CS4400
DEEP REINFORCEMENT LEARNING

Lecture 5: Advanced DQN

Wendelin Böhmer

<j.w.bohmer@tudelft.nl>

30th of November 2023

Content of this lecture

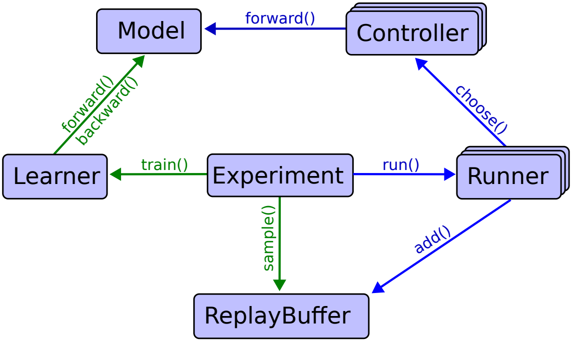
5.1

Advanced DQN

Q-learning extensions

5.1 Synchronous software architecture

- Alternate `run()` and `train()`
  - `Learner` on GPU, `Runner` on CPU/GPU
  - large `ReplayBuffer` on CPU
  - many parallel `Runner` possible

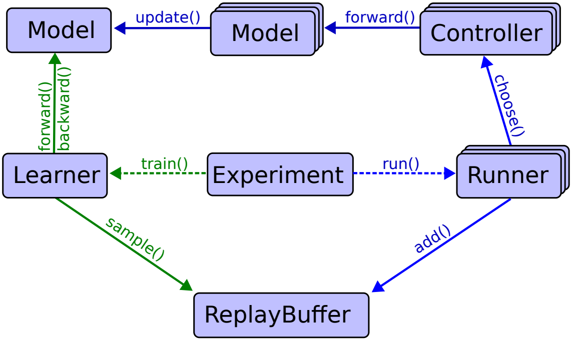


assignment sheet 2

see Daley and Amato (2021) for state-of-the-art DQN architectures

Asynchronous software architecture

- Endless `run()` and `train()` in parallel threads
  - `Controller` regularly update copy of `Model`
  - `Runner` can run on different servers
  - multiple `Learner` regularly average `Model`



see Daley and Amato (2021) for state-of-the-art DQN architectures

5.1 Double Q-learning

- Uncertain bootstrapping *overestimates* values
  - amplifies over multiple time-steps

$$\mathbb{E}\Big[\max_a Q(s,a)\Big] \quad \geq \quad \max_a \mathbb{E}[Q(s,a)]$$

assignment sheet 2

DDQN by van Hasselt et al. (2016)

5.1 Double Q-learning

- Uncertain bootstrapping *overestimates* values
  - amplifies over multiple time-steps

$$\mathbb{E}\big[\max_a Q(s,a)\big] \quad \geq \quad \max_a \mathbb{E}[Q(s,a)]$$

- Double Q-learning: use independent estimate Q'
  - Q and Q' are alternately trained
  - requires twice as many transitions

$$\mathbb{E}\big[Q\big(s, \arg\max_a Q'(s,a)\big)\big] \quad \approx \quad \max_a \mathbb{E}\big[Q(s,a)\big]$$

assignment sheet 2

DDQN by van Hasselt et al. (2016)

5.1 Double Q-learning

- Uncertain bootstrapping *overestimates* values
  - amplifies over multiple time-steps

$$\mathbb{E}\big[\max_a Q(s,a)\big] \quad \geq \quad \max_a \mathbb{E}[Q(s,a)]$$

- Double Q-learning: use independent estimate Q'
  - Q and Q' are alternately trained
  - requires twice as many transitions

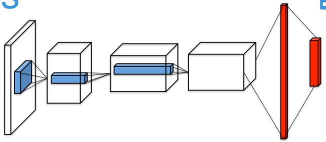
$$\mathbb{E}\big[Q\big(s, \arg\max_a Q'(s,a)\big)\big] \quad \approx \quad \max_a \mathbb{E}\big[Q(s,a)\big]$$

