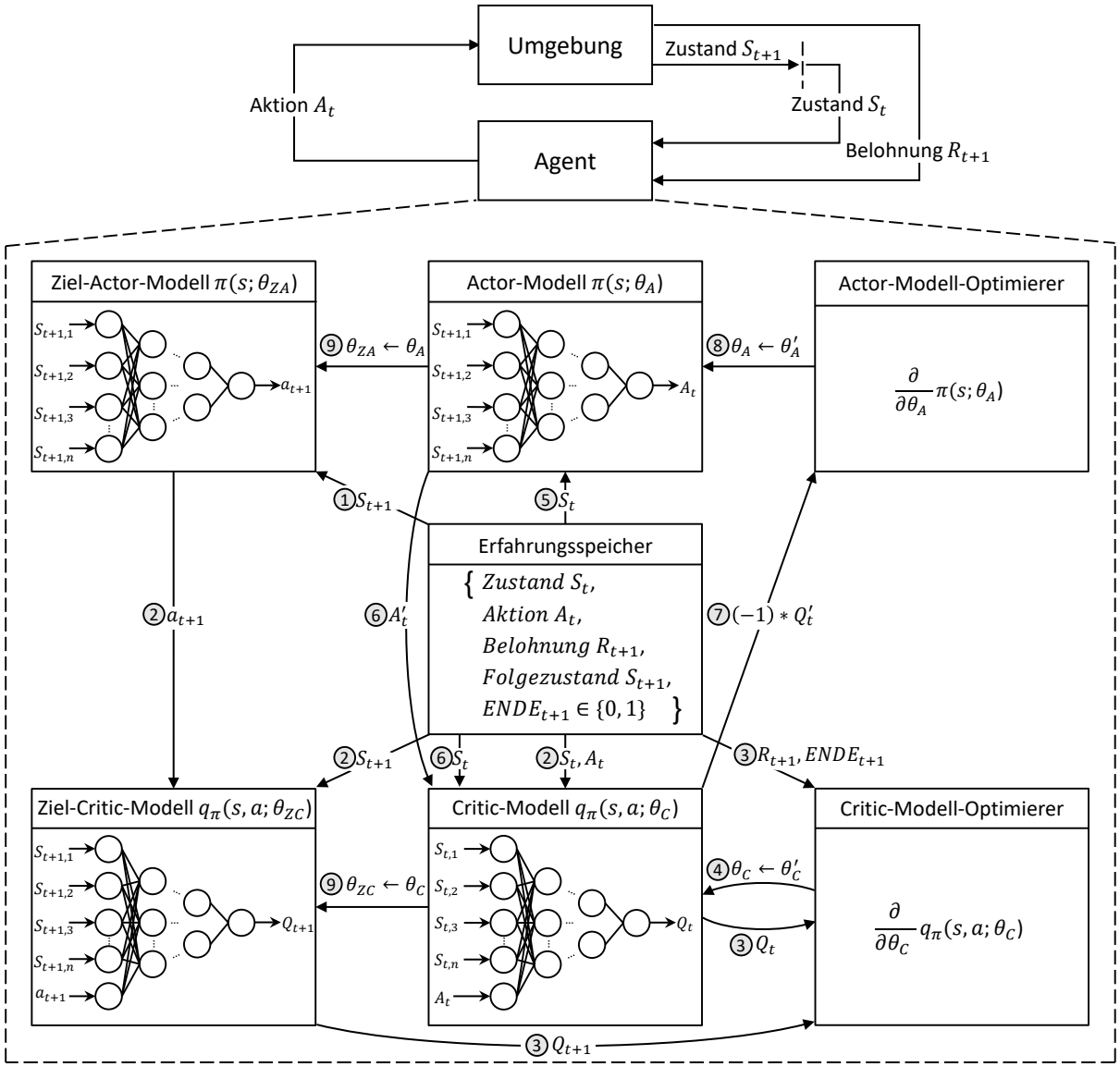


Abbildung D-1. DDPG-Architektur und algorithmischer Ablauf

Im Folgenden soll der Trainingsprozess des DDPG-Algorithmus kurz dargelegt werden. Analog zum DQN-Algorithmus generiert der Agent durch Interaktion mit der Umgebung Beobachtungen, die im Erfahrungsspeicher gesammelt werden. Ebenfalls immer dann, wenn eine neue Beobachtung dem Erfahrungsspeicher hinzugefügt wird und der Erfahrungsspeicher bereits eine ausreichende Anzahl an Beobachtungen

enthält, wird ein Trainingsschritt durchgeführt. Hierfür wird ein Los mit zufälligen Beobachtungen aus dem Erfahrungsspeicher generiert. In der ersten Phase des Trainingsprozesses wird das Critic-Modell $q_\pi(s, a; \theta_C)$ aktualisiert. Zu diesem Zweck werden die Aktionsnutzenwerte Q_t und Q_{t+1} ermittelt. Um Q_t zu erhalten, werden der aktuelle Zustand S_t und die gewählte Aktion A_t vorwärts durch das Actor-Modell propagiert. Für die Ermittlung von Q_{t+1} wird zunächst mithilfe des Ziel-Actor-Modells $\pi(s; \theta_{ZA})$ die Auswahl einer Aktion a_{t+1} für den Folgezustand S_{t+1} prognostiziert. Danach werden S_{t+1} und a_{t+1} vorwärts durch das Ziel-Critic-Modell $q_\pi(s, a; \theta_{ZC})$ propagiert, um Q_{t+1} zu erhalten. Um das Fehlersignal des Critic-Modells zu ermitteln, wird zunächst die Zielgröße U_t für die kumulative Belohnung in der Zukunft über die folgende Formel prognostiziert.

$$U_t = R_{t+1} + (1 - ENDE_{t+1})\gamma Q_{t+1} \tag{D.1}$$

Der Fehler der Ausgabeschicht des Critic-Modells berechnet sich analog zum DQN-Algorithmus gemäß Formel (3.8). Durch Rückpropagierung werden die Gradienten des Critic-Modells bestimmt und dessen Parameter aktualisiert. Nun beginnt die zweite Phase des Trainingsprozesses, in der das Actor-Modell $\pi(s; \theta_A)$ aktualisiert wird. Hierfür wird zunächst S_t vorwärts durch das Actor-Modell propagiert, um die Aktivierung $a^{(l)}$ aller Actor-Neuronen für die spätere Gradientenbildung zu erhalten. Die resultierende Aktion A'_t wird zusammen mit S_t vorwärts durch das aktualisierte Critic-Modell propagiert. Die Critic-Ausgabe Q'_t repräsentiert zeitgleich das Fehlersignal, das rückwärts durch das Actor-Modell propagiert wird. Im Gegensatz zum DQN-Algorithmus werden die Ziel-Modelle $\pi(s; \theta_{ZA})$ und $q_\pi(s, a; \theta_{ZC})$ nicht nach n Zeitschritten mit den aktuellen Parametern des Actor- bzw. Critic-Modells überschrieben. Stattdessen werden die Parameter der Ziel-Modelle nach jedem Zeitschritt durch Polyak-Mittelung (OpenAI 2018a) der Actor- bzw. Critic-Parameter sowie der Parameter des entsprechenden Ziel-Modells aktualisiert. Die Mittelwertbildung erfolgt unter Verwendung des Parameter τ, der entweder nahe, jedoch größer null (Lillicrap et al. 2015) oder nahe, jedoch kleiner eins ist (OpenAI 2018a). Gemäß dem ersten Fall $0 < \tau \ll 1$ und am Beispiel des Actor-Modells erfolgt die Aktualisierung des Ziel-Modells nach der folgenden Rechenvorschrift.

$$\theta'_{ZA} = \tau\theta_A + (1 - \tau)\theta_{ZA} \tag{D.2}$$

Der Parameter τ trägt somit Sorge, dass die Ziel-Modell-Parameter nur inkrementell in Richtung der Actor-Parameter aktualisiert werden. Sofern für τ der umgekehrte Fall $0 \ll \tau < 1$ gilt, werden die Koeffizienten von θ_A und θ_{ZA} vertauscht. Die Berechnungsvorschrift des Ziel-Critic-Modells gestaltet sich nach demselben Prinzip.

Eine populäre Abwandlung des DDPG-Algorithmus ist der »Twin Delayed DDPG« (TD3) -Algorithmus (Fujimoto et al. 2018). Ergänzend zu DDPG verwendet TD3 ein zweites Critic-Modell (zzgl. des entsprechenden Zielmodells) zur Prognose der Aktionsnutzen. Die Einführung eines zweiten Critic-Modells wird dadurch motiviert, dass beim konventionellen DDPG-Algorithmus das Critic-Modell dazu neigen kann, den Nutzen von Aktionen zu hoch einzuschätzen, wodurch die Lösungsgüte der Entscheidungspolitik beeinträchtigt wird (OpenAI 2018b). Im TD3-Algorithmus wird für jede Aktion der kleinere Wert aus beiden prognostizierten Nutzen verwendet, um die kumulierte Gesamtbelohnung in der Zukunft zu schätzen. Analog zum ursprünglichen DQN-Algorithmus werden zudem die Ziel-Modelle nicht nach jedem Zeitschritt, sondern nach n Zeitschritten aktualisiert.

ANHANG E – EXKURS: BAYES'SCHE OPTIMIERUNG

Neben der Kreuzentropie-Methode und den klassifikationsbasierten Ansätzen (siehe Abschnitt 3.4.1 im Hauptwerk) repräsentiert die Bayes'sche Optimierung das dritte grundständige RL-Verfahren, das gradientenfrei und modellsuchend arbeitet. Die Bayes'sche Optimierung (BO) (Mockus 1975; 1989) konstruiert ein probabilistisches Modell für eine zu optimierende (i. d. R. stark verrauschte und / oder aufwändig zu evaluierende) Funktion[1] $f(x_t)$, mit welchem auf die potenziell besten Lösungskandidaten geschlossen werden soll.

