CS4400
DEEP REINFORCEMENT LEARNING

Lecture 3: Deep Q-Learning

Wendelin Böhmer

<j.w.bohmer@tudelft.nl>

21st of November 2023

Content of this lecture

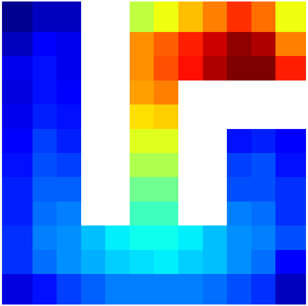
3.1

Deep Q-Learning

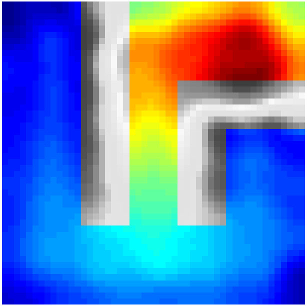
Value Approximation

Residual value gradients

- TD-error regression of data set $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1}, a_{t+1})\}_{t=1}^n$
  - sampled from one or more trajectories based on π



discrete value function



approximated value function

residual algorithms converge, but not necessarily to the correct values, as s_t and s_{t+1} are not independent (Baird, 1995)

3.1 Residual value gradients

- TD-error regression of data set $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1}, a_{t+1})\}_{t=1}^{n}$
  - sampled from one or more trajectories based on π
- Mean-squared error between value $v_\theta(s) \approx V^\pi(s)$ and target

$$\mathcal{L}[\theta] \quad := \quad \mathbb{E}_{\mathcal{D}}\Big[\Big(\overbrace{\underbrace{r_t + \gamma\, v_\theta(s_{t+1})}_{\text{target } y_t} - v_\theta(s_t)}^{\text{TD-error}}\Big)^2\Big]$$

residual algorithms converge, but not necessarily to the correct values, as s_t and s_{t+1} are not independent (Baird, 1995)

3.1 Residual value gradients

- TD-error regression of data set $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1}, a_{t+1})\}_{t=1}^{n}$
  - sampled from one or more trajectories based on π
- Mean-squared error between value $v_\theta(s) \approx V^\pi(s)$ and target

$$\mathcal{L}[\theta] \quad := \quad \mathbb{E}_{\mathcal{D}}\Big[\Big(\overbrace{\underbrace{r_t + \gamma\, v_\theta(s_{t+1})}_{\text{target } y_t} - v_\theta(s_t)}^{\text{TD-error}}\Big)^2\Big]$$

- SARSA: MSE between Q-value $q_\theta(s,a) \approx Q^\pi(s,a)$ and target
  - also the name of an on-policy control algorithm

$$\mathcal{L}[\theta] \quad := \quad \mathbb{E}_{\mathcal{D}}\Big[\Big(\underbrace{r_t + \gamma\, q_\theta(s_{t+1}, a_{t+1})}_{\text{target } y_t} - q_\theta(s_t, a_t)\Big)^2\Big]$$

residual algorithms converge, but not necessarily to the correct values, as s_t and s_{t+1} are not independent (Baird, 1995)

3.1 Residual gradients break causality

- Let's derive the Q-learning update rule for table $\mathbf{Q} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$
  - after observing transition $s_t, a_t \to r_t, s_{t+1}$ we minimize

$$\mathcal{L}[\mathbf{Q}] := \big(r_t + \gamma Q_{(s_{t+1},a^*)} - Q_{(s_t,a_t)}\big)^2\,, \qquad a^* := \arg\max_{a\in\mathcal{A}} Q_{(s_{t+1},a)}$$

- One gradient-descent step $\mathbf{Q} \leftarrow \mathbf{Q} - \frac{\alpha}{2}\nabla_{\mathbf{Q}}\mathcal{L}[\mathbf{Q}]$

$$Q_{(s,a)} \quad\leftarrow\quad Q_{(s,a)} - \alpha\big(r_t + \gamma Q_{(s_{t+1},a^*)} - Q_{(s_t,a_t)}\big)\big(\gamma\delta\big(\substack{a=a^*\\ s=s_{t+1}}\big) - \delta\big(\substack{a=a_t\\ s=s_t}\big)\big)$$

residual algorithms converge, but not necessarily to the correct values, as s_t and s_{t+1} are not independent (Baird, 1995)