- Deep Double Q-learning (DDQN)
  - target network is Q and current network is Q'
  - not uncorrelated, but works in practice

assignment sheet 2

DDQN by van Hasselt et al. (2016)

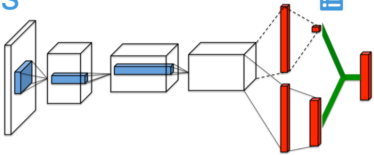
5.1 Dueling network architectures



- Estimating $Q(s,a)$ for many actions
  - only one head at a time updated
  - update all actions with same successor state s'?
- $Q^\pi(s,a) =: V^\pi(s) + A^\pi(s,a)$
  - advantage of action $A^\pi(s,a)$
  - $Q^*(s,a^*) = V^*(s) \Rightarrow A^*(s,a^*) = 0$
  - estimating $A^* + V^*$ easier than Q^*?

images and dueling architectures from Wang et al. (2016)

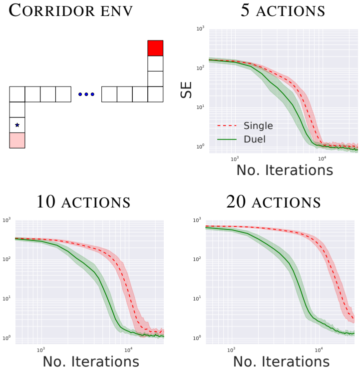
5.1 Dueling network architectures



- Estimating $Q(s,a)$ for many actions
  - only one head at a time updated
  - update all actions with same successor state s'?

- $Q^\pi(s,a) =: V^\pi(s) + A^\pi(s,a)$
  - advantage of action $A^\pi(s,a)$
  - $Q^*(s,a^*) = V^*(s) \Rightarrow A^*(s,a^*) = 0$
  - estimating $A^* + V^*$ easier than Q^*?

- Dueling architectures: $Q(s,a) :=$
  - $V(s) + \left(A(s,a) - \max\limits_{a \in \mathcal{A}} A(s,a)\right)$
  - $V(s) + \left(A(s,a) - \frac{1}{|\mathcal{A}|} \sum\limits_{a \in \mathcal{A}} A(s,a)\right)$



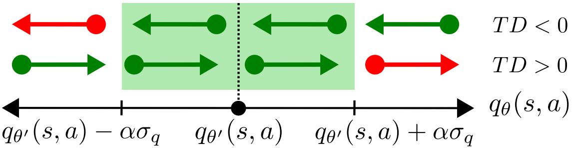
images and dueling architectures from Wang et al. (2016)

5.1 Trust-region online bootstrapping

- Target networks θ' stabilize, but delay value propagation
- Bootstrapping with (detached) *online network* θ must be stabilized
  - mask out TD losses when online and TD error very different, when:

$$
\begin{aligned}
\left|q_\theta(s_t,a_t) - q_{\theta'}(s_t,a_t)\right| \quad &> \quad \alpha \overbrace{\sqrt{\mathbb{V}[q_\theta(s,a) | (s,a) \sim \mathcal{D}]}}^{\sigma_q} \\
\operatorname{sgn}\Big(q_\theta(s_t,a_t) - q_{\theta'}(s_t,a_t)\Big) \quad &\neq \quad \operatorname{sgn}\Big(\underbrace{q_\theta(s_t,a_t) - \big(r_t + \gamma \max_a q_\theta(s_{t+1},a)\big)}_{TD_t}\Big)
\end{aligned}
$$

  - only gradient steps towards trust-region allowed



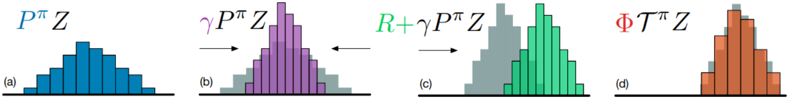
method and figure replicated from Badia et al. (2020)