Das Modell wird häufig durch einen Gauß-Prozess repräsentiert (Qian und Yu 2021). Dem Gauß-Prozess liegt die Überlegung zugrunde, dass jeder Wert $y_t = f(x_t)$ der nachzubildenden Funktion einer jeweils eigenen normalverteilten Zufallsvariable Y_t unterliegt, die durch ihren Erwartungswert μ_t und ihre Varianz σ_t^2 charakterisiert wird. Die Stelle y_t ist hierbei ein konkreter Wert (Realisation) aus Y_t. Für die nachzubildende Funktion wird angenommen, dass sie unendlich viele Punkte $(x_t,\ y_t = f(x_t)\ )$ besitzt. Eine gewöhnliche n-dimensionale Normalverteilung $\mathcal{N}_n(\vec{\mu}, \Sigma)$ ist über ihren endlichen Erwartungswertvektor $\vec{\mu}$: $\mathbb{R}^n$ und ihre Kovarianzmatrix Σ: $\mathbb{R}^{n\times n}$ definiert. In der Numerik können unendlich dimensionale Normalverteilungen $(n = \infty)$ nicht abgebildet werden. Ein Gauß-Prozess $\mathcal{GP}\big(\mu(x), \Sigma(x, x')\big)$ umgeht dieses Problem, indem Erwartungswerte und Kovarianzen der Funktionsausgaben y nicht über einen dimensionsbeschränkten Vektor bzw. eine dimensionsbeschränkte Matrix, sondern über eine Erwartungswertfunktion $\mu(x)$ und eine Kovarianzfunktion $\Sigma(x, x')$ abgebildet werden, die in Abhängigkeit von der zugehörigen Funktionsstelle x und einer Vergleichsstelle x' (zur Messung der Kovarianz) definiert sind. Für die Stellen x und x' wird ebenfalls angenommen, dass es sich um Realisationen von normalverteilten Zufallsvariablen X und X' handelt. Ferner repräsentieren X und X' multivariate Normalverteilungen, sofern jede Stelle x eines jeden Punktes (x, y) mehrdimensional ist.

Im Rahmen der BO wird angenommen, dass die Funktionswerte y, die durch den Gauß-Prozess prognostiziert werden, stets eindimensional sind. Ungeachtet dessen kann ein Gauß-Prozess als multivariate Normalverteilung (Gauß-Verteilung) von y mit unendlich vielen Dimensionen versinnbildlicht werden. Im Kontrast zum multivariaten Fall von X_t repräsentiert jede Dimension der multivariaten Normalverteilung des Gauß-Prozesses einen Wert Y_t der nachzubildenden Funktion $f(x)$. Praktisch ist ebenfalls ein Gauß-Prozess nicht in der Lage, die gesuchte Funktion über unendlich viele Punkte abzubilden. Stattdessen verallgemeinert der Gauß-Prozess durch seine Erwartungswert- und Kovarianzfunktion das Konzept der multivariaten Normalverteilung auf eine solche Weise, dass über eine beliebig große und flexibel erweiterbare Indexmenge $T = \{Y_t \ \forall\ t = (0, \dots, |T|)\}$ die gesuchte Funktion über T Stützpunkte approximiert wird (Rasmussen und Williams 2008, S. 13). Hierbei entspricht eine Stichprobe aus dem Gauß-Prozess einer zufälligen Funktion, die durch T Stützpunkte beschrieben wird.

Die Erwartungswertfunktion $\mu(x) = \mathbb{E}[f(x)]$ gibt den erwarteten Funktionswert y an der Stelle x zurück und ist vergleichsweise einfach zu formulieren. Zu Beginn, wenn noch keine Beobachtungen mit $f(x)$ vorliegen, wird gewöhnlich angenommen, dass die Erwartungswertfunktion an jeder Stelle den Wert null besitzt $(\mu(x) = 0)$ (Rasmussen und Williams 2008, S. 27; Görtler et al. 2019). Diese initiale Annahme ist deshalb zulässig, da im weiteren Verlauf die Erwartungswerte auf Basis von Beobachtungen aktualisiert werden. Eine Herausforderung hinsichtlich der problemspezifischen Adaption von Gauß-Prozessen ist die Identifikation einer geeigneten Kovarianzfunktion[2], welche die Beziehung der verschiedenen

[1] Im RL-Kontext handelt es sich bei der zu approximierenden Funktion um die Agentenumgebung.

[2] In der wissenschaftlichen Literatur wird der Begriff »Kernel«-Funktion gleichermaßen verwendet.

Zufallsvariablen Y_t untereinander möglichst präzise beschreibt. Konkret gibt die Kovarianzfunktion $\Sigma(x, x') = \mathbb{E}\big[\big(f(x) - \mu(x)\big)\big(f(x') - \mu(x')\big)\big]$ die erwartete Kovarianz zwischen zwei Stützwerten $y = f(x)$ und $y' = f(x')$ in Abhängigkeit von den Eingaben x und x' und ohne Information über die Ergebnisse von $f(x)$ und $f(x')$ zurück. Eine Übersicht zu verschiedenen Kovarianzfunktionen präsentieren bspw. Rasmussen und Williams (2008, 81 ff.). Eine weit verbreitete und einfach verständliche Kovarianzfunktion ist die quadratisch-exponentielle Kovarianz, die gemäß Formel (E.1) definiert ist (Garnett 2015a).

$$\Sigma(x, x') = e^{-1/2\|x - x'\|^2} \tag{E.1}$$

Hierbei berechnet $\|x - x'\|$ die euklidische Distanz zwischen x und x'. Formel (E.1) erlaubt ebenfalls, dass für x und x' Vektoren mit mehreren Stellen eingesetzt werden. Angenommen $x \in \mathbb{R}^n$ und $x' \in \mathbb{R}^m$, so berechnet $\Sigma(x, x')$ eine $(n \times m)$-Matrix. Ferner kann ebenfalls jede Stelle in x bzw. x' mehrdimensional sein. Gemäß dem Fall, dass jede Stelle in x und x' jeweils k Dimensionen besitzt, sodass $x \in \mathbb{R}^{n \times k}$ bzw. $x' \in \mathbb{R}^{m \times k}$ gilt, berechnet $\Sigma(x, x')$ ebenfalls eine $(n \times m)$-Matrix. Die Kovarianz zwischen zwei Stellen wird hierbei über deren Dimensionen gemittelt.

Das BO-Verfahren wird zum Anlernen des Gauß-Prozesses verwendet. Analog zur Kreuzentropie-Methode aktualisiert BO das probabilistische Modell durch die stichprobenartige Evaluation von Lösungskandidaten. Das BO-Verfahren unterscheidet sich von der Kreuzentropie-Methode insbesondere durch zwei Aspekte.

Erstens dient das BO-konstruierte probabilistische Modell nicht der Generierung von Stichproben für die Eingabe x, sondern der Prognose von Ausgabewerten $\mathbb{E}[f(x)]$ in Abhängigkeit von einer gegebenen Eingabe x. Nichtsdestotrotz zielt das BO-Verfahren letztendlich auf die Identifikation einer optimalen Eingabe x ab. Im Gegensatz zur Kreuzentropie-Methode werden Lösungskandidaten jedoch über ein zusätzliches Verfahren generiert, z. B. durch eine Rastersuche über den Eingabedatenraum (Bergstra et al. 2011), via Monte-Carlo-Simulation (Snoek et al. 2012) oder mittels eines anderen Verfahrens (Shahriari et al. 2016).

Zweitens werden Lösungskandidaten nicht in arbiträrer Reihenfolge durch $f(x)$ evaluiert, sondern mithilfe einer sogenannten »Akquisitionsfunktion« ausgewählt. Die Akquisitionsfunktion bestimmt die nächste zu evaluierende Lösung aus der Menge von Lösungskandidaten, z. B. gemäß der höchsten Wahrscheinlichkeit für eine Verbesserung (Kushner 1964) oder gemäß der höchsten erwarteten Verbesserung (Mockus et al. 1978). Exemplarisch soll im Folgenden die Arbeitsweise der Akquisitionsfunktion anhand der höchsten erwarteten Verbesserung erklärt werden (Garnett 2015b).

Angenommen sei ein Minimierungsproblem $\min f$, bei dem auf Basis vorausgegangener Beobachtungen…

$$\mathcal{D} = \mathcal{X}_\mathcal{D} \cup \mathcal{Y}_\mathcal{D} = \{(x_d, y_d = f(x_d)) \; \forall \; d = (0, \dots, t-1)\} \tag{E.2}$$

…die bisher beste Lösung $y^* = f(x^*)$ gefunden wurde. Mithilfe eines beliebigen Verfahrens wurde eine Menge von Lösungskandidaten $\mathcal{X}_\mathcal{C} = \{x_c \forall \; c = (0, \dots, |\mathcal{X}_\mathcal{C}|)\}$ generiert. Gesucht wird Lösungskandidat $x \in \mathcal{X}_\mathcal{C}$, der die größte Verbesserung $u(x)$ verspricht.

$$u(x) = \max_{x \in \mathcal{X}_C} (0, y^* - f(x)) \tag{E.3}$$

Hierbei repräsentiert $f(x)$ die zu minimierende Fitnessfunktion zur Bewertung von Lösungskandidaten, die eine hohe Laufzeitkomplexität besitzt und deshalb durch einen Gauß-Prozess angenähert werden soll.

