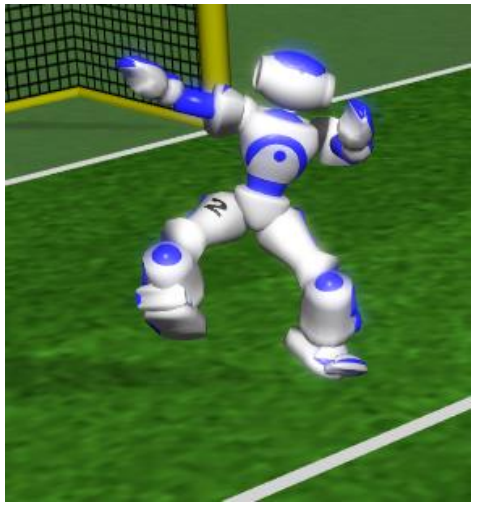
HOCH
SCHULE
OFFEN
BURG

Deep Reinforcement Learning

Prof. Dr. Klaus Dorer



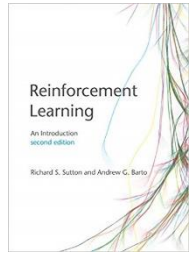
Übersicht

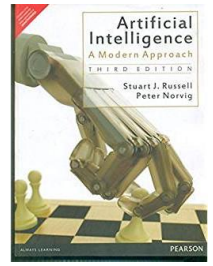
- Reinforcement Learning
  - State-Value Learning
    - Passive agent
    - Active agent
  - Q-Learning
  - Deep Reinforcement Learning
  - Examples

- Goal
  - Understand how Reinforcement Learning works
  - Know the applicability of DRL

Literature

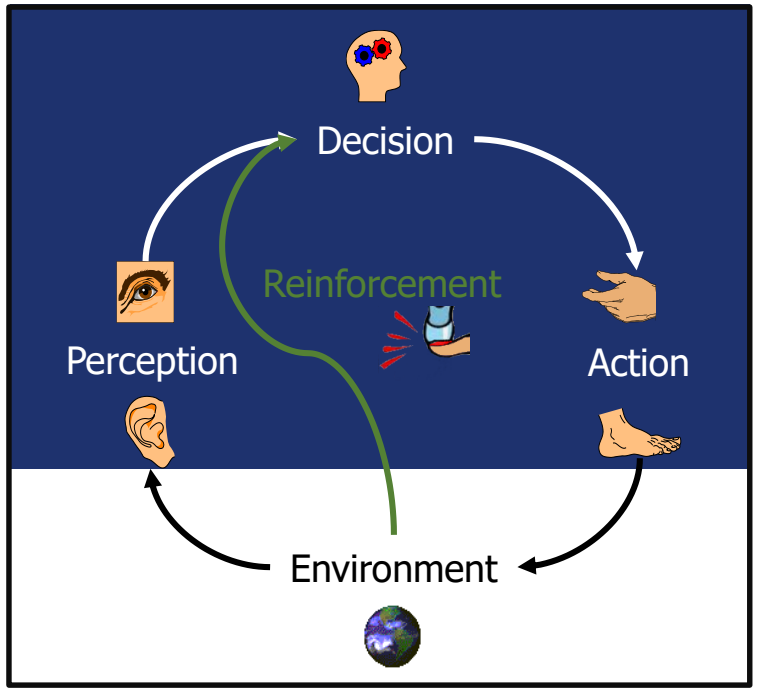
- Books
  - Richard Sutton, Andrew Barto (2018) Reinforcement Learning: An Introduction. http://incompleteideas.net/book/bookdraft2017nov5.pdf
  - Stuart Russell, Peter Norvig (2015) Artificial intelligence : a modern approach. Pearson Verlag.
- Papers
  - Gerald Tesauro (1995) Temporal difference learning and td-gammon. Communications of the ACM, 38(3):58–68, 1995.
  - Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, Riedmiller (2013) Playing Atari with Deep Reinforcement Learning. https://arxiv.org/pdf/1312.5602v1.pdf
- Playgrounds
  - https://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html





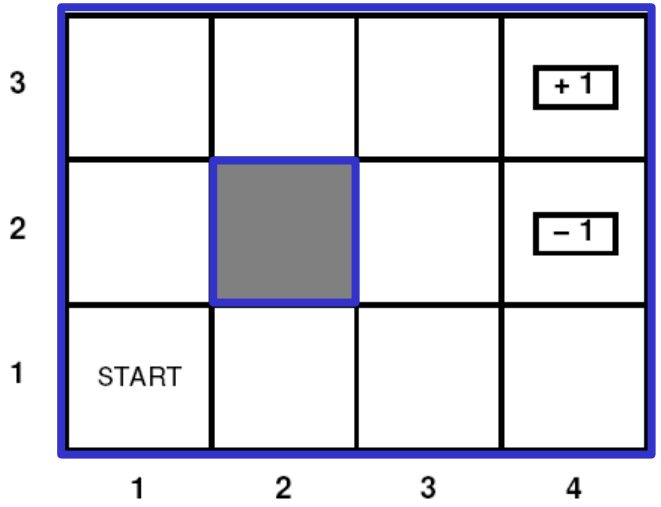
Reinforcement Learning

- Agents in a stochastic environment
  - Sense their environment
  - Act and change the environment
  - Receive a positive or negative reward (reinforcement) from the environment or a trainer
  - $a_0\ o_1\ r_1\ a_1\ o_2\ r_2\ a_2\ o_3\ r_3\ a_3$
- Goal
  - Maximize the long term reward
- Learning is difficult, because
  - there is no feedback on what should be done, only a number
  - the feedback might be received after some time (temporal credit assignment problem)

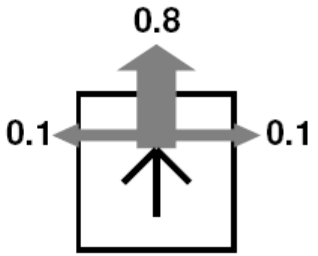


State-Value Agent

- Learns the utility of states
- Example domain
  - 4*3 world with one good state (+1) and one bad state (-1) (episode reward) (a)
  - Other states have reward -0.04 (continuous or step reward)
  - Actions North, East, South, West
    - Are stochastic (b)
    - Driving against a wall keeps the robot in the same state



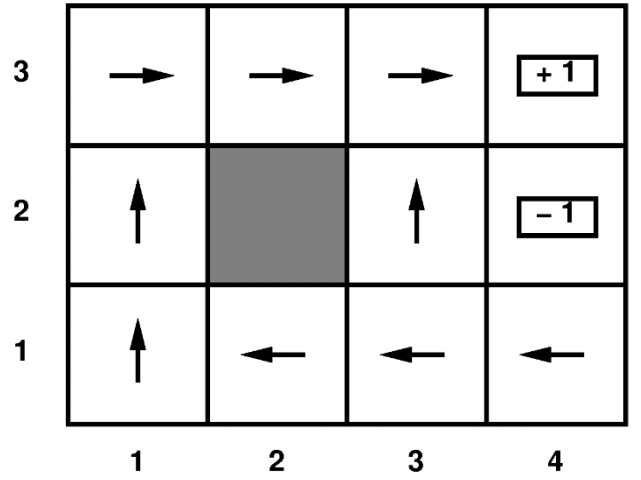
(a)



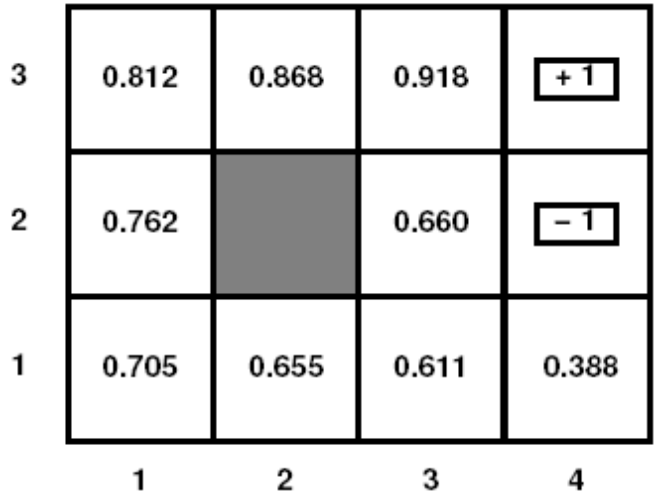
(b)

Passive State-Value Agent

- Strategy (Policy) is fixed and injected from outside
  - Agent only observes



- Goal
  - Learn how good the strategy is, i.e. learn the value of each state under the injected strategy



Terms

- Sequence (Episode, Epoch, Trajectory)
  - Sequence of states and actions ending in a terminal state
  - (1,1)→North→(1,2)→ North→(1,3)→East→(2,3)→East→(3,3)→East→(4,3)
  - (1,1)→ North→(1,2)→ North→(1,3)→ East→(1,2)→ North→(1,3)→ East→(2,3)→ East→(3,3)→ East→(4,3)
- Reward, reinforcement
  - positive oder negative feedback
  - $(1,1)_{-0.04}$→North→$(1,2)_{-0.04}$→ North→$(1,3)_{-0.04}$→East→$(2,3)_{-0.04}$→East→$(3,3)_{-0.04}$→East→$(4,3)_{+1}$
- Reward-to-go (rtg)
  - Sum of all reinforcements starting from the current ending at the terminal state
  - $(1,1)_{0.80}$→North→$(1,2)_{0.84}$→North→$(1,3)_{0.88}$→East→$(2,3)_{0.92}$→East→$(3,3)_{0.96}$→East→$(4,3_{)+1}$
  - $(1,1)_{\mathbf{0.72}}$→North→$(1,2)_{\mathbf{0.76}}$→North→$(1,3)_{\mathbf{0.80}}$→East→$(1,2)_{\mathbf{0.84}}$→North→$(1,3)_{\mathbf{0.88}}$→ East→$(2,3)_{\mathbf{0.92}}$→ East→$(3,3)_{\mathbf{0.96}}$→East→$(4,3)_{\mathbf{+1}}$

Passive State-Value Agent

- Decides on fixed policy π
- The ‚magic' happens in the update(..) function
  - Least Mean Squares (LMS)
  - Adaptive Dynamic Programming (ADP)
  - Temporal Difference Learning (TD)

```
public IOperator decide(IProblemState currentState)
{
     // add new perception to the list of perceptions
     percepts.add(currentState);

     // update the value function
     valueFunctionUpdateStrategy.update(currentState, percepts, lastAction);

     // end of episode?
     if (currentState.checkGoalState()) {
         percepts.clear();
     }

     lastAction = decideOnAction(currentState);
     return lastAction;
}
```

Least Mean Squares (LMS)

- Assumption: observed future reward directly corresponds to the utility of a state
- The utility of a state is then the running average of the reward to go: $U_i = \frac{U_i \cdot n_i + Rtg_i}{n_i + 1}$
- Limitations
  - Only works in episodic environments
  - Only works, if the end of an episode is reached

```
public void update(IProblemState currentState, List<IProblemState> percepts, IOperator lastAction)
{
  if (currentState.checkGoalState()) {
    float rewardToGo = 0.0f;
    for (int i = percepts.size() - 1; i >= 0; i--) {
      IProblemState state = percepts.get(i);
      // calculate the reward to go
      rewardToGo += state.getReinforcement();

      // calculate the running average
      int count = state.getExplorationCount();
      state.setUtility((state.getUtility() * count + rewardToGo) / (count + 1);
      state.onExploration();
} } }
```

Least Mean Squares (LMS)

- Example
  - Calculate reward-to-go
  - Calculate adjusted utility
  - $U_i = \frac{U_i \cdot n + Rtg_i}{n+1}$

Old Utility U	0.5	0.6	0.8	0.9	0.9	1.0
Visit count n	60	40	30	20	15	10
Sequence	(1,1)	(1,2)	(1,3)	(2,3)	(3,3)	(4,3)
Reinforcement r	-0.04	-0.04	-0.04	-0.04	-0.04	1.0
Reward-to-go						
New Utility (LMS)						

