HPI Hasso Plattner Institut
Digital Engineering · Universität Potsdam

Dynamic Programming and Reinforcement Learning

Week 5b: Deep Q-Networks

Rainer Schlosser und Alexander Kastius

Enterprise Platform and Integration Concepts

19.05.22

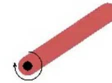
Yet Unsolved Issues?

HPI Hasso Plattner Institut

Still unsolved:

- State Space Complexity
  - Many Dimensions
  - Continuous Values
- Current methods require discretization and become intractable at some point

- Continuous Control
  - Action Space might consist of continuous values as well
  - Can be discretized sometimes, which prevents us from finding the actual optimal policy



Deep Q-Networks

Given: Network Q with params ϕ, a learning rate η_t, an optimizer which can minimize the measured error.

1. Observe s_t, choose a_t according to the current policy
2. Observe r_t, s_{t+1}
3. Update the Q-value estimate:

$$\phi \leftarrow -\eta_t \nabla_\phi \left(\boldsymbol{r_t} + \boldsymbol{\gamma} \max_{\boldsymbol{a} \in \boldsymbol{A}} \boldsymbol{Q}(\boldsymbol{s_{t+1}}, \boldsymbol{a}; \boldsymbol{\phi}) - \boldsymbol{Q}(\boldsymbol{s_t}, \boldsymbol{a_t}; \boldsymbol{\phi}) \right)^{\boldsymbol{2}} + \phi$$

4. Repeat from 1. with each new transition

SARSA and Value Estimation

Hasso Plattner Institut

SARSA with Gradients

Given: Network Q with params ϕ, a learning rate η_t, an optimizer which can minimize the measured error.

1. Observe s_t, choose a_t according to the current policy
2. Observe r_t, s_{t+1}, choose a_{t+1}
3. Update the Q-value estimate:
$$\phi \leftarrow -\eta_t \nabla_\phi \left(r_t + \gamma \boldsymbol{Q}(\boldsymbol{s_{t+1}}, \boldsymbol{a_{t+1}}; \boldsymbol{\phi}) - Q(s_t, a_t; \phi)\right)^2 + \phi$$
4. Repeat from 1. with each new transition

Value Estimation with Gradients

Given: Network V with params ϕ, a learning rate η_t, an optimizer which can minimize the measured error.

1. Observe s_t, choose a_t according to the policy
2. Observe r_t, s_{t+1}
3. Update the value estimate:
$$\phi \leftarrow -\eta_t \nabla_\phi \left(r_t + \gamma \boldsymbol{V}(\boldsymbol{s_{t+1}}; \boldsymbol{\phi}) - V(s_t; \phi)\right)^2 + \phi$$
4. Repeat from 1. with each new transition

Dis-/Advantages of Gradient Based Methods

HPI Hasso Plattner Institut

Advantages

- The network allows high dimensional inputs. We can efficiently work on states with many dimensions and compute values for them.
- As we are adjusting the estimate for other states as well implicitly, we make use of the networks ability to generalize. A similar, but yet unseen state hopefully leads to a similar estimate.
- This is highly flexible with regard to the network setup.

Disadvantages

- We loose all convergence guarantees. As the estimate is not only changed for the observed state, the estimate for other states might actually become worse than it was before.
- We introduce many more hyperparameters, with poor choices resulting in poor performance.
- The whole system turns into a blackbox. Not only the process is hard to predict, now the agent is hard to predict as well.

Network Implementation

Hasso Plattner Institut

How does the actual network look like?

2 Options:

Action a is an input and the network outputs the value for the given combination of s, a.

The input consists of only s and the network outputs the values of all actions a as vector.

Differences?

Network Implementation

How does the actual network look like?

2 Options:

Action a is an input and the network outputs the value for the given combination of s, a.

Requires one forward-pass for each action, makes max. op expensive.

Adjusting the value of the action influences every single parameter.

The input consists of only s and the network outputs the values of all actions a as vector.