In den folgenden Formeln werden $\mathcal{X}_c$, $\mathcal{X}_\mathcal{D}$und $\mathcal{Y}_\mathcal{D}$ nicht als Mengen, sondern als Vektoren oder Matrizen[3] von Funktionsstellen bzw. als Vektor von bekannten Funktionswerten behandelt. Für die Auswahl des Lösungskandidaten, anhand dessen das Modell trainiert werden soll, wird die Akquisitionsfunktion $a_{EI}(\mathcal{X}_c)$ formuliert, die in vektorisierter Form die erwartete Verbesserung EI aller Lösungen $f(\mathcal{X}_c)$ für alle Kandidaten $\mathcal{X}_c$ prognostiziert.

$$a_{EI}(\mathcal{X}_C) = \mathbb{E}[u(\mathcal{X}_c) \mid \mathcal{X}_c, \mathcal{D}] = \int_{-\infty}^{y^*}(y^* - f(\mathcal{X}_c)) * \mathcal{N}_{|\mathcal{X}_c|}\big(f(\mathcal{X}_c); \mu(\mathcal{X}_c), \Sigma(\mathcal{X}_c, \mathcal{X}_c)\big)\mathrm{d}f(\mathcal{X}_c) \quad \text{(E.4)}$$

$$= \big(y^* - \mu(\mathcal{X}_c)\big) * \Phi_{|\mathcal{X}_c|}\big(y^*; \mu(\mathcal{X}_c), \Sigma(\mathcal{X}_c, \mathcal{X}_c)\big) + \Sigma(\mathcal{X}_c, \mathcal{X}_c) * \mathcal{N}_{|\mathcal{X}_c|}\big(y^*; \mu(\mathcal{X}_c), \Sigma(\mathcal{X}_c, \mathcal{X}_c)\big) \quad \text{(E.5)}$$

Es lässt sich zeigen, dass durch Auflösung des partiellen Integrals in Formel (E.4) die zu approximierende Fitnessfunktion $f(\mathcal{X}_c)$ eliminiert werden kann (Jones 2001). Die resultierende Akquisitionsfunktion in Formel (E.5) besteht aus zwei Termen. Der linke Term repräsentiert die Verteilungsfunktion $\Phi_{|\mathcal{X}_c|}\big(y^*; \mu(\mathcal{X}_c), \Sigma(\mathcal{X}_c, \mathcal{X}_c)\big)$ der multivariaten Normalverteilung $\mathcal{N}_{|\mathcal{X}_c|}\big(\mu(\mathcal{X}_c), \Sigma(\mathcal{X}_c, \mathcal{X}_c)\big)$ an der Stelle y^* in Abhängigkeit von dem Erwartungswert $\mu(\mathcal{X}_c)$ und der Varianz $\Sigma(\mathcal{X}_c, \mathcal{X}_c)$[4]. Die Verteilungsfunktion gibt an, wie viel Prozent aller Werte von $\mathcal{N}_{|\mathcal{X}_c|}\big(\mu(\mathcal{X}_c), \Sigma(\mathcal{X}_c, \mathcal{X}_c)\big)$ unter y^* liegen. Der linke Term wird mit der Differenz zwischen $\mu(\mathcal{X}_c)$ und y^* gewichtet und begünstigt somit die Tiefenuntersuchung einer bestimmten Lösungsregion. Der Term gewinnt für die Kalkulation von EI an Signifikanz, je besser $\mu(\mathcal{X}_c)$ im Vergleich zu y^* ist. Der rechte Term repräsentiert die Wahrscheinlichkeitsdichtefunktion $\mathcal{N}_{|\mathcal{X}_c|}\big(y^*; \mu(\mathcal{X}_c), \Sigma(\mathcal{X}_c, \mathcal{X}_c)\big)$ derselben multivariaten Normalverteilung an der Stelle y^*. Sie gibt die Position von y^* auf der Glockenkurve von $\mathcal{N}_{|\mathcal{X}_c|}\big(\mu(\mathcal{X}_c), \Sigma(\mathcal{X}_c, \mathcal{X}_c)\big)$ zurück und somit die Wahrscheinlichkeit, dass y^* aus $\mathcal{N}_{|\mathcal{X}_c|}\big(\mu(\mathcal{X}_c), \Sigma(\mathcal{X}_c, \mathcal{X}_c)\big)$ gezogen wird. Der rechte Term wird mit der Varianz $\Sigma(\mathcal{X}_c, \mathcal{X}_c)$ der geschätzten Fitnessfunktion gewichtet. Aufgrund der Tatsache, dass der Term mit einer zunehmenden Streuung der Werte wächst, fördert dieser eine breite Exploration des Lösungsraums (Brochu et al. 2010). Die Evaluation einer Eingabe x mithilfe von Formel (E.5) erfordert die Berechnung von $\mu(\mathcal{X}_c)$ und $\Sigma(\mathcal{X}_c, \mathcal{X}_c)$. Eine exemplarische Berechnungsvorschrift für $\Sigma(\mathcal{X}_c, \mathcal{X}_c)$ wurde bereits in Formel (E.1) dargestellt. Auf die Erwartungswerte der zu testenden Stellen $\mu(\mathcal{X}_C)$, für die initial Nullwerte angenommen wurden, kann auf Basis der vorliegenden Beobachtungen $\mathcal{D}$ gemäß der folgenden Formel geschlossen werden (Rasmussen und Williams 2008, S. 16).

$$\mu(\mathcal{X}_c) = \mathbb{E}[f(\mathcal{X}_c)| \mathcal{X}_c, \mathcal{D}] = \Sigma(\mathcal{X}_c, \mathcal{X}_\mathcal{D}) * \Sigma(\mathcal{X}_\mathcal{D}, \mathcal{X}_\mathcal{D})^{-1} * \mathcal{Y}_\mathcal{D} \quad \text{(E.6)}$$

Die Intention von Formel (E.6) ist, dass die bekannten Werte $\mathcal{Y}_\mathcal{D}$ mit dem Verhältnis aus der erwarteten Kovarianz zwischen $f(x)$ an den Stellen $\mathcal{X}_c$ und $\mathcal{X}_\mathcal{D}$ und der erwarteten Varianz von $f(x)$ an der Stelle $\mathcal{X}_\mathcal{D}$ multipliziert werden, um auf den bedingten erwarteten Ausgabevektor $\mu(\mathcal{X}_c)$ an den Stellen $\mathcal{X}_c$ zu schließen.

Zusammengefasst wird zu Beginn des BO-Verfahrens eine bestimme Anzahl von Lösungskandidaten $\mathcal{X}_c$ generiert. Mindestens ein Lösungskandidat wird mithilfe von $f(x)$ evaluiert, um mindestens eine Beobachtung in $\mathcal{D}$ zu erhalten. Evaluierte Lösungskandidaten werden aus $\mathcal{X}_c$ entfernt. Das BO-Verfahren läuft darauffolgend über eine benutzerdefinierte Anzahl T Iterationen. In jeder Iteration t wird *(i)* zunächst durch Maximierung der Akquisitionsfunktion über alle Lösungskandidaten in $\mathcal{X}_c$ der nächste zu evaluierende Lösungskandidat x_t ausgewählt und aus $\mathcal{X}_c$ entfernt, *(ii)* darauffolgend der Lösungskandidat durch die Agentenumgebung $f(x_t)$ evaluiert und schließlich *(iii)* die resultierende Beobachtung (x_t, y_t)

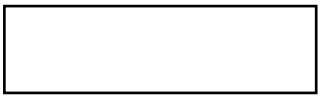
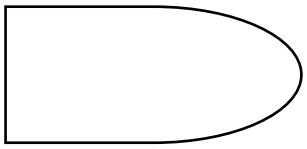
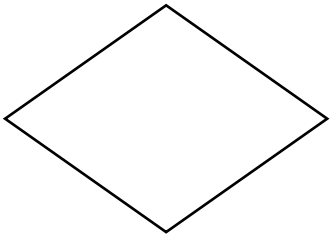
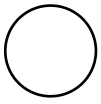
[3] Sofern alle Stellen in $\mathcal{X}_\mathcal{D}$ und $\mathcal{X}_c$ mehrdimensional sind.

[4] Die Kovarianz einer Variable mit sich selbst entspricht der Varianz der Variable.

zu $\mathcal{D}$ hinzugefügt. Ferner wird *(iv)* die beste Beobachtung y^* durch y_t überschrieben, sofern y_t besser als y^* ist. Durch die Aktualisierung von $\mathcal{X}_{\mathcal{C}}$ und $\mathcal{D}$ ändern sich in der darauffolgenden Iteration die erwarteten Verbesserungen EI der verbleibenden Lösungskandidaten in $\mathcal{X}_{\mathcal{C}}$. Nach T Iterationen wird y^* als Endergebnis ausgegeben.

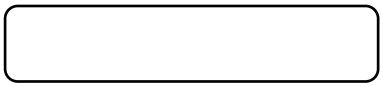
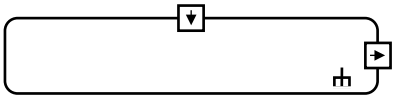
ANHANG F – PROGRAMMABLAUFPLAN-NOTATION

Tabelle F-1. Notation für Programmablaufpläne

Symbol	Bedeutung
	Start- bzw. Endknoten
	Eingangs- bzw. Ausgangsdaten
	Prozess
	Verschachtelter Prozess (Prozess, der wiederum in Unterprozesse gegliedert ist)
	Terminierte bzw. nichtterminierte Verzögerung ▪ Terminiert: Verzögerung endet nach vordefinierter Zeit ▪ Nichtterminiert: Verzögerung endet in Abhängigkeit von einem eintretenden Ereignis
	Verzweigung
	Terminierte Schleife (For-Schleife)
	Schnittstelle zu benachbartem Programmablaufplan

ANHANG G – UML-NOTATION

Tabelle G-1. Notation für UML-Aktivitätsdiagramme

Symbol	Bedeutung
	Startknoten
	Globaler Endknoten
	Lokaler Endknoten
	Aktivität
	Verschachtelte Aktivität mit einem Eingang und einem Ausgang (Aktivität, die wiederum in Aktivitäten gegliedert ist)
	Verzweigung

ANHANG H – DETAILLIERTE PROBLEMDATEN UND ERGEBNISSE ZU ABSCHNITT 6.1

Tabelle H-1. Operationspläne für die Probleminstanzen 5J×5M, 8J×8M, 10J×5M und 20J×5M (in Anlehnung an Rajkumar et al. 2010)

Auftrag j	Operationsplan v	Operationen o
1, 6[*1], 11[*2], 16[*3]	1	{3, 4, 1, 2}
	2	{4, 3, 1, 2}
	3	{2, 4, 1, 3}
	4	{3, 1, 4, 2}
2, 7[*1], 12[*2], 17[*3]	1	{1, 2, 3}
	2	{1, 3, 2}
	3	{3, 2, 1}
	4	{2, 1, 3}
3, 8[*1], 13[*2], 18[*3]	1	{2, 4, 1, 3}
	2	{1, 4, 2, 3}
	3	{1, 4, 5}
	4	{4, 2, 1, 3}
4, 9[*2], 14[*2], 19[*3]	1	{1, 2, 4, 3}
	2	{1, 2, 3, 4}
	3	{4, 2, 1, 3}
	4	{3, 2, 4, 1}
5, 10[*2], 15[*2], 20[*3]	1	{3, 1, 2}
	2	{3, 4}
	3	{3, 2, 1}
	4	{1, 3, 2}

[*1] Nur Probleminstanz 8J×8M, 10J×5M und 20J×5M

[*2] Nur Probleminstanz 10J×5M und 20J×5M

[*3] Nur Probleminstanz 20J×5M

Tabelle H-2. Operationszeiten und Fertigstellungsfristen für die Probleminstanzen 5J×5M, 8J×8M, 10J×5M und 20J×5M (in Anlehnung an Rajkumar et al. 2010)

