Neural Networks

Lecture 11: Neural Reinforcement Architectures

http://www.informatik.uni-hamburg.de/WTM/

Motivation & State of the Art

- Human level control in complex environments [Minh et al. 2015]
- Basic idea: Learn from rewards
  - Advantages
    - No expert knowledge needed
  - Drawbacks
    - High training times
    - Lots of „useless“ or dangerous actions

Benefits of (Deep) Reinforcement Learning

- No annotated training data needed
  - E.g. complex control problems
  - Pilot an airplane - crash gives negative reward
- Learning of action sequences
- Model free
- End-to-end learning

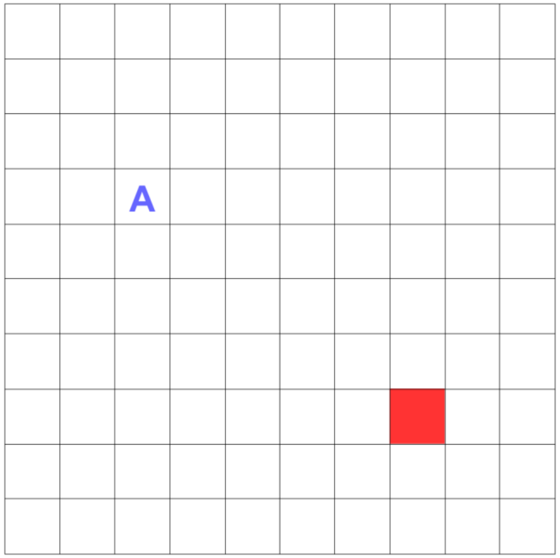
Overview

- Reinforcement learning
- Neural architecture for reinforcement learning
- Deep reinforcement learning – continuous states
  - Stability
- Continuous deep reinforcement learning
  - Actor-Critic architecture
- Related approaches

Part 1:

Reinforcement Learning

A Navigation Example



- Actions:
  - Up ↑
  - Down ↓
  - Left ←
  - Right →

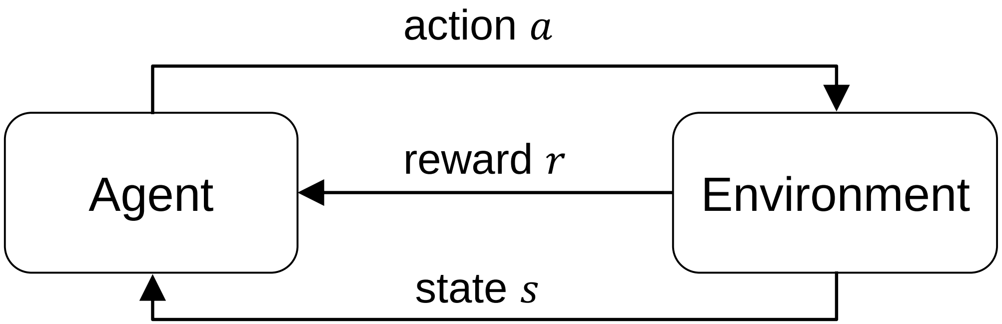
- Objective: get to the target position as quickly as possible

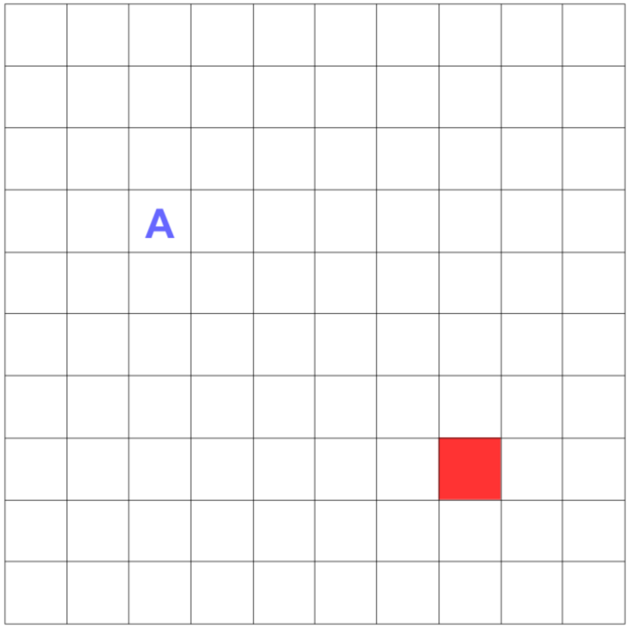
- Condition: no knowledge about the environment or actions outcome, only at the end of the sequence

- Method: Reinforcement Learning

Agent-Environment Interaction

- Reinforcement Learning (RL):
  - Perceive state s
  - Select and perform an action a
  - Sometimes receive reward r

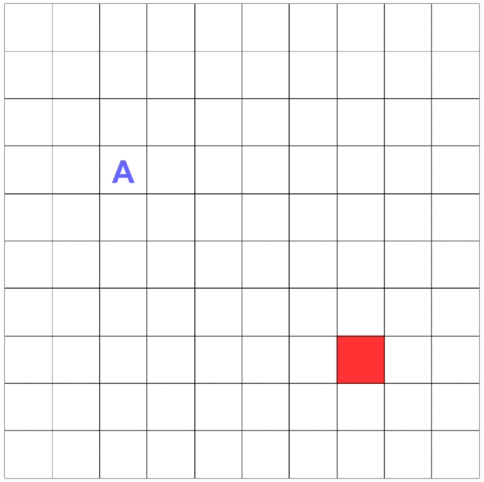




- Markov Decision Process (MDP):
  - fixed transition probabilities
  - fixed reward probabilities
  - independent of previous states

[Sutton, Barto 98]

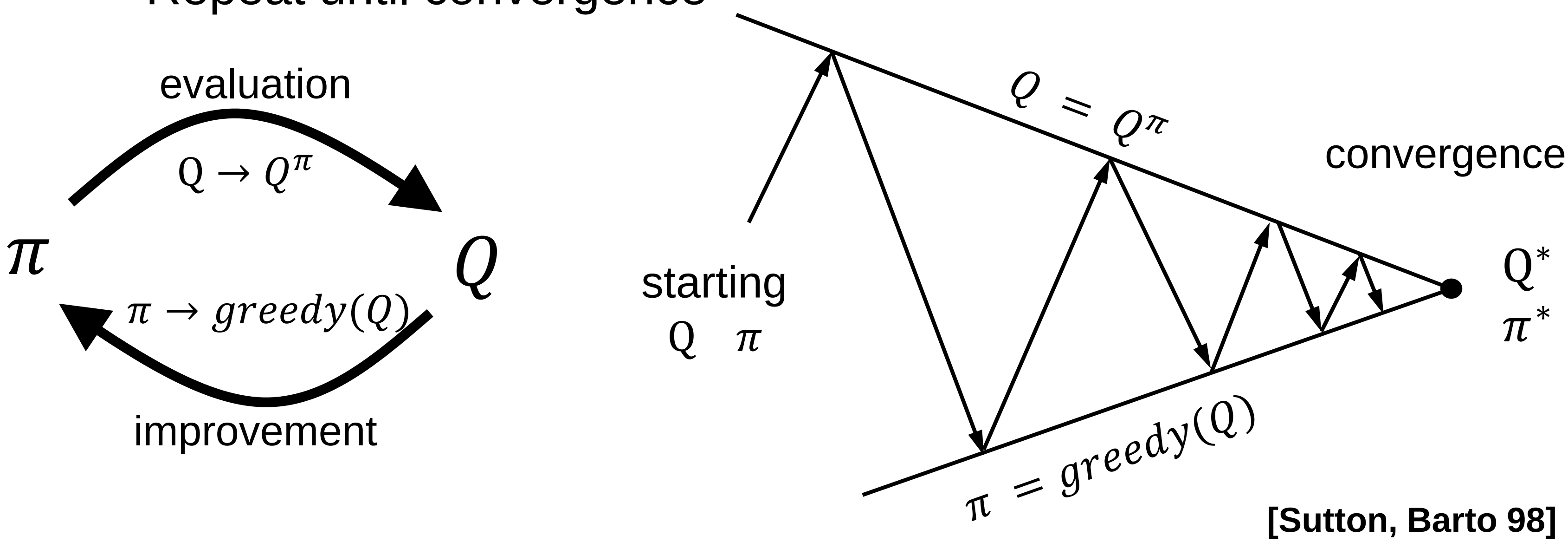
Elements for Learning



- Feedback in the form of reward r
  - Different ways to model rewards
- Stopping conditions
  - When to stop taking actions
  - E.g. when does a vacuum-bot stop to clean
- Action-value function $Q(s, a)$
  - How „valuable“ is an action a in state s?
  - Alternative: State-value function $V(s)$
- A strategy to discover solutions policy π
  - E.g. greedy strategy: $\pi(s) = argmax_a\, Q(s, a)$

