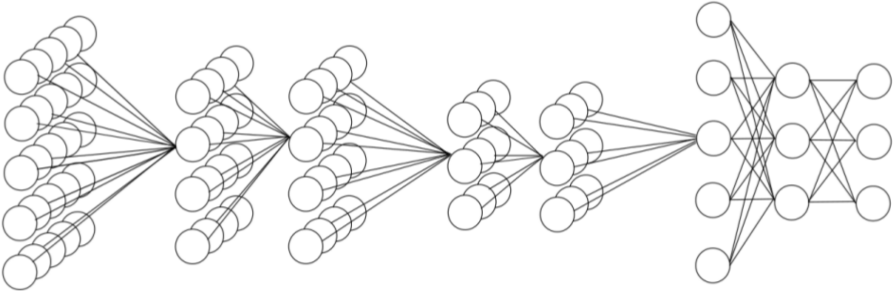
Deep Reinforcement Learning with Continuous Actions

Simon Ramstedt
simonramstedt@gmail.com
Supervisors: Simone Parisi, Gerhard Neumann



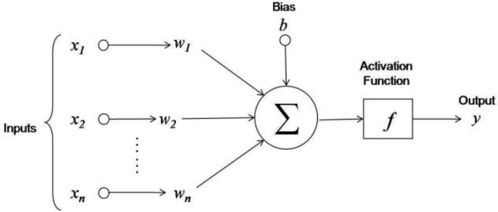
Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

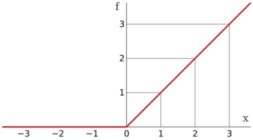
- Neural Networks work well with **high dimensional** real world data
- Reinforcement Learning for sequential decision making proved successful for robot learning
- Deep Reinforcement Learning successes (at least for discrete actions): AlphaGo and Atari games
- Continuous actions needed for many control tasks

Neural Networks

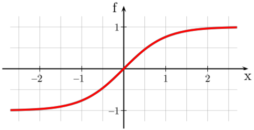
TECHNISCHE
UNIVERSITÄT
DARMSTADT



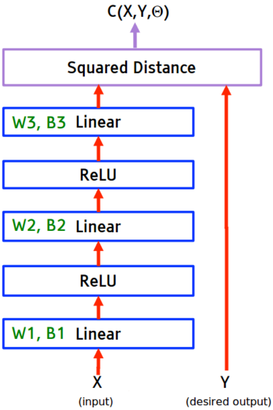
$y = f(\sum_i w_i x_i + b)$



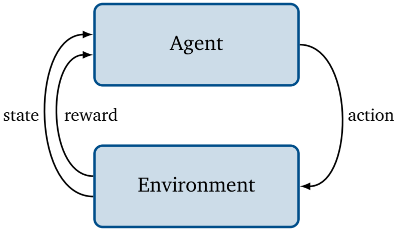
(a) ReLU: $f(x) = \max(0, x)$.



(b) Tanh: $f(x) = \tanh(x)$.



Reinforcement Learning



Objective: Maximize the return (i.e. sum of rewards)

$$R = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$$

Policy

$\pi : s_t \rightarrow a_t$

Value function

$V(s_t) = \mathbb{E}[R|s_t]$

Action value function

$Q(s_t, a_t) = \mathbb{E}[R|s_t, a_t]$

Transition model

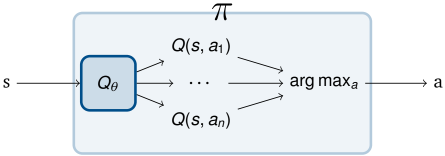
$M : s_t, a_t \rightarrow s_{t+1}, r_{t+1}$

Deep Q-Network

Q-learning with a Neural Network

TECHNISCHE UNIVERSITÄT DARMSTADT

Bellman equation: $Q^{\pi}(s_t, a_t) = \mathbb{E}[r(s_t, a_t) + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1}))]$



- Problem: correlated samples
- Solution: experience replay
- Problem: recursive Q targets
- Solution: target weights $\theta' = \mathrm{EMA}(\theta)$