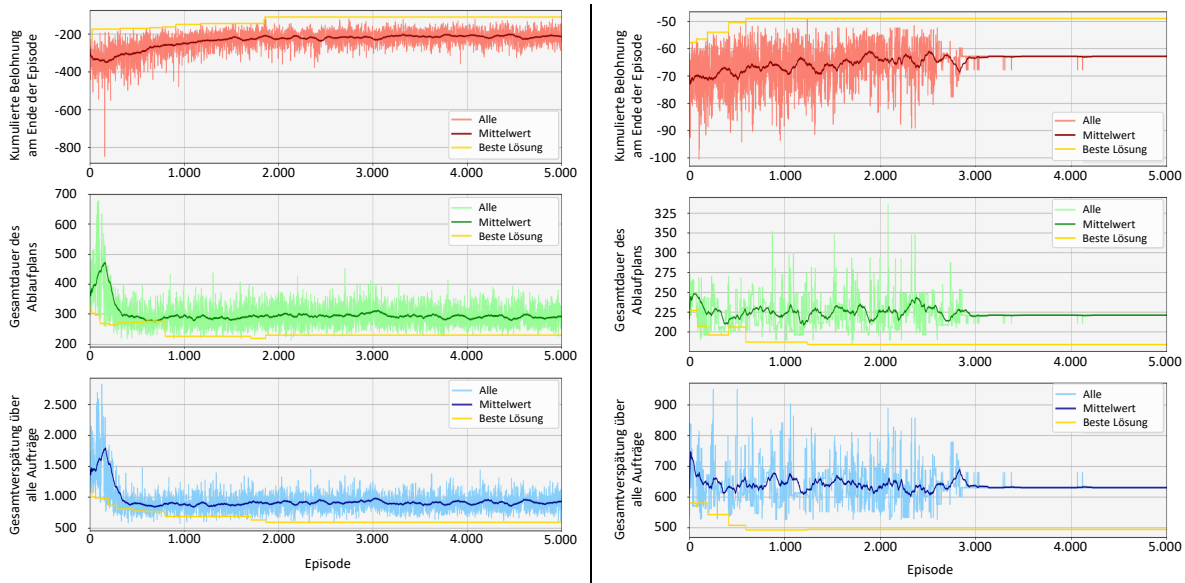
Auftrag j	Operation o	Produktionsressource i					Fertigstellungsfrist d_j
		1	2	3, 6[*4]	4, 7[*4]	5, 8[*4]	
		Operationszeit $o_{i,j,o}$					
1, 6[*1], 11[*2], 16[*3]	1	57	40	88	62	77	100 bzw. 500[*3]
	2	7	10	11	10	5	
	3	95	74	76	71	93	
	4	24	18	22	28	26	
2, 7[*1], 12[*2], 17[*3]	1	84	76	68	98	84	120 bzw. 400[*3]
	2	20	10	15	20	19	
	3	91	88	98	87	90	
3, 8[*1], 13[*2], 18[*3]	1	65	87	58	80	74	120 bzw. 500[*3]
	2	46	21	38	52	18	
	3	19	22	19	14	22	
	4	73	56	64	72	60	
	5	96	98	96	95	98	
4, 9[*2], 14[*2], 19[*3]	1	13	7	13	12	11	80 bzw. 600[*3]
	2	52	64	97	47	40	
	3	20	30	17	11	14	
	4	94	66	80	79	95	
5, 10[*2], 15[*2], 20[*3]	1	94	97	55	78	85	110 bzw. 700[*3]
	2	31	23	19	42	17	
	3	88	65	76	64	80	
	4	88	74	90	92	75	

[*1] Nur Probleminstanz 8J×8M, 10J×5M und 20J×5M

[*2] Nur Probleminstanz 10J×5M und 20J×5M

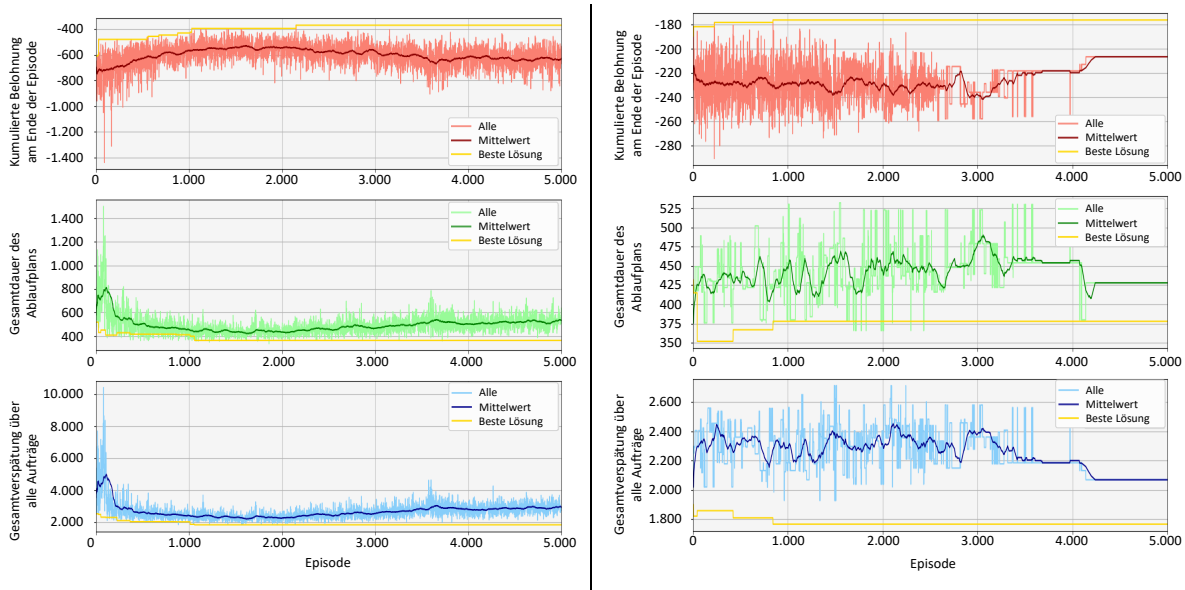
[*3] Nur Probleminstanz 20J×5M

[*4] Nur Probleminstanz 8J×8M



(a) (b)

Abbildung H-1. Trainingsmetriken der Probleminstanz 8J×8M für (a) den Agenten der Ressourcenbelegungsplanung und (b) den Agenten für die Auswahl von Operationsplänen



(a) (b)

Abbildung H-2. Trainingsmetriken der Probleminstanz 10J×5M für (a) den Agenten der Ressourcenbelegungsplanung und (b) den Agenten für die Auswahl von Operationsplänen

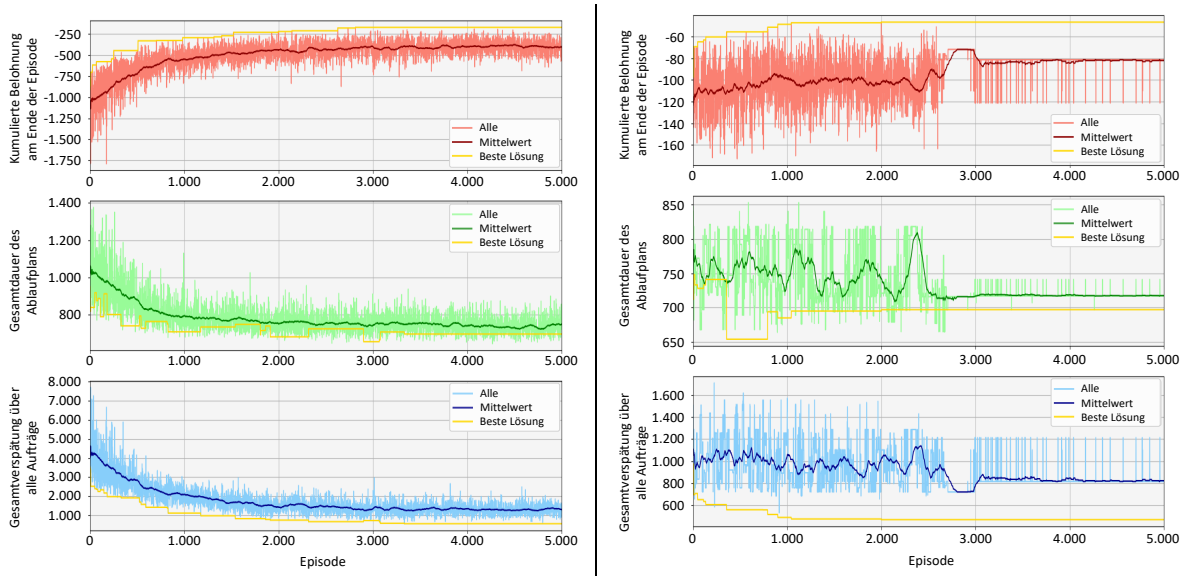


Abbildung H-3. Trainingsmetriken der Probleminstanz 20J×5M für (a) den Agenten der Ressourcenbelegungsplanung und (b) den Agenten für die Auswahl von Operationsplänen

Tabelle H-3. Mittelwerte, Standardabweichungen und Konfidenzintervalle (Konfidenzniveau: 95 %) der Gesamtdauer des Ablaufplans C_{max} für die Probleminstanz 20J×5M mit dynamischen Auftragshorizont über jeweils 100 Replikationen

Anzahl zusätzlicher Aufträge	DQN-Agent			Regelbasierte Strategie		
	μ	σ	Konfidenzintervall	μ	σ	Konfidenzintervall
5	870,28	23,53	[865,61; 874,95]	922,55	24,05	[917,78; 927,32]
10	1.037,56	28,27	[1031,95; 1043,17]	1.101,81	23,67	[1097,11; 1106,51]
15	1.223,9	30,19	[1217,91; 1229,89]	1.280,96	29,29	[1275,15; 1286,77]
20	1.409,87	37,27	[1402,47; 1417,27]	1.636,42	26,8	[1631,1; 1641,74]

Tabelle H-4. Mittelwerte, Standardabweichungen und Konfidenzintervalle (Konfidenzniveau: 95 %) der Gesamtverspätung über alle Aufträge T für die Probleminstanz 20J×5M mit dynamischen Auftragshorizont über jeweils 100 Replikationen