Generalized Policy Interaction

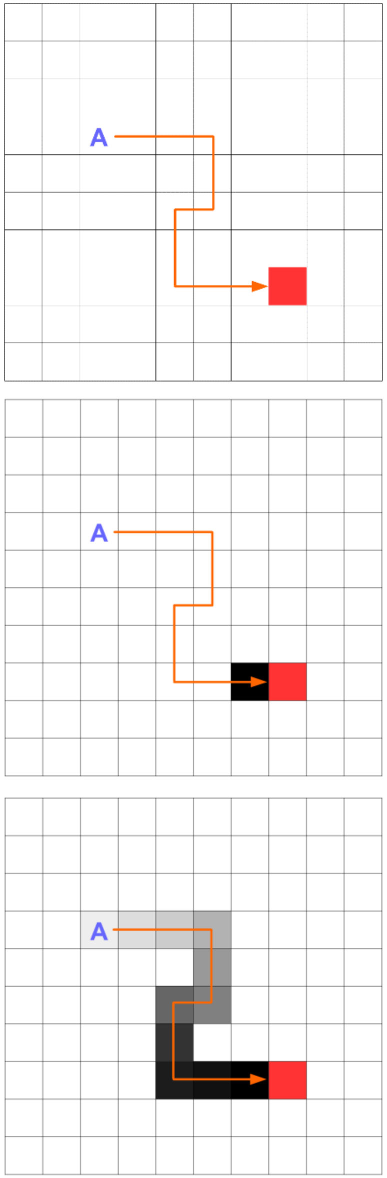
- Value functions $Q(s,a)$ and policy π closely interact during search for optimal functions Q^*, π^* that perform optimal actions
  - π selects action based on Q
    (i.e. the action *a* with the highest Q(s,a) for a given state *s*)
  - Q is updated by actual reward
  - Update to Q changes action selection of π
  - Repeat until convergence



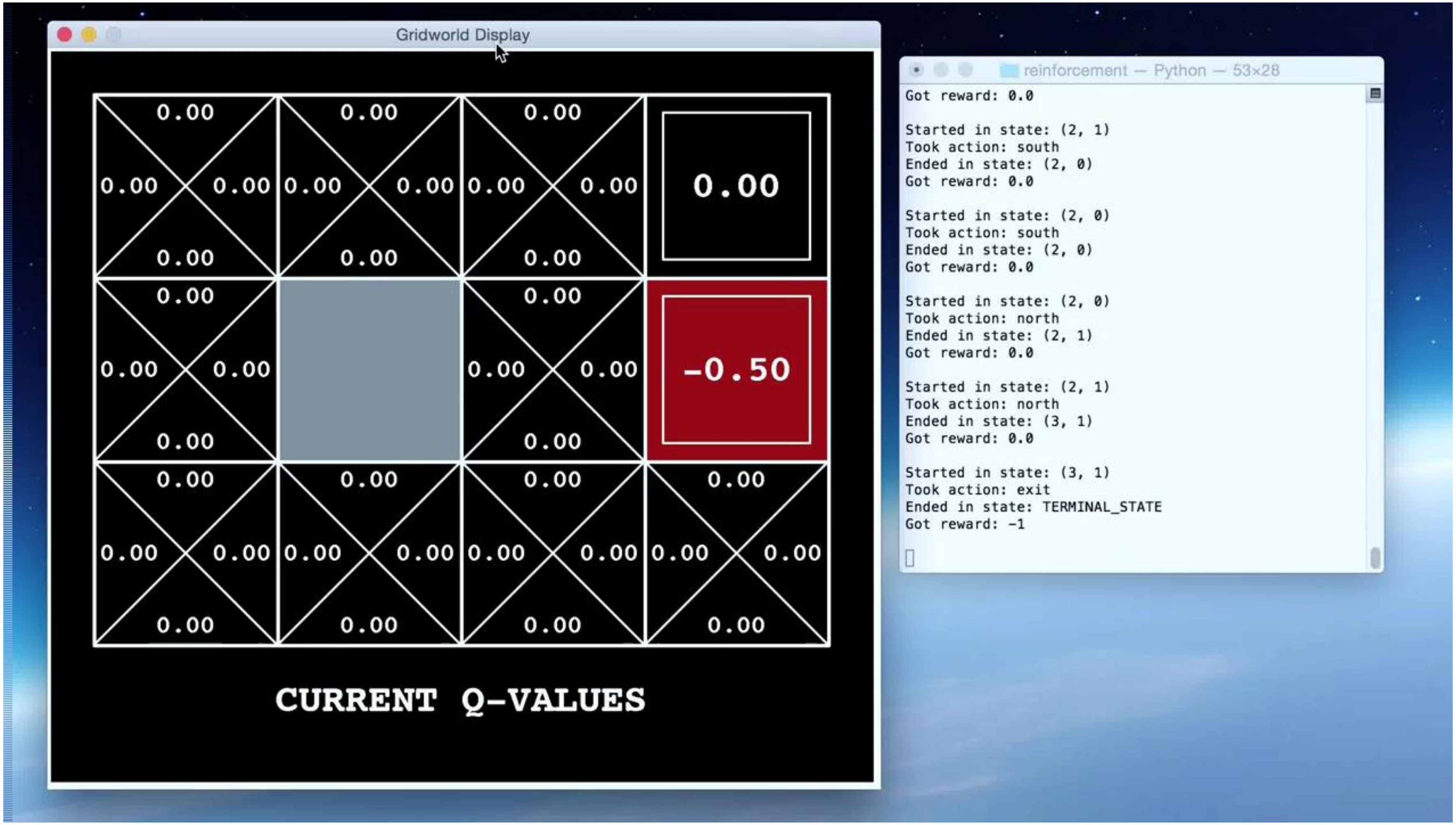
[Sutton, Barto 98]

How to Find Q-Function

- With some luck the agent reaches the goal receiving a reward r
- Learn → update the expectation
  $$Q(s,a) \leftarrow r$$
- What to do without immediate reward after action?
  Temporal-difference learning:
  $$Q(s,a) \leftarrow r + \gamma\, Q(s',a')$$
- γ: importance of future reward
  $0 < \gamma < 1$ discount factor
  large γ → far-sighted, planning
  small γ → impatient robots



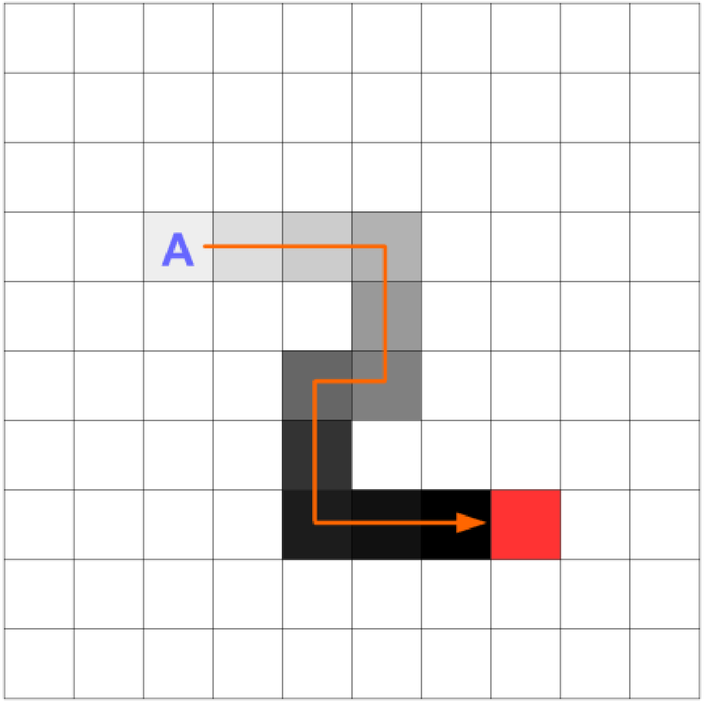
Gridworld Example



[https://www.youtube.com/watch?v=RTu7G0y4Os4]

Optimal Policy

- How to find the optimal path?