Residual gradients break causality

- Let's derive the Q-learning update rule for table $\mathbf{Q} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$
  - after observing transition $s_t, a_t \to r_t, s_{t+1}$ we minimize

$$\mathcal{L}[\mathbf{Q}] := \Big(r_t + \gamma Q_{(s_{t+1},a^*)} - Q_{(s_t,a_t)}\Big)^2\,, \qquad a^* := \arg\max_{a\in\mathcal{A}} Q_{(s_{t+1},a)}$$

- One gradient-descent step $\mathbf{Q} \leftarrow \mathbf{Q} - \frac{\alpha}{2}\nabla_{\mathbf{Q}}\mathcal{L}[\mathbf{Q}]$

$$Q_{(s,a)} \quad\leftarrow\quad Q_{(s,a)} - \alpha\Big(r_t + \gamma Q_{(s_{t+1},a^*)} - Q_{(s_t,a_t)}\Big)\Big(\gamma\delta\big({}_{s=s_{t+1}}^{a=a^*}\big) - \delta\big({}_{s=s_t}^{a=a_t}\big)\Big)$$

$$Q_{(s_t,a_t)} \quad\leftarrow\quad (1-\alpha)Q_{(s_t,a_t)} + \alpha\Big(r_t + \gamma Q_{(s_{t+1},a^*)}\Big) \qquad \checkmark$$

residual algorithms converge, but not necessarily to the correct values, as s_t and s_{t+1} are not independent (Baird, 1995)

3.1 Residual gradients break causality

- Let's derive the Q-learning update rule for table $\mathbf{Q} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$
  - after observing transition $s_t, a_t \to r_t, s_{t+1}$ we minimize

$$\mathcal{L}[\mathbf{Q}] := \Big(r_t + \gamma Q_{(s_{t+1},a^*)} - Q_{(s_t,a_t)}\Big)^2, \qquad a^* := \arg\max_{a\in\mathcal{A}} Q_{(s_{t+1},a)}$$

- One gradient-descent step $\mathbf{Q} \leftarrow \mathbf{Q} - \frac{\alpha}{2}\nabla_{\mathbf{Q}}\mathcal{L}[\mathbf{Q}]$

$$\begin{aligned}
Q_{(s,a)} &\leftarrow Q_{(s,a)} - \alpha\Big(r_t + \gamma Q_{(s_{t+1},a^*)} - Q_{(s_t,a_t)}\Big)\Big(\gamma\delta\big({}^{a=a^*}_{s=s_{t+1}}\big) - \delta\big({}^{a=a_t}_{s=s_t}\big)\Big) \\
Q_{(s_t,a_t)} &\leftarrow (1-\alpha)Q_{(s_t,a_t)} + \alpha\Big(r_t + \gamma Q_{(s_{t+1},a^*)}\Big) \qquad \checkmark \\
Q_{(s_{t+1},a^*)} &\leftarrow (1-\alpha\gamma^2)Q_{(s_{t+1},a^*)} - \alpha\gamma\Big(r_t - Q_{(s_t,a_t)}\Big) \qquad \times
\end{aligned}$$

- $Q_{(s_{t+1},a^*)}$ does not depend on $Q_{(s_t,a_t)}$!
  - residual gradients break causality
  - in practice slows down learning considerably

residual algorithms converge, but not necessarily to the correct values, as s_t and s_{t+1} are not independent (Baird, 1995)