Temporal Difference Learning

- Uses observed state transitions to adjust state values
- $U_i = U_i + \alpha(n_i) \cdot (R_i + U_j\text{-}U_i)$
- Instead of taking all possible outcomes into account, TD Learning just uses the observed transitions
- Makes use of that unlikely transitions happen rarely, so values will converge
- A learning rate decreasing with the number of visits n_i, helps to converge U_i to its real values
- Does not require a transition model of the environment
- Works without episodes

Temporal Difference Learning

- Example
  - $U_i = U_i + \alpha(n_i) \cdot (R_i + U_j\text{-}U_i)$

Old Utility	0.5	0.6	0.8	0.9	0.9	1.0
State	(1,1)	(1,2)	(1,3)	(2,3)	(3,3)	(4,3)
Action	North	North	North	East	East	
Reinforcement	-0.04	-0.04	-0.04	-0.04	-0.04	1.0
New Utility (α=0.5)						

Temporal Difference Learning

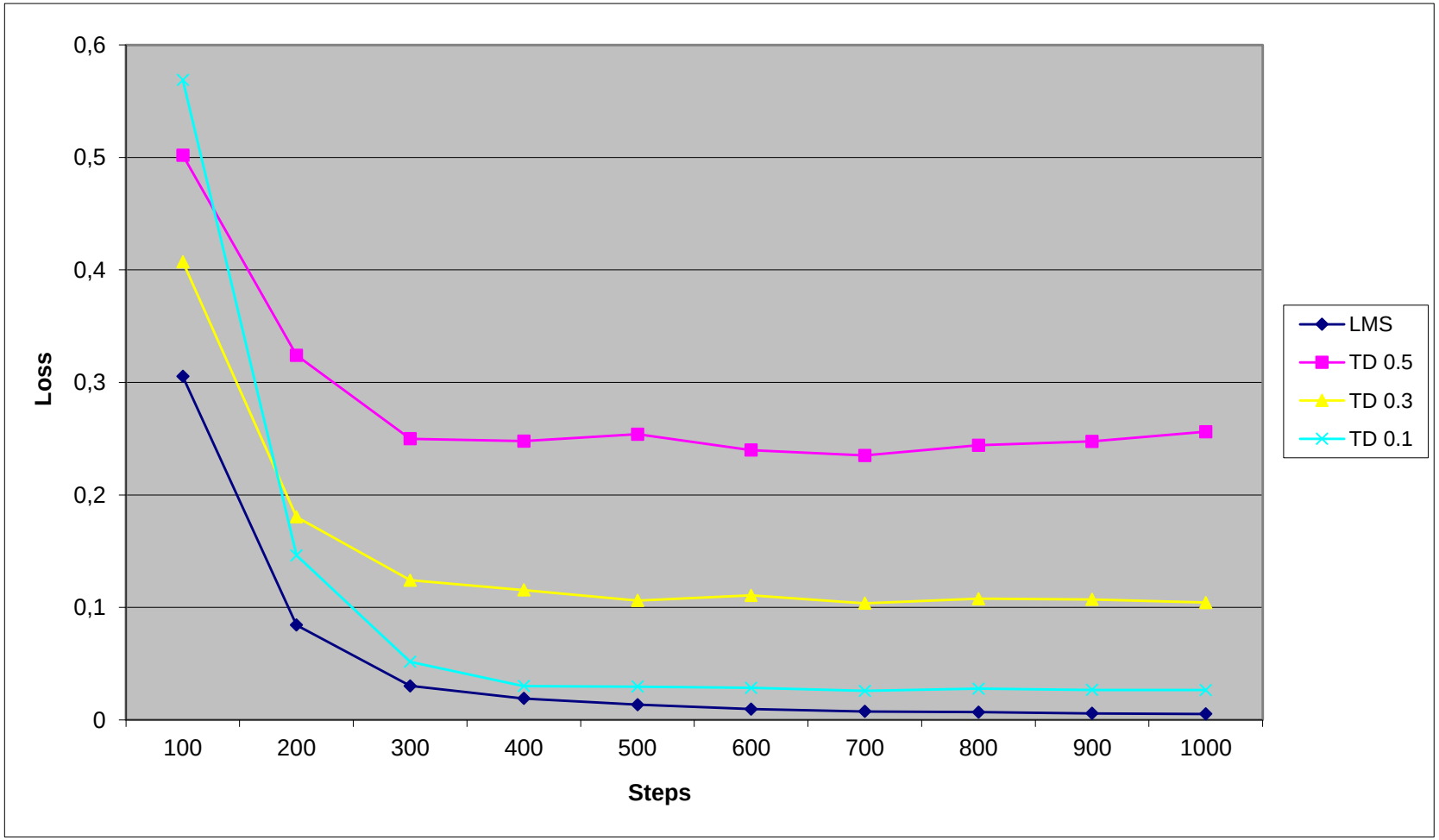
- Implementation

```
public void update(IProblemState currentState, List<IProblemState> percepts, Ioperator lastAction)
{
  if (currentState.checkGoalState()) {
    // for goal states we can simply calculate the running average
    int count = currentState.getExplorationCount();
    currentState.setUtility((currentState.getUtility()*count + currentState.getReinforcement())/(count +1));
    currentState.onExploration();
  }

  if (percepts.size() > 1) {
    // we have a state transition so apply TD update rule
    IProblemState penultimateState = percepts.get(percepts.size() - 2);
    penultimateState.onExploration();
    float learnrate = learnrateStrategy.getAlpha(penultimateState.getExplorationCount());
    float utility = penultimateState.getUtility();
    float reinforcement = penultimateState.getReinforcement();
    // td update rule
    penultimateState.setUtility(utility + learnrate *
                          (reinforcement + currentState.getUtility() - utility));
  }
}
```

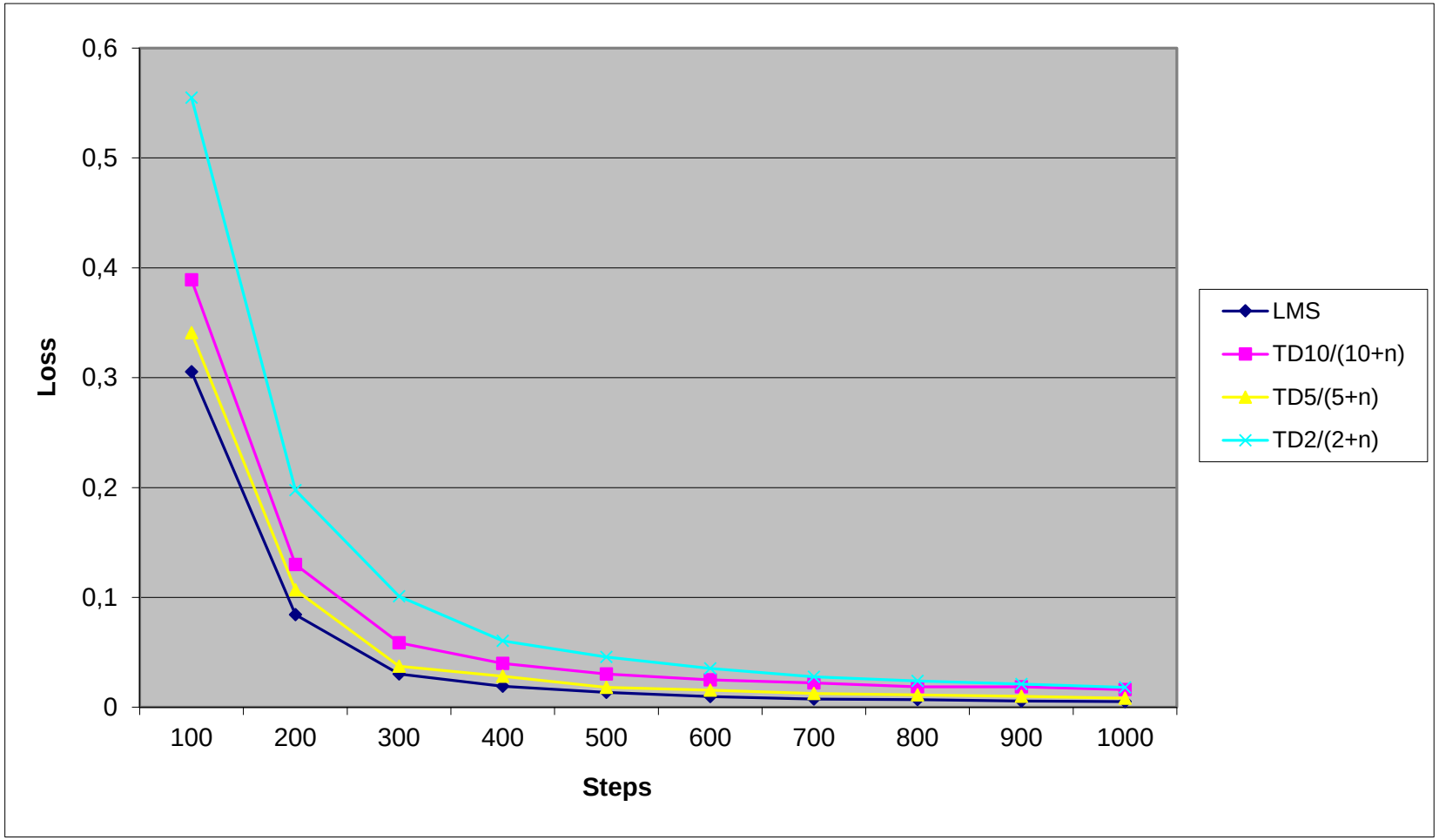
Results

- LMS versus TD with various fixed learning rates



Results

- LMS versus TD with decreasing learning rates



Active State-Value Agent

- An active agent has to decide on its own for which action to take
- Requires to know the state transition model M_{ij}^{a}

```
protected IOperator decide(IProblemState currentState)
{
  // learn model
  TransitionModel transitionModel = model.get(currentState);
  if (transitionModel == null) {
    transitionModel = new TransitionModel();
    model.put(currentState, transitionModel);
  }
  // determine best action
  List<IOperator> actions = currentState.getOperators();
  float bestExpectedUtility = Float.NEGATIVE_INFINITY;
  IOperator bestAction = actions.get(0);
  for (IOperator action : actions) {
    // check all possible successor states
    float expectedUtility = transitionModel.getExpectedUtility(action);
    if (expectedUtility > bestExpectedUtility) {
      bestAction = action;
      bestExpectedUtility = expectedUtility;
    }
  }
  return bestAction;
}
```

Active State-Value Agent

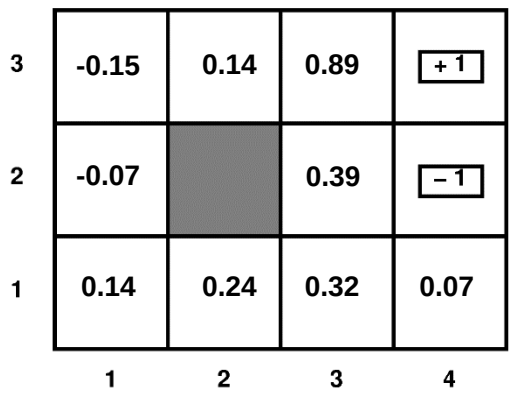
- The agent will try to maximize its utility
- Expected Utility: $eu_{a,i} = \sum_j M_{ij}^a U_j$
- Utility of a state: $U_i = R_i + \max_a \sum_j M_{ij}^a U_j$
- Example: Agent on (3,1)
  - $eu_{North,(3,1)}$ =0.8*0.660 + 0.1*0.388 + 0.1*0.655 = 0.632
  - $eu_{East,(3,1)}$ =0.8*0.388 + 0.1*0.611 + 0.1*0.660 = 0.438
  - $eu_{South,(3,1)}$ =0.8*0.611 + 0.1*0.655 + 0.1*0.388 = 0.593
  - $eu_{West,(3,1)}$ =0.8*0.655 + 0.1*0.660 + 0.1*0.611 = 0.651
  - max_a -> West
  - U(3,1) = -0.04 + 0.651 = 0.611