- Random exploration can find better solutions
- Exploitation vs. exploration

Action Selection Policies

Action selection:

- randomly
- greedy $\arg\max_{i'} Q(s_j, a_{i'})$
- ϵ-greedy
- Boltzmann (soft-max)

$$P(a_i = 1) = \frac{e^{h_i/\tau}}{\sum_k e^{h_k/\tau}},\ h_n = Q(s,a_n)$$

large $\tau \rightarrow$ large $\epsilon \rightarrow$ exploration

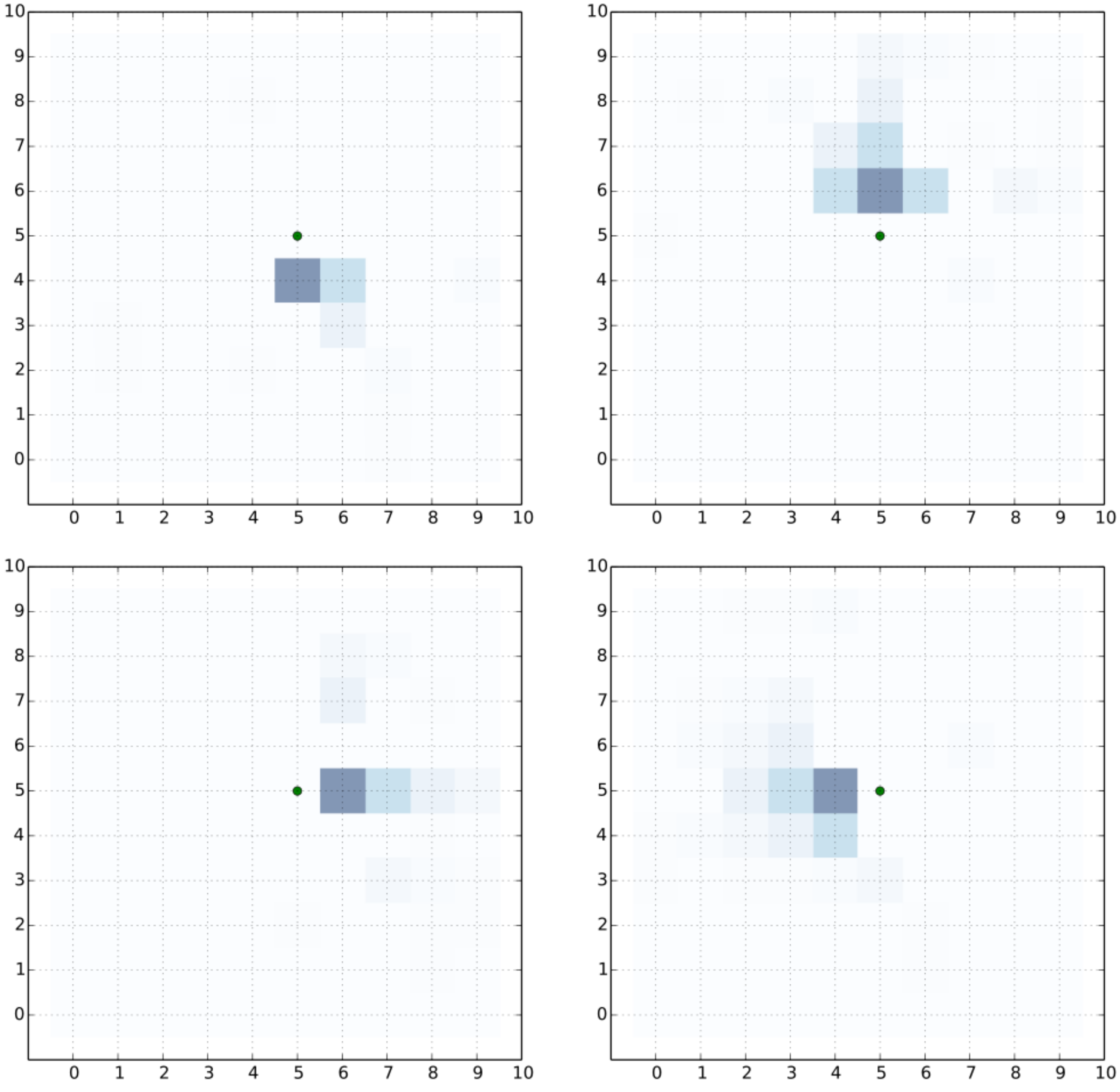
small $\tau \rightarrow$ small $\epsilon \rightarrow$ exploitation

```
ε-greedy

if random(0,1) < ε:
  choose random action
else
  chose argmaxi' Q(sj,ai)

small chance to perform
random action; else
greedy
```

After Enough Exploration



Part 2:

Neural Architecture for Reinforcement Learning

How does the neural architecture work?

- Classification or regression?
  - Regression for function approximation
- What function is approximated
  - Q-Function Q(a,s)
- What information is available after each action?
  - Reward & state
- Input and output to neural network?
  - Input: (Action &) state   Output: Q-value for each action

Implementing Reinforcement Learning Algorithms

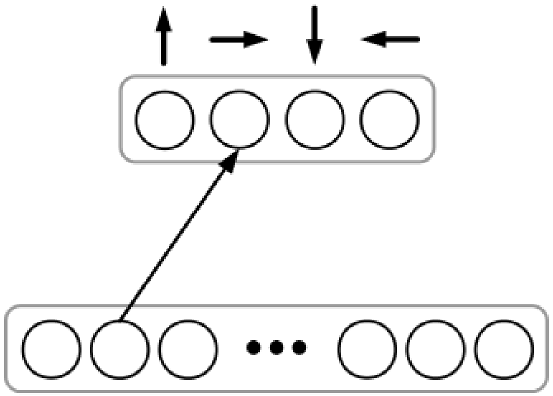
$$Q(s, a) = \sum_{k,l} w_{k,l} a_k s_l$$

w: weight
a: action
s: state

$$s_j = \begin{cases} 1, & \text{agent in position } j \\ 0, & \text{else} \end{cases}$$

$$a_i = \begin{cases} 1, & \text{agent takes action } i \\ 0, & \text{else} \end{cases}$$

Q-value for each action



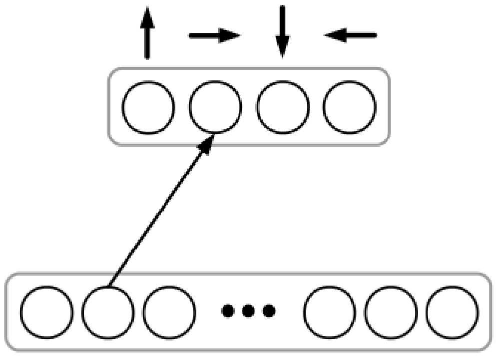
States

Learning Rule

- current estimated at position j and action i: $Q(s, a)$
- Q is estimated as function of the weights: $Q(s, a) = w_{ij} a_i s_j$
- expected reward 1 step ahead: $r + \gamma Q(s', a')$
- error function: $E = {}^1\!/\!_2(\underbrace{r + \gamma Q(s', a') - Q(s, a)}_{\text{value prediction error } \delta})^2$
- weight update by gradient descent on error:

$$\Delta w_{ij} \approx -\frac{\partial E}{\partial w_{ij}} = \delta \underbrace{a_i s_j}_{Hebb}$$

$$w_{ij} += \Delta w_{ij}$$



Details of Learning Rule

- Update
  - state s → s’ by action a
  - Next action a‘

- $E = \frac{1}{2}\left(r + \gamma Q(s', a') - Q(s, a)\right)^2$

- How to compute Q(s‘,a‘)?
  - Selection of next action a‘ ← Different TD learning alg.
  - Value of Q(s‘,a‘) ← Neural network