5.1 Distributional Q-learning

- Expectations over returns not always enough
  - e.g. for risk-sensitive or constrained RL

5.1 Distributional Q-learning

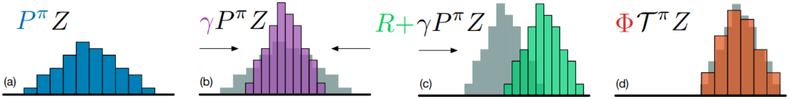
- Expectations over returns not always enough
  - e.g. for risk-sensitive or constrained RL
- Estimate distribution over possible returns $Z_t = \sum_{i=0}^{\infty} \gamma^i R_{t+i} \in \mathbb{R}$
  - $\mathbb{P}(Z_t | s_t, a_t) \;\overset{D}{=}\; \mathbb{P}(R_t | s_t, a_t) \;+\; \gamma\, \mathbb{P}(Z_{t+1} | s_{t+1}, a_{t+1})$



originally by Bellemare et al. (2017), see Lyle et al. (2019) for a recent comparison with DQN

Distributional Q-learning

- Expectations over returns not always enough
  - e.g. for risk-sensitive or constrained RL
- Estimate distribution over possible returns $Z_t = \sum\limits_{i=0}^{\infty} \gamma^i R_{t+i} \in \mathbb{R}$
  - $\mathbb{P}(Z_t | s_t, a_t) \quad \overset{D}{=} \quad \mathbb{P}(R_t | s_t, a_t) \;+\; \gamma \, \mathbb{P}(Z_{t+1} | s_{t+1}, a_{t+1})$

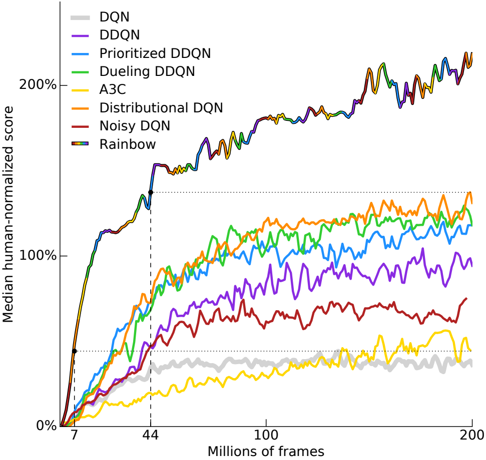


- Many ways to model the return-distribution
  - categorical distribution (Bellemare et al., 2017)
  - quantile regression (Dabney et al., 2018)
  - moment matching (Nguyen-Tang et al., 2021)

originally by Bellemare et al. (2017), see Lyle et al. (2019) for a recent comparison with DQN

5.1 Modern DQN versions

- Combine DQN extensions
- e.g. RAINBOW (2018)
  - double Q-learning
  - prioritized replay
  - dueling architectures
  - distributional DQN
- e.g. Agent57 (2020)
  - trust-region
  - short-term memory
  - exploration
  - episodic memory
  - meta-controllers



see Deepmind's blog about Agent57 for a good overview

RAINBOW and image by Hessel et al. (2018), Agent57 by Badia et al. (2020), newest results in Kapturowski et al. (2023)

5.1 Summary

- Standard architecture can be easily parallelized
- Double Q-learning reduces overestimation
- Dueling networks for many similar actions
- Trust-regions for faster value propagation
- Distributional Q-learning to estimate risk
- Modern DQN versions combine tricks to learn superhuman AI

Learning Objectives

LO5.1: Explain when the discussed extensions are beneficial
LO5.2: Implement and test double Q-learning