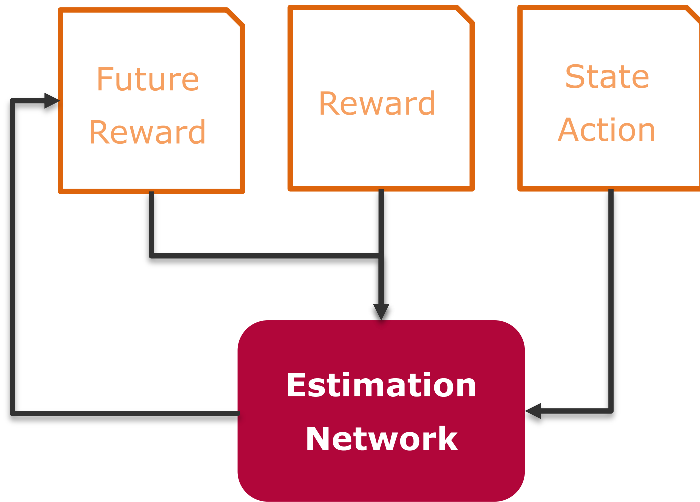
Only a single forward pass for maximization operation necessary.

Adjusting one action influences the other actions, but only one set of weights in the last layer is changed.

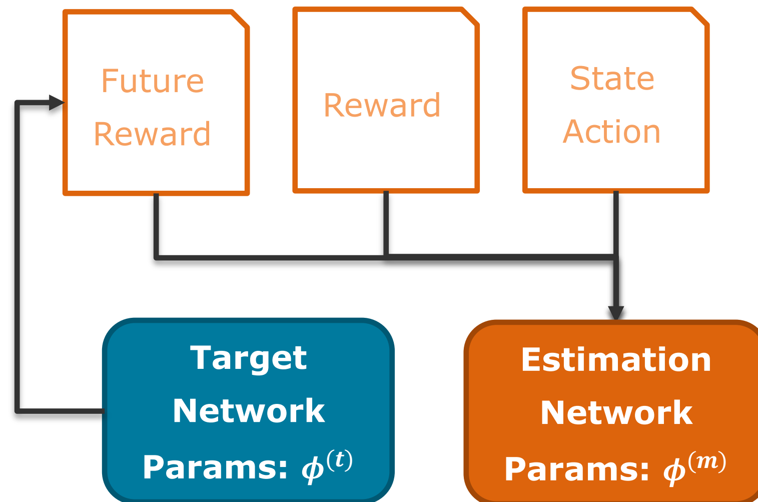
Double Deep Q-Networks

Hasso Plattner Institut

Old architecture



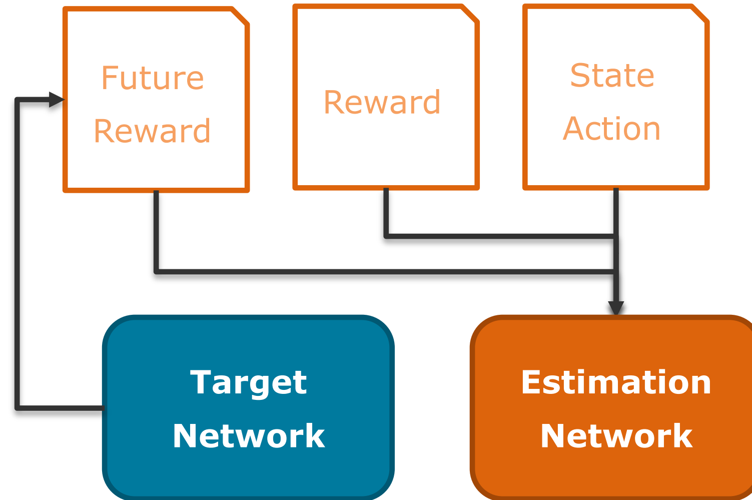
New architecture



$$\phi^{(m)} \leftarrow -\eta \nabla_{\phi^{(m)}} \left(\left(r_t + \gamma Q\left(s_{t+1}, \operatorname*{argmax}_{a' \in A} Q\left(s_{t+1}, a'; \phi^{(m)}\right); \phi^{(t)} \right) \right) - Q\left(s_t, a_t; \phi^{(m)}\right) \right)^2 + \phi^{(m)}$$

Double Deep Q-Networks

- Update occurs by only computing the target value by using the target network
- Target network is not updated immediately
- Instead, either full or moving average updates are applied every few steps
- $\phi_t = w\phi_m + (1 - w)\phi_t,\ w \in (0, 1)$



Experience Replay

- Data Usage is still inefficient.
- Network estimation shows the disadvantage of possible overfitting to very recent tuples.
- Computing gradients for a single tuple is inefficient as well.
- Idea: **Introduce replay buffer**
- Every few tuples, take last collected tuples and some random samples from the replay buffer to learn on a whole batch.

Replay Buffer

s_t	a_t	r_t	s_{t+1}
s_1	a_1	r_1	s_2
s_2	a_2	r_2	s_3
s_3	a_3	r_3	s_4
s_4	a_4	r_4	s_5
…	…	…	…

Prioritized Experience Replay

HPI Hasso Plattner Institut

- Idea: There are tuples rarely seen in processes and thus rarely used for training. In many cases, this leads to poor estimations on those combinations.
- Solution: **Prioritize those tuples in the selection process for the training op.**
- The TD-error does provide us with a measure of how wrong an estimate is.
- We have to store this either on collection or at an update.
- Priority p_t = The TD-error measured last time the tuple was used.

Replay Buffer

s_t	a_t	r_t	s_{t+1}	p_t
s_1	a_1	r_1	s_2	td_1
s_2	a_2	r_2	s_3	td_2
s_3	a_3	r_3	s_4	td_3
s_4	a_4	r_4	s_5	td_4
…	…	…	…	…

Prioritized Experience Replay – Sampling

- Problem: Greedily selecting by the error causes the same tuples to be selected over and over again until their error is diminished. This can cause overfitting.
- Solution: We randomly sample the tuples. The priority of being chosen is relative to the measured error (j is the index of a tuple in the store, N the number of tuples stored, p_j is the priority mentioned aforehand, α determines the shape of the sampling probabilities):

$$P(j) = \frac{p_j^\alpha}{\sum_{k=1}^{N} p_k^\alpha}$$

- As our buffer contains all p_k we can simply sample from it by computing a number between 0 and 1 and then searching for the element at the corresponding relative position in the buffer.

Prioritized Experience Replay – Reweighting

HPI Hasso Plattner Institut

- Problem: Changing the data distribution introduces a bias to the weight updates.
- Solution: Adjust the TD-error by an importance sampling factor (j is the index of a tuple in the store, N the number of tuples stored, $\beta \in [0,1]$):

$$w_j = \left(\frac{1}{N} * \frac{1}{P(j)}\right)^{\beta}$$

β is an annealing factor, that moves towards 1 during the course of learning. This is possible, as a small bias is not that important in the beginning of the learning process.

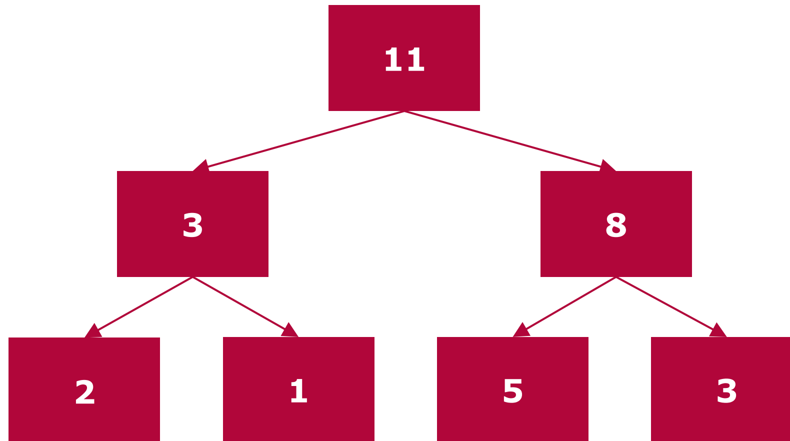
How to apply the weight updates?

As done for other examples of importance sampling: Multiply the TD-error with it before computing updates.

$$\phi \leftarrow -\eta_t w \nabla_\phi \left(\boldsymbol{r_t} + \boldsymbol{\gamma} \max_{\boldsymbol{a \in A}} \boldsymbol{Q}(\boldsymbol{s_{t+1}}, \boldsymbol{a}; \boldsymbol{\phi}) - \boldsymbol{Q}(\boldsymbol{s_t}, \boldsymbol{a_t}; \boldsymbol{\phi})\right)^2 + \phi$$

Prioritized Experience Replay – Data Structure