Anzahl zusätzlicher Aufträge	DQN-Agent			Regelbasierte Strategie		
	μ	σ	Konfidenzintervall	μ	σ	Konfidenzintervall
5	2384,53	373,7	[2310,38; 2458,68]	3537,37	447,44	[3448,59; 3626,15]
10	6185,5	637,33	[6059,04; 6311,96]	7732,02	665,39	[7599,99; 7864,05]
15	11339,6	879,6	[11165,08; 11514,14]	12944	887,45	[12767,86; 13120,04]
20	17460,8	1057,05	[17251,01; 17670,49]	19132,8	998,47	[18934,63; 19330,87]

ANHANG I – DETAILLIERTE ERGEBNISSE ZU ABSCHNITT 6.2

Tabelle I-1. Mittelwert, Standardabweichung und aggregiertes Konfidenzintervall (Stichprobenumgang: 300, Konfidenzniveau: 95 %) der Gesamtverspätung über alle Aufträge T des PPO-trainierten Agenten über zehn Replikationen sowie Vergleichswerte von verschiedenen Prioritätsregeln für verschiedene PMP-Instanzen mit $w_d = 0,1$ und $D = 0,15$

$\|J\|$	$\|M\|$	$\|b=0\|$	PPO-Agent		EDD		LST		SPT		MAS_PAR	
			μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
5	6	40	105,01	79,56	947,66	111,21	1.015,52	155,16	1.113,71	114,40	999,38	105,44
		45	181,81	56,14	982,27	163,39	1.068,85	178,47	1.310,78	172,12	1.002,43	116,28
		50	203,65	68,30	1.071,65	193,96	1.135,48	174,07	1.473,94	141,29	1.114,39	134,99
		55	209,96	91,68	1.137,56	147,04	1.172,18	195,31	1.680,15	125,12	1.101,14	145,67
		60	281,29	84,84	1.165,22	199,66	1.241,07	208,38	1.809,82	278,43	1.124,08	94,58
	7	40	90,87	49,19	862,11	116,01	933,11	142,12	921,80	119,40	961,36	101,43
		45	127,06	56,07	896,38	135,53	943,78	119,15	1.092,57	119,65	993,81	99,31
		50	136,91	54,40	988,84	151,26	1.077,90	135,48	1.255,34	142,18	1.023,08	113,38
		55	160,11	51,20	1.006,54	126,50	1.134,72	90,72	1.399,42	181,63	1.013,49	115,36
		60	188,10	74,86	949,08	190,80	1.061,17	157,70	1.404,05	188,22	1.070,77	105,01
	8	40	83,50	30,03	736,70	132,21	833,10	80,81	811,31	83,45	1.018,41	125,05
		45	102,24	41,47	827,13	128,39	965,27	69,61	968,26	126,37	1.019,74	92,06
		50	136,41	51,14	961,07	118,91	1.028,25	146,78	1.132,64	104,91	1.033,72	91,27
		55	148,24	55,91	967,85	130,69	1.069,68	104,44	1.273,59	81,39	1.020,22	109,37
		60	167,68	52,61	1.015,24	216,46	1.023,54	160,03	1.370,19	161,20	1.061,48	104,85
7	6	40	171,90	74,63	1.019,17	129,35	1.020,85	141,83	1.195,66	119,80	1.243,71	149,21
		45	197,27	79,73	1.072,04	209,95	1.190,77	168,56	1.244,88	152,90	1.270,05	120,93
		50	232,26	108,65	1.159,07	143,37	1.256,84	104,90	1.466,85	122,70	1.246,88	196,54
		55	238,22	91,61	1.255,13	172,30	1.305,75	133,23	1.748,01	209,70	1.362,38	147,36
		60	334,54	84,78	1.396,02	234,78	1.478,73	137,30	1.955,75	132,56	1.491,25	176,39
	7	40	124,57	48,04	918,45	125,25	959,31	143,54	1.033,87	107,75	1.261,37	115,59
		45	151,14	53,86	965,44	117,78	991,14	138,34	1.164,29	127,32	1.283,55	126,57
		50	189,21	62,86	1.014,72	165,63	1.158,79	142,47	1.318,66	178,50	1.243,50	204,24
		55	229,23	75,52	1.106,96	117,52	1.114,39	79,41	1.506,03	179,98	1.342,80	146,69
		60	256,88	47,00	1.223,13	164,19	1.258,00	167,06	1.683,35	95,77	1.402,80	142,65
	8	40	128,83	53,93	844,80	74,52	861,99	118,39	924,22	83,15	1.180,98	95,34
		45	136,32	53,44	961,39	104,25	988,03	89,75	1.020,02	115,97	1.267,58	99,17
		50	192,46	66,87	1.022,18	127,59	1.056,11	160,68	1.194,89	166,59	1.353,57	114,47
		55	216,37	88,89	1.086,25	98,32	1.122,12	125,73	1.321,09	171,69	1.377,39	137,92
		60	262,63	78,62	1.200,79	144,25	1.220,77	160,42	1.478,13	145,31	1.473,38	138,91
Mittelwert:			179,49	65,53	1.025,36	146,37	1.089,57	137,66	1.309,11	141,65	1.178,62	125,53
Konfidenzintervall:			[172,04; 186,94]		[1008,73; 1041,99]		[1073,93; 1105,21]		[1293,02; 1325,20]		[1164,36; 1192,88]	

Tabelle I-2. Mittelwert, Standardabweichung und aggregiertes Konfidenzintervall (Stichprobenumgang: 300, Konfidenzniveau: 95 %) der Gesamtverspätung über alle Aufträge T des PPO-trainierten Agenten über zehn Replikationen sowie Vergleichswerte von verschiedenen Prioritätsregeln für verschiedene PMP-Instanzen mit $w_d = 0,1$ und $D = 0,25$

$\|J\|$	$\|M\|$	$\|b=0\|$	PPO-Agent		EDD		LST		SPT		MAS_PAR	
			μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
5	6	40	137,32	65,40	898,43	110,71	961,56	165,11	1.118,44	114,09	1.001,69	105,86
		45	177,62	72,63	917,55	164,47	966,40	149,55	1.314,04	173,30	1.010,21	114,19
		50	204,35	90,87	990,30	190,89	988,30	155,84	1.480,67	145,58	1.120,08	134,47
		55	220,37	88,81	1.037,36	153,50	1.077,87	185,51	1.688,24	127,47	1.103,42	145,60
		60	267,31	103,51	1.048,63	199,98	1.152,64	124,64	1.819,34	288,92	1.127,11	90,03
	7	40	91,61	48,65	816,82	112,21	871,22	104,07	923,91	117,38	961,17	101,12
		45	127,13	61,04	839,84	137,20	908,45	106,80	1.096,87	126,75	997,78	102,34
		50	151,09	84,30	919,58	155,08	1.053,41	188,76	1.260,81	147,11	1.031,64	111,88
		55	195,47	70,71	920,65	127,59	966,65	125,58	1.406,79	187,05	1.021,22	115,78
		60	203,55	90,63	849,86	189,26	913,18	145,33	1.413,92	193,89	1.072,57	105,92
	8	40	82,24	51,98	707,47	115,03	755,94	157,89	813,75	81,51	1.018,26	123,75
		45	117,44	44,74	776,81	132,13	909,89	109,29	970,64	123,64	1.019,56	93,51
		50	142,27	53,38	899,50	120,18	956,75	89,64	1.136,97	109,43	1.037,64	91,41
		55	155,35	51,18	892,94	136,67	956,96	82,67	1.282,42	85,81	1.027,52	110,28
		60	186,51	66,65	928,09	216,91	929,90	192,35	1.379,28	160,03	1.067,19	107,57
7	6	40	182,12	73,55	974,88	138,78	1.019,79	184,13	1.203,02	120,75	1.250,00	154,78
		45	228,77	98,75	1.016,90	224,05	1.121,61	186,31	1.253,78	158,78	1.275,19	128,19
		50	251,74	105,37	1.086,08	157,88	1.164,95	157,42	1.476,33	124,90	1.251,26	203,67
		55	288,66	108,13	1.165,05	182,70	1.171,69	143,82	1.754,22	210,50	1.369,73	154,78
		60	338,80	105,22	1.288,49	250,36	1.296,50	149,74	1.963,90	126,18	1.498,07	185,74
	7	40	150,46	54,63	876,78	133,65	852,17	105,67	1.038,35	112,48	1.265,92	121,95
		45	161,13	48,72	915,16	125,56	896,84	166,64	1.170,95	132,62	1.287,09	134,37
		50	195,47	53,51	951,52	172,18	1.031,32	133,23	1.326,15	176,91	1.245,38	207,83
		55	250,43	74,65	1.029,70	128,55	1.086,59	106,88	1.513,00	179,04	1.348,66	150,44
		60	270,72	56,57	1.130,13	171,47	1.081,58	157,66	1.690,67	96,56	1.412,23	153,41
	8	40	130,35	49,17	790,75	83,83	831,81	96,99	930,28	89,66	1.184,75	102,81
		45	168,28	58,98	916,17	109,73	906,67	103,66	1.025,25	123,71	1.274,45	112,09
		50	186,06	87,39	966,19	138,72	1.037,95	103,80	1.199,32	170,60	1.355,51	118,06
		55	218,71	77,84	1.019,50	108,94	1.043,84	110,45	1.328,47	174,71	1.378,84	140,27
		60	260,30	70,99	1.121,11	154,97	1.145,42	107,01	1.484,56	144,18	1.479,09	147,23
Mittelwert:			191,39	72,27	956,41	151,44	1.001,93	136,55	1.315,48	144,12	1.183,11	128,98
Konfidenzintervall:			[183,18; 199,60]		[939,20; 973,62]		[986,42; 1017,44]		[1299,11; 1331,85]		[1168,46; 1197,76]	

Tabelle I-3. Mittelwert, Standardabweichung und aggregiertes Konfidenzintervall (Stichprobenumgang: 300, Konfidenzniveau: 95 %) über alle Aufträge T des PPO-trainierten Agenten über zehn Replikationen sowie Vergleichswerte von verschiedenen Prioritätsregeln für verschiedene PMP-Instanzen mit $w_d = 0,2$ und $D = 0,15$