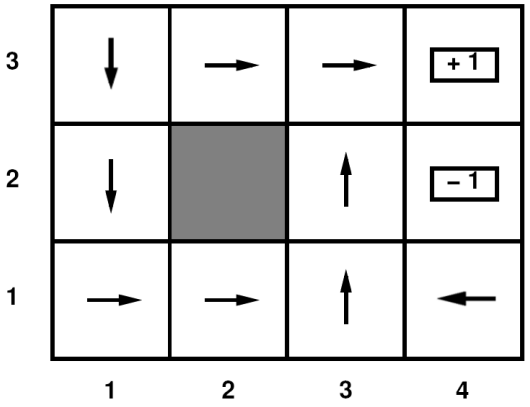
  - $eu_{East,(2,3)}$ =

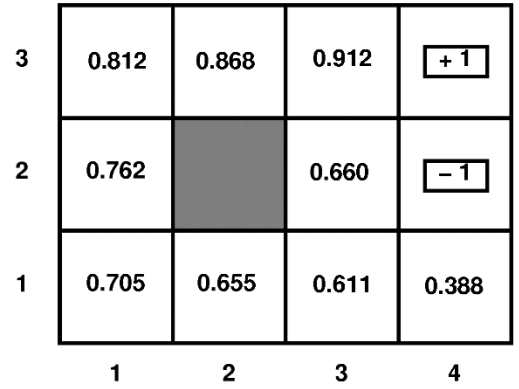
3	0.812	0.868	0.912	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Exploration

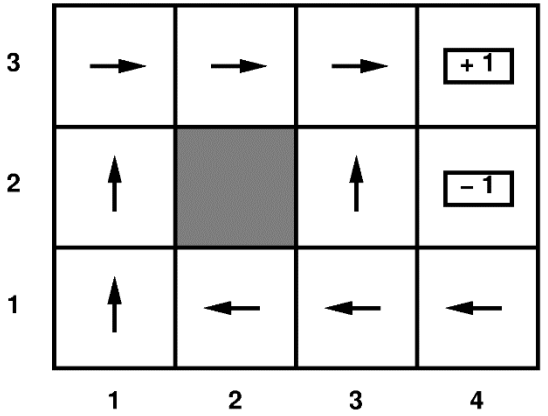
- Actions have two effects
  - Collect reward through an episode
  - Collect knowledge about the world
- Tradeoff between
  - Immediate utility (greedy, exploit)
  - Long term utility (explore to exploit later)
- Ideally, the agent explores initially and exploits once knowing the environment well







Utilities



Policy

Q-Learning
Learning the Action-Value Function

- Instead of learning state-values and the transition model to calculate expected utility, we can directly learn action-values
- Q(a,i) is the utility of performing action a in state i
- The utility of a state is then the maximal Q-value of all actions possible in that state $U_i = \max_a Q_i^a$

- Does not require to learn or know the transition model
- Q-values can be learned directly from reinforcements
- No prediction possible

Temporal Difference Q-Learning

- Update rule for TD Q-Learning
  - $Q_i^a = Q_i^a + \alpha(n_i) \cdot (R_i + \max_{a'} Q_j^{a'} - Q_i^a)$

```
public void update(IProblemState currentState, List<IProblemState> percepts, IOperator lastAction)
{
  if (currentState.checkGoalState()) {
    // for goal states we can simply calculate the running average
    float reinforcement = currentState.getReinforcement();
    IOperator bestAction = qTable.getBestAction(currentState);
    float oldUtility = qTable.getUtility(currentState, bestAction);
    float count = qTable.getExplorationCount(currentState, bestAction);
    float newUtility = (oldUtility * count + reinforcement) / (count + 1);
    qTable.update(currentState, bestAction, newUtility);
  }

  if (percepts.size() > 1) {
    // we have a state transition so apply TD update rule
    IProblemState penultimateState = percepts.get(percepts.size() - 2);
    float learnrate = learnrateStrategy.getAlpha(qTable.getExplorationCount(penultimateState, lastAction));
    float oldUtility = qTable.getUtility(penultimateState, lastAction);
    float successorUtility = qTable.getBestUtility(currentState);
    float reinforcement = penultimateState.getReinforcement();
    float newUtility = oldUtility + learnrate * (reinforcement + successorUtility - oldUtility);
    qTable.update(penultimateState, lastAction, newUtility);
  }
}
```

Temporal Difference Q-Learning

- Example
  - $Q_i^a = Q_i^a + \alpha(n_i) \cdot (R_i + \max_{a'} Q_j^{a'} - Q_i^a)$

Sequence		(1,1)	(1,2)	(1,3)	(2,3)	(3,3)	(4,3)
Action		North	North	East	East	East	
Old Q values	North	0.5	0.6	0.7	0.75	0.8	1.0
	East	0.35	0.5	0.8	0.9	0.9	1.0
	South	0.3	0.4	0.5	0.75	0.2	1.0
	West	0.4	0.5	0.6	0.6	0.7	1.0
Reinforcement		-0.04	-0.04	-0.04	-0.04	-0.04	1.0
New Q value (α=0.5)							

Generalisation for RL

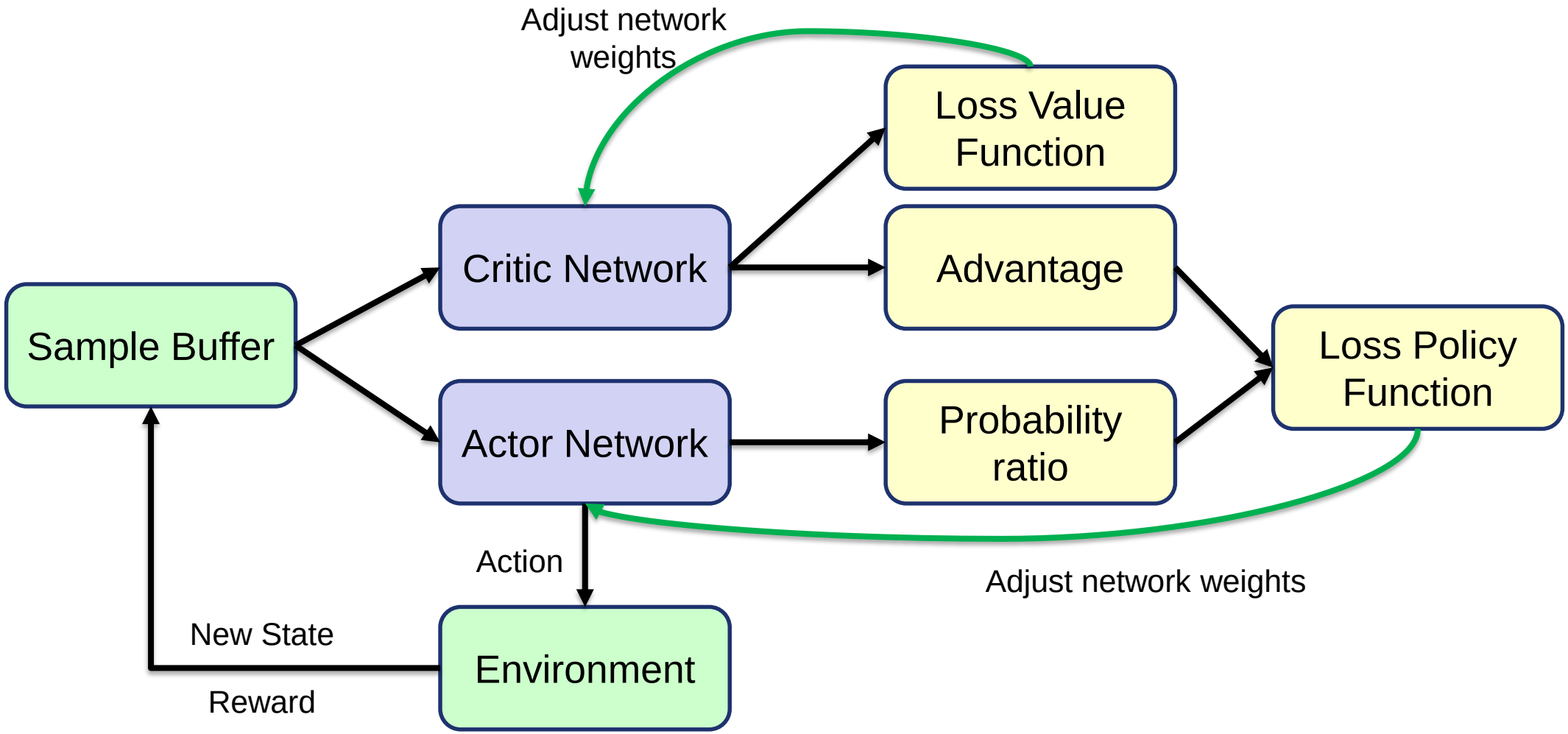
- So far we assume that the values the agent learned (U, M, R, Q) are stored in tables (explicit representation)
- This only works in very small environments
- For bigger environments we require an implicit representation of the functions
- Example Chess
  - Ca 10^{120} states (explicit representation)
  - Linear weighted feature vector
    - $U(i) = w_1f_1(i) + w_2f_2(i) + \ldots + w_nf_n(i)$ (implizite Repräsentation)
    - With n=10 features one can play reasonably
  - Neural Networks allow to learn functions

Deep Reinforcement Learning

- Learn the mapping from observation to action with a (deep) neural network
- Problems
  - We do not have labled data
  - Inputs and Outputs may be continuous [-1..1]
- Algorithms

	Value Based	Policy Based	Actor-Critic
On-Policy	• Monte Carlo Learning (MC) • TD(0) • SARSA • Expected SARSA • n-Step TD/SARSA • TD(λ)	• REINFORCE • REINFORCE with Advantage	• A3C • A2C • TRPO • PPO
Off-Policy	• Q-Learning • DQN • Double DQN • Dueling DQN		• DDPG • TD3 • SAC • IMPALA

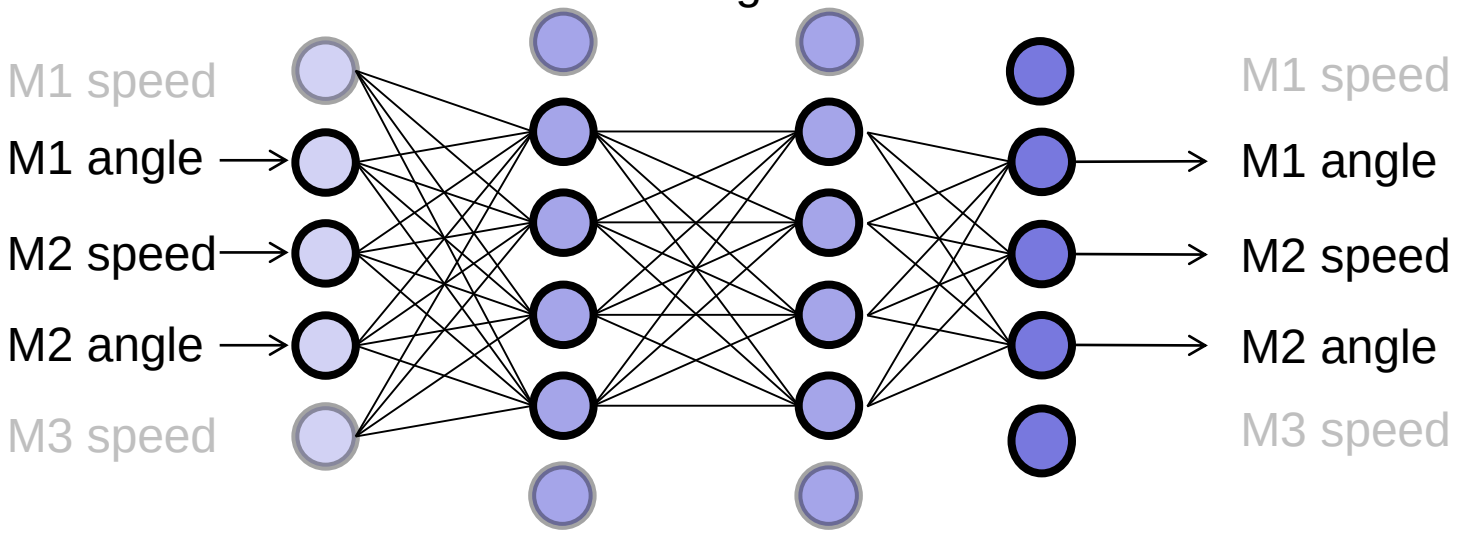
Example Actor-Critic Algorithm: PPO [Schulman et al. 2017]