Variations of TD Learning Algorithms

- Q-learning: update based on next best possible estimates

$Q(s,a) \leftarrow \eta\ (r\ +\ \gamma\ \max_a(Q(s',a')) - Q(s,a))$

- SARSA: update estimates based on next chosen action

$Q(s,a) \leftarrow \eta\ (r\ +\ \gamma\ Q(s',a')\ -\ Q(s,a))$

- Actor-Critic: on-policy with two memories

$V(s) \leftarrow \eta\ \left(r\ +\ \gamma\ V'(s) -\ V\ (s)\right)$

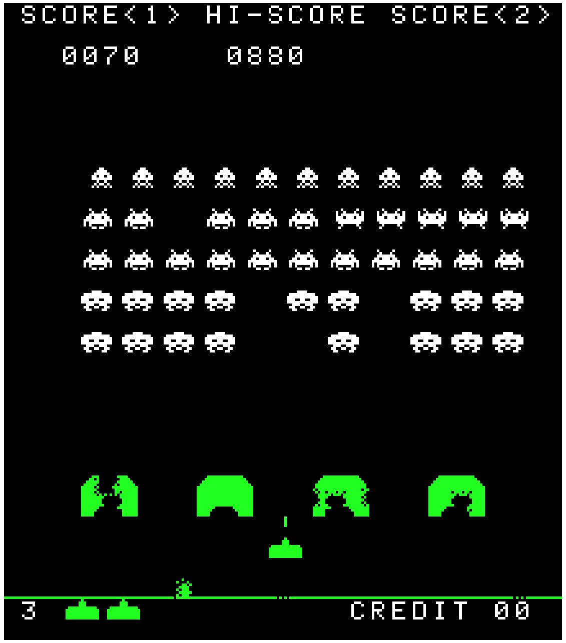
Update actor $A(s)$ depending on prediction error δ

Part 3:

Deep Reinforcement Learning

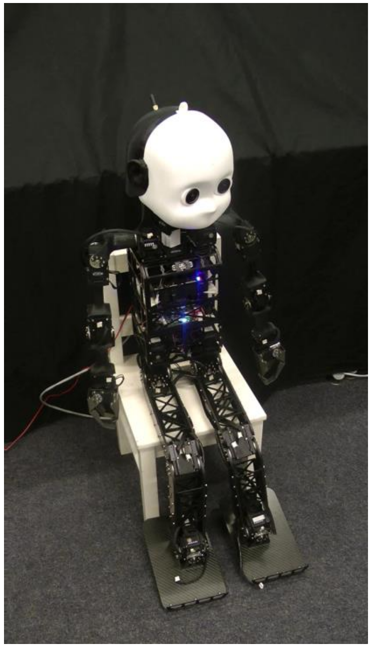
Motivation for Deep Reinforcement Learning

- How many states?



Source: Wikipedia

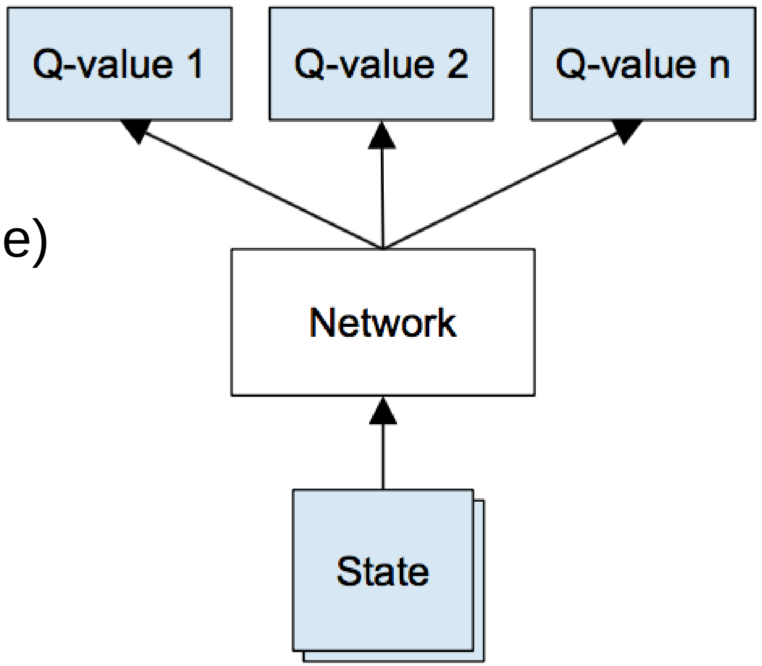
Source: sourceforge.net/projects/torcs/



Deep Q-Learning

- Convolutional Neural Networks (CNN) provide good features for highly structured data

- Example: Breakout
  - **Input:** game state s (video feed/image)
  - **Output:** Q-value for each possible action a (left/right)



⇨ Learns without **any** domain knowledge, only looking at pixels

[Google DeepMind 2013]

Deep Reinforcement Learning

$$Q(s_t, a_t)$$

Not stable!

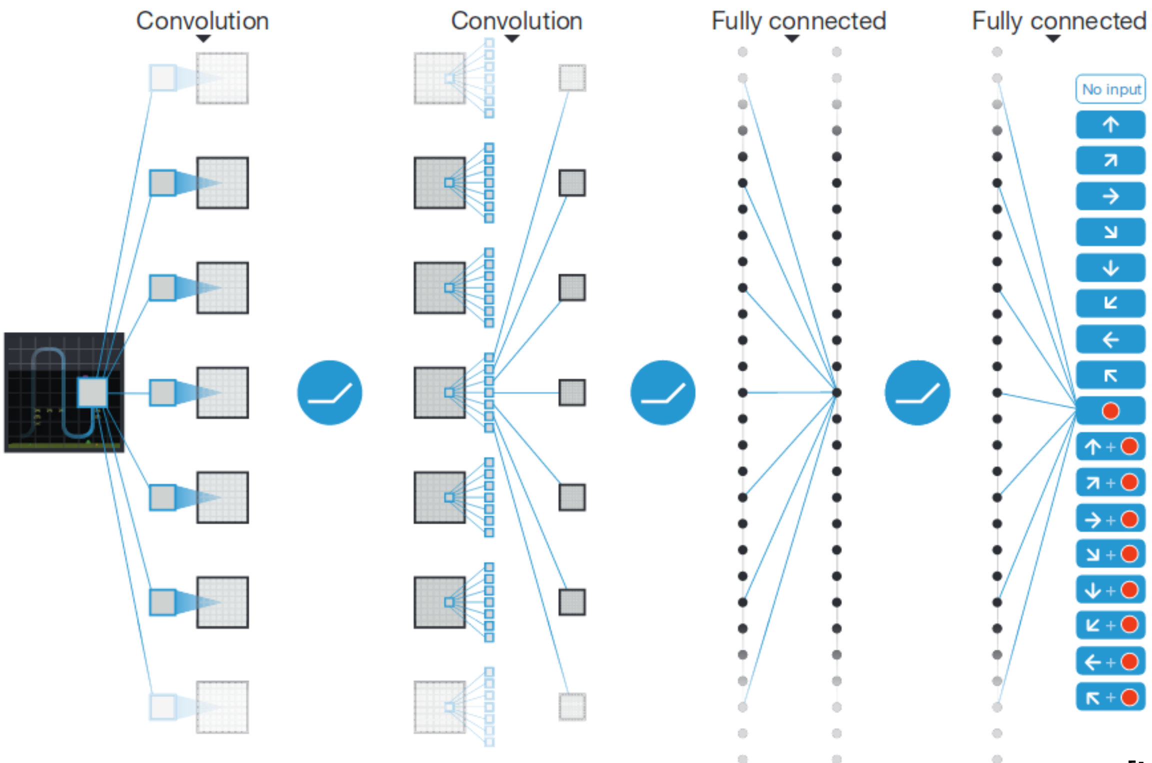
Approximate with neural network

Try all actions, choose best one

Translation invariance!
Velocities!