$\|J\|$	$\|M\|$	$\|b=0\|$	PPO-Agent		EDD		LST		SPT		MAS_PAR	
			μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
5	6	40	290,90	93,71	1.242,10	122,45	1.323,04	167,01	1.325,22	118,13	1.150,58	103,71
		45	360,14	94,34	1.331,28	185,05	1.444,35	195,08	1.557,39	189,06	1.178,50	113,47
		50	386,96	122,69	1.484,26	211,02	1.566,60	193,70	1.764,14	152,03	1.281,55	129,81
		55	492,75	110,96	1.609,78	162,13	1.658,31	223,06	2.010,39	130,99	1.276,40	141,14
		60	549,66	182,66	1.687,44	227,92	1.782,94	249,68	2.187,15	282,84	1.312,53	100,52
	7	40	208,12	72,86	1.073,81	110,32	1.150,54	155,07	1.093,65	132,11	1.099,20	99,41
		45	272,03	86,24	1.184,13	145,28	1.250,18	125,58	1.300,79	120,94	1.160,28	95,01
		50	323,62	105,13	1.343,82	167,46	1.458,44	148,04	1.509,37	150,66	1.196,30	115,08
		55	381,64	93,56	1.408,16	143,45	1.564,98	98,17	1.684,82	198,54	1.197,88	121,46
		60	416,67	69,29	1.376,89	219,64	1.519,26	178,57	1.717,58	185,63	1.258,50	114,60
	8	40	177,33	51,40	935,88	105,08	1.036,57	70,55	961,26	89,96	1.149,39	120,12
		45	255,55	59,06	1.046,78	162,10	1.185,38	99,15	1.145,25	141,00	1.174,06	91,40
		50	316,44	87,73	1.252,25	129,90	1.345,76	149,87	1.351,00	110,30	1.206,80	96,22
		55	310,07	49,31	1.322,22	140,49	1.449,28	111,80	1.537,81	86,49	1.214,84	120,88
		60	362,15	76,83	1.415,67	239,88	1.439,21	185,52	1.670,07	162,19	1.271,64	111,33
7	6	40	357,23	111,50	1.325,32	134,25	1.326,89	137,41	1.410,82	120,49	1.422,44	147,02
		45	392,45	129,54	1.442,17	234,68	1.583,10	191,33	1.500,46	161,59	1.472,38	116,63
		50	477,57	174,45	1.593,83	167,17	1.709,83	116,10	1.766,32	137,10	1.468,88	204,22
		55	590,26	154,83	1.745,48	196,69	1.810,62	155,60	2.086,57	224,95	1.594,55	160,71
		60	647,91	135,45	1.953,83	264,18	2.053,06	154,41	2.328,40	120,70	1.737,15	183,30
	7	40	295,80	65,69	1.098,91	125,20	1.182,85	162,87	1.209,17	113,83	1.424,86	116,11
		45	332,54	75,38	1.255,28	124,62	1.294,82	150,34	1.388,42	126,10	1.471,62	131,64
		50	397,56	58,66	1.372,24	183,28	1.544,08	149,49	1.583,69	196,45	1.454,54	208,94
		55	463,18	90,08	1.524,03	133,99	1.552,35	85,12	1.799,17	196,84	1.568,96	166,29
		60	503,30	41,08	1.701,11	180,96	1.753,52	191,04	2.009,64	101,84	1.647,60	146,87
	8	40	250,84	80,61	1.037,62	86,97	1.077,64	148,38	1.076,23	85,03	1.330,96	97,78
		45	306,25	102,04	1.203,96	96,47	1.280,60	88,49	1.209,87	112,87	1.452,44	98,31
		50	357,60	110,13	1.324,04	136,00	1.368,97	173,32	1.425,27	178,94	1.557,48	118,37
		55	414,02	120,76	1.459,65	109,60	1.514,24	135,53	1.582,20	193,65	1.603,48	155,19
		60	496,68	110,67	1.636,04	159,89	1.672,18	180,73	1.765,86	165,70	1.712,10	155,50
Mittelwert:			379,57	97,22	1.379,60	160,20	1.463,32	152,37	1.565,27	149,57	1.368,26	129,37
Konfidenzintervall:			[368,52; 390,62]		[1361,4; 1397,8]		[1446,01; 1480,63]		[1548,28; 1582,26]		[1353,56; 1382,96]	

Tabelle I-4. Mittelwert, Standardabweichung und aggregiertes Konfidenzintervall (Stichprobenumgang: 300, Konfidenzniveau: 95 %) über alle Aufträge T des PPO-trainierten Agenten über zehn Replikationen sowie Vergleichswerte von verschiedenen Prioritätsregeln für verschiedene PMP-Instanzen mit $w_d = 0,2$ und $D = 0,25$

$\|J\|$	$\|M\|$	$\|b=0\|$	PPO-Agent		EDD		LST		SPT		MAS_PAR	
			μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
5	6	40	279,82	106,73	1.194,54	121,43	1.266,77	195,30	1.331,10	119,25	1.150,84	102,83
		45	383,78	86,94	1.269,54	186,78	1.341,44	171,12	1.563,92	186,38	1.174,89	112,10
		50	400,64	95,24	1.410,06	207,04	1.421,08	171,96	1.772,46	152,70	1.288,85	127,54
		55	519,63	145,82	1.514,77	169,40	1.570,07	208,89	2.023,01	129,82	1.286,85	143,12
		60	562,01	147,66	1.573,48	231,58	1.698,21	148,87	2.201,11	288,61	1.324,05	95,63
	7	40	227,62	78,23	1.029,54	107,83	1.126,38	160,98	1.098,19	128,25	1.101,20	99,45
		45	267,36	72,59	1.128,70	147,38	1.201,27	145,39	1.306,22	127,43	1.156,82	97,11
		50	331,30	84,14	1.277,39	171,64	1.430,50	208,33	1.515,43	148,84	1.195,28	113,10
		55	410,57	112,16	1.326,32	144,38	1.393,79	139,89	1.692,65	198,48	1.199,27	119,45
		60	424,08	87,73	1.278,80	222,12	1.364,64	170,15	1.729,26	190,26	1.264,57	112,89
	8	40	198,78	57,36	898,18	96,51	956,36	156,08	964,43	88,17	1.150,35	118,64
		45	262,29	74,33	997,26	166,35	1.181,80	120,55	1.148,95	135,95	1.174,70	92,17
		50	307,33	106,89	1.192,01	130,85	1.266,61	99,47	1.356,06	112,82	1.205,43	94,58
		55	357,97	63,54	1.249,07	146,74	1.330,18	90,40	1.544,31	88,17	1.212,33	121,05
		60	431,51	99,86	1.330,23	241,67	1.346,44	212,71	1.682,06	161,84	1.270,21	110,47
7	6	40	364,04	100,46	1.282,00	143,33	1.331,08	205,61	1.416,22	123,34	1.429,00	153,92
		45	431,75	128,42	1.392,29	253,08	1.511,68	207,88	1.509,09	170,45	1.477,54	127,50
		50	493,06	130,56	1.528,62	185,52	1.621,23	186,82	1.775,97	139,53	1.468,52	213,63
		55	611,18	172,81	1.662,50	211,31	1.684,39	162,03	2.094,02	225,65	1.598,80	167,95
		60	656,08	123,74	1.852,12	285,45	1.869,05	171,54	2.340,14	108,05	1.747,18	195,99
	7	40	303,24	58,26	1.059,06	132,34	1.111,35	110,18	1.215,37	118,00	1.429,20	124,25
		45	347,36	91,78	1.206,11	133,45	1.238,22	195,31	1.394,59	134,83	1.478,06	140,37
		50	383,30	76,18	1.311,47	190,58	1.417,50	138,38	1.589,97	194,80	1.455,59	213,18
		55	447,30	89,33	1.452,30	147,90	1.535,05	118,92	1.806,77	196,92	1.568,30	170,99
		60	543,10	92,93	1.614,54	192,84	1.565,65	186,87	2.017,98	102,05	1.651,43	158,03
	8	40	269,93	61,56	998,80	85,92	1.038,93	95,30	1.079,99	92,42	1.334,99	104,42
		45	320,66	94,72	1.160,14	103,90	1.197,46	149,37	1.216,53	121,99	1.457,12	110,10
		50	393,39	109,82	1.269,39	147,87	1.357,61	117,15	1.431,52	184,51	1.558,85	124,64
		55	442,42	135,08	1.394,33	119,32	1.429,43	117,80	1.587,68	196,45	1.605,83	158,76
		60	520,87	104,15	1.561,40	174,32	1.597,28	119,32	1.773,59	165,89	1.715,59	162,44
Mittelwert:			396,41	99,64	1.313,83	166,63	1.380,05	156,08	1.572,62	151,06	1.371,05	132,88
Konfidenzintervall:			[385,09; 407,73]		[1294,9; 1332,76]		[1362,32; 1397,78]		[1555,46; 1589,78]		[1355,95; 1386,15]	