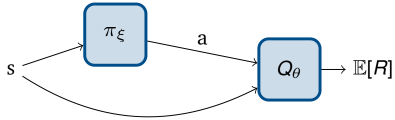
Deep Q-Network

for *every timestep* t **do**

- Select action $a_t = \epsilon$-greedy $[\text{argmax}_a\, Q(s_t, a|\theta)]$
- Execute a_t and observe reward r_t and next state s_{t+1}
- Store transition (s_t, a_t, r_t, s_{t+1}) in D
- Sample random minibatch of m transitions from D
- Set targets $y_j = r_j + \gamma \max_a Q(a, s_{j+1}|\theta')$
- Perform gradient descent on cost $C = \frac{1}{m}\sum_j (y_j - Q(s_j, a_j|\theta))^2$ with respect to θ
- Update $\theta' \leftarrow \text{LP}(\theta)$

end

Deep Deterministic Policy Gradient (DDPG)



- Policy gradient $\nabla_\xi \mathbb{E}[R] = \nabla_a Q \cdot \nabla_\xi \pi$
- Use tricks from DQN

- 20+ different motor control tasks (pole swingup, cart-pole, double pole, quadruped balance, ...)
- Evaluated on state inputs and pixel inputs
- Smooth reward functions
- Performance: max. $\mathbf{10^6}$ timesteps of interaction needed

Deep Deterministic Policy Gradient

TECHNISCHE UNIVERSITÄT DARMSTADT

for *every timestep t* **do**

- Select action $a_t = \pi(s_t) + \mathcal{M}_t$
- Execute a_t and observe reward r_t and next state s_{t+1}
- Store transition (s_t, a_t, r_t, s_{t+1}) in D
- Sample random minibatch of m transitions from D
- Set Q targets $y_j = r_j + \gamma Q(s_{j+1}, \pi(s_{j+1}|\xi')\ |\theta')$
- Perform gradient descent on cost $C = \frac{1}{m}\sum_j (y_j - Q(s_j, a_j|\theta))^2$ with respect to θ
- Perform gradient ascent on $Q(s_j, \pi(s_j|\xi)\ |\theta)$ with respect to ξ
- Update $\theta' \leftarrow \mathrm{LP}(\theta)$
- Update $\xi' \leftarrow \mathrm{LP}(\xi)$

end

Experiments

DDPG sounds nice but ...

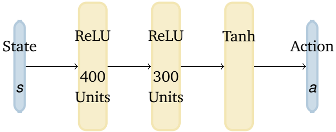
TECHNISCHE UNIVERSITÄT DARMSTADT

- How hard is it to implement?
- How data efficient is it?
- What about sparse reward functions?
- How robust is it to hyperparameters variations?
- Other problems?
- How can it be improved?

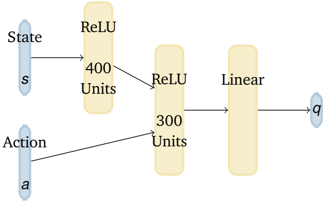
Experiments

Setup

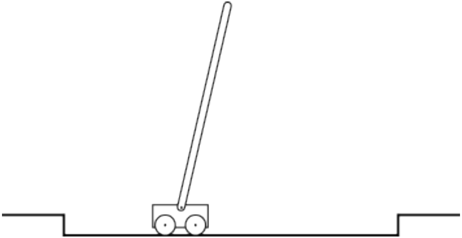
Policy network layout:



Q network layout:



Cart pole swingup and balance



s_0 : position of the cart

s_1 : angle of the pole

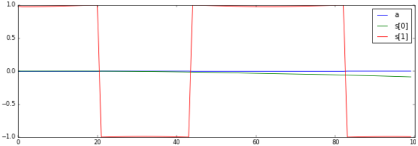
$s_2 = \dot{s}_0$

$s_3 = \dot{s}_1$

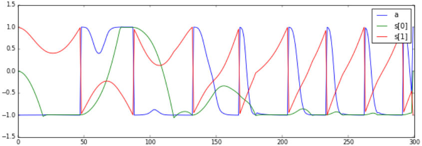
a : force applied to the cart

Experiments

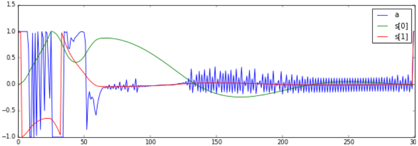
Trajectories



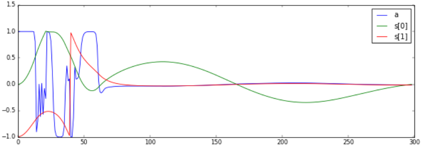
(a) Trajectory after 0 timesteps of learning.