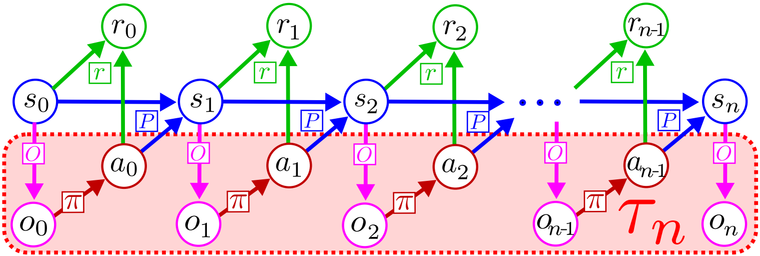
5.2

Advanced DQN

Partial observablity

5.2 Partially observable MDP

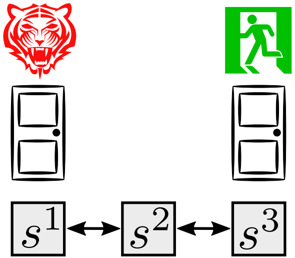
- POMDP: MDP + observations $o_t \sim O(\cdot|s_t) \in \mathcal{O}$
  - true state not observed by agent
  - histories $\tau_t := (o_0, a_0, \ldots, o_t) \in (\mathcal{O} \times \mathcal{A})^t \times \mathcal{O}$



see e.g. Spaan (2012) for an introduction to POMDP

5.2 Partially observable MDP

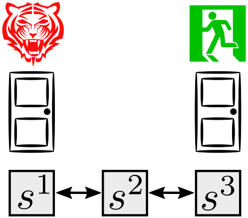
- POMDP: MDP + observations $o_t \sim O(\cdot|s_t) \in \mathcal{O}$
  - true state not observed by agent
  - histories $\tau_t := (o_0, a_0, \ldots, o_t) \in (\mathcal{O} \times \mathcal{A})^t \times \mathcal{O}$
  - example: escape tiger by only observing doors

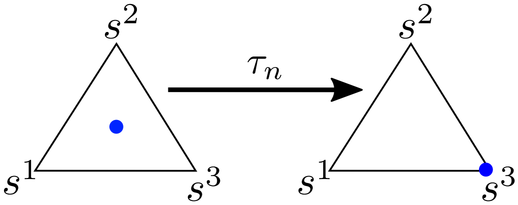


see e.g. Spaan (2012) for an introduction to POMDP

5.2 Partially observable MDP

- POMDP: MDP + observations $o_t \sim O(\cdot|s_t) \in \mathcal{O}$
  - true state not observed by agent
  - histories $\tau_t := (o_0, a_0, \ldots, o_t) \in (\mathcal{O} \times \mathcal{A})^t \times \mathcal{O}$
  - example: escape tiger by only observing doors
- Equivalent deterministic MDP over belief-distributions b
  - $b(s|\tau_t) := \mathbb{P}(s_t = s | b_0, o_0, a_0, \ldots, o_t)$
  - $b(s|\tau_{t+1}) \propto O(o_{t+1}|s) \int P(s|s', a_t)\, b(s'|\tau_t)\, ds'$





see e.g. Spaan (2012) for an introduction to POMDP

5.2 Partially observable MDP

- POMDP: MDP + observations $o_t \sim O(\cdot|s_t) \in \mathcal{O}$
  - true state not observed by agent
  - histories $\tau_t := (o_0, a_0, \ldots, o_t) \in (\mathcal{O} \times \mathcal{A})^t \times \mathcal{O}$
  - example: escape tiger by only observing doors
- Equivalent deterministic MDP over belief-distributions b
  - $b(s|\tau_t) := \mathbb{P}(s_t = s | b_0, o_0, a_0, \ldots, o_t)$
  - $b(s|\tau_{t+1}) \propto O(o_{t+1}|s) \int P(s|s', a_t)\, b(s'|\tau_t)\, ds'$
- History τ_t is *sufficient statistic* of belief $b(s|\tau_t)$
  - $r(\tau_t, a) = \int r(s,a)\, b(s|\tau_t)\, ds$
  - $P(\tau_{t+1}|\tau_t, a_t) = \iint O(o_{t+1}|s')\, P(s'|s, a_t)\, b(s|\tau_t)\, ds\, ds'$
  - $Q^*(\tau_t, a) = r(\tau_t, a) + \gamma \int P(\tau_{t+1}|\tau_t, a) \max_{a' \in \mathcal{A}} Q^*(\tau_{t+1}, a')\, d\tau_{t+1}$