Convolutional neural network

Deep Reinforcement Learning Architecture

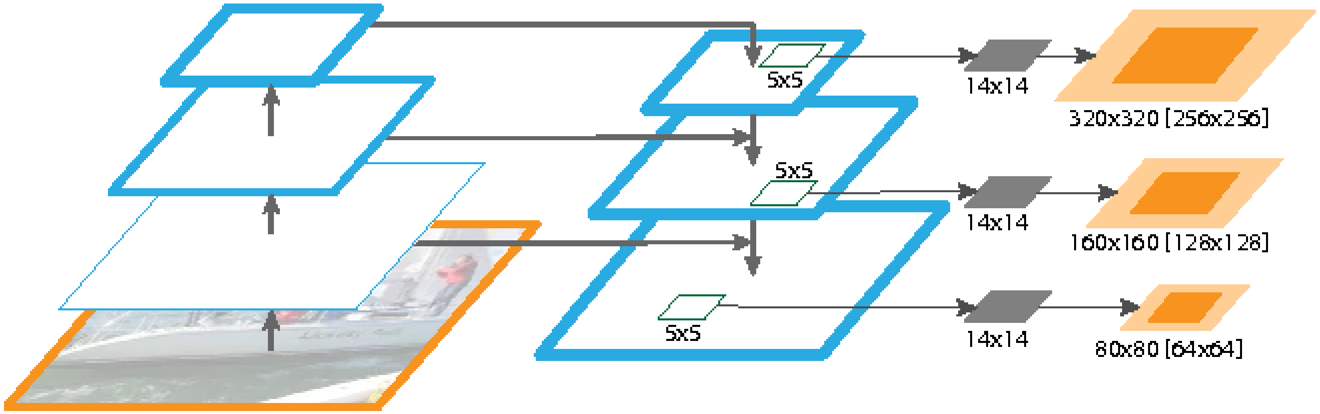


[Mnih et al. 2015]

- CNN → object identification and localization
- MLP → reinforcement learning
- End-to-end training of CNN + MLP → deep RF

Translational Invariance & Velocities

- No pooling layers
  - Less translational invariance
  - Alternatives: Pyramid Networks



[Lin et al. 2017]

- Blending of ~4 consecutive frames
  - Compensate flickering
  - Represent velocities

Stability

- Why is stability a problem?

- Q is updated using Q
  - $E = \frac{1}{2}\left(r + \gamma Q(s', a') - Q(s, a)\right)^2$
  - One lucky/unlucky episode can strongly influence future learning

- Coupled episodes
  - Learning experience does not represent the problem properly
  - E.g. exploration limited to some part of the state space „*Race car stuck in left turn*“

Stability: Replay Memory

- Replay Memory
  - Buffer of size n
    - $n \sim 10^6$
    - Use random actions to fill buffer with samples
    - Draw x random samples for training step

- Prioritized replay memory [Schaul et al. 2015]
  - What samples to keep?
  - What samples to delete?
  - What samples to prioritize?

Stability: Double Q-Learning & Q-Freezing

- Consistent targets during training
  - Q-Freezing[Mnih et al. 2015]
    - Keep constant copy of Q for some time (freezing)
    - Update after time interval
  - Double Q learning[van Hasselt et al. 2016]
    - Use 2 Q-functions
    - One function updates the other

Part 4:

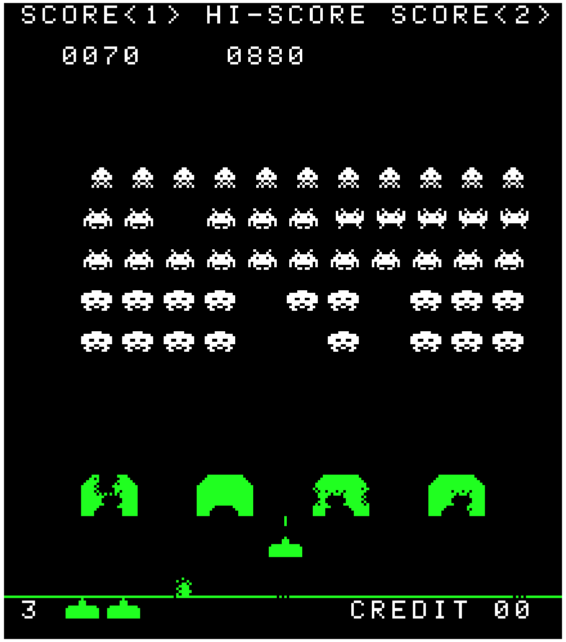
Continuous Deep Reinforcement Learning

Motivation for Continuous Deep RL

- What are the actions?

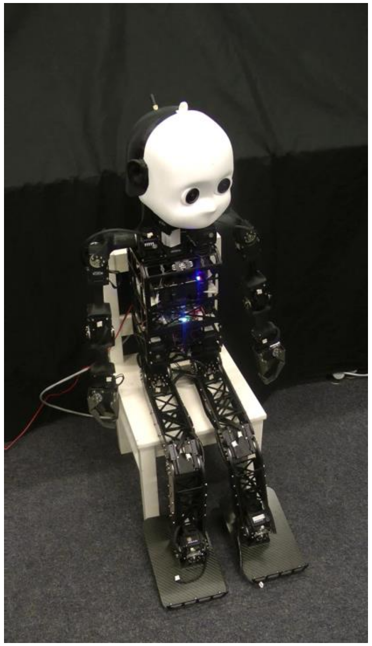
▲▼◀▶●

Curse of dimensionality



Source: Wikipedia

Source: sourceforge.net/projects/torcs/



Deep Reinforcement Learning

$$Q(s_t, a_t)$$

Not stable!

Approximate with neural network

Curse of dimensionality !

Try all actions, choose best one

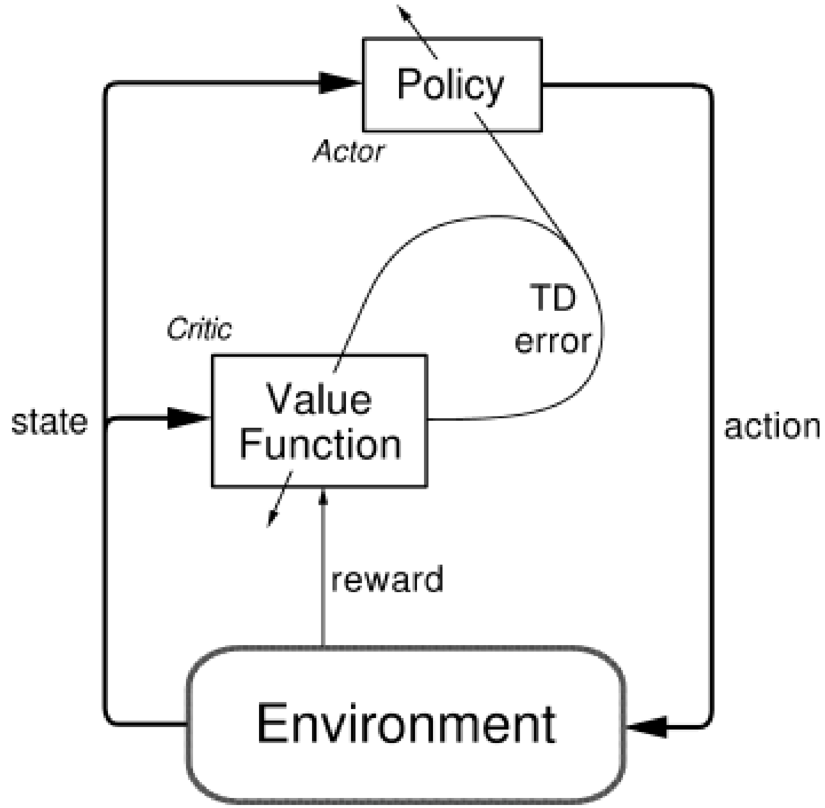
Translation invariance!
Velocities!