3.1 Value semi-gradients

core concept: Bootstrapping

- causality: future values do not depend on past values
- separate bootstrapping network $v_{\theta'}$ with parameters θ'

$$\mathcal{L}[\theta] \quad := \quad \mathbb{E}_{\mathcal{D}}\Big[\Big(\underbrace{r_t + \gamma\, v_{\theta'}(s_{t+1})}_{\text{bootstrapped target}} - v_\theta(s_t)\Big)^2\Big]$$

3.1 Value semi-gradients

core concept: Bootstrapping

- causality: future values do not depend on past values
- separate bootstrapping network $v_{\theta'}$ with parameters θ'

$$\mathcal{L}[\theta] \quad := \quad \mathbb{E}_{\mathcal{D}}\Big[\Big(\underbrace{r_t + \gamma\, v_{\theta'}(s_{t+1})}_{\text{bootstrapped target}} - v_\theta(s_t)\Big)^2\Big]$$

- Semi-gradient TD-learning: $\theta' = \theta$
  - `targets.detach()` prevents gradient propagation
  - learns much faster
  - convergence for linear models

Gordon (1995) showed faster learning; for the convergence proof see Tsitsiklis and Van Roy (1997)

Value semi-gradients

core concept: Bootstrapping

- causality: future values do not depend on past values
- separate bootstrapping network $v_{\theta'}$ with parameters θ'

$$\mathcal{L}[\theta] \quad := \quad \mathbb{E}_{\mathcal{D}}\Big[\Big(\underbrace{r_t + \gamma\, v_{\theta'}(s_{t+1})}_{\text{bootstrapped target}} - v_\theta(s_t)\Big)^2\Big]$$

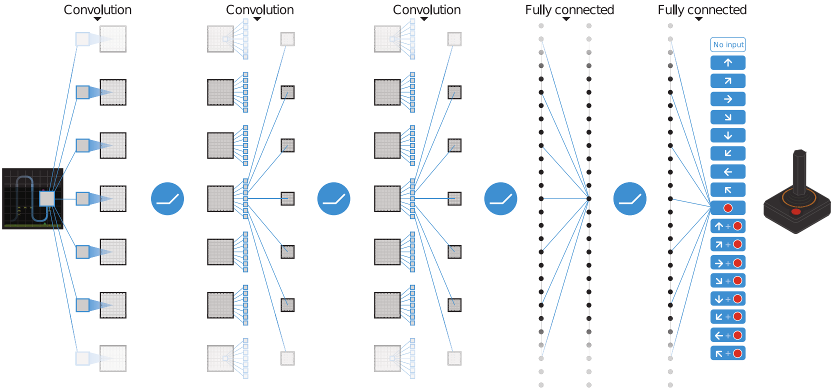
- Semi-gradient TD-learning: $\theta' = \theta$
  - `targets.detach()` prevents gradient propagation
  - learns much faster
  - convergence for linear models
- Neural-fitted Q-iteration (NFQ): update $\theta' \leftarrow \theta$ after convergence
  - first successful deep RL algorithm, but slow iterations

Neural-fitted Q-iteration by Riedmiller (2005)

3.1 Online Q-learning

- Semi-gradient Q-learning using NN with one output per action

$$q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t) \;:=\; \max_\pi \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\Big|\, \begin{matrix}\boldsymbol{s}_t \\ \boldsymbol{a}_t\end{matrix}\Big] \;\equiv\; r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma\, \mathbb{E}\Big[\max_{\boldsymbol{a}'} q_\theta(\boldsymbol{s}_{t+1}, \boldsymbol{a}')\Big]$$



originally by Watkins and Dayan (1992), deep version by Mnih et al. (2013, figure source), see Sutton and Barto (2018) for details

3.1 Online Q-learning

- Semi-gradient Q-learning using NN with one output per action

$$\underbrace{q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t)}_{\text{value}} := \max_\pi \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \Big| \begin{smallmatrix} \boldsymbol{s}_t \\ \boldsymbol{a}_t \end{smallmatrix} \Big] \equiv \underbrace{r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma\, \mathbb{E}\Big[\max_{\boldsymbol{a}'} q_\theta(\boldsymbol{s}_{t+1}, \boldsymbol{a}') \Big]}_{\text{target}}$$