see e.g. Spaan (2012) for an introduction to POMDP

core concept: Deep Recurrent Q-Networks

- Use LSTM to encode the history $\tau_t = (o_0, a_0, \dots, o_t)$
  - $\boldsymbol{h}_t = \mathrm{LSTM}(\boldsymbol{h}_{t-1}, \underbrace{a_{t-1}, o_t}_{\text{input}}) \in \mathbb{R}^m, \quad \boldsymbol{h}_0 = \mathrm{LSTM}(\underbrace{\boldsymbol{0}, 0,}_{\text{init}} o_0)$

- End-to-end Q-learning of histories

$$\mathcal{L}[\theta] \;=\; \mathbb{E}\Big[\sum_{t=0}^{n}\Big(r_t + \gamma \max_{a \in \mathcal{A}} q_{\theta'}(\boldsymbol{h}_{t+1}, a) - q_\theta(\boldsymbol{h}_t, a_t)\Big)^2 \,\Big|\, \tau_n \sim \mathcal{D}\Big]$$

- How does this change the replay buffer?

Deep Recurrent Q-networks (DRQN) by Hausknecht and Stone (2015)

core concept: Deep Recurrent Q-Networks

- Use LSTM to encode the history $\tau_t = (o_0, a_0, \ldots, o_t)$
  - $\boldsymbol{h}_t = \text{LSTM}(\boldsymbol{h}_{t-1}, \underbrace{a_{t-1}, o_t}_{\text{input}}) \in \mathbb{R}^m, \quad \boldsymbol{h}_0 = \text{LSTM}(\underbrace{\boldsymbol{0}, 0,}_{\text{init}} o_0)$

- End-to-end Q-learning of histories

$$\mathcal{L}[\theta] \;=\; \mathbb{E}\Big[\sum_{t=0}^{n}\Big(r_t + \gamma \max_{a\in\mathcal{A}} q_{\theta'}(\boldsymbol{h}_{t+1}, a) - q_\theta(\boldsymbol{h}_t, a_t)\Big)^2 \,\Big|\, \tau_n \sim \mathcal{D}\Big]$$

- How does this change the replay buffer?
  - sample mini-batch of episodes, not transitions
  - contiguous tensor stores episodes of varying length
  - zero-out non-existent losses afterwards

Deep Recurrent Q-networks (DRQN) by Hausknecht and Stone (2015)

5.2 DRQN implementation (1)

- We start with a normal DQN gradient descent

```python
import torch as th
batch = self.replay_buffer.sample()
targets = ???
current_v = ???
loss = mse_loss(current_v, targets.detach())
# gradient descent step on supervised regression loss
optimizer.zero_grad()
(loss).backward()
optimizer.step()
```

5.2 DRQN implementation (2)

- Replay buffer returns batch of episodes (e.g. time-dim=0)
- Tensor slice `[:-1]` and `[1:]` for 'current' and 'next'

```python
import torch as th
batch = self.replay_buffer.sample()
acts = batch['actions']
values = ???
# derive the regression targets from the values
max_v = ~batch['terminals'] * values.max(dim=-1, keepdim=True)[0]
targets = (batch['rewards'][:-1] + gamma * max_v[1:])
# derive the MSE loss to be optimized
current_v = values[:-1].gather(dim=-1, index=acts[:-1])
loss = mse_loss(current_v, targets.detach())
# gradient descent step on supervised regression loss
optimizer.zero_grad()
(loss).backward()
optimizer.step()
```