Convolutional neural network

Curse of Dimensionality

- How to deal with continuous actions?
  - E.g. Joint configuration of robot
- Discretization of actions
  - E.g. +5º / -5º for specific joint
- Number of possible actions for robot with 30 joints?
  - Number of actions with 1º discretization?
- Large number of possible actions makes deep RL infeasible

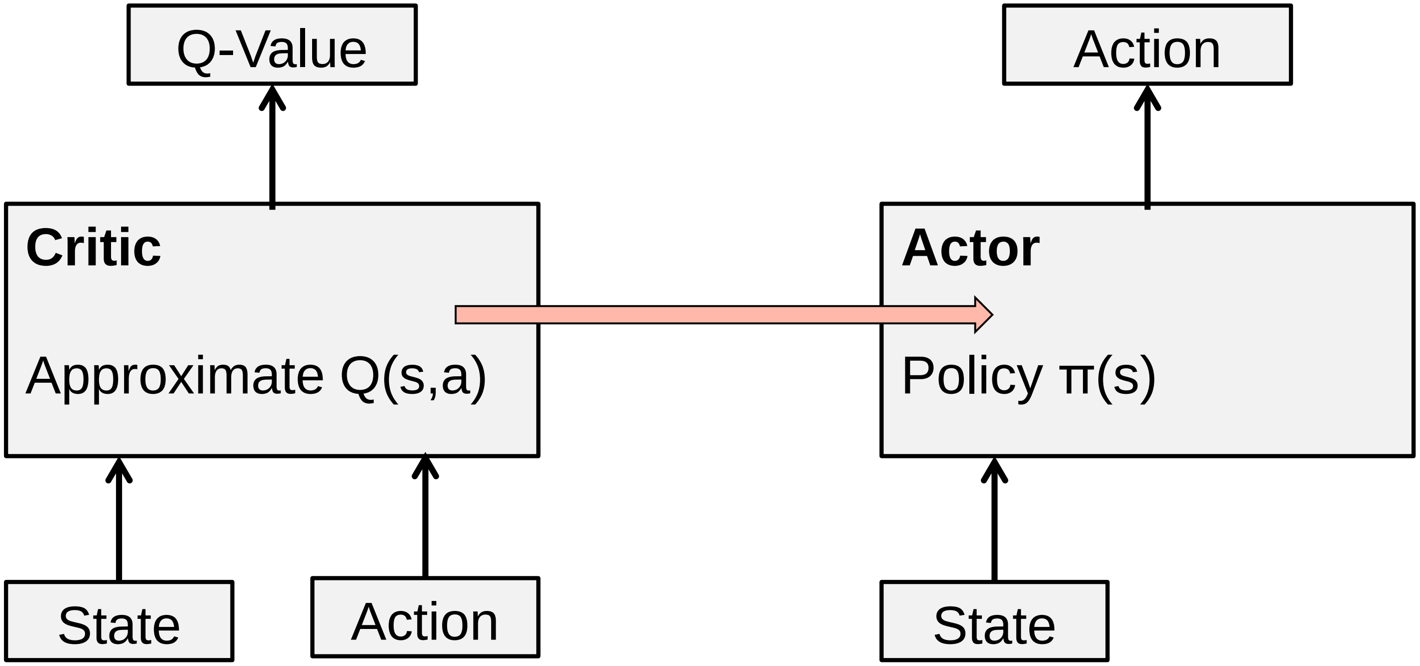
Actor-Critic Learning



[Sutton, Barto 98]

Actor-Critic Learning

- Separate networks for critic and actor



Actor-Critic Explained

- Critic can be trained
  - ~ Q(s,a)
- Actor can not be trained (directly)
  - No information on correct action

- Transfer of gradient from critic to actor
    - Critic is differentiable
      (e.g. gradient computation for update weights of critic network)
    - Critic is also differentiable w.r.t. actions
    - This gradient can be used to update the actor

Stability, again!

- Target Networks
  - Keep target network constant during training phase

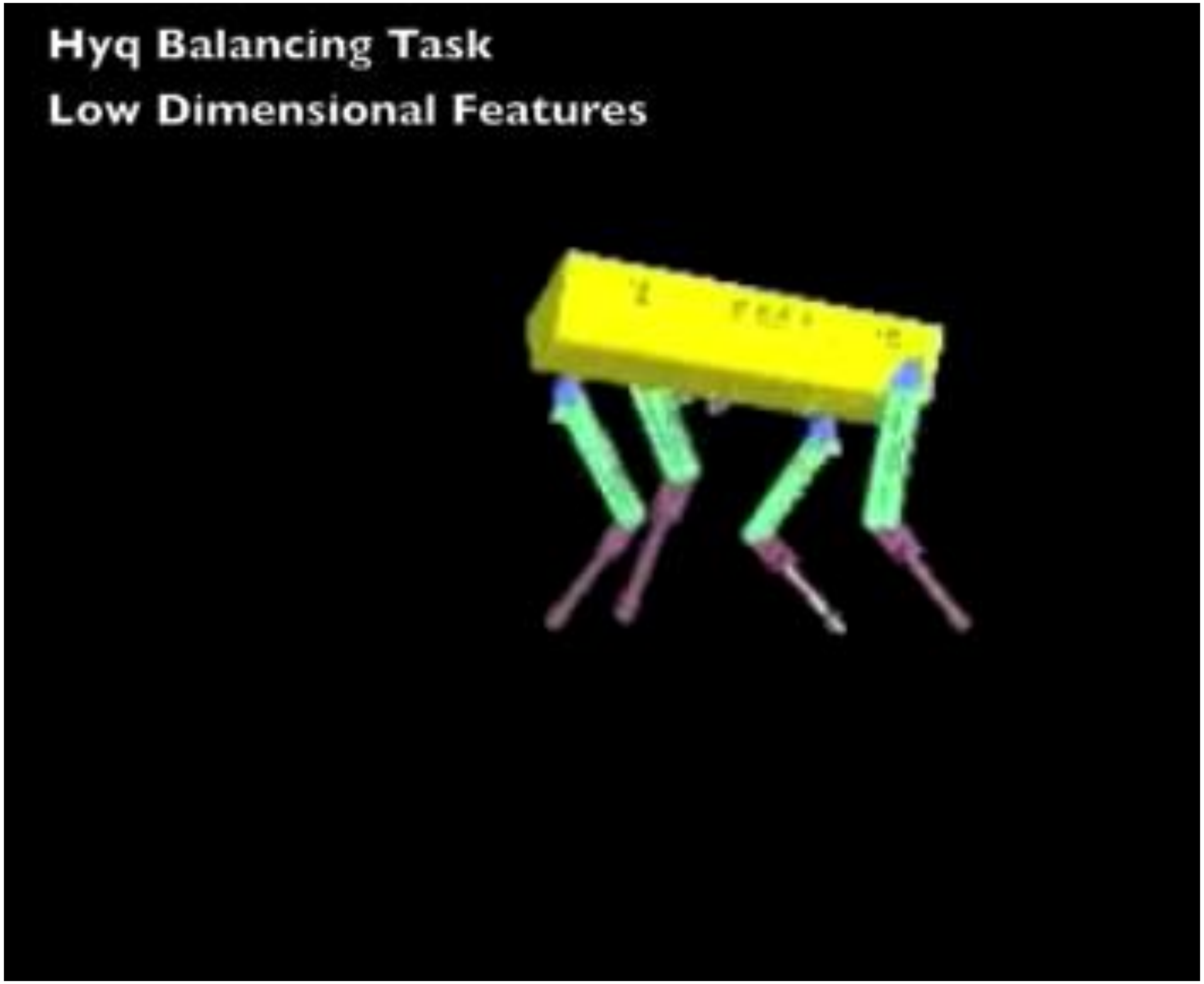
  - Gradually adjust target network towards updated network
    - θ model parameters
    - τ update factor
      e.g. $\tau = 0.001$