ANHANG J – DETAILLIERTE ERGEBNISSE ZU ABSCHNITT 6.3.3

Tabelle J-1. Lösungsgüte des NEAT-Algorithmus in den Experimenten 01–05

EXP	Kürzel	Lauf	Datensatz 1		Datensatz 2		Datensatz 3		Datensatz 4	
			C_{max}	T	C_{max}	T	C_{max}	T	C_{max}	T
01	PRE-SEQ-KNN& MIN-WL	01	20.026	0	23.277	0	23.613	0	22.288	1.457
		02	22.291	3.374	23.056	0	24.587	0	22.688	1.732
		03	20.026	0	23.277	0	23.613	0	21.886	231
		04	20.026	0	23.277	0	24.357	0	22.182	231
		05	20.026	0	23.277	0	23.613	0	24.997	7.221
		06	20.663	0	22.938	0	24.272	0	20.886	51
		07	21.737	0	27.388	0	23.853	0	21.480	588
		08	20.026	0	23.277	0	23.595	0	22.153	231
		09	20.026	0	23.277	0	23.613	0	21.886	231
		10	20.026	0	23.277	0	23.595	0	22.153	231
02	FIFO& J-ALLOK-KNN	01	22.564	12.002	24.594	29.982	22.785	53.741	23.043	324.466
		02	23.855	3.119	24.414	32.432	20.778	34.965	20.512	170.977
		03	23.782	8.172	23.245	32.094	21.639	46.679	24.569	358.037
		04	24.216	5.304	25.445	24.038	22.360	39.113	22.757	205.228
		05	29.418	19.891	24.848	25.034	21.994	39.039	24.021	260.269
		06	23.515	8.308	28.956	39.896	21.883	31.446	24.321	224.271
		07	24.362	12.926	24.670	27.487	23.118	41.012	22.524	207.173
		08	27.481	11.057	27.219	24.650	22.037	37.792	20.956	133.018
		09	25.831	14.485	25.120	31.028	22.829	36.780	19.983	187.091
		10	24.550	10.865	25.322	37.377	23.402	33.183	24.299	328.523
03	LIFO& J-ALLOK-KNN	01	31.274	184.056	32.833	232.775	37.397	229.864	38.962	193.290
		02	23.635	296.082	31.541	245.924	27.531	253.491	32.254	343.961
		03	23.126	315.732	24.299	265.930	28.065	246.900	26.283	123.942
		04	25.025	195.524	25.422	249.982	27.886	223.800	27.286	221.304
		05	29.444	218.283	28.347	251.263	36.876	230.548	34.458	165.774
		06	31.426	245.617	34.421	250.423	33.112	238.973	22.028	192.880
		07	21.607	225.409	26.362	277.584	26.781	214.685	25.395	139.326
		08	22.959	290.981	32.843	232.778	31.444	233.629	30.354	152.602
		09	30.073	305.669	27.189	272.396	36.689	235.346	35.947	284.044
		10	21.583	239.592	30.973	231.052	30.476	233.986	25.730	77.839
04	EDD& J-ALLOK-KNN	01	24.839	12.237	26.305	1.033	24.251	2.925	24.126	510
		02	25.857	5.741	27.260	1.052	26.666	3.130	27.491	5.125
		03	23.216	4.058	27.025	1.033	25.438	3.459	23.435	174
		04	25.120	5.830	27.405	203	25.401	3.400	24.131	18
		05	21.820	313	26.360	1.514	25.947	3.191	24.496	2.167
		06	23.465	6.168	25.491	1.606	24.512	2.637	22.632	1.100
		07	26.588	1.366	26.957	476	23.189	4.133	25.989	2.110
		08	25.893	2.914	26.958	714	27.146	3.122	24.391	1.770
		09	27.100	3.141	26.807	1.162	24.380	2.264	22.829	1.232
		10	24.329	31.167	27.719	3.400	25.103	4.051	23.220	496
05	SPT& J-ALLOK-KNN	01	27.541	60.392	27.954	49.834	26.296	61.946	22.377	36.929
		02	25.330	59.396	23.766	53.377	23.657	60.934	26.010	59.667
		03	23.754	47.464	24.968	43.560	28.450	58.615	25.265	32.425
		04	24.869	56.979	23.472	50.485	23.949	49.820	22.290	34.454
		05	25.156	47.908	23.434	51.845	24.548	63.279	22.816	34.929
		06	24.578	90.750	23.328	48.567	24.645	52.530	24.300	24.802
		07	29.220	123.561	24.303	50.058	24.524	52.658	23.149	30.004
		08	25.917	91.948	25.010	46.100	23.109	67.495	25.903	31.680
		09	23.521	51.319	23.981	50.234	23.477	65.104	23.800	34.541
		10	27.602	57.852	30.022	58.094	27.140	73.520	26.506	46.034

Tabelle J-2. Lösungsgüte des NEAT-Algorithmus in den Experimenten 06–10

EXP	Kürzel	Lauf	Datensatz 1		Datensatz 2		Datensatz 3		Datensatz 4	
			C_{max}	T	C_{max}	T	C_{max}	T	C_{max}	T
06	FIFO& F-ALLOK-KNN	01	23.872	3.177	24.334	23.508	19.838	47.612	19.893	278.837
		02	20.935	1.243	23.497	23.690	20.134	46.778	18.708	307.650
		03	19.111	5.279	23.084	22.417	20.351	47.388	18.933	268.496
		04	25.671	8.997	24.245	18.972	22.067	49.691	21.054	275.766
		05	20.110	4.248	23.216	27.994	20.677	44.252	18.966	221.461
		06	21.467	14.878	23.509	25.966	22.219	43.522	18.711	243.454
		07	19.385	5.143	24.333	26.000	20.704	45.389	18.622	269.445
		08	24.600	3.602	23.125	25.685	19.915	46.386	19.151	262.534
		09	20.404	10.630	23.358	21.120	20.021	47.881	18.777	258.135
		10	20.049	7.184	22.660	23.481	21.095	38.633	18.964	196.469
07	LIFO& F-ALLOK-KNN	01	22.122	527.663	30.162	594.746	26.823	484.844	27.514	235.475
		02	18.396	603.495	24.489	606.616	23.292	516.196	20.624	223.408
		03	26.020	671.074	28.894	609.046	24.253	512.644	21.051	247.812
		04	20.483	515.390	28.112	574.580	24.585	504.915	20.017	280.617
		05	23.210	582.161	29.133	592.988	21.986	521.506	25.185	452.329
		06	34.026	730.690	30.162	587.157	27.037	457.545	19.948	234.652
		07	21.170	482.938	26.554	585.086	28.957	491.000	20.890	192.435
		08	18.801	523.742	26.872	564.874	29.635	461.847	24.078	279.587
		09	18.249	561.416	28.596	583.468	23.771	510.387	20.185	325.177
		10	20.371	560.776	28.596	569.649	22.441	520.199	18.676	257.765
08	EDD& F-ALLOK-KNN	01	21.674	9.735	25.803	305	21.518	399	22.711	6.310
		02	22.930	0	23.995	345	22.033	238	24.653	4.249
		03	19.831	1.393	24.689	113	21.533	406	20.605	1.019
		04	19.919	1.238	27.322	480	21.632	0	19.379	0
		05	20.254	0	24.263	72	22.290	1.120	19.643	2.333
		06	20.024	2.818	25.809	354	23.333	933	20.134	1.843
		07	21.601	889	26.696	401	21.994	691	21.483	0
		08	19.078	2.437	26.696	512	21.573	437	19.149	745
		09	20.091	2.490	25.474	4	22.032	438	19.952	563
		10	22.079	2.232	25.022	233	22.988	593	19.901	1.486
09	SPT& F-ALLOK-KNN	01	22.036	104.659	26.124	103.917	23.803	117.741	21.791	108.844
		02	22.878	144.335	27.038	118.977	23.118	102.453	19.528	95.355
		03	24.830	109.568	27.578	113.977	23.659	110.264	29.776	123.326
		04	22.883	105.980	26.482	107.454	22.880	111.278	20.299	101.420
		05	23.752	89.963	26.575	109.931	26.090	113.942	20.582	89.186
		06	21.263	104.024	25.597	113.217	23.891	112.549	21.014	93.715
		07	21.536	91.291	26.753	113.882	22.966	114.632	22.393	89.165
		08	23.695	128.116	25.105	120.472	24.401	107.302	25.622	105.425
		09	20.944	95.779	27.655	113.042	24.838	113.496	20.878	93.272
		10	20.758	112.526	26.352	103.175	24.373	116.262	20.396	83.850
10	PRE-SEQ-KNN& J-ALLOK-KNN	01	22.045	990	25.085	27	25.695	1.587	25.000	205
		02	22.904	4.311	26.164	146	25.533	1.626	24.594	686
		03	21.588	41.617	23.074	183	24.977	1.523	22.939	425
		04	22.597	3.040	27.081	1.086	26.724	2.369	25.694	8.441
		05	24.463	1.991	26.789	0	26.059	2.800	26.940	7.181
		06	22.545	18.542	23.057	1.551	25.729	1.818	20.980	1.631
		07	26.510	4.229	26.949	1.555	26.115	5.945	28.345	4.129
		08	23.993	7.375	27.433	28	26.761	668	25.820	3.242
		09	24.295	3.835	27.492	867	25.736	318	22.987	0
		10	28.025	68.950	26.354	3.632	25.109	3.908	22.525	2.527

Tabelle J-3. Lösungsgüte des NEAT-Algorithmus in den Experimenten 10–13

EXP	Kürzel	Lauf	Datensatz 1		Datensatz 2		Datensatz 3		Datensatz 4	
			C_{max}	T	C_{max}	T	C_{max}	T	C_{max}	T
11	POST-SEQ-KNN& J-ALLOK-KNN	01	21.194	6.061	24.800	790	26.041	1.379	27.943	87.656
		02	21.792	1.126	26.533	2.979	24.252	4.540	23.556	1.833
		03	23.872	5.876	27.073	2.011	27.265	223	26.834	4.116
		04	22.572	547	24.611	1.998	22.978	656	22.857	16.891
		05	23.297	4.184	24.654	2.963	26.076	2.651	21.320	6.710
		06	23.294	64.943	24.029	366	25.983	717	25.428	109.789
		07	23.868	26.842	25.128	617	24.773	1.252	20.510	17.752
		08	20.860	4.777	25.625	1.647	24.706	953	24.156	12.954
		09	20.702	5.948	24.699	1.237	23.862	765	21.999	12.779
		10	23.269	3.674	25.900	1.751	21.461	560	20.022	8.748
12	PRE-SEQ-KNN& F-ALLOK-KNN	01	20.049	14.285	22.791	72	20.278	0	19.587	764
		02	19.776	2.037	24.359	0	22.120	602	18.774	1.196
		03	18.480	2.336	25.785	25	20.324	206	19.523	666
		04	19.777	98	21.713	0	20.399	132	20.859	0
		05	19.795	2.484	23.692	0	20.525	209	19.012	2.102
		06	19.757	719	23.997	0	20.810	60	20.850	5.193
		07	21.609	10.273	26.621	0	21.754	273	19.030	1.655
		08	19.453	15.318	25.324	0	20.796	4	19.559	0
		09	17.605	5.141	22.144	0	20.980	211	18.780	3.804
		10	20.705	846	25.589	64	21.372	0	18.723	107
13	POST-SEQ-KNN& F-ALLOK-KNN	01	21.464	3.348	23.333	79	21.669	492	18.928	331
		02	20.686	8.261	25.232	183	21.032	724	19.197	1.387
		03	20.514	0	24.809	7	20.233	344	20.768	111
		04	22.113	7.326	26.748	0	20.525	411	20.004	11.091
		05	18.275	969	24.887	0	21.057	614	19.722	483
		06	18.663	203	24.845	172	20.530	320	18.529	1.568
		07	22.018	11.157	26.167	0	21.054	474	23.699	2.005
		08	19.069	909	26.703	0	21.236	492	19.331	345
		09	24.407	765	22.855	0	21.032	209	18.360	14.238
		10	30.176	76.848	25.912	0	23.627	615	30.837	228.864