End-to-end regression of Q-value function `q` in `environment`

```python
from torch.optim import RMSprop
from torch.nn.functional import mse_loss
optimizer = RMSprop(q.parameters())
while True: # sample episode from environment
  state, action, reward, term, next = environment.sample(q)
  # compute left and right side of Bellman eq.
  value = q(state).gather(dim=-1, index=action)
  target = reward + gamma * (~term * q(next).max(dim=-1)[0])
  # gradient descent step on supervised regression loss
  optimizer.zero_grad()
  mse_loss(value, target.detach()).backward()
  optimizer.step()
```

originally by Watkins and Dayan (1992), deep version by Mnih et al. (2013, figure source), see Sutton and Barto (2018) for details

3.1 Online Q-learning

- Semi-gradient Q-learning using NN with one output per action

$$\underbrace{q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t)}_{\text{value}} := \max_\pi \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \Big| \begin{matrix}\boldsymbol{s}_t \\ \boldsymbol{a}_t\end{matrix}\Big] \equiv \underbrace{r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma\, \mathbb{E}\Big[\max_{\boldsymbol{a}'} q_\theta(\boldsymbol{s}_{t+1}, \boldsymbol{a}')\Big]}_{\text{target}}$$

End-to-end regression of Q-value function `q` in `environment`

```
from torch.optim import RMSprop
from torch.nn.functional import mse_loss
optimizer = RMSprop(q.parameters())
while True: # sample episode from environment
  state, action, reward, term, next = environment.sample(q)
  # compute left and right side of Bellman eq.
  value = q(state).gather(dim=-1, index=action)
  target = reward + gamma * (~term * q(next).max(dim=-1)[0])
  # gradient descent step on supervised regression loss
  optimizer.zero_grad()
  mse_loss(value, target.detach()).backward()
  optimizer.step()
```

- `term` boolean indicates terminal state. Why do we need this?

originally by Watkins and Dayan (1992), deep version by Mnih et al. (2013, figure source), see Sutton and Barto (2018) for details

3.1 Summary

- Residual value gradients are slow
- We use in practice semi-gradients with `.detach()`
- Bootstrapping varies from semi-gradients to NFQ
- Q-networks have one head per action

Learning Objectives

LO3.1: Derive and implement semi-gradients of a (Q-)value function
LO3.2: Define and implement Q-network architectures

3.2

Deep Q-Learning

Stabilization Techniques

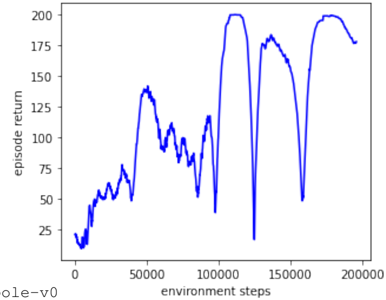
3.2 Catastrophic forgetting

- Online learning violates ML assumptions
  - regression targets (bootstrapping) not stationary
  - greedy policy changes training distribution
  - networks “forget” old samples
  - transitions are not i.i.d.

modified from image source: wikipedia.org

3.2 Catastrophic forgetting

- Online learning violates ML assumptions
  - regression targets (bootstrapping) not stationary
  - greedy policy changes training distribution
  - networks “forget” old samples
  - transitions are not i.i.d.
- Naive online Q-learning is not stable!



experiment on `Cartpole-v0`


modified from image source: wikipedia.org

3.2 On- and Off-policy sampling

- No training/test set split in RL
  - all data is sampled “live” from envionment
  - instead: “who” sampled episodes?

3.2 On- and Off-policy sampling

- No training/test set split in RL
  - all data is sampled “live” from envionment
  - instead: “who” sampled episodes?