    - $\theta' \leftarrow \tau\theta + (1-\tau)\,\theta'$, $\tau \ll 1$

Step by Step

- Update the critic by minimizing the loss w.r.t. Q-value estimation
- Then the actor policy is updated using the sampled policy gradient
- Update actor target network (modulated by τ)
- Update critic target network (modulated by τ)

Example

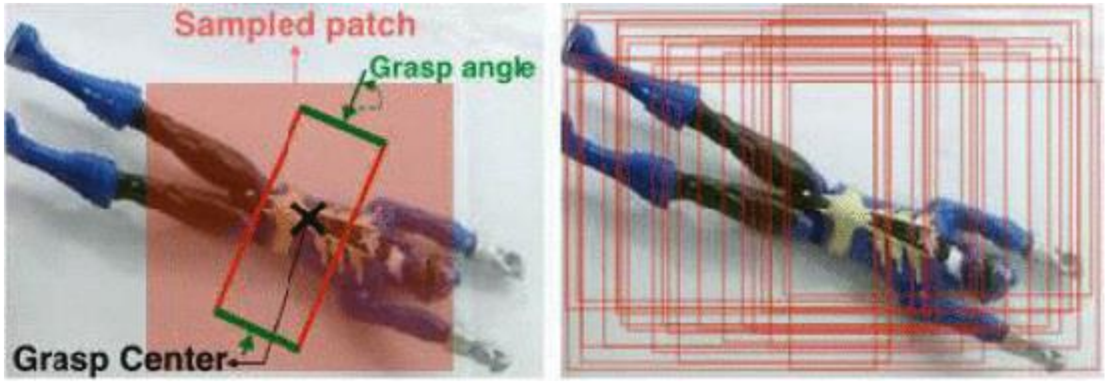


Part 5:

Related Approaches

Learning to Grasp from 50k Tries and 700 Robot Hours [Pinto et al. 2016]

- Platform
  - Two fingered parallel gripper
  - Grasping from top



[Pinto et al. 2016]

- CNN-based classifier
  - Image patch → Grasp likelihood of different grasp angles

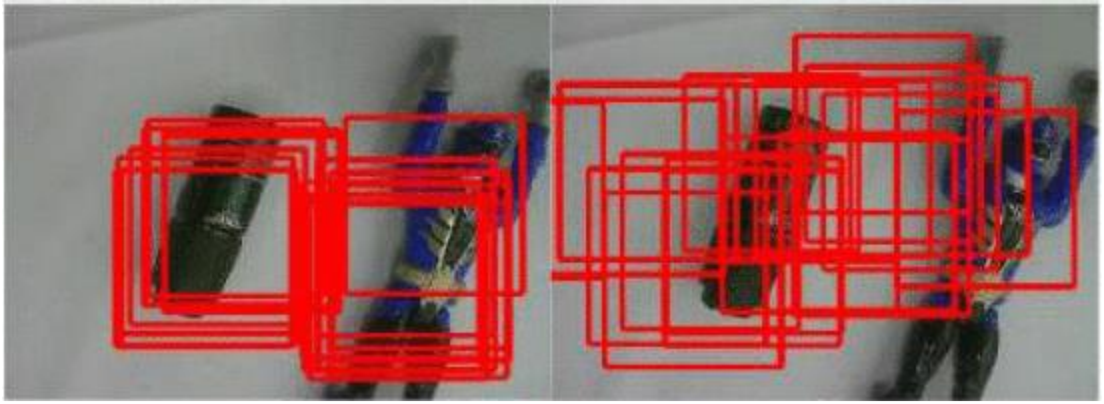
- At test time
  - Sample patches at different positions
  - Choose top graspable location & gripper angle

Learning to Grasp from 50k Tries and 700 Robot Hours [Pinto et al. 2016]

- Multi-Staged Learning
  - Previously trained network used to collect samples for training next stage of network

patches from learnt algorithm

random patches



[Pinto et al. 2016]

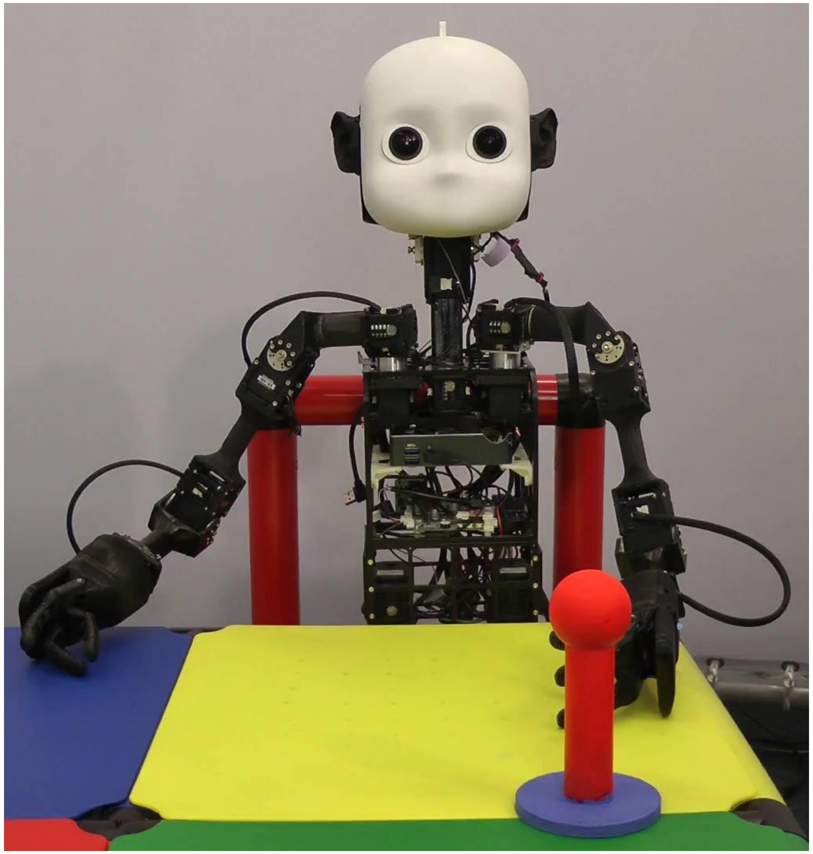
Combining Advantages of Supervised and Unsupervised Learning

- Robot generates annotated training data through interaction with the environment
  - Goals
    - No annotated data needed
    - No information about kinematic needed (model free)
    - As fast as supervised training
    - Minimal human supervision
    - Minimal damage potential