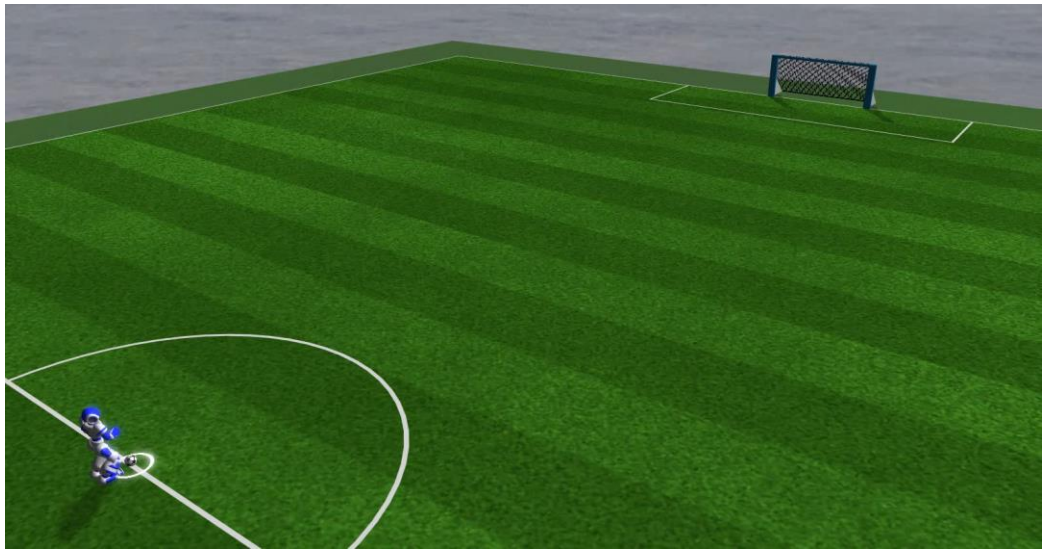
DRL Example: RoboCup

- Observation Space
  - For each motor (24) angle and speed
  - Accelerometer and Gyroskop
  - Foot force sensors
  - Torso up, ball position, counter
- Action Space
  - For each motor angle and speed
- Network
  - 120 – 64 – 64 – 28 -> 13568 trainable weights

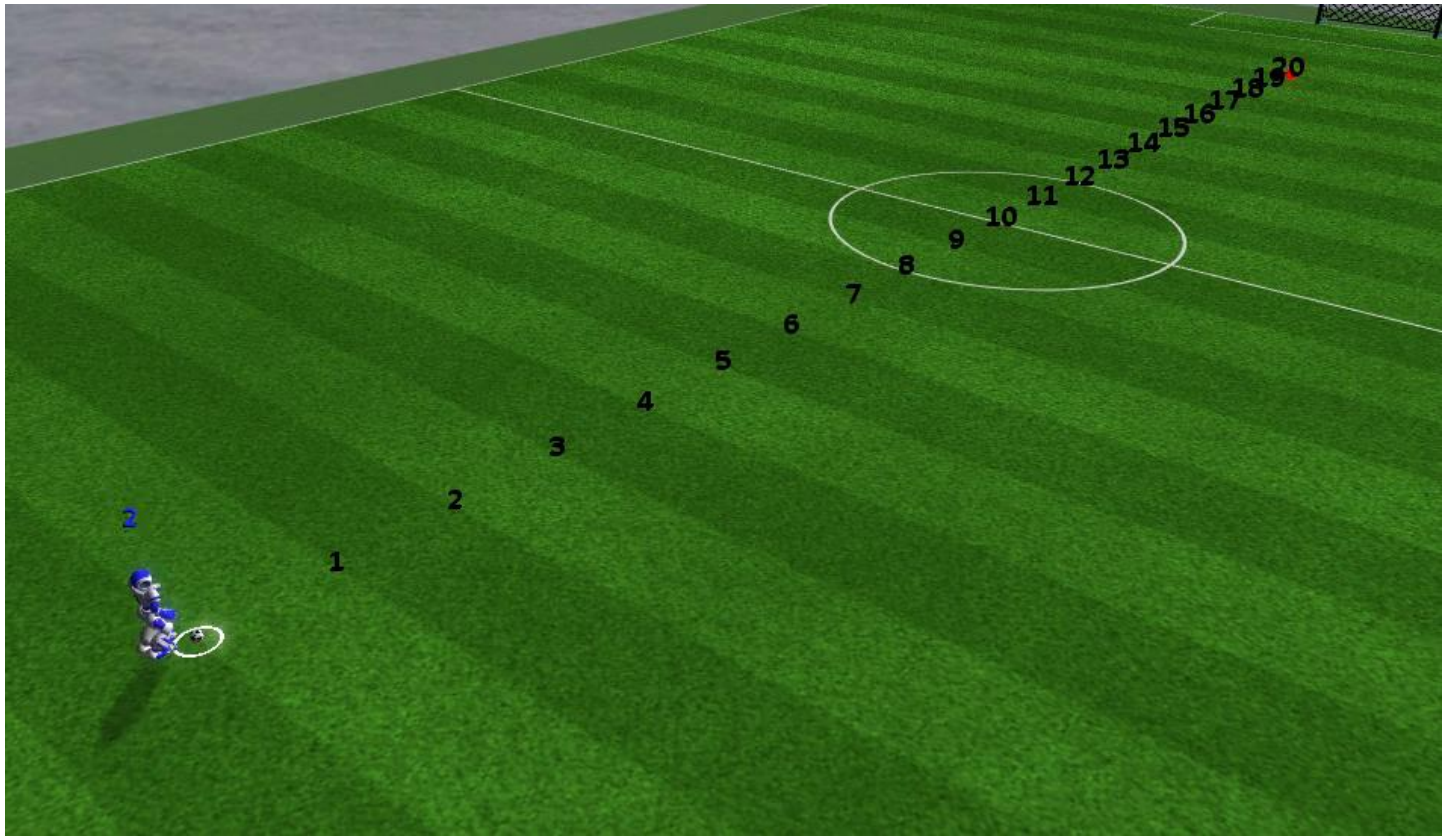
Index	Count	Observation
0	1	Counter
1-4	4	**head joints***
5-20	16	**arm joints***
21-48	28	**hip, knee, ankle, toe joints***
49-54	6	3D relative ball position*
55-102	48	**foot and toe force sensors***
103-108	6	**accelerometer***
109-114	6	**gyroscope***
115-116	2	torso up vector x,y
117	1	ball relative angle
118	1	desired kick direction (-90..90°, relative)
119	1	desired kick distance (0..20m)