5.2 DRQN implementation (3)

- LSTM Q-value function q on observations o_t and actions a_{t-1}
- $\boldsymbol{h}_t = \mathrm{LSTM}(\boldsymbol{h}_{t-1}, a_{t-1}, o_t)$ starts with o_0 and $a_{-1} = \mathbf{0}$

```python
import torch as th
batch = self.replay_buffer.sample()
acts = batch['actions']
# compute all recurrent Q-values with function q
first_act = th.zeros(1, *acts.shape[1:])  # a_{-1} = 0
delayed_acts = th.cat([first_act, acts[:-1]], dim=0)
values = q(th.cat([delayed_acts, batch['observations']], dim=-1))[0]
# derive the regression targets from the values
max_v = ~batch['terminals'] * values.max(dim=-1, keepdim=True)[0]
targets = (batch['rewards'][:-1] + gamma * max_v[1:])
# derive the MSE loss to be optimized
current_v = values[:-1].gather(dim=-1, index=acts[:-1])
loss = mse_loss(current_v, targets.detach())
# gradient descent step on supervised regression loss
optimizer.zero_grad()
(loss).backward()
optimizer.step()
```

5.2 DRQN implementation (4)

- Multiplying with a `mask` tensor removes 'invalid' entries
- Normalize loss by number of 'valid' TD-errors
- Can be implemented with any DQN extension

```
import torch as th
batch = self.replay_buffer.sample()
acts, mask = batch['actions'], batch['mask'][:-1]
# compute all recurrent Q-values with function q
first_act = th.zeros(1, *acts.shape[1:])  # a_{-1} = 0
delayed_acts = th.cat([first_act, acts[:-1]], dim=0)
values = q(th.cat([delayed_acts, batch['observations']], dim=-1))[0]
# derive the regression targets from the values
max_v = ~batch['terminals'] * values.max(dim=-1, keepdim=True)[0]
targets = (batch['rewards'][:-1] + gamma * max_v[1:]) * mask
# derive the MSE loss to be optimized
current_v = values[:-1].gather(dim=-1, index=acts[:-1]) * mask
loss = mse_loss(current_v, targets.detach(), reduction='sum')
# gradient descent step on supervised regression loss
optimizer.zero_grad()
(loss / mask.sum()).backward()
optimizer.step()
```

Question: Design a neural architecture

- Design a neural network architecture for the shown task
  - agent observes (ambiguous) first-person images
  - actions $\boldsymbol{a} \in \mathbb{R}^2$: move forward by a_1, rotate by a_2
  - collecting apples and reaching goal rewarded
- Choose a fitting neural architecture for the PPO algorithm
  - sketch outputs and which modules feed into each other

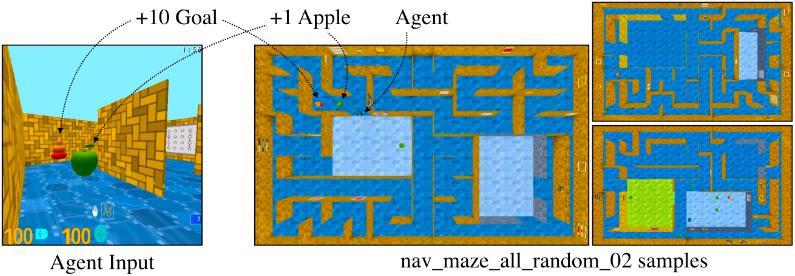
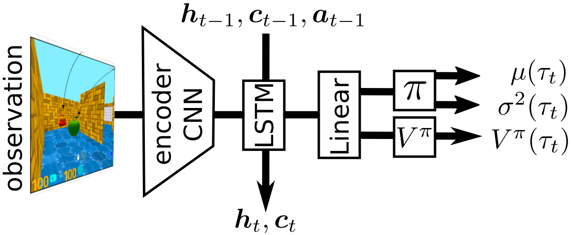


image from the Deepmind Control Suite (Jaderberg et al., 2017)