- On-policy sampling evaluates the sampled policy π
  - \+ relatively stable and low-dimensional
  - \- must resample once policy changes

$$V^\pi(s_t) = \mathbb{E}\Big[r_t + \gamma V^\pi(s_{t+1}) \,\Big|\, \substack{a \sim \pi(\cdot|s_t) \\ r_t = r(s_t, a),\; s_{t+1} \sim P(\cdot|s_t, a)}\Big]$$

3.2 On- and Off-policy sampling

- No training/test set split in RL
  - all data is sampled "live" from envionment
  - instead: "who" sampled episodes?

- On-policy sampling evaluates the sampled policy π
  - \+ relatively stable and low-dimensional
  - \- must resample once policy changes

$$V^\pi(s_t) = \mathbb{E}\Big[r_t + \gamma V^\pi(s_{t+1}) \,\Big|\, \substack{a \sim \pi(\cdot|s_t) \\ r_t = r(s_t, a),\; s_{t+1} \sim P(\cdot|s_t, a)} \Big]$$

core concept: Off-policy sampling evaluates *any* policy π

- \- larger input space (actions)
- \+ can reuse old/other's experiences

$$Q^\pi(s_t, a_t) = \mathbb{E}\Big[r_t + \gamma Q^\pi(s_{t+1}, a') \,\Big|\, \substack{r_t = r(s_t, a_t),\; s_{t+1} \sim P(\cdot|s_t, a_t) \\ a' \sim \pi(\cdot|s_{t+1})} \Big]$$

3.2 Experience replay

- How can we exploit that Q-learning is **off-policy**?

image source: wikipedia.org

(3.2) Experience replay

- How can we exploit that Q-learning is **off-policy**?

core concept:
Experience replay buffers

- remember the last n transitions
- mini-batches of i.i.d. transitions
- always include last episode

experience replay buffers introduced by Lin (1992)

3.2 Experience replay

- How can we exploit that Q-learning is **off-policy**?

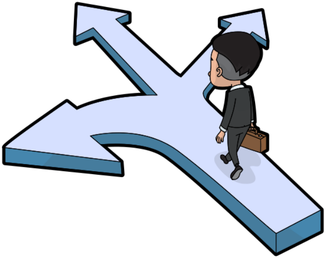
core concept:
Experience replay buffers

- remember the last n transitions
- mini-batches of i.i.d. transitions
- always include last episode

- Prioritized experience replay buffers
  - choose transitions with high errors more often
  - more susceptible to catastrophic forgetting

basic idea called *prioritized sweeping* (Moore and Atkeson, 1993), prioritized replay introduced by (Schaul et al., 2015)

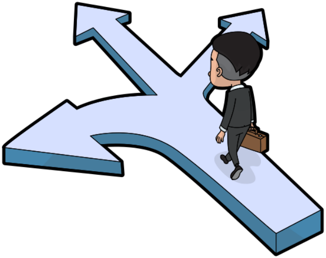
- Recap: ϵ-greedy exploration policy
  - act randomly with probability ϵ
  - act greedily with probability $1 - \epsilon$
  - linearly decay ϵ over n_ϵ steps



more on exploration in Lecture 5, image source: www.wikipedia.org

3.2 Exploration and neural networks

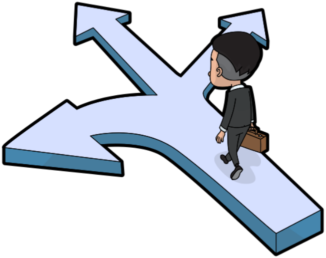
- Recap: ϵ-greedy exploration policy
  - act randomly with probability ϵ
  - act greedily with probability $1 - \epsilon$
  - linearly decay ϵ over n_ϵ steps
- Greedy policy $\rightarrow$ catastrophic forgetting
  - buffer fills with the same state-actions
  - other actions "forget" their value
  - maximum can select "drifting" actions



more on exploration in Lecture 5, image source: www.wikipedia.org

Exploration and neural networks

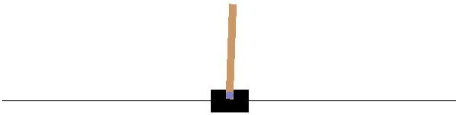
- Recap: ϵ-greedy exploration policy
  - act randomly with probability ϵ
  - act greedily with probability $1 - \epsilon$
  - linearly decay ϵ over n_ϵ steps
- Greedy policy $\rightarrow$ catastrophic forgetting
  - buffer fills with the same state-actions
  - other actions “forget” their value
  - maximum can select “drifting” actions
- Never stop exploring!
  - stop decay at some ϵ_{min}
  - measure greedy policy in test episodes

