Praktikum: Selbstlernende Systeme

Acrot-Critic

08.01.2019

Universität Augsburg
Institut für Informatik
Lehrstuhl für Organic Computing

Gliederung

ORGANIC COMPUTING

1. Actor-Critic

2. A2C und A3C

3. Implementierung von A2C

4. Quellen

Actor-Critic

Reinforcement Learning

ORGANIC COMPUTING

Value based

- Q-learning, Deep Q-Learning
- Value function: mappt jedes Zustands-Aktions-Paar auf einen Wert
- beste Aktion $\rightarrow$ größter Wert
- funktioniert gut mit einem finiten Set von Aktionen

Policy based

- REINFORCE
- direktes parametrisieren und optimieren der Policy
- keine Value function
- nützlich wenn der Aktionsraum kontinuierlich oder stochastisch ist
- Problem: gute Score function (gesamter EpisodenReward)

Actor-Critic

ORGANIC COMPUTING

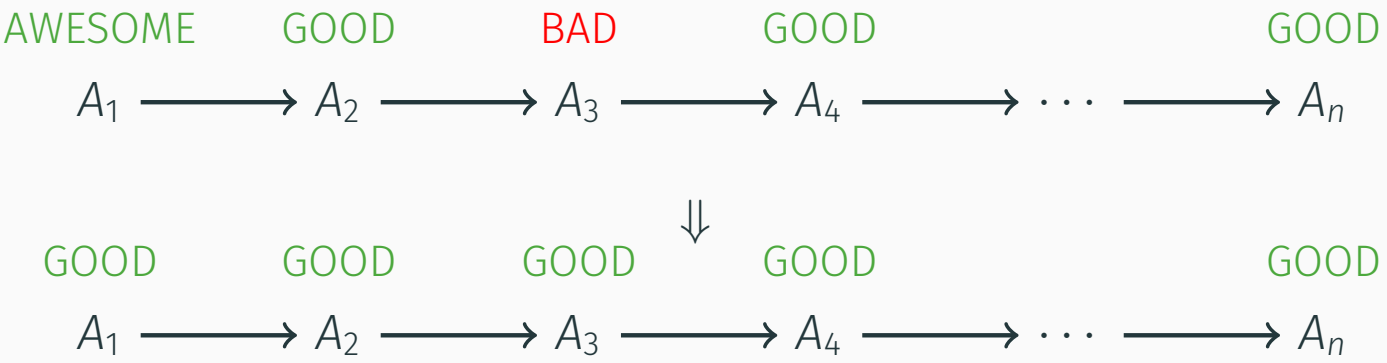
beide Methoden haben Nachteile
$\Rightarrow$ hybride Methode: Actor-Critic
zwei neuronale Netze:

- Actor: kontrolliert wie sich der Agent verhält (policy-based)
- Critic: misst wie gut die gewählte Aktion ist (value-based)

Das Problem mit Policy Gradient

ORGANIC COMPUTING

- Berechnung des Rewards erfolgt am Ende der Episode
- man könnte Annehmen: großes $R(t) \rightarrow$ alle gewählten Aktionen waren gut



- Alle Aktionen werden durchschnittlich als *GOOD* gewertet
- gesamter Reward entscheidend
- $\rightarrow$ viel Samples nötig $\rightarrow$ langsames Lernen