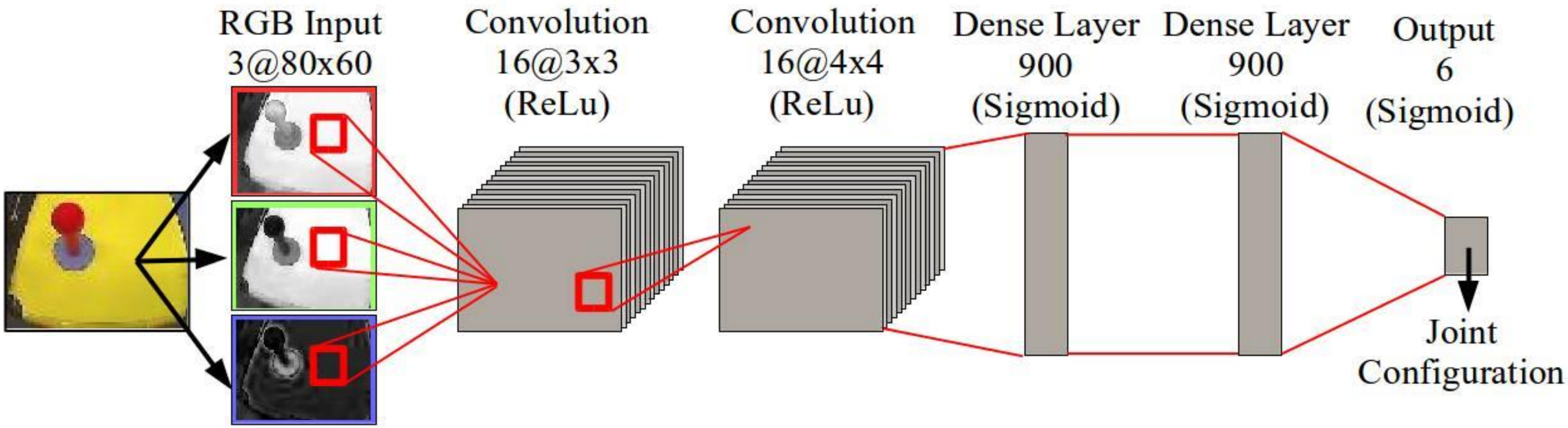
Experimental Setup: Gathering Training Data

Interaction with the environment

1. Move hand to home position
2. Close hand
    1. Select position on table from memory
    2. Places object on table
    3. Remove hand …
    4. and record training data (image – joint value pairs)
    5. Grasp abject again (with last used joint values)
3. Repeat

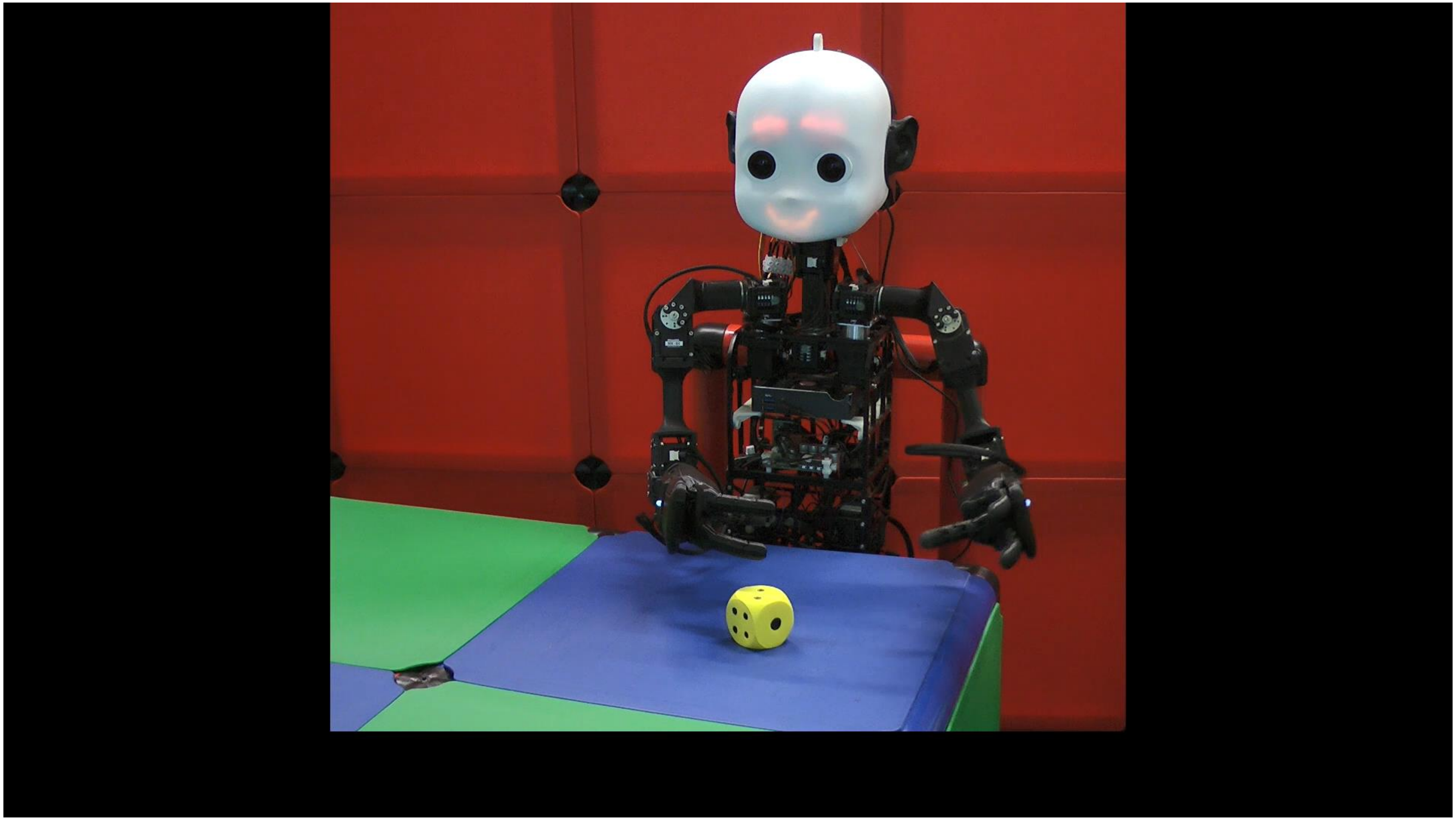


Neural Architecture



- One neural network
  - Supervised training
  - Output: Actions (joint configurations)

NICO Playing with Die



Perspective on RL: Developmental Robotics

- Learn and teach - don‘t program
  - “*autonomously acquire an increasingly complex set of sensorimotor and mental capabilities*” [Cangelosi & Schlesinger 2015]

- Advantages
  - HRI: Suitable for non-expert users
    - *„I failed to grasp the cup, can you please put it in my hand?“*
    - Teacher as intuitive role
  - Scientific exchange [Lungarella et al. 2013]
    - Evaluate models from developmental sciences
    - Adapt findings from ontogenetic development

Part 6:

Current Research Challenges

Current Research Challenges & Directions

- Minimize (physical) training time:
  - Transfer from simulator → real world
  - Transfer between different tasks
  - Use pretrained networks (e.g. vision networks)
- → Lifelong learning

- Optimize exploration & learning
  - Reward curiosity [Hafez et al. 2018]
  - Provide simplified training instances [Kerzel et al. 2018]
  - Alter episodes stored in replay memory → Hindsight Experience Replay (HER)

Dynamic Target Adjustment

Hindsight Experience Replay

- How to learn from failures?
  - Set a goal
  - Perform a sequence of actions
  - Does the outcome achieve the goal?
    - Yes: great – use the sample for learning
    - No: great – pretend that you wanted to achieve the outcome and learn from that

- Advantage
  - Every learning episode is valuable

Hindsight Experience Replay

Source: www.youtube.com/watcg?v=aKSILzbAqJs

Achievements & Challenges

- Achievements
  - Learn from raw sensory input
  - Domains with simple rules
    - E.g. “stay in the middle of the road on the race track”
  - Learn from observed expert behavior

- Challenges
  - Time consuming
  - Transfer and reuse of learned knowledge
  - Design of reward functions

Conclusion

- RL: Solution is not determined by labelled data (SL) but by exploratively acting in an environment, maximizing reward.
- Neural networks provide the ability to *learn* state space representations instead of hand-crafted engineering.
- Different architectures allow continuous state spaces and actions.
- Drawback: Slower to train, difficult to analyze.

References

- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

- Pinto, L., & Gupta, A. (2016, May). Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In Robotics and Automation (ICRA), 2016 IEEE International Conference on (pp. 3406-3413). IEEE.

- Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016). End-to-end training of deep visuomotor policies. Journal of Machine Learning Research, 17(39), 1-40.