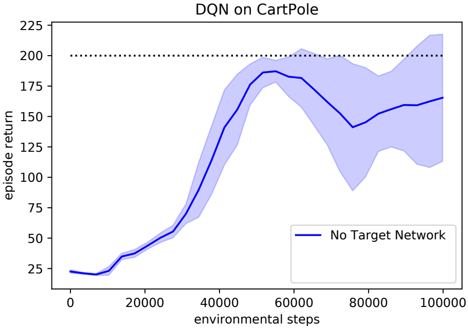


more on exploration in Lecture 5, image source: www.wikipedia.org

3.2 Target networks

- Semi-gradient Q-learning
  - with replay buffer
  - update changes targets
  - unstable learning
  - example `Cartpole-v0`



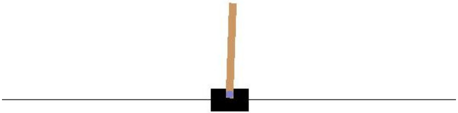


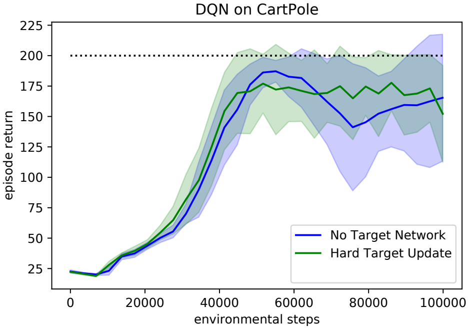
assignment sheet 2

mean and standard deviation over 10 histogram-smoothed seeds

Target networks

- Semi-gradient Q-learning
  - with replay buffer
  - update changes targets
  - unstable learning
  - example `Cartpole-v0`
- Hard target update
  - use "old" values as targets
  - every n steps: $\theta' \leftarrow \theta$
  - here $n = 10$



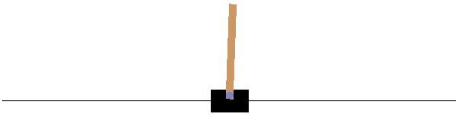


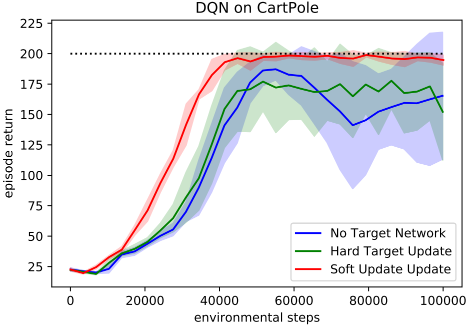
assignment sheet 2

hard target updates first in Mnih et al. (2013)

3.2 Target networks

- Semi-gradient Q-learning
  - with replay buffer
  - update changes targets
  - unstable learning
  - example `Cartpole-v0`





- Hard target update
  - use “old” values as targets
  - every n steps: $\theta' \leftarrow \theta$
  - here $n = 10$

- Soft target update
  - $\theta' \leftarrow (1-\eta)\theta' + \eta\theta$
  - here $\eta = 0.1$

assignment sheet 2

soft target updates first in Lillicrap et al. (2016), hard target updates first in Mnih et al. (2013)

3.2 Summary

- Values can only be estimated on-policy, Q-values off-policy
- Naive online deep Q-leaning is *not stable*
- Catastrophic forgetting requires regular visitations
- Stabilizes with replay buffers and target networks