Genetic vs DRL Kick



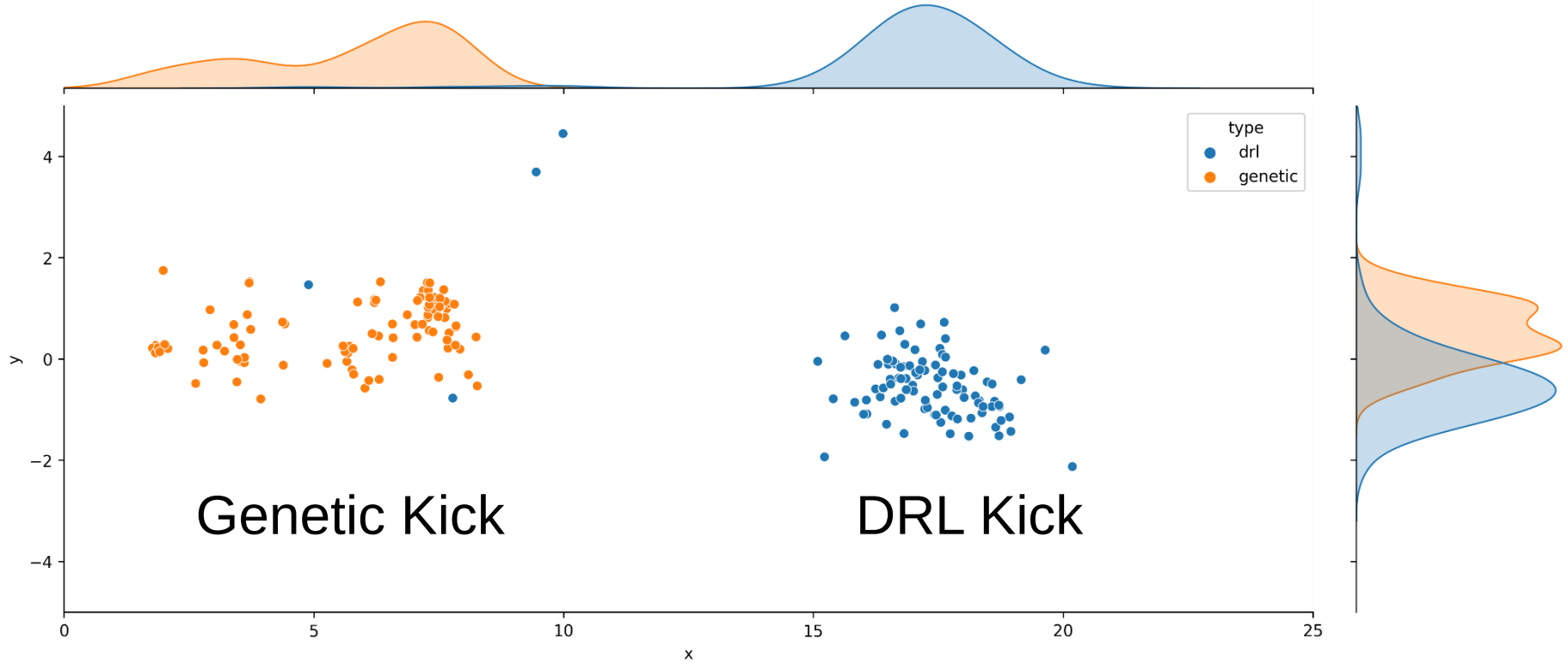
2019: 8m Genetic



2022: 20m DRL

Genetic vs DRL Kick

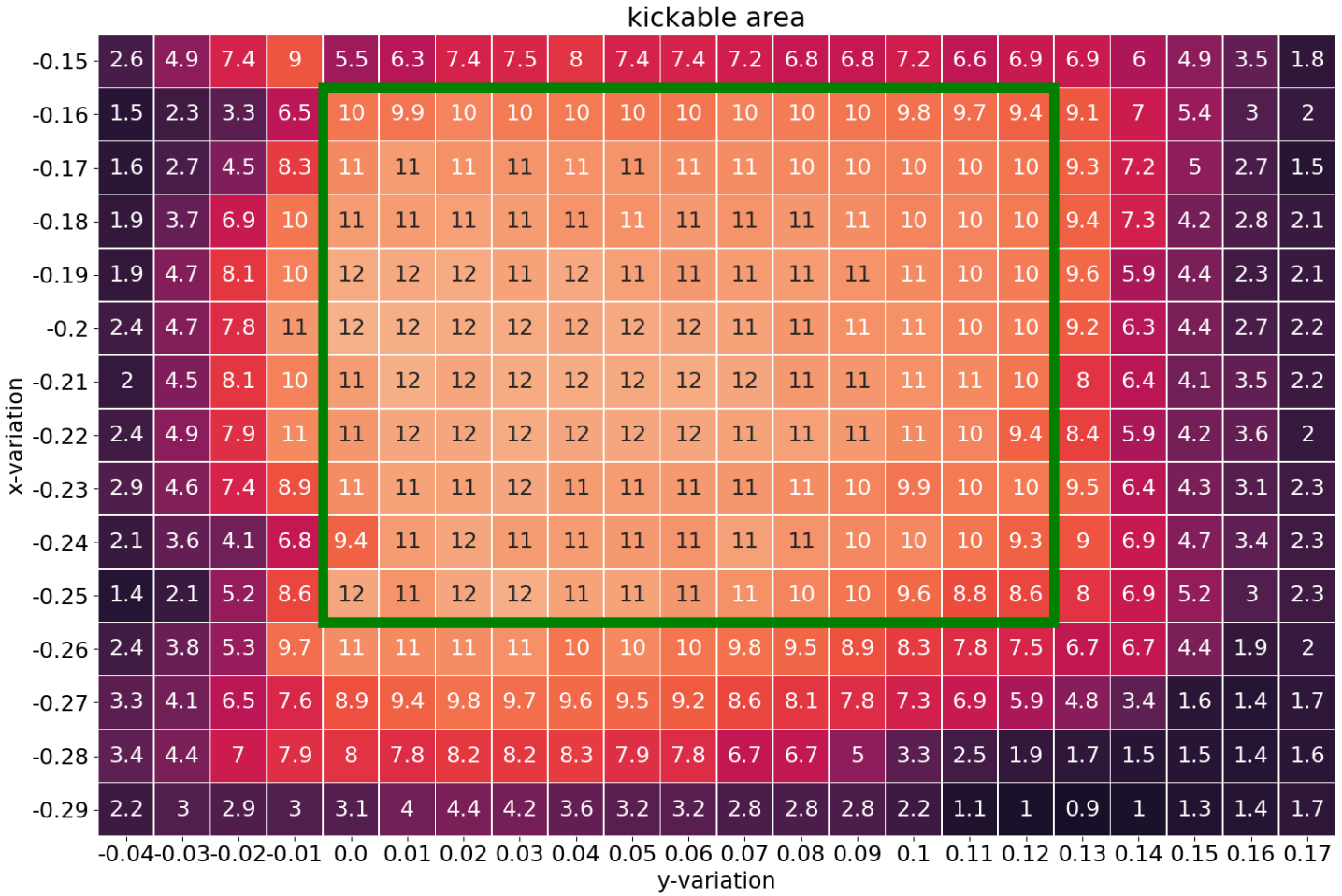
- Comparison to existing kick (learned with genetic algorithms)



Spitznagel, Weiler, Dorer (2021) Deep Reinforcement Multi-Directional Kick-Learning of a Simulated Robot with Toes. IEEE ICARSC 2021.

Genetic vs DRL Kick

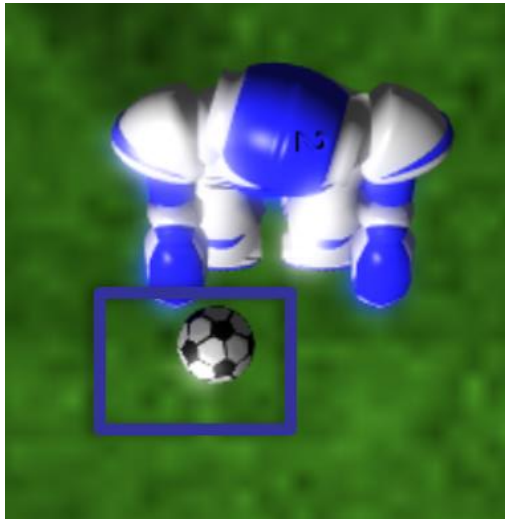
- Kickable area
  - Right foot is kicking
  - Each square represents the relative position of the ball to the torso
  - 50 kicks per square
  - Reward = distancex - |distancey| + 2 * stable
  - Green rectangle: area into which the player was beamed during learning

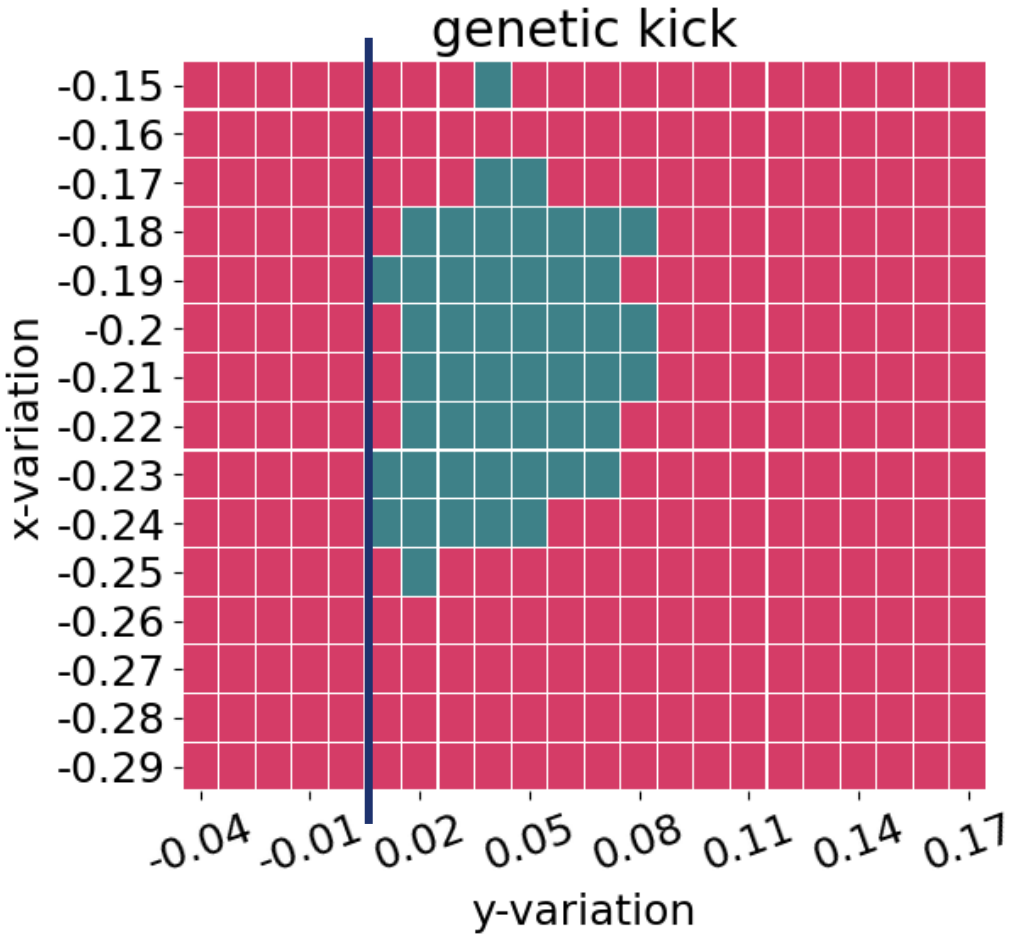


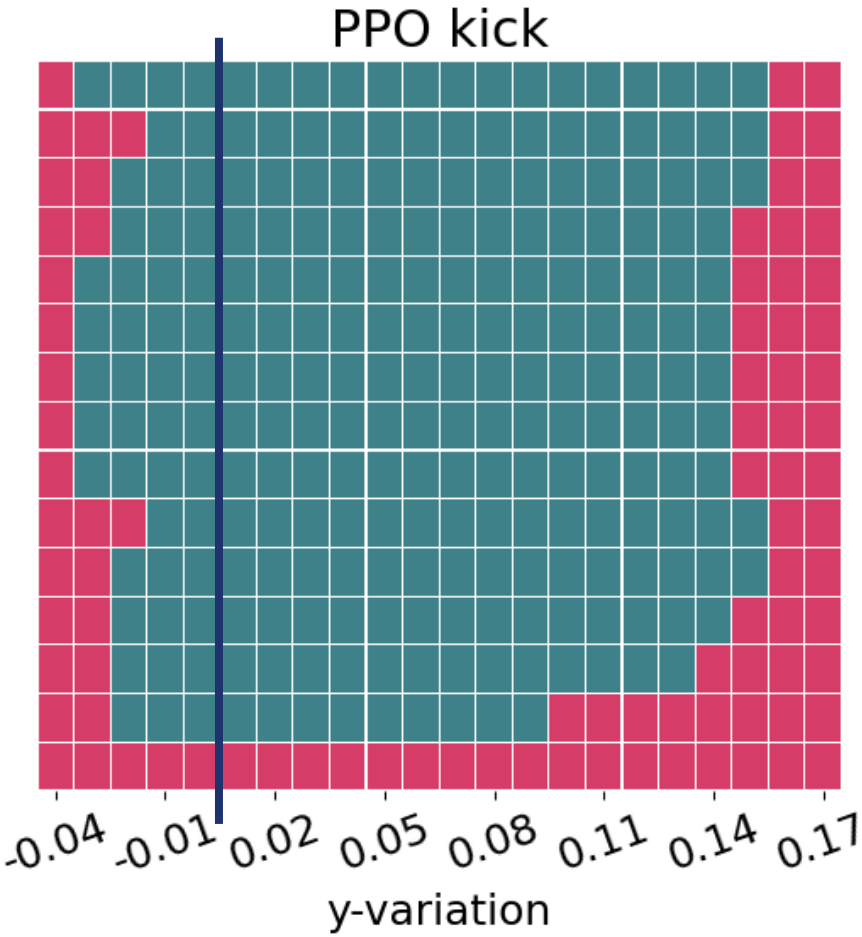
Spitznagel, Weiler, Dorer (2021) Deep Reinforcement Multi-Directional Kick-Learning of a Simulated Robot with Toes. IEEE ICARSC 2021.

Genetic vs DRL Kick

- Kickable area

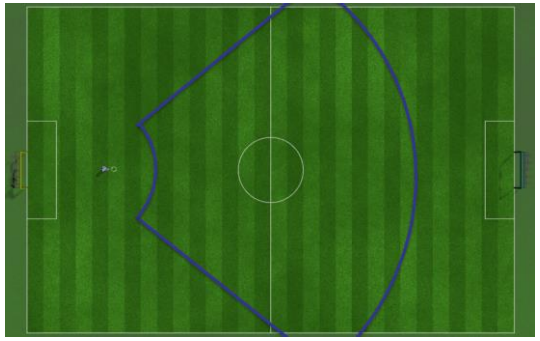






Results: Flexibility

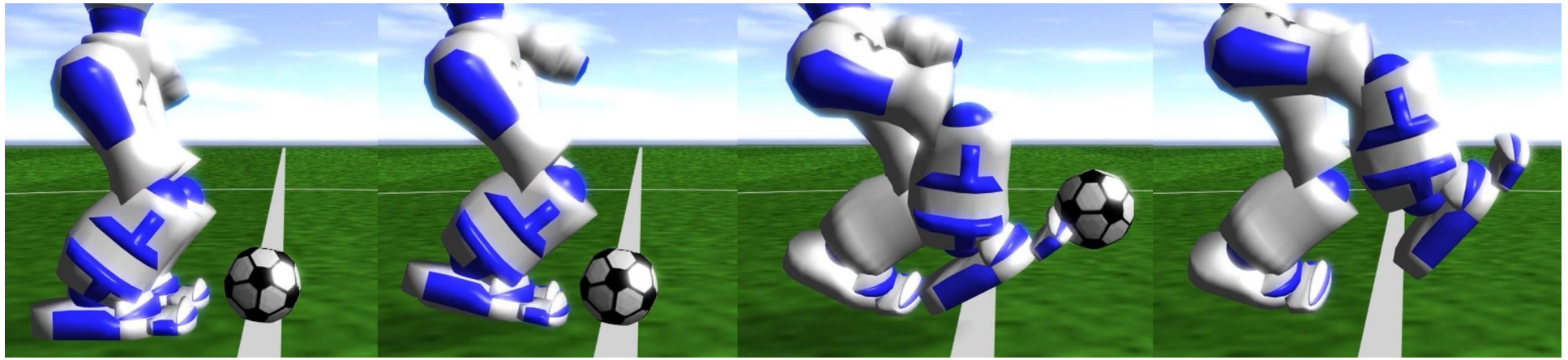
- Multi-directional kick
  - x-axis: relative kick direction, y-axis: kick distance
  - Reward = $100*(d_0-d)/d_0$ – fallenPenalty
  - 100 kicks per square