Actor-Critic: Einführung

ORGANIC COMPUTING

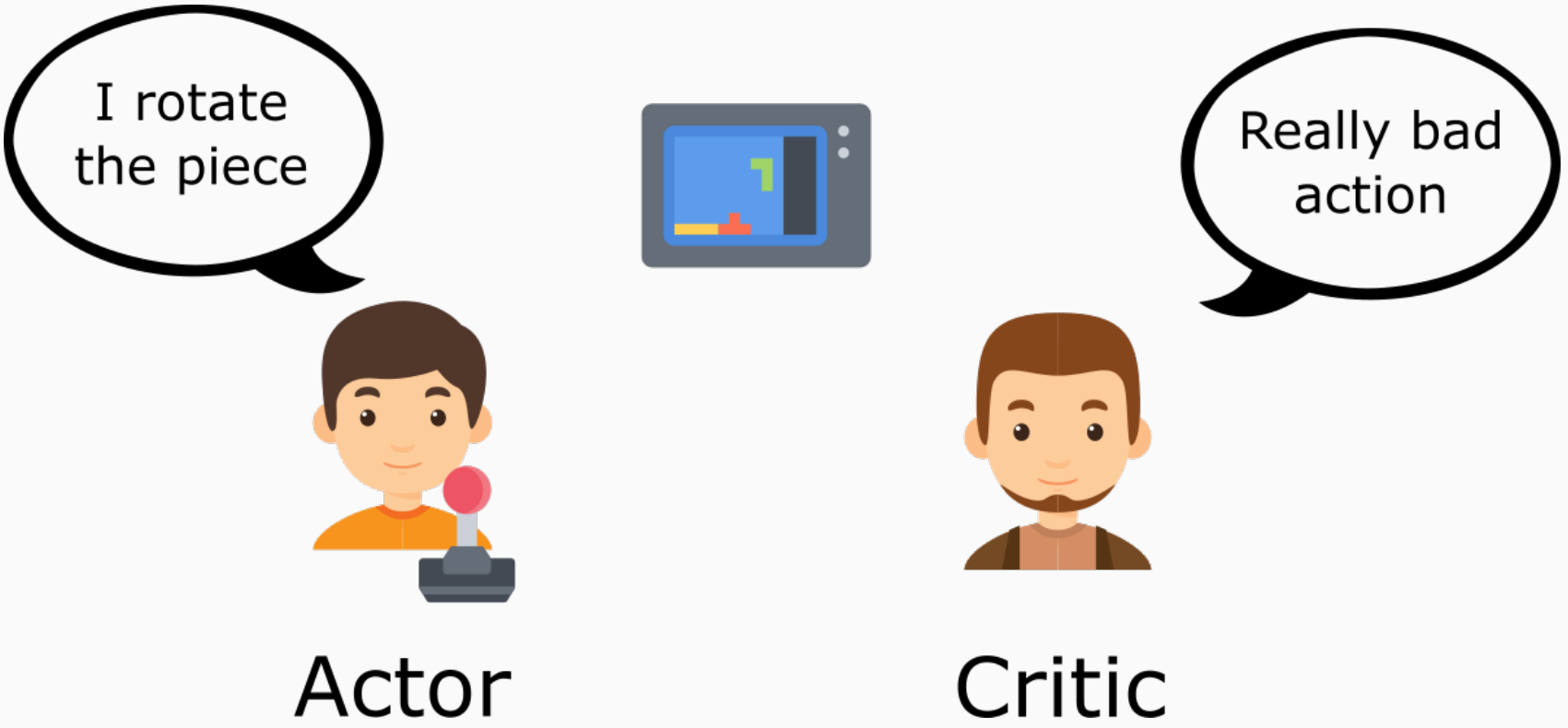
- Update in jedem Zeitschritt (TD Learning)

$$\Delta\theta = \alpha * \nabla_\theta * (\log \pi(S_t, A_t, \theta)) * R(t)$$

$$\Delta\theta = \alpha * \nabla_\theta * (\log \pi(S_t, A_t, \theta)) * Q(S_t, A_t)$$

- gesamter Reward $R(t)$ kann nicht benutzt werden
- Critic Modell $\rightarrow$ approximiert die Value function
- Value function ersetzt den Reward

Wie funktioniert Actor-Critic?



Wie funktioniert Actor-Critic?(2)

ORGANIC COMPUTING

- zu Beginn kein Wissen wie man spielt $\rightarrow$ zufällig Aktionen ausprobieren
- Critic überwacht und gibt Feedback $\rightarrow$ Policy updaten
- Critic lernt besseres Feedback zu geben $\rightarrow$ Critic updaten
- Actor:

$$\pi(s, a, \theta)$$

- Critic:

$$\hat{Q}(s, a, w)$$

- $\Rightarrow$ 2 Gewichts Sets θ und w

Update Regeln

θ und w müssen separat upgedatet werden:

Policy Update:

$$\Delta\theta = \alpha \nabla_\theta (\log \pi_\theta(a|s)) \underbrace{\hat{Q}_w(s,a)}_{\substack{\textit{Q learning} \\ \textit{function} \\ \textit{approximation}}}$$