Learning Objectives

LO3.3: Explain the difference between on-policy and off-policy sampling
LO3.4: Explain why experience replay buffers and target networks stabilize

3.3

Deep Q-Learning

Deep Q-Networks

core concept: Deep Q-networks (DQN)

- Denotes a *family* of online deep Q-learning algorithms
  - originally hard target update, 1 update/step, visual input
- Samples either an episode or n steps between m updates

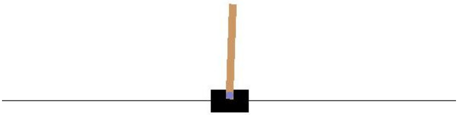
```python
# Using mini-'batch' of transitions from replay buffer
batch = self.replay_buffer.sample()
targets = batch['rewards'] + self.gamma * (~batch['terminals'] \
            * self.target_q(batch['next_states']).max(dim=-1)[0])
values = q(batch['states']).gather(dim=-1, index=batch['actions'])
# Backpropagate loss
self.optimizer.zero_grad()
mse_loss(values, targets.detach()).backward()
self.optimizer.step()
# Update target network (hard or soft)
self.target_model_update()
```

assignment sheet 2

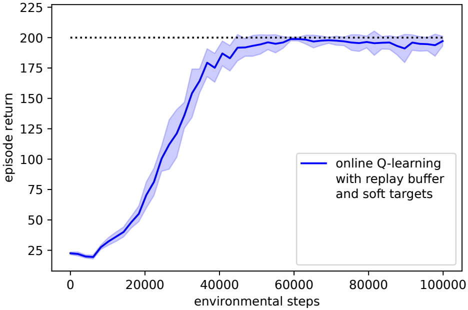
first working DQN variant by Mnih et al. (2013, 2015)

Update frequency

- Sample efficiency?
  - example `Cartpole-v0`
  - constant exploration $\epsilon = 0.1$



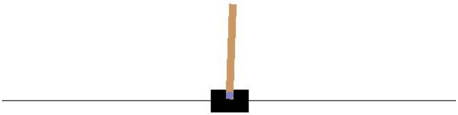
1. Sampling entire episodes
   - 1 update/epsiode



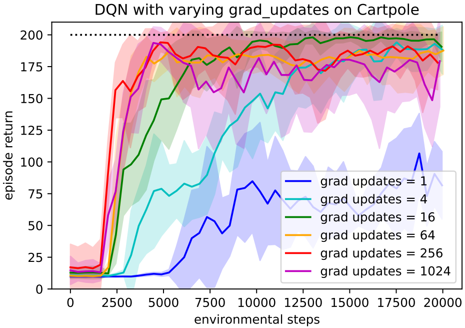
mean and standard deviation over 10 histogram-smoothed seeds

3.3 Update frequency

- Sample efficiency?
  - example `Cartpole-v0`
  - constant exploration $\epsilon = 0.1$



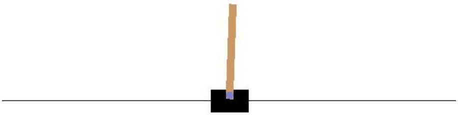
1. Sampling entire episodes
   - n updates/episode
   - faster learning for $n \leq 256$
   - less stable for $n > 16$



mean and standard deviation over 10 histogram-smoothed seeds

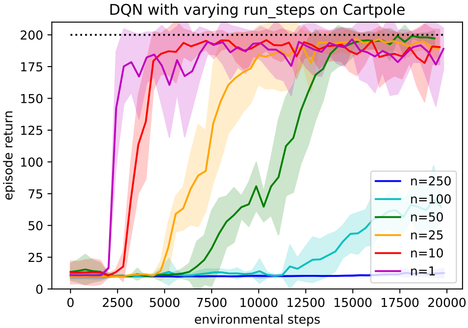
3.3 Update frequency

- Sample efficiency?
  - example `Cartpole-v0`
  - constant exploration $\epsilon = 0.1$



