Continuous Deep Q-Learning ~~with Model-based Acceleration~~

Sebastian Mueller

May 30, 2018

Motivation

- Many real world tasks are continuous
- Model-free RL: + no feature engineering
  - high sample complexity
- Model-based RL: + more efficient
  - model limits performance of policy
- Goal: Combine both advantages
  - Derive continous variant of Q-Learning
  - Decrease sample complexity

Motivation

Common approaches in continuous domains:

- policy gradient descend
- actor-critic-methods

$\Rightarrow$ high sample complexity
What about Q-Learning?

Motivation

What about Q-Learning?

- off-policy algorithm
- only one optimization goal
- for discrete domains

Normalized Advantage Function (NAF)

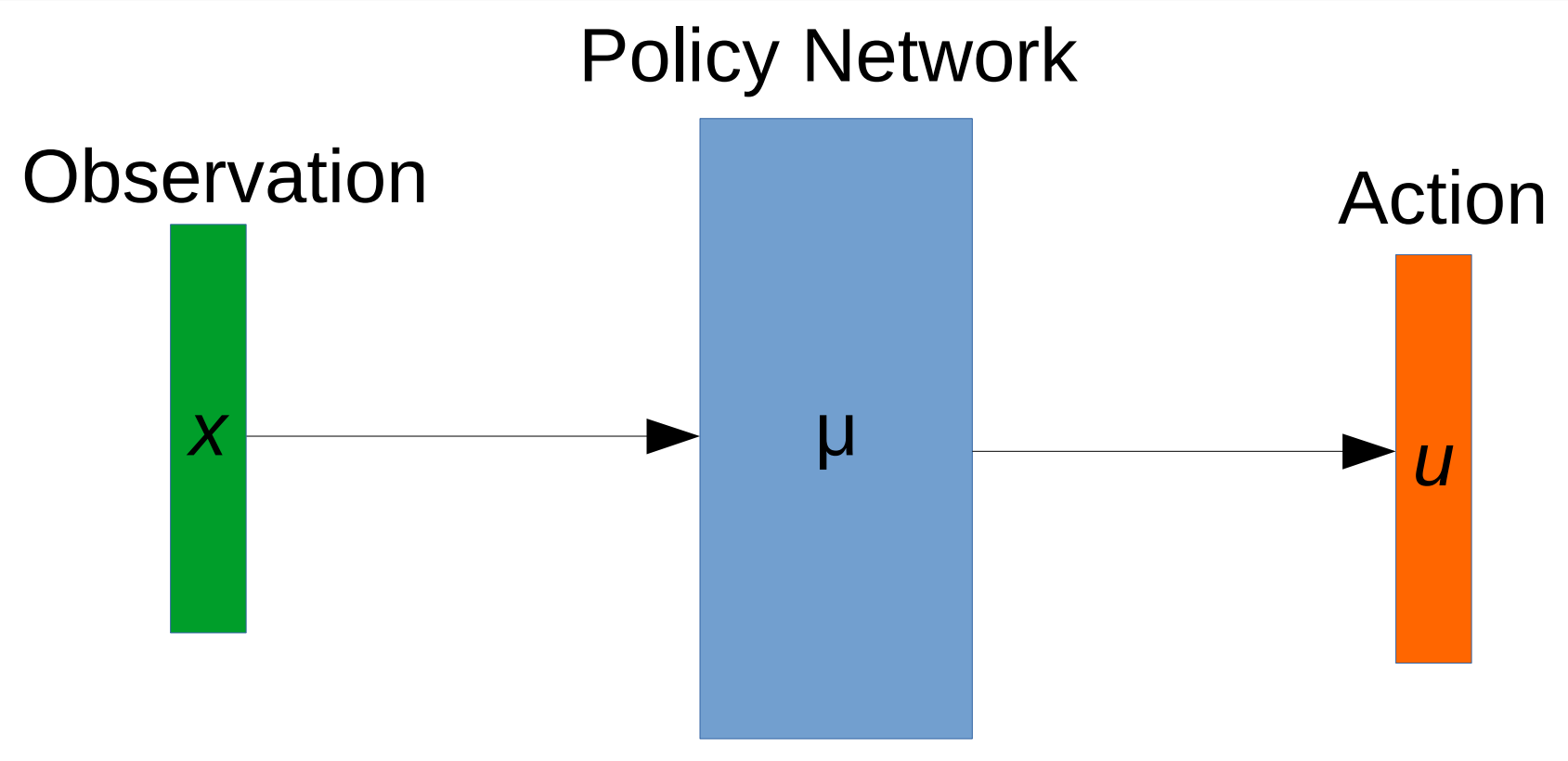
- Classical Q-Learning:
  $Q(x_t, u_t) = Q(x_t, u_t) + \alpha[R_{t+1} + \gamma \max_a Q(x_{t+1}, a) - Q(x_t, u_t)]$
- Decomposing Q (Baird III (1993), Advantage Updating):
  $Q(x_t, u_t) = \underbrace{A(x_t, u_t)}_{advantage-term} + \underbrace{V(x_t)}_{state-value-term}$
  with $A(x_t, u_t) = Q(x_t, u_t) - V(x_t)$
- Normalized Advantage Function:
  $Q(x, u|\theta^Q) = A(x, u|\theta^A) + V(x|\theta^V)$
  $A(x, u|\theta^A) = -\frac{1}{2}(u - \underbrace{\mu(x|\theta^\mu)}_{policy})^T \underbrace{\mathbf{P}(x|\theta^P)}_{positive-definite} (u - \mu(x|\theta^\mu))$