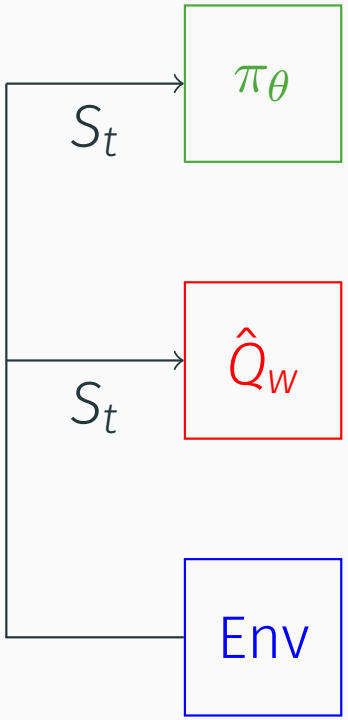
Value Update:

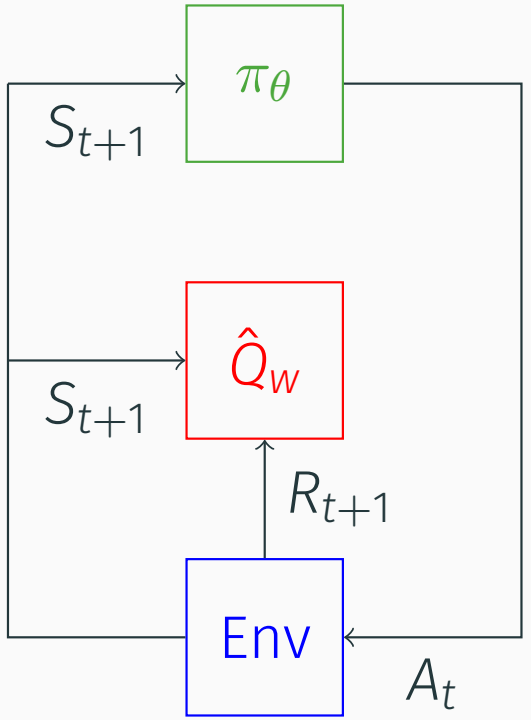
$$\Delta w = \beta \underbrace{(R(s_t,a_t) + \gamma \hat{Q}_w(s_{t+1},a_{t+1}) - \hat{Q}_w(s_t,a_t))}_{\textit{TD error}} \underbrace{\nabla_w \hat{Q}_w(s_t,a_t)}_{\substack{\textit{Gradient of} \\ \textit{value function}}}$$