(b) Swingup after $\sim 10,000$ timesteps.



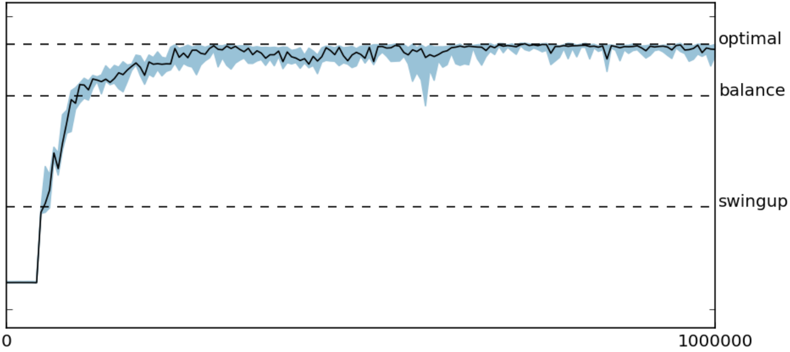
(c) Balance after $\sim 100,000$ timesteps.



(d) Near optimal after $\sim 200,000$ timesteps.

Results

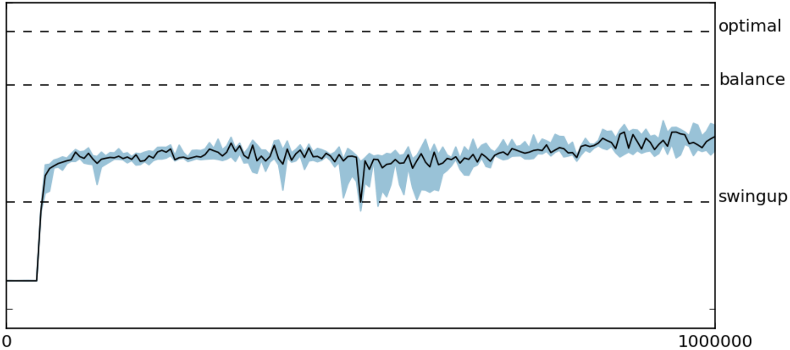
DDPG in Cartpole



$$r = 0.5 \cdot \cos(s_1) - 0.03 \cdot a^2 - 0.015 \cdot |s_0| - 0.2 \cdot s_3^2 \quad \Big| \quad t_{\text{warmup}} = 50,000$$

Results

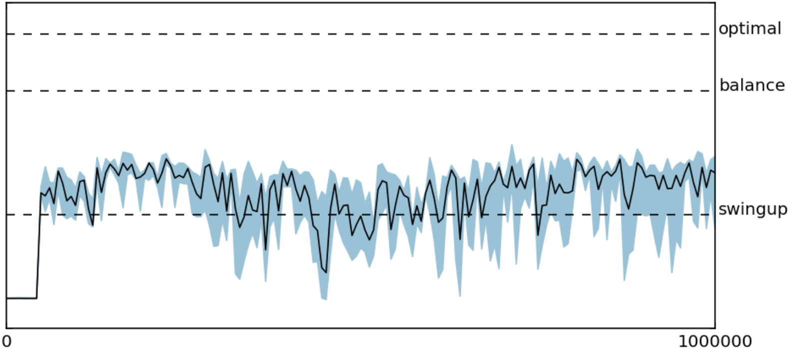
DDPG in Cartpole without target networks



$r = 0.5 \cdot \cos(s_1) - 0.03 \cdot a^2 - 0.015 \cdot |s_0| - 0.2 \cdot s_3^2$ | $t_{\text{warmup}} = 50,000$