Continuous Q-Learning with NAF

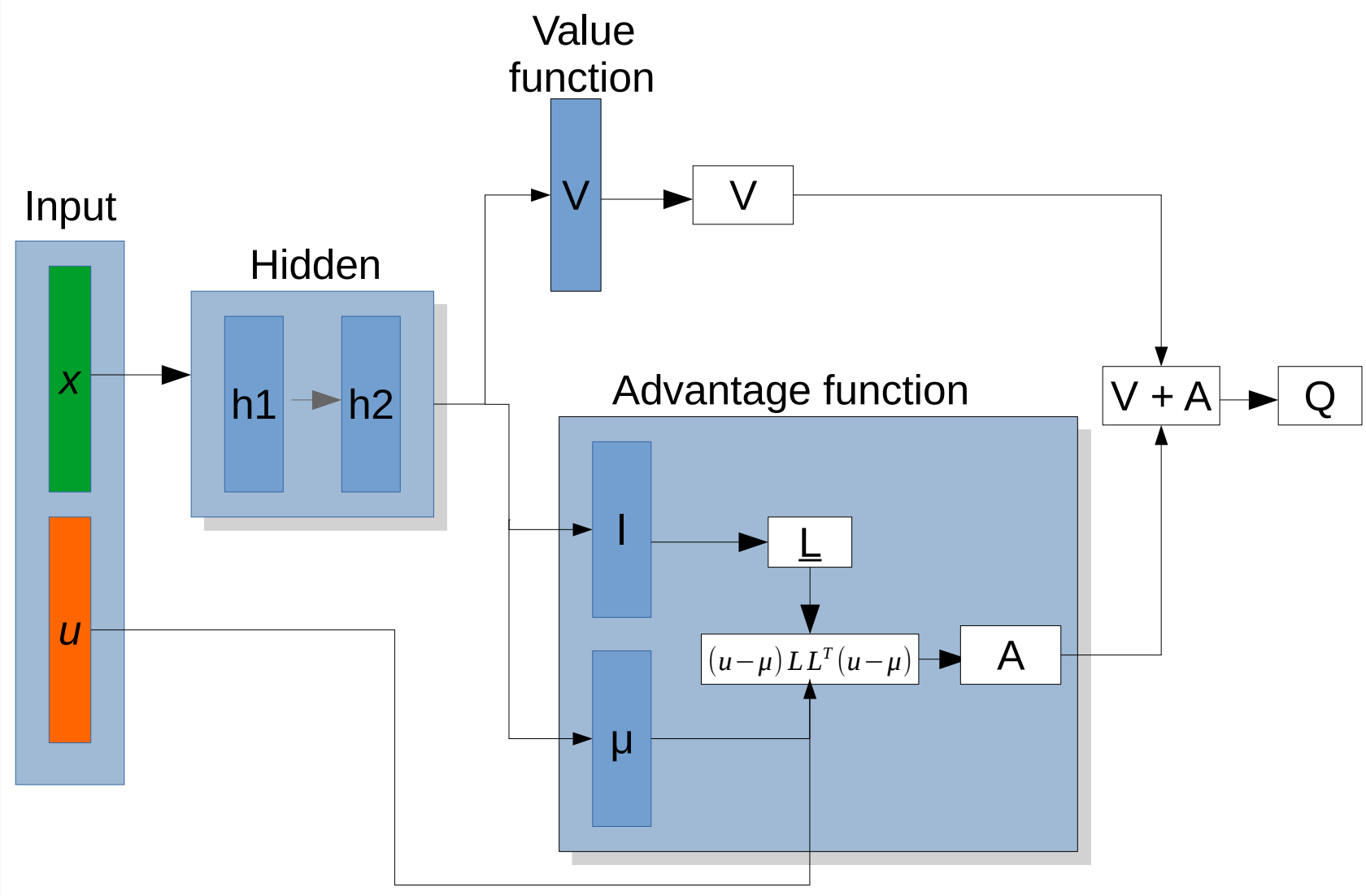
Algorithm 1 Continuous Q-Learning with NAF