$\alpha =$ Policy Learningrate

$\beta =$ Value Learningrate

Der Actor Critic Prozess

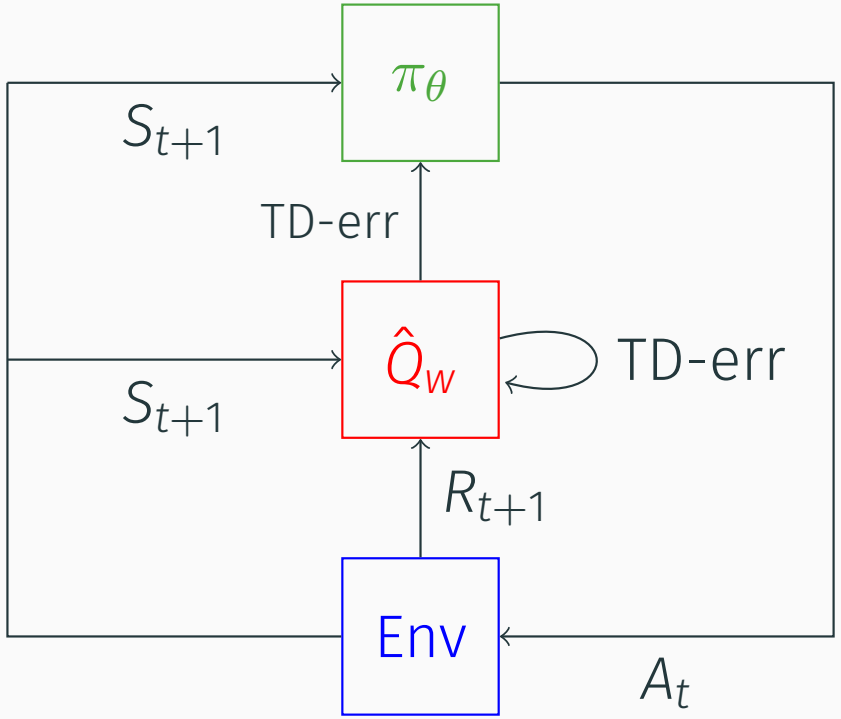




- in jedem Zeitschritt t
- gebe Zustand S_t vom Environment zu Actor und Critic

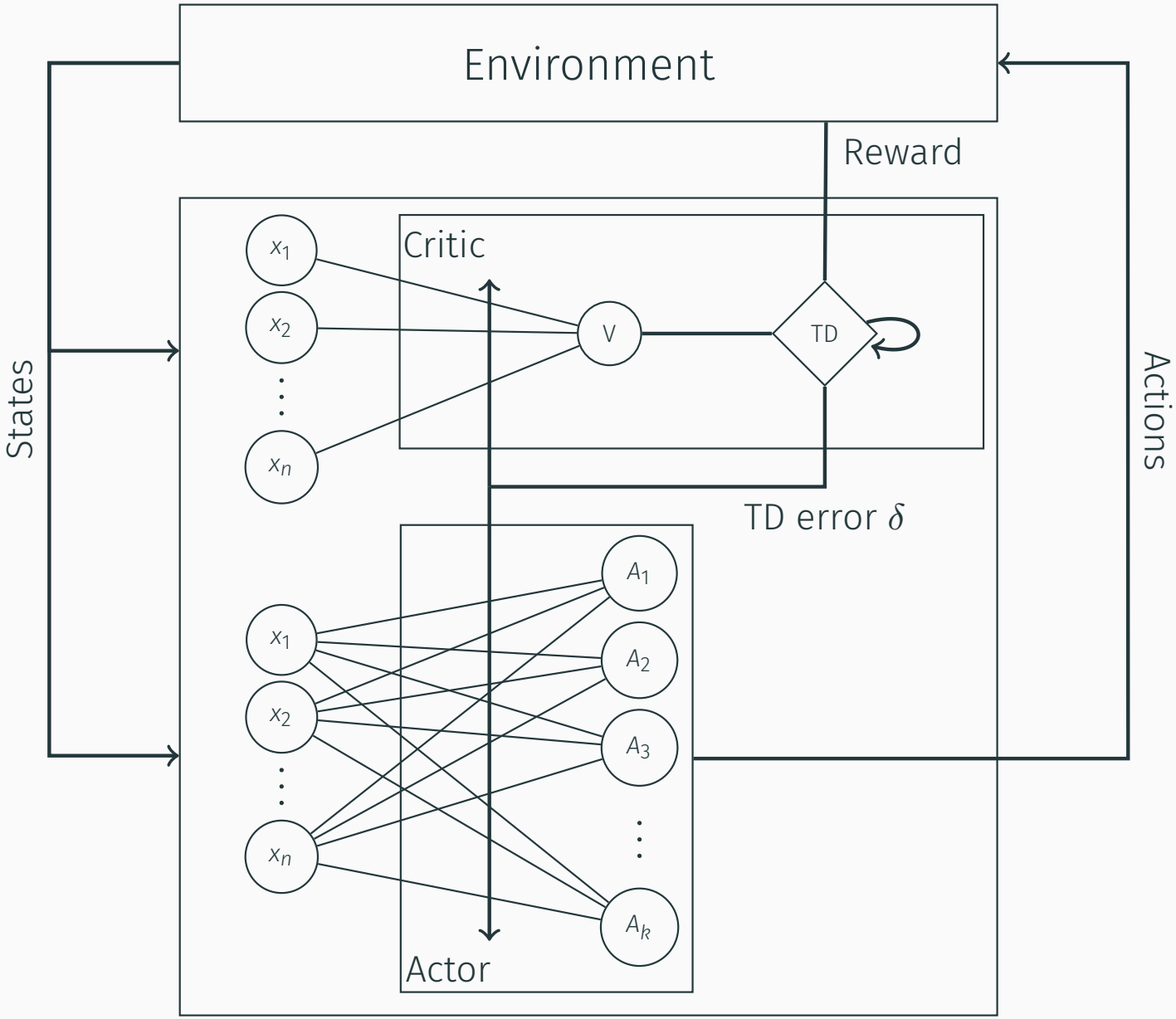
- Actor gibt eine Aktion A_t aus
- Actor und Critic bekommen neuen Zustand S_{t+1}
- Critic bekommt Reward R_{t+1}