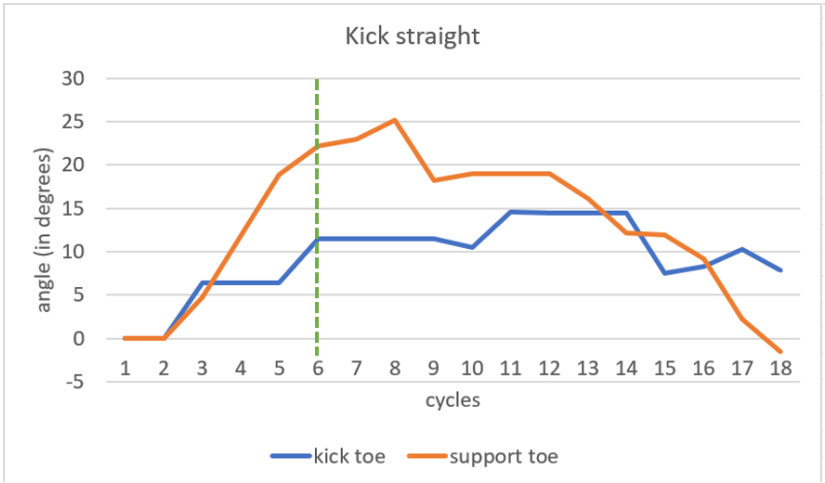


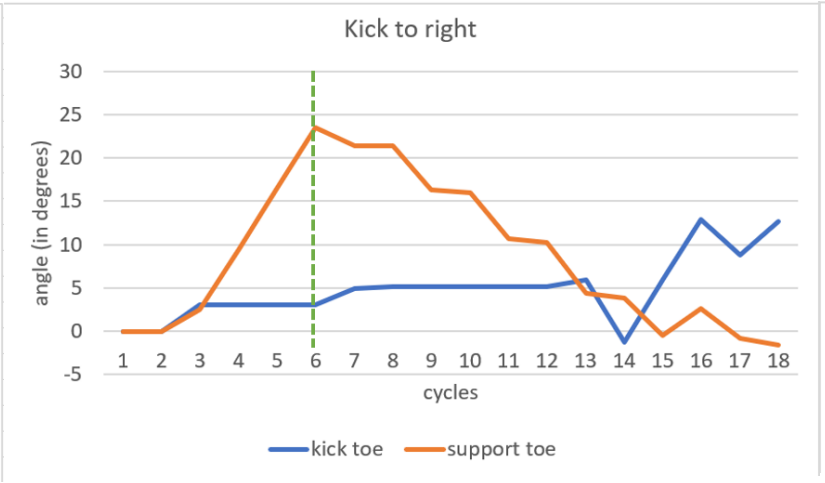
	-45	-40	-35	-30	-25	-20	-15	-10	-5	0	5	10	15	20	25	30	35	40	45
3	50.128	51.764	57.595	63.607	58.772	53.812	61.842	63.492	61.82	64.816	64.711	61.841	66.207	69.868	70.471	72.092	77.937	65.233	66.17
4	69.912	75.018	84.187	88.026	85.856	86.09	88.069	88.751	84.201	84.394	85.603	87.738	86.364	90.459	89.918	89.851	86.372	81.584	73.303
5	69.341	65.63	78.183	84.33	84.244	83.574	86.309	86.607	84.909	82.522	83.681	81.847	84.072	83.264	82.772	87.191	81.64	79.82	81.725
6	72.144	73.079	79.267	81.238	82.556	80.771	85.564	86.765	87.677	88.977	85.701	86.251	87.912	84.394	89.312	88.182	86.443	86.565	75.99
7	80.228	83.917	84.942	87.749	82.5	88.35	88.691	92.172	91.974	90.758	89.388	90.228	92.012	90.243	90.289	91.572	89.946	80.645	78.161
8	82.583	85.826	87.806	89.517	90.73	89.742	93.14	93.51	94.501	93.99	92.446	93.527	91.337	93.91	94.71	93.772	90.557	82.334	76.369
9	84.146	87.015	87.926	93.267	92.239	93.067	91.764	94.588	94.929	95.252	94.34	95.034	95.934	94.618	95.403	93.003	91.119	81.427	74.768
10	80.101	88.936	91.196	91.191	93.536	94.26	94.746	93.223	94.665	95.438	95.881	95.178	94.281	95.739	95.036	93.461	88.1	81.786	78.318
11	81.189	85.224	92.209	95.325	95.221	94.235	95.39	95.147	96.312	96.209	92.141	95.409	95.606	95.349	95.045	91.227	86.878	80.749	75.597
12	80.01	83.373	87.268	94.837	96.478	95.091	95.778	95.558	96.419	96.407	95.259	94.053	95.551	91.449	92.323	90.241	84.899	76.231	73.112
13	78.176	82.215	86.889	93.582	93.226	95.344	95.218	96.033	96.406	95.118	94.012	94.339	93.311	91.506	89.875	87.095	84.449	78.491	70.777
14	74.009	75.568	88.258	91.173	94.197	92.542	92.465	95.215	95.09	94.028	92.625	93.131	92.907	92.929	86.717	84.454	80.387	75.835	69.517
15	75.497	82.205	82.169	88.631	91.738	92.956	94.302	93.629	92.694	91.742	91.312	91.825	91.141	94.122	91.775	82.505	78.386	74.44	63.478
16	65.396	78.107	81.246	86.277	90.33	91.166	92.056	91.103	91.085	91.05	91.44	91.72	92.315	91.969	89.505	83.452	77.927	70.019	63.675
17	66.123	77.047	82.561	86.425	87.079	90.394	90.167	93.264	94.528	93.276	95.434	95.349	95.171	88.524	84.604	77.989	75.458	70.311	63.637
18	66.504	72.118	75.727	85.095	89.12	90.524	92.795	94.169	94.12	95.053	94.936	89.812	90.371	89.153	83.422	81.34	70.193	67.498	58.92
19	58.584	66.39	75.504	82.921	84.71	85.779	91.098	91.134	91.594	90.585	90.755	90.707	88.088	84.75	81.37	77.087	73.66	64.31	58.348
20	59.217	65.211	73.193	78.42	82.007	82.74	85.2	87.843	90.489	87.523	90.069	89.431	82.88	81.647	77.468	69.646	70.093	60.763	54.831
21	53.795	66.459	69.376	76.497	76.654	79.507	84.852	82.417	87.673	82.566	84.585	84.003	81.295	78.882	74.976	72.826	67.319	61.297	53.551

DRL Kick

- NAOToe







Spitznagel, Weiler, Dorer (2021) Deep Reinforcement Multi-Directional Kick-Learning of a Simulated Robot with Toes. IEEE ICARSC 2021.

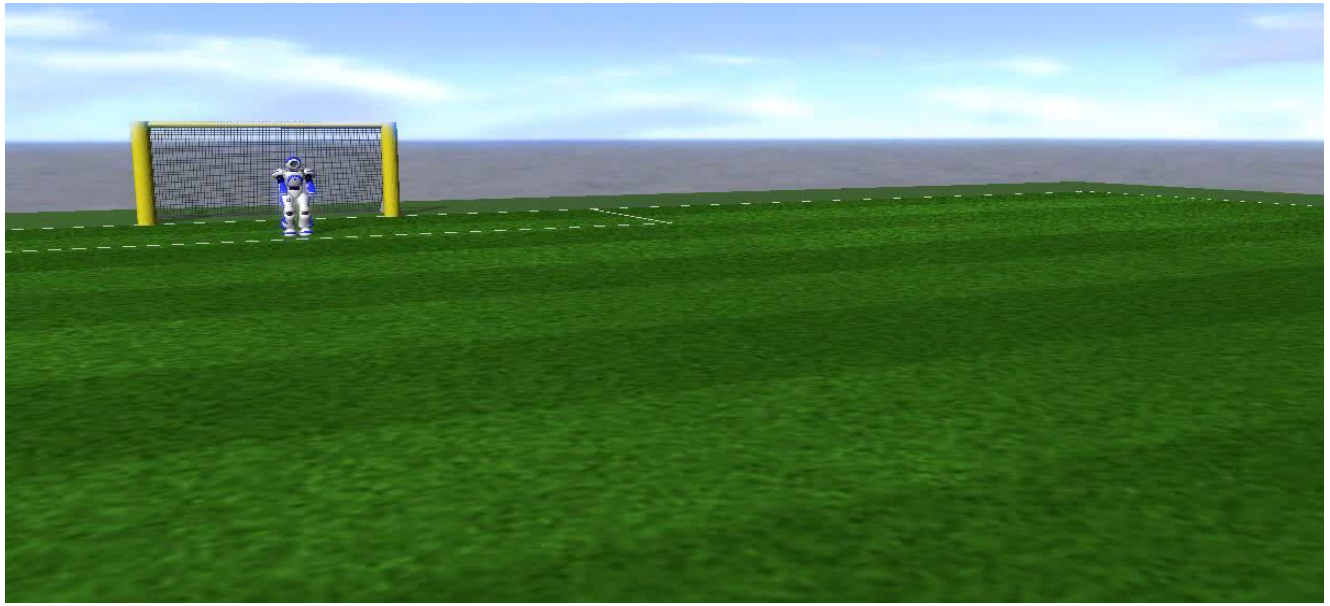
DRL Kick

DRL Kick

- 2023

DRL Walk

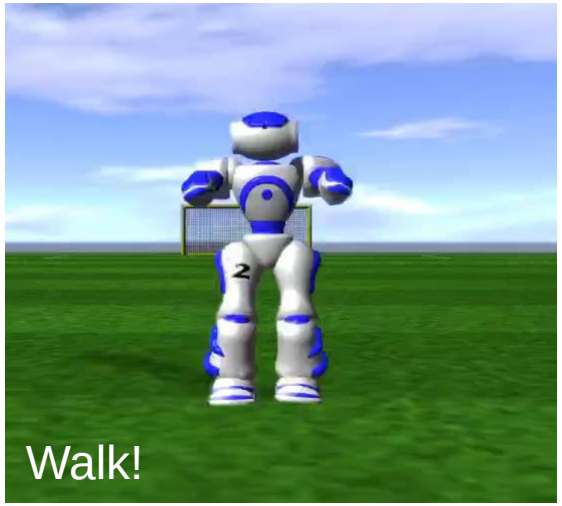
- 2018
  - 1 m/s
  - Genetic, time for learning 8h

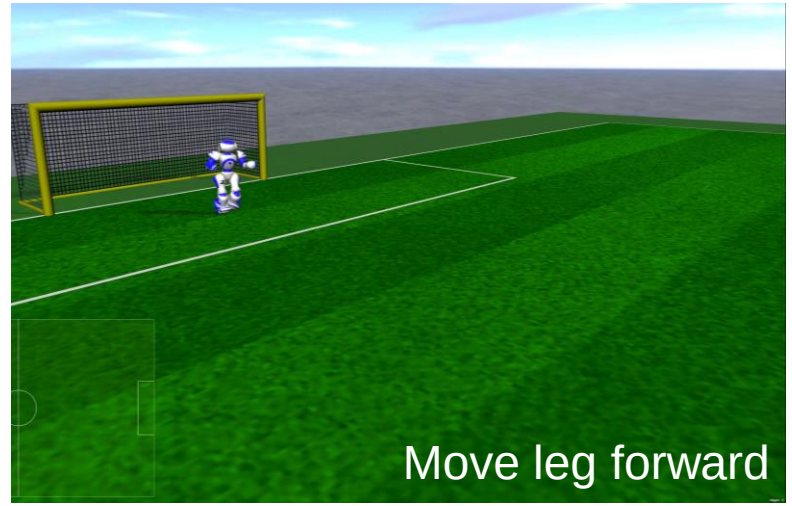


- 2020
  - 2 m/s
  - DRL, time for learning 8d

Lessons Learned

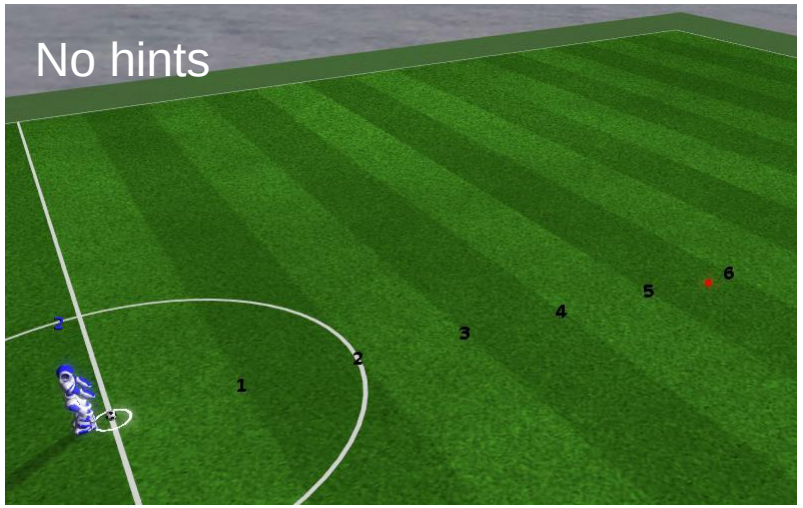
- Reward function plays significant role
  - Hints help sometimes
  - But may be missleading at other times