Results

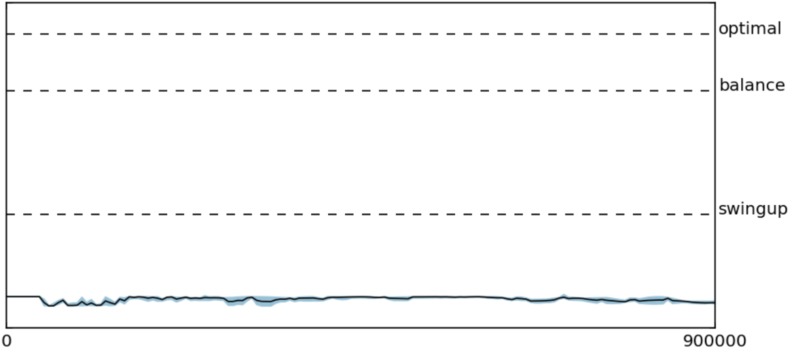
DDPG in Cartpole without replay memory



$$r = 0.5 \cdot \cos(s_1) - 0.03 \cdot a^2 - 0.015 \cdot |s_0| - 0.2 \cdot s_3^2 \quad \Big| \quad t_{\text{warmup}} = 50,000$$

Results

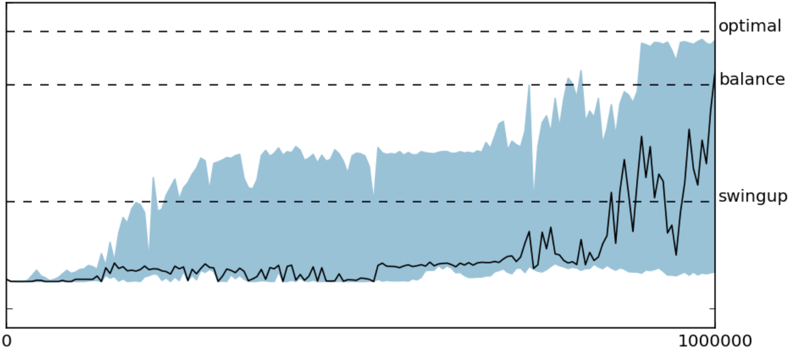
DDPG in Cartpole with sparse reward function



$r = 1 \cdot (s_1 < 0.01) - 0.03 \cdot a^2$ | $t_{\text{warmup}} = 10,000$

Results

DDPG in Cartpole with sparse reward function

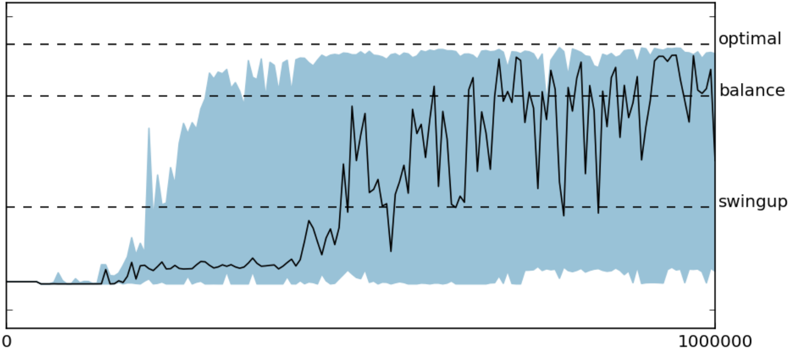


$r = \mathbf{4} \cdot (s_1 < 0.01) - 0.03 \cdot a^2$

$t_{\text{warmup}} = 10{,}000$

Results

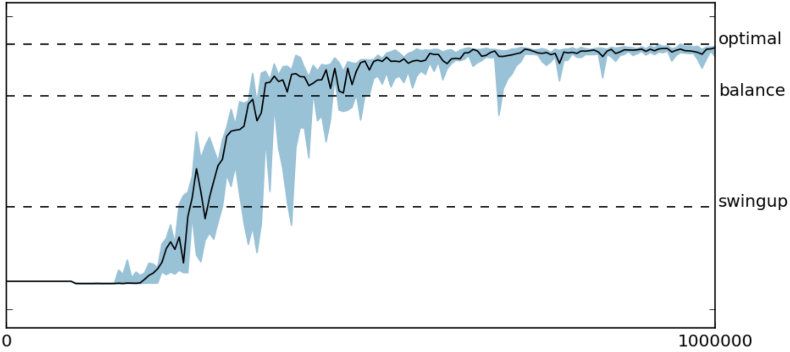
DDPG in Cartpole with sparse reward function