Der Actor Critic Prozess(2)



- Critic berechnet den Wert den diese Aktion erbringt
- Actor und Critic verbessern ihre Hypothese

Actor Critic ANN



A2C und A3C

REINFORCE

REINFORCE-Update:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

- Return $v_t = Q_{\pi\theta}(s_t, a_t)$
- v_t = zukünftiger diskontierter Reward beginnend bei t
- jeder sofortige Reward = Zufallsvariable (Env und π stochastisch)
- $\Rightarrow$ v_t hängt von allen zukünftigen Zuständen ab $\rightarrow$ hohe Varianz (Schwankung)

Baseline

ORGANIC COMPUTING

- Wenn man eine Baseline $B(s)$ von v_t abzieht:
  - Kann man die Varianz verringern
  - ohne dabei das Ergebnis zu verändern
  - Eine gute Baseline ist die Value function $B(s) = V_{\pi\theta}(s)$
- $V_{\pi\theta}(s)$ ist der erwartete Reward beginnend in Zustands s zum Zeitpunkt t
- $v_t - V_{\pi\theta}(s)$ entfernt die Abhängigkeit von allen Zukünftigen Zuständen von v_t
- Das Ergebnis hängt nur von Zustand s ab $\rightarrow$ kleinere Varianz

Advantage

$$A(s,a) = \underbrace{Q(s,a)}_{Q-Value} - \underbrace{V(s)}_{avg\ Value}$$

- Verbesserung im Vergleich zum durchschnittlichen Wert des Zustands wenn man Aktion a wählt
- extra Reward wenn man a wählt
- $A(s,a) > 0 \rightarrow$ Gradient in diese Richtung verschieben
- $A(s,a) < 0 \rightarrow$ Gradient andere Richtung verschieben

Advantage(2)

Problem: zwei Value functions - $Q(s, a)$ und $V(s)$

Lösung: TD error

$$A(s, a) = Q(s, a) - V(s)$$

$$= \underbrace{r + \gamma V(s') - V(s)}_{\textit{TD Error}}$$

Asynchron vs. Synchron

ORGANIC COMPUTING

- A3C - Asynchronous Advantage Actor Critic:
  - kein Experience Replay
  - unterschiedliche Agenten laufen parallel
  - mehrere Instanzen des Environments
  - Jeder Worker updatet das globale Netz asynchron
- A2C - Advantage Actor Critic:
  - genauso wie A3C
  - Außer: synchrones Update des globalen Netzwerks
  - Warten bis alle Worker fertig sind mit dem Training
  - Gradient-Durchschnitt verwenden