Tabelle J-4. Lösungsgüte des NEAT-Algorithmus in Experiment 12 PRE-SEQ-KNN&F-ALLOK-KNN unter Anwendung der zweiten explorationsorientierten Hyperparameter-Konfiguration (in Anlehnung an Lang et al. 2021)

Lauf	Datensatz 1		Datensatz 2		Datensatz 3		Datensatz 4	
	C_{max}	T	C_{max}	T	C_{max}	T	C_{max}	T
01	25.414	135.966	23.633	0	20.265	3	20.032	1.745
02	21.368	0	24.286	0	21.617	4	18.928	12.974
03	18.247	438	23.977	74	22.174	0	20.531	10.194
04	17.863	0	24.912	0	22.405	0	18.975	0
05	19.923	5.828	23.795	0	20.83	131	19.755	0
06	19.477	4.932	26.654	57	20.378	105	19.732	6.025
07	18.557	0	24.467	0	21.878	0	19.115	0
08	21.437	1.171	24.699	4	20.987	0	22.187	1.416
09	22.011	1.871	24.518	41	21.48	0	19.741	0
10	20.247	469	24.806	66	21.663	0	24.742	45.441

Tabelle J-5. Ergebnisdaten des Lösungsverfahrensvergleichs in Abbildung 6-16 (Abschnitt 6.3.4)

Lösungsverfahren	Datensatz 1		Datensatz 2		Datensatz 3		Datensatz 4	
	C_{max}	T	C_{max}	T	C_{max}	T	C_{max}	T
Fertigungsplanung des Unternehmens	23.513	198.783	25.447	271.700	23.626	31.372	23.539	257.376
EDD&MIN-WL	23.586	86.490	26.662	149.141	25.756	149.148	20.507	11.518
SPT&MIN-WL	21.154	0	26.226	4.833	22.603	6.000	21.145	964
ISBO-Heuristik	19.354	148	21.819	0	19.979	536	18.806	639
Simulated Annealing	21.930	0	23.108	0	23.059	0	20.562	0
Tabu Search	19.669	0	25.142	0	22.507	0	21.610	0
Genetischer Algorithmus	16.467	0	19.274	0	18.459	0	17.526	0
A2C	17.693	0	21.355	2.371	19.159	2.413	18.729	0
NEAT (EXP 01)	20.663	0	22.938	0	24.272	0	20.886	51
NEAT (EXP 12)	18.557	0	24.467	0	21.878	0	19.115	0

LITERATURVERZEICHNIS

Alpaydin, E. (2019): Maschinelles Lernen. 2. Auflage. Berlin: de Gruyter.

Bergstra, J., Bardenet, R., Bengio, Y. und Kégl, B. (2011): Algorithms for Hyper-Parameter Optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger und Killian Q. (Hg.): Advances in Neural Information Processing Systems 24. 25th Conference on Neural Information Processing Systems (NeurIPS 2011). Granada, ESP, 12.-17. Dezember. Red Hook, NY, USA: Curran Associates, Inc. Online verfügbar unter: https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

Brochu, E., Cora, V. M. und Freitas, N. de (2010): A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. Online verfügbar unter: https://arxiv.org/pdf/1012.2599.

Cho, K., van Merrienboer, B., Bahdanau, D. und Bengio, Y. (2014): On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In: Wu, D., Carpuat, M., Carreras, X. und Maria Vecchi, E. (Hg.): Proceedings of the Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8). Doha, Qatar, 25. Oktober. Stroudsburg, PA, USA: Association for Computational Linguistics, S. 103–111.

Fujimoto, S., van Hooof, H. und Merger, D. (2018): Addressing Function Approximation Error in Actor-Critic Methods. In: Dy, J. und Krause, A. (Hg.): Proceedings of the 35th International Conference on Machine Learning (ICML 2018). Stockholm, SWE, 10.-15. Juli, S. 1587–1596.

Garnett, R. (2015a): CSE 515T: Bayesian Methods in Machine Learning - Spring 2015. Lecture 11: Gaussian Process Regression. Vorlesung. Washington University in St. Louis, St. Louis, MO, USA. Department of Computer Science and Engineering. Online verfügbar unter: https://www.cse.wustl.edu/~garnett/cse515t/spring_2017/files/lecture_notes/9.pdf.

Garnett, R. (2015b): CSE 515T: Bayesian Methods in Machine Learning – Spring 2015. Lecture 11: Bayesian Optimization. Vorlesung. Washington University in St. Louis, St. Louis, MO, USA. Department of Computer Science and Engineering. Online verfügbar unter: https://www.cse.wustl.edu/~garnett/cse515t/spring_2015/files/lecture_notes/12.pdf.

Gers, F. A., Schmidhuber, J. und Cummins, F. (2000): Learning to Forget: Continual Prediction with LSTM. In: *Neural Computation* 12 (10), S. 2451–2471.

Görtler, J., Kehlbeck, R. und Deussen, O. (2019): A Visual Exploration of Gaussian Processes. In: *Distill* 4 (4). Online verfügbar unter https://distill.pub/2019/visual-exploration-gaussian-processes/.

Haykin, S. (1999): Neural Networks. A Comprehensive Foundation. 2. Auflage. Delhi: Pearson Education.

Hochreiter, S. (1991): Untersuchungen zu dynamischen neuronalen Netzen. Diplomarbeit. Technische Universität München, München. Institut für Informatik. Online verfügbar unter: http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf.

Hochreiter, S. und Schmidhuber, J. (1997): Long Short-Term Memory. In: *Neural Computation* 9 (8), S. 1735–1780.

Jones, D. R. (2001): A Taxonomy of Global Optimization Methods based on Response Surfaces. In: *Journal of Global Optimization* 21 (4), S. 345–383.

Kelley, H. J. (1960): Gradient Theory of Optimal Flight Paths. In: *ARS Journal* 30 (10), S. 947–954.

Kriesel, D. (2007): Ein kleiner Überblick über Neuronale Netze. Online verfügbar unter: http://www.dkriesel.com/_media/science/neuronalenetze-de-zeta2-2col-dkrieselcom.pdf.

Kruse, R., Borgelt, C., Braune, C., Klawonn, F., Moewes, C. und Steinbrecher, M. (2015): Computational Intelligence. Eine methodische Einführung in künstliche neuronale Netze, evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze. 2. Auflage. Wiesbaden: Springer Vieweg.

Kushner, H. J. (1964): A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. In: *Journal of Basic Engineering* 86 (1), S. 97–106.

Lang, S., Reggelin, T., Schmidt, J., Müller, M. und Nahhas, A. (2021): NeuroEvolution of Augmenting Topologies for Solving a Two-Stage Hybrid Flow Shop Scheduling Problem: A Comparison of Different Solution Strategies. In: *Expert Systems with Applications* 172, Aufsatznummer 114666.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y. et al. (2015): Continuous control with deep reinforcement learning. Online verfügbar unter: http://arxiv.org/pdf/1509.02971v6.

Mockus, J. B. (1975): On Bayesian Methods for Seeking the Extremum. In: Marčuk, G. I. (Hg.): Optimization Techniques. IFIP Technical Conference, Novosibirsk, July 1-7, 1974. Berlin, Heidelberg: Springer, S. 400–404.

Mockus, J. B. (1989): Bayesian Approach to Global Optimization. Theory and Applications. Dordrecht, NL: Springer Netherlands.

Mockus, J. B., Tiesis, V. und Zilinskas, A. (1978): The Application of Bayesian Methods for Seeking the Extremum. In: Dixon, L. C. und Szegö, G. P. (Hg.): Towards Global Optimisation 2. A Collection of Proceedings From Two Workshops Held in Varenna, Italy, in June 1976 and at the University of Bergamo, Italy, in July 1977, and Various Other Research Papers. Amsterdam, NL: North-Holland Publishing, S. 117–129.

Ng, A. (2012): Machine Learning. Neural Networks: Learning. Stanford University. Online verfügbar unter: https://www.coursera.org/learn/machine-learning/supplement/FklyY/lecture-slides.

OpenAI (2018a): Deep Deterministic Policy Gradient. OpenAI. Online verfügbar unter: https://spinningup.openai.com/en/latest/algorithms/ddpg.html.

OpenAI (2018b): Twin Delayed DDPG. OpenAI. Online verfügbar unter: https://spinningup.openai.com/en/latest/algorithms/td3.html.

Qian, H. und Yu, Y. (2021): Derivative-Free Reinforcement Learning: A Review. In: *Frontiers of Computer Science* 15 (6), Aufsatznummer 156336.

Rajkumar, M., Asokan, P., Page, T. und Arunachalam, S. (2010): A GRASP Algorithm for the Integration of Process Planning and Scheduling in a Flexible Job-Shop. In: *IJMR* 5 (2), S. 230–251.

Rasmussen, C. E. und Williams, C. K. I. (2008): Gaussian Processes for Machine Learning. 3. Auflage. Cambridge, MA, USA: MIT Press.

Rosenblatt, F. (1958): The Perceptron. A Probabilistic Model for Information Storage and Organization in the Brain. In: *Psychological Review* 65 (6), S. 386–408.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. und Klimov, O. (2017): Proximal Policy Optimization Algorithms. Online verfügbar unter: https://arxiv.org/pdf/1707.06347.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P. und Freitas, N. de (2016): Taking the Human Out of the Loop: A Review of Bayesian Optimization. In: *Proceedings of the IEEE* 104 (1), S. 148–175.

Snoek, J., Larochelle, H. und Adams, R. P. (2012): Practical Bayesian Optimization of Machine Learning Algorithms. In: Pereira, F., Burges, C. J. C., Bottou, L. und Weinberger, K. Q. (Hg.): Advances in Neural Information Processing Systems 25. 26th Conference on Neural Information Processing Systems (NeurIPS 2012). Lake Tahoe, NV, USA, 03.-08. Dezember. Red Hook, NY, USA: Curran Associates, Inc., S. 1–9.

Werbos, P. J. (1982): Applications of advances in nonlinear sensitivity analysis. In: Drenick, R. F. und Kozin, F. (Hg.): System Modeling and Optimization. Proceedings of the 10th IFIP Conference New York City, USA, August 31 - September 4, 1981, Bd. 38. Berlin, Heidelberg: Springer (Lecture Notes in Control and Information Sciences, 38), S. 762–770.