Lessons Learned

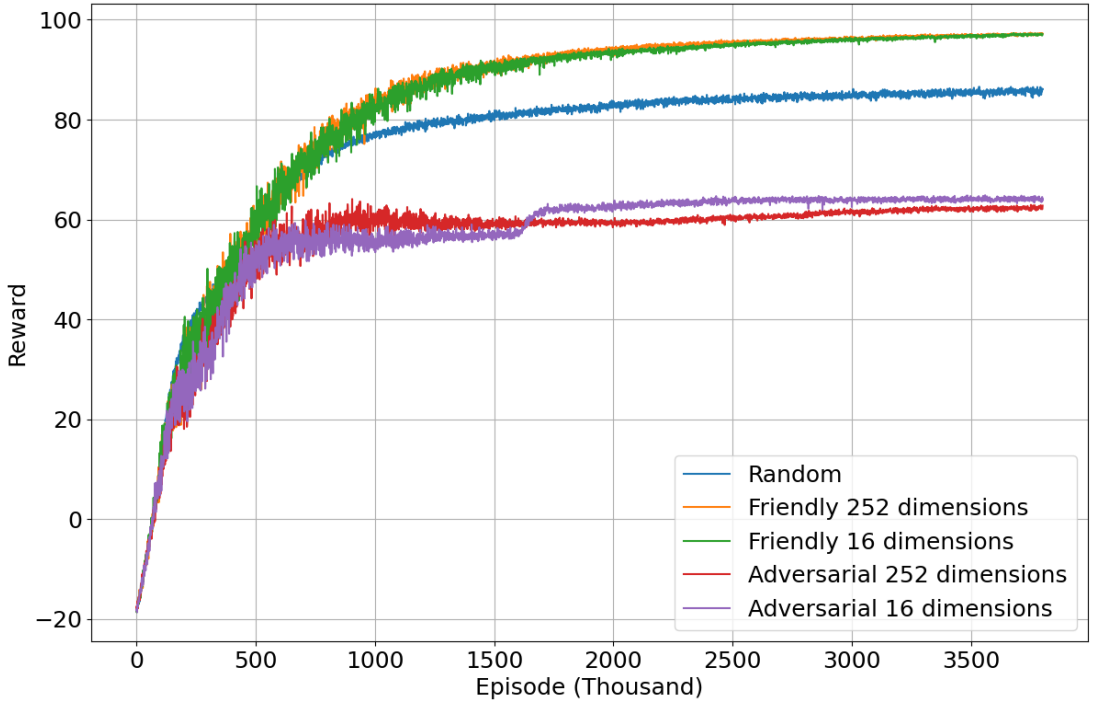
- Reward function plays significant role
  - Hints help sometimes
  - But may be missleading at other times



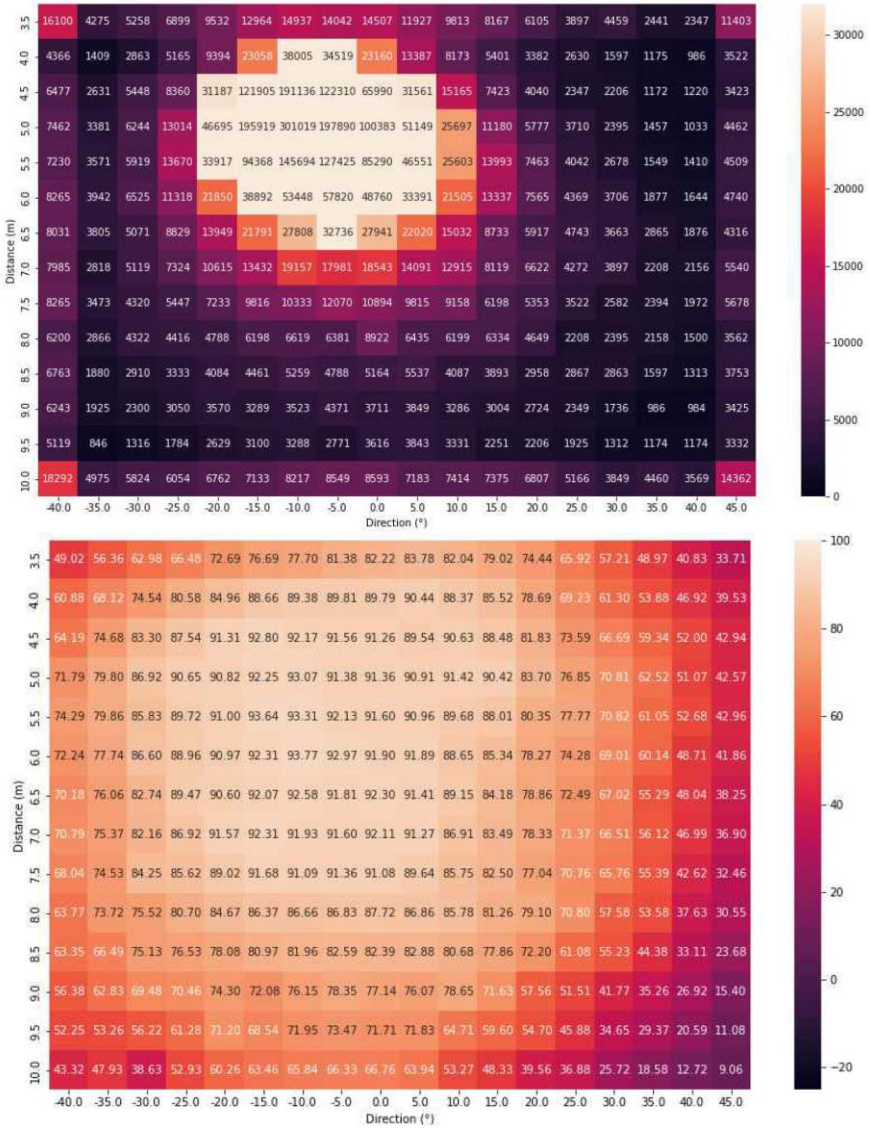
Lessons Learned

Kiefer, Dorer (2023) Double Deep Reinforcement Learning. To appear: Proceedings of the IEEE ICARSC-23 conference (Best Paper Award).

- Learning curves may be deceiving

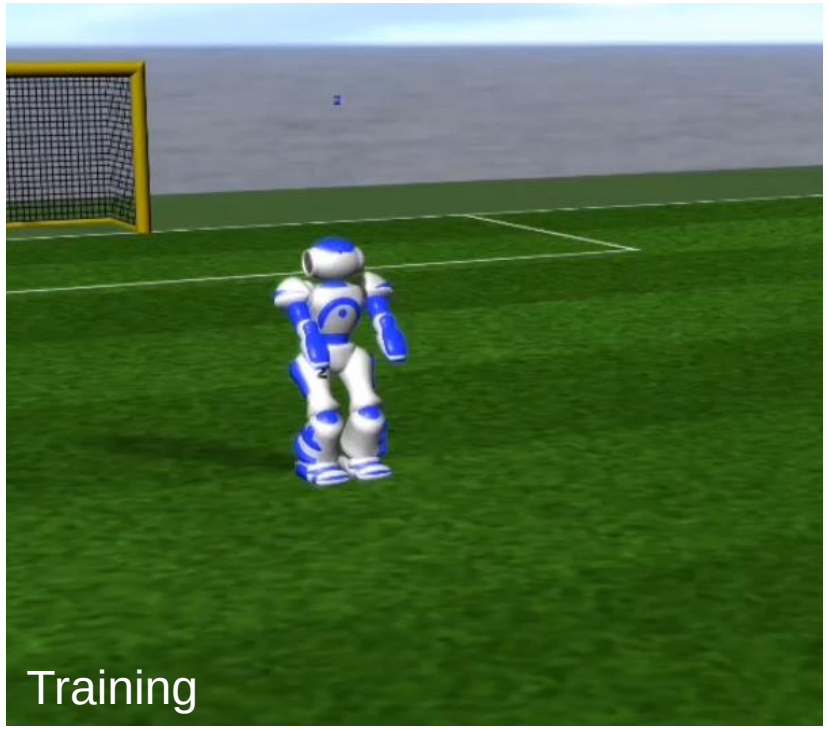


71
86



Lessons Learned

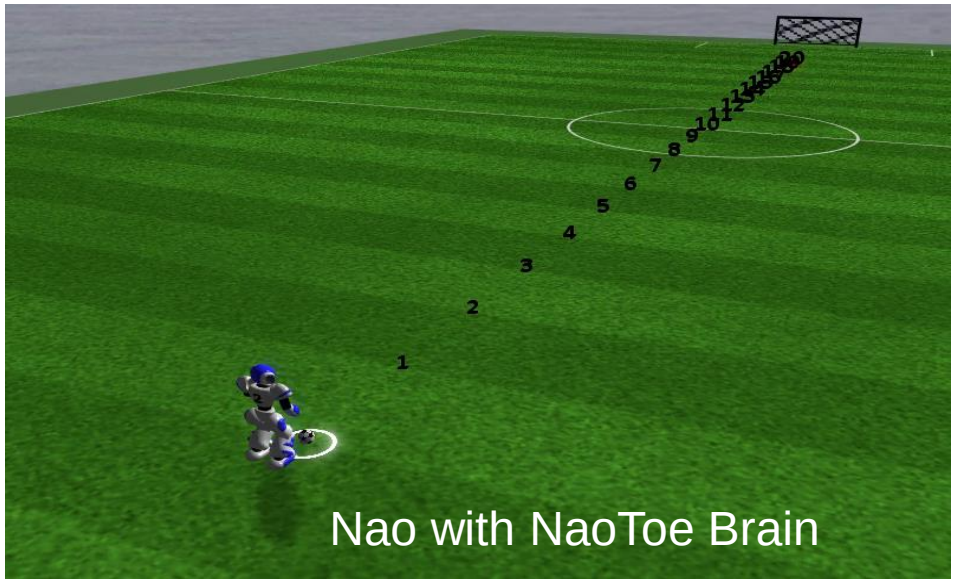
- Learning a behavior is only the first step
  - Example get up from front



Lessons Learned

- Weaknesses in the simulation are exploited without any mercy

- Results are highly optimized for the specific system used in training
  - Behaviors trained for a robot with toes do not transfer well to one without