Randomly initialize normalized Q network $Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q)$.
Initialize target network Q' with weight $\theta^{Q'} \leftarrow \theta^Q$.
Initialize replay buffer $R \leftarrow \emptyset$.
for episode=1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $\boldsymbol{x}_1 \sim p(\boldsymbol{x}_1)$
  for t=1, T **do**
    Select action $\boldsymbol{u}_t = \mu(\boldsymbol{x}_t|\theta^\mu) + \mathcal{N}_t$
    Execute $\boldsymbol{u}_t$ and observe r_t and $\boldsymbol{x}_{t+1}$
    Store transition $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1})$ in R
    for iteration=1, I **do**
      Sample a random minibatch of m transitions from R
      Set $y_i = r_i + \gamma V'(\boldsymbol{x}_{i+1}|\theta^{Q'})$
      Update θ^Q by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(\boldsymbol{x}_i, \boldsymbol{u}_i|\theta^Q))^2$
      Update the target network: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    end for
  end for
end for

Architecture: Choosing an action

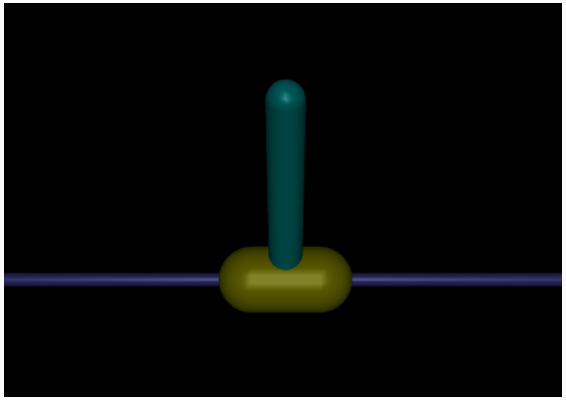
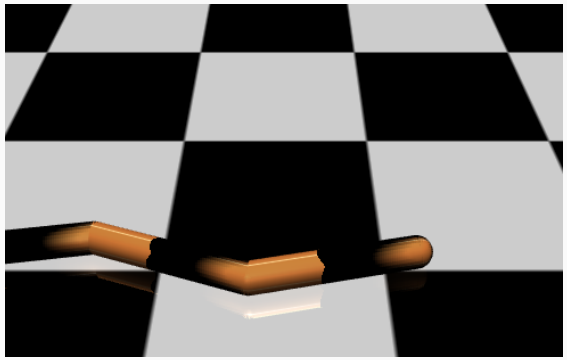
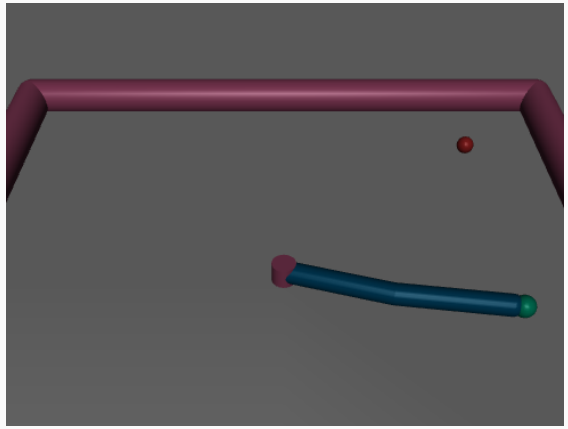
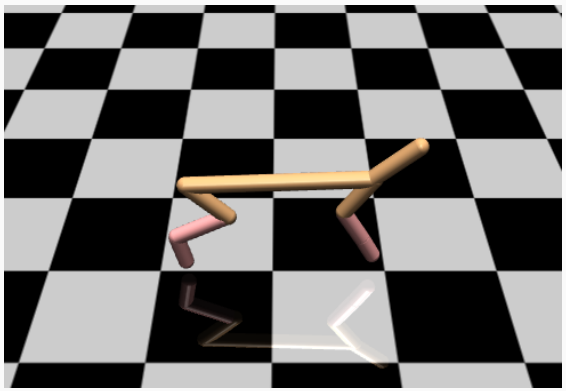
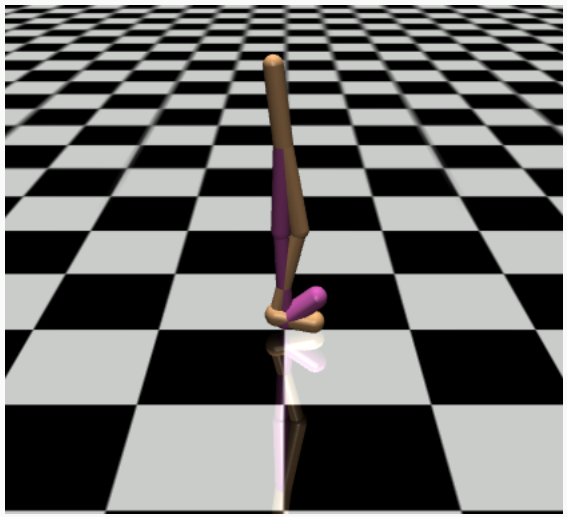
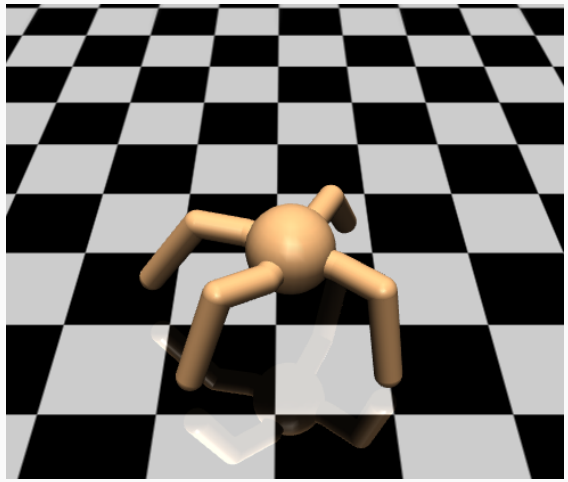


Architecture: Computing the Q-Value



Experiments

Networks were trained on various locomotion tasks in the gym/ mujoco framework

Experiments

Environment	Observation size	Action size
Inverted Pendulum	4	1
Swimmer	7	2
Reacher	10	2
Half Cheetah	16	6
Walker2d	16	6
Ant	110	8