Nachteile A3C

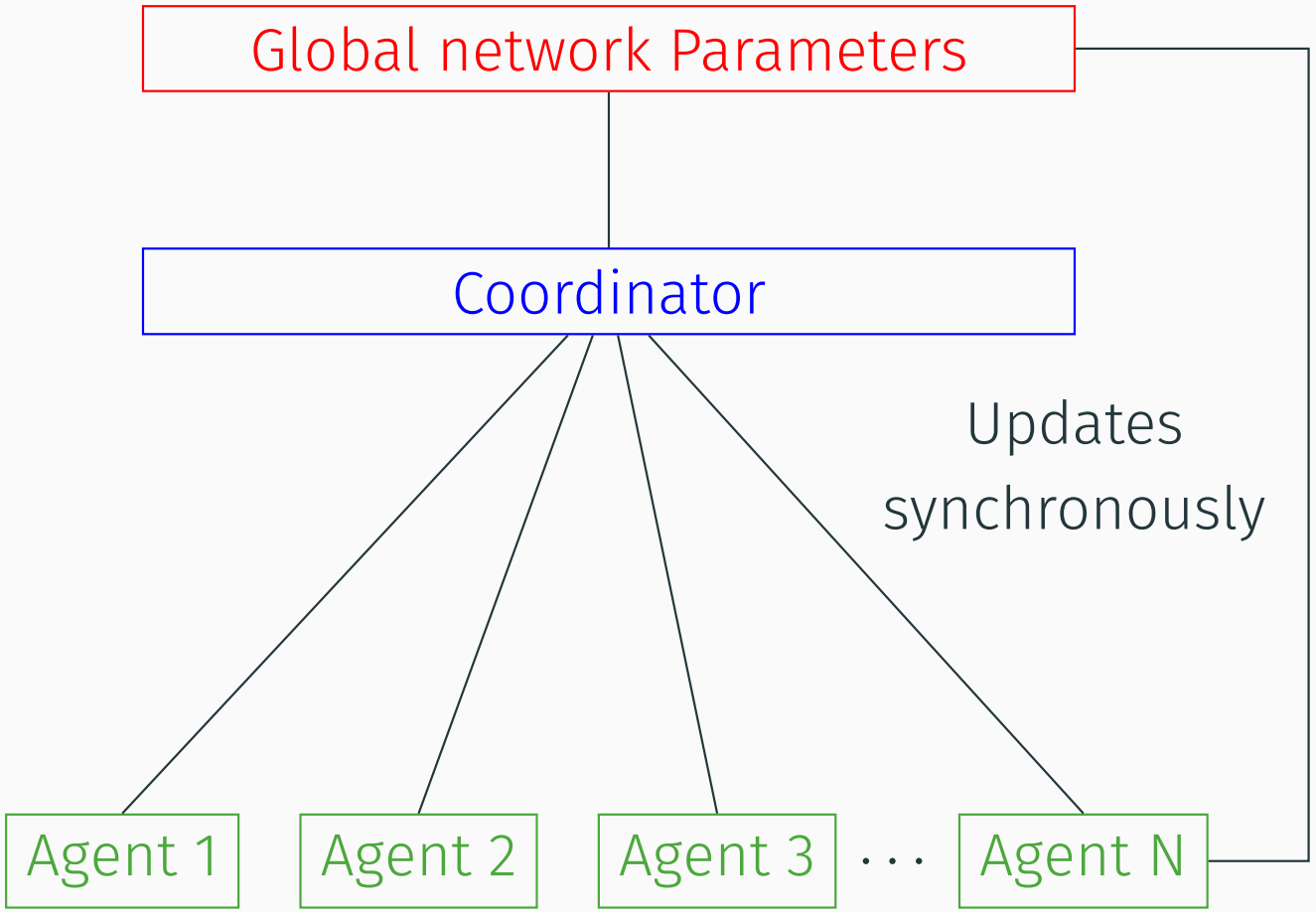
ORGANIC COMPUTING

- Jeder Agent kommuniziert unabhängig mit den globalen Variablen
- es kann passieren, dass die Agenten mit unterschiedlichen Versionen der Policy arbeiten
- $\Rightarrow$ nicht ganz optimales aggregiertes Update

$\Downarrow$

A2C verwendet einen Koordinator der abwartet bis alle Agenten eine Episode beendet haben