Example RoboCup

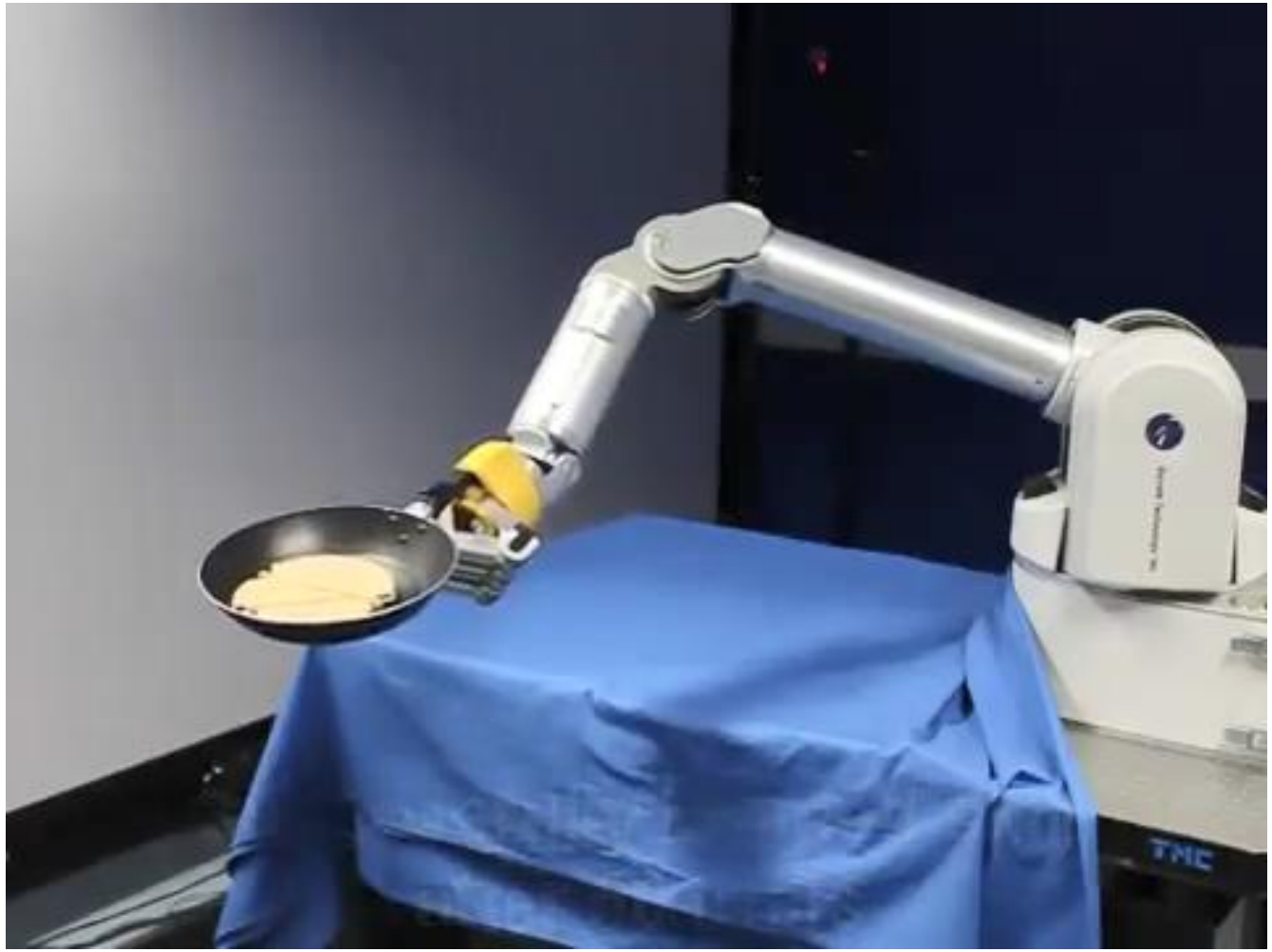
2022 Semi-final against the reigning world champion from USA

Real Robot Examples

- Hochschule Offenburg

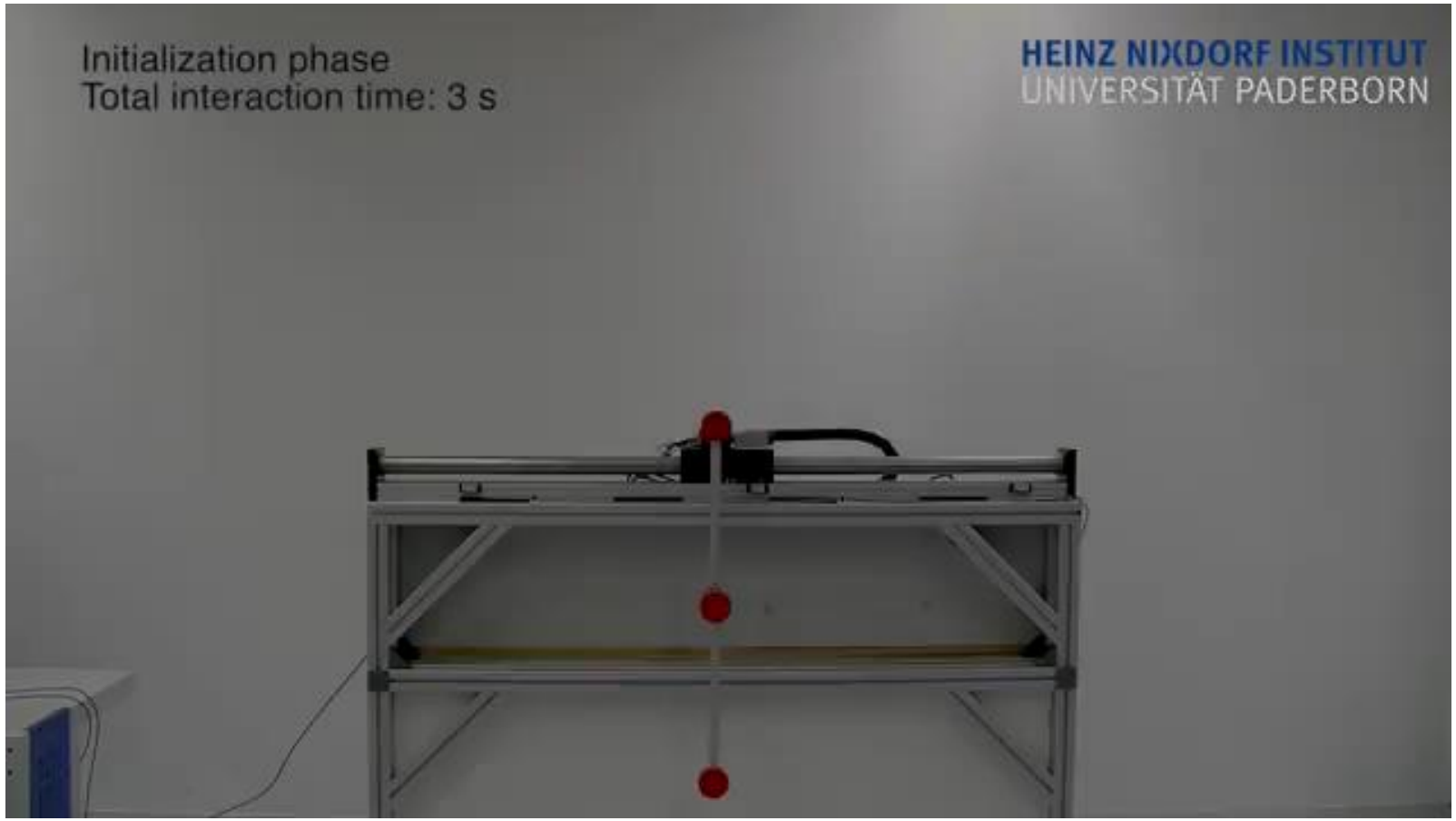
Real Robot Examples

- Italian Institute of Technology



Real Robot Examples

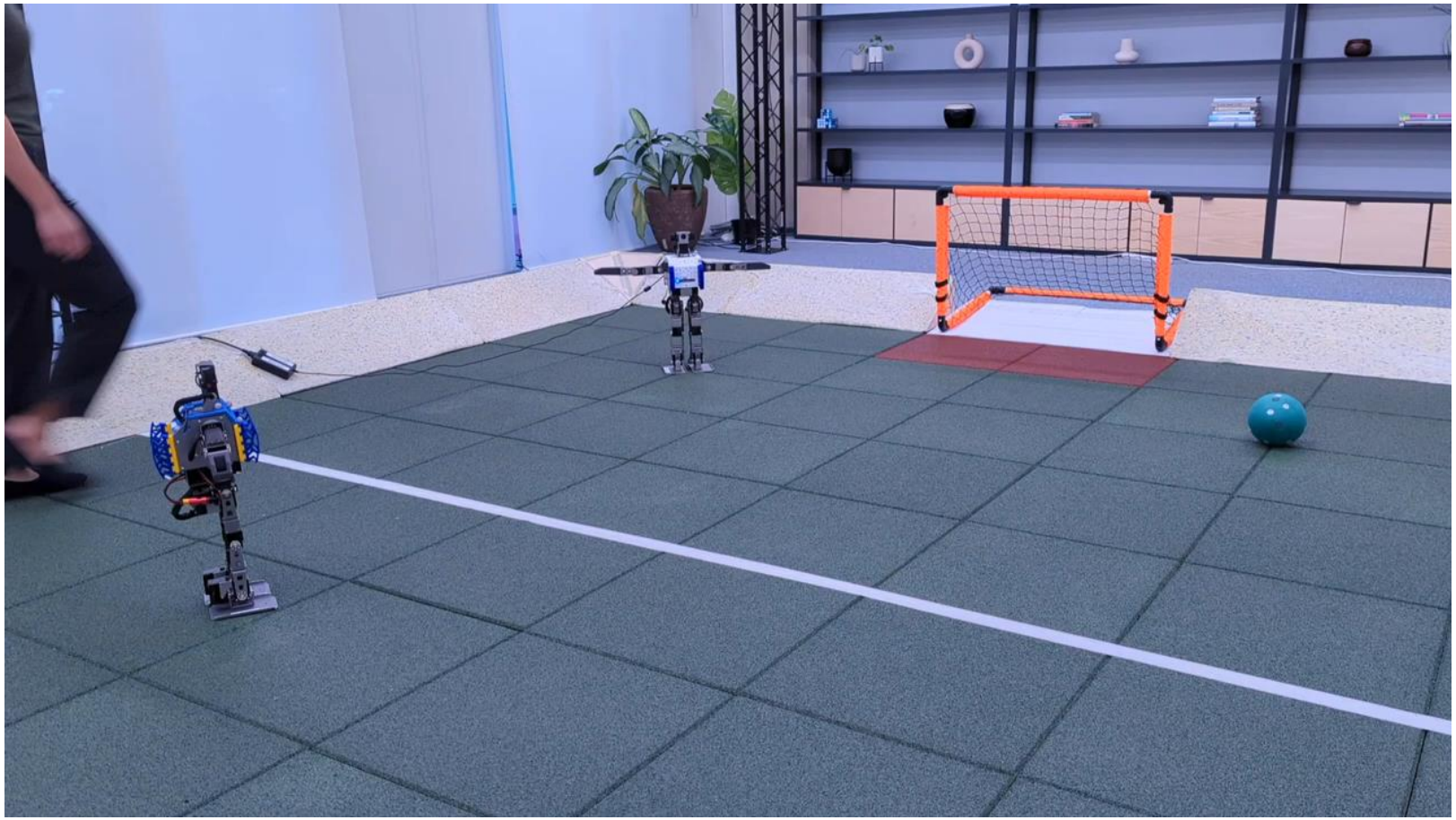
- Uni Paderborn / Heinz-Nixdorf-Institut



https://www.youtube.com/watch?v=N-yrQu9zuOI

Real Robot Examples

- DeepMind



https://www.youtube.com/watch?v=KSvLcr5HtNc&t=27s

Haarnoja et al. (2023) Learning Agile Soccer Skills for a Bipedal Robot with Deep Reinforcement Learning. arXiv:2304.13653

Summary

- State-value function can be learnd with timely delayed reinforcements
- Q-Learning learns the action-value function
- Deep Reinforcement Learning uses neural networks to generalize over states

- For real live
  - The value of a domain depends on what you do with it

Sources

- Martin Spitznagel, David Weiler, Klaus Dorer (2021) Deep Reinforcement Multi-Directional Kick-Learning of a Simulated Robot with Toes. IEEE ICARSC 2021.
- Stuart Russell, Peter Norvig (2015) Artificial intelligence : A Modern Approach. Pearson Verlag.