$r = 4 \cdot (s_1 < 0.01) - 0.03 \cdot a^2$ | $t_{\text{warmup}} = \mathbf{50,000}$

Results

DDPG in Cartpole with sparse reward function

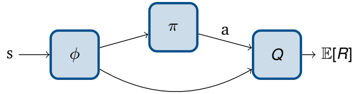


$r = 4 \cdot (s_1 < 0.01) - 0.03 \cdot a^2$ | $t_{\text{warmup}} = \mathbf{100,000}$

Future Work

Increase data efficiency

Share weights:



Learn a **transition model**

Use it to compute another policy gradient (e.g. Heess et. al., 2015)

$$\begin{aligned}\mathbb{E}[R] &= r(s_t, a_t) + \lambda V(M(s_t)) \\ \nabla_\xi \mathbb{E}[R] &= \nabla_\xi r(s_t, a_t) + \lambda \nabla_\xi \pi(s_t) \cdot \nabla_a M(s_t, a_t) \cdot \nabla_{s_{t+1}} V(s_{t+1})\end{aligned}$$

Future Work

TECHNISCHE UNIVERSITÄT DARMSTADT

Faster reward propagation via **n-step updates** on recent trajectories (e.g. Mnih et. al., 2016)
$Q^\pi(s_t, a_t) = \mathbb{E}[r_t + \gamma^1 r_{t+1} + \cdots + \gamma^n r_{t+n} + \gamma^{n+1} Q^\pi(s_{t+n+1}, a_{t+n+1})]$

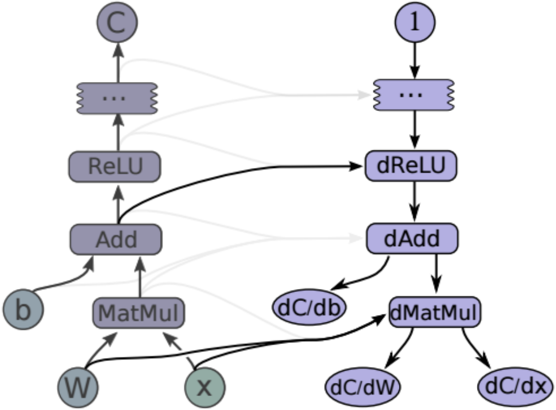
Stochastic neural networks:
a) Additive noise in the layers (works with standart backpropagation)
b) Stochastic backpropagation (Kingma & Welling, 2013) (Rezende et. al., 2014)

Thanks

Questions or Feedback ?

- Thesis and slides: https://github.com/simonramstedt/bt
- Code: https://git.ias.informatik.tu-darmstadt.de/SimonR/DRLTF
- Deep Reinforcement Learning papers overview:
  https://github.com/junhyukoh/deep-reinforcement-learning-papers
- DQN: Mnih et. al., Human-level control through deep reinforcement learning, 2015
- DDPG: Lillicrap et. al., Continuous control with deep reinforcement learning, 2015
- A3C: Mnih et. al, Asynchronous Methods for Deep Reinforcement Learning, 2016
- Prioritized Replay: Schaul et. al., Prioritized Experience Replay, 2015

Automatic differentiation



Deep Q-Network (DQN)

on the Arcade Learning Environment (ALE)

Atari Environment:

- **Discrete** actions (buttons on Atari controller)
- Learning from **raw pixel** input
- Using only the game score as reward signal
- Performance: max. $\mathbf{10^8}$ timesteps of interaction needed

Method	Training Time	Mean	Median
DQN (from [Nair et al., 2015])	8 days on GPU	121.9%	47.5%
Gorila [Nair et al., 2015]	4 days, 100 machines	215.2%	71.3%
Double DQN [Van Hasselt et al., 2015]	8 days on GPU	332.9%	110.9%
Dueling Double DQN [Wang et al., 2015]	8 days on GPU	343.8%	117.1%
Prioritized DQN [Schaul et al., 2015]	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%