Question: Design a neural architecture

- Design a neural network architecture for the shown task
  - agent observes (ambiguous) first-person images
  - actions $\boldsymbol{a} \in \mathbb{R}^2$: move forward by a_1, rotate by a_2
  - collecting apples and reaching goal rewarded
- Choose a fitting neural architecture for the PPO algorithm
  - sketch outputs and which modules feed into each other



$$\boldsymbol{a}_t \sim \mathcal{N}\big(\cdot \,|\mu(\tau_t), \sigma^2(\tau_t)\big)$$

image from the Deepmind Control Suite (Jaderberg et al., 2017)

5.2 Summary

- POMDP induce deterministic Belief-MDP
- Histories are a sufficient statistic for the belief
- DRQN encodes histories with LSTMs
- LSTM, CNN and other modules can be combined

Learning Objectives

LO5.3: Explain why deep recurrent Q-networks solve POMDP
LO5.4: Design neural architectures for POMDP algorithms

5.2 Next lecture

- Next lecture:
  continuous actions!

- Don't forget assingment sheet 2

- Questions? Ask them here:
  `answers.ewi.tudelft.nl`

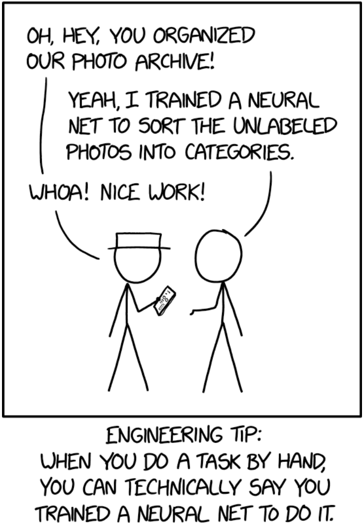


image source: `xkcd.com`

References I

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, volume 119 of *Proceedings of Machine Learning Research*, pages 507–517, 2020. URL https://arxiv.org/abs/2003.13350.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 449–458, 2017. URL https://proceedings.mlr.press/v70/bellemare17a.html.

Will Dabney, Georg Ostrovski, David Silver, and Remi Munos. Implicit quantile networks for distributional reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1096–1105. PMLR, 2018. URL https://proceedings.mlr.press/v80/dabney18a.html.

Brett Daley and Christopher Amato. Human-level control without server-grade hardware, 2021. URL https://arxiv.org/abs/2111.01264.

Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposia*, pages 29–37, 2015. URL http://www.aaai.org/ocs/index.php/FSS/FSS15/paper/view/11673.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, pages 3215–3222, 2018. URL https://arxiv.org/abs/1710.02298.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *5th International Conference on Learning Representations, ICLR*. OpenReview.net, 2017. URL https://arxiv.org/abs/1611.05397.

Steven Kapturowski, Víctor Campos, Ray Jiang, Nemanja Rakicevic, Hado van Hasselt, Charles Blundell, and Adria Puigdomenech Badia. Human-level atari 200x faster. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2209.07550.

References II

Clare Lyle, Marc G. Bellemare, and Pablo Samuel Castro. A comparative analysis of expected and distributional reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4504–4511, Jul. 2019. doi: 10.1609/aaai.v33i01.33014504. URL https://ojs.aaai.org/index.php/AAAI/article/view/4365.

Thanh Nguyen-Tang, Sunil Gupta, and Svetha Venkatesh. Distributional reinforcement learning via moment matching. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):9144–9152, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/17104.

M. T. J. Spaan. Partially observable markov decision processes. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning: State of the Art*, pages 387–414. Springer Verlag, 2012. URL https://www.st.ewi.tudelft.nl/mtjspaan/pub/Spaan12pomdp.pdf.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, pages 2094–2100, 2016. URL https://arxiv.org/abs/1509.06461.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *the 33rd International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, 2016. URL http://proceedings.mlr.press/v48/wangf16.html.