1. Sampling entire episodes
   - n updates/episode
   - faster learning for $n \leq 256$
   - less stable for $n > 16$

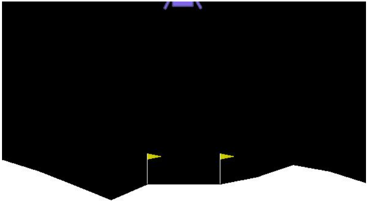
2. Sampling n steps/update
   - old episodes continued
   - faster learning
   - less stable for $n < 25$



mean and standard deviation over 10 histogram-smoothed seeds

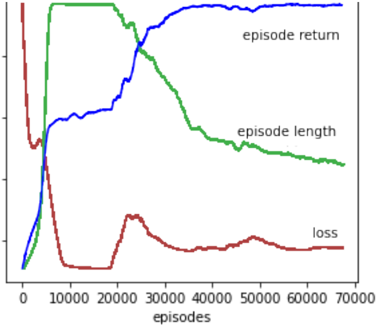
3.3 DQN example: Lunar Lander

- Lunar lander on the moon
  - state: pos., vel. and rot.
  - actions exhaust costly fuel
  - reward for landing between flags
  - punishment for crashing



- 3 phases of learning
  1. random initialization
  2. learned to hover
  3. fine-tune landing

- Loss often increases
  - whenever sampling changes
  - e.g. phase transitions



assignment sheet 2

3.3 Debugging recommendations

- **failure:** policy does not pick up
  - raise learning rate α of gradient descend
  - raise γ to consider far future rewards
  - transform network inputs to zero-mean and unit variance
  - transform reward to be in $[-1, 1] \subset \mathbb{R}$ (be careful!)
  - increase exploration time
  - increase episode length (if possible)
  - decrease **or** increase number of layers/neurons

- **instability:** policy unlearns after a while
  - lower learning rate α of gradient descent
  - lower γ to reduce error propagation
  - slow target network adaptation
  - increase mini-batch size
  - increase replay buffer size
  - increase final exploration ϵ

3.3 Summary

- DQN stabilizes with replay buffers and target networks
- Update frequency crucial for efficiency/stability
- Learning happens in phases, indicated by increasing losses
- Choosing working hyper-parameters requires a lot of experience

Learning Objectives

LO3.5: Implement and evaluate DQN with replay buffer and target network

3.3 Next lecture

- Next lecture: **why do we call it *deep learning*?**
- This Thursday is **tutorial**
  - submit **assignment 1**!
  - until start of tutorial
  - answer can be incorrect
- Questions? Ask them here: `answers.ewi.tudelft.nl`

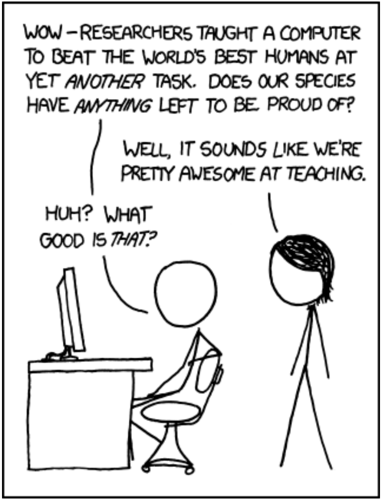


image source: `xkcd.com`

References I

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. *Machine Learning*, 1995.

Geoffrey J. Gordon. Stable function approximation in dynamic programming. In *Proc. 12th International Conference on Machine Learning*, pages 261–268, 1995.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016. URL http://arxiv.org/abs/1509.02971.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3): 293–321, 1992.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602. NIPS Deep Learning Workshop 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

A.W. Moore and C.G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

Martin Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In *Machine Learning: ECML 2005*, pages 317–328. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-31692-3.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2015. URL http://arxiv.org/abs/1511.05952.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

John N. Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in Neural Information Processing Systems*, 1997.

Christopher Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.