A2C - Architektur



⇒ Training zusammenhängender und schneller

Implementierung von A2C

N-step return

ORGANIC COMPUTING

- normalerweise: 1-step return - $Q(s, a), V(s)$

$$V(s_0) \leftarrow r_0 + \gamma V(s_1)$$

- mann kann auch mehr steps verwenden

$$V(s_0) \leftarrow r_0 + \gamma r_1 + \gamma^2 V(s_2)$$

- Allgemein: N-step return

$$V(s_0) \leftarrow r_0 + \gamma r_1 + \cdots + \gamma^n V(s_n)$$

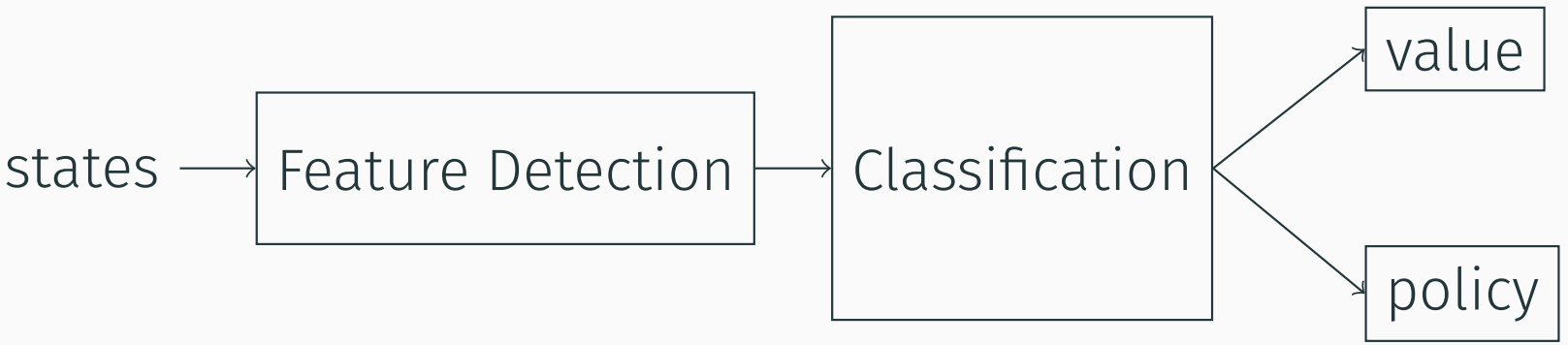
N-step return - Vor- und Nachteile

ORGANIC COMPUTING

- Änderungen werden schneller propagiert
  - unerwarteter Reward tritt auftbt
  - 1-step return: Value function ändert sich langsam (1 Schritt rückwärts/Iteration)
  - n-step return: Veränderung wird pro Iteration n Schritte zurückpropagiert
- höhere Varianz $\rightarrow$ Value hängt von einer Kette von Aktionen ab, welche in viele unterschiedliche zustände führen können
- Verzögert

Ein Netz

- Feature Detection und Klassifikation $\rightarrow$ Verständnis vom Problem
- Bei zwei Netzen muss das Problem unabhängig voneinander 2x gelernt werden $\Rightarrow$ doppelter Trainingsaufwand
- Lösung: ein Netz mit Verzweigung am Ende (2 Outputs)



Loss Function

$$L = L_\pi + c_{val} L_{val} + c_H H$$

L_π: Policy Loss

c_{val}: Konstante (0.5)

L_{val}: Value Loss

v_H: Konstante (0.01)

H: Policy Entropy

Policy Loss

ORGANIC COMPUTING

$$L_\pi = -J(\pi)$$

$$J(\pi) = E\big[A(s,a)\nabla \log \pi(a|s)\big]$$

Durchschnitt aller Samples eines Batches:

$$L_\pi = -\frac{1}{n}\sum_{i=1}^{n} A(s_i,a_i) * \log \pi(a_i,s_i)$$

Achtung: $A(s,a)$ ist hier eine Konstante: tf.stop_gradient()

Value Loss

ORGANIC COMPUTING

Analog zu Q-Learning
n-step scenario:

$$V(s_0) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^{n-1} r_{n-1} + \gamma^n V(s_n)$$

Fehler/Advantage:

$$err = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^{n-1} r_{n-1} + \gamma^n V(s_n) - V(s_0)$$

Mean squarred error:

$$L_{val} = \frac{1}{n} \sum_{i=1}^{n} err_i^2$$

Entropy

ORGANIC COMPUTING

- Entropy zur Loss function hinzuzufügen verbessert die Exploration und verhindert vorzeitiges konvergieren zu einer suboptimalen Policy

$$H(\pi(s)) = -\sum_{k=1}^{n} \pi(s)_k * \log \pi(s)_k$$

  $\pi(s)_k$: Wahrscheinlichkeit für die $k-$te Aktion in zustand s
- $\min H = [1, 0, 0, 0]$
- $\max H = [0.25, 0.25, 0.25, 0.25]$
- $\Rightarrow$ durch das maximieren $(-H)$ der Entropy bewegen wir die Policy weg vom deterministischen Zustand
- Durchschnitt über alle Samples

$$L_{reg} = -\frac{1}{n} \sum_{i=1}^{n} H(\pi(s))$$

EnvVector

ORGANIC COMPUTING

- managed die einzelnen Instanzen des Environments
- jedes Environment wird von einem Worker gesteuert (eigener Thread)
- Kommunikation:

```
1 from multiprocessing import Process, Pipe
```

- Aktionen der Worker:
  1. reset
  2. step
  3. close

Quellen

Bilder

ORGANIC COMPUTING

- https://medium.freecodecamp.org/an-introduction-to-policy-gradients-with-cartpole-and-doom-495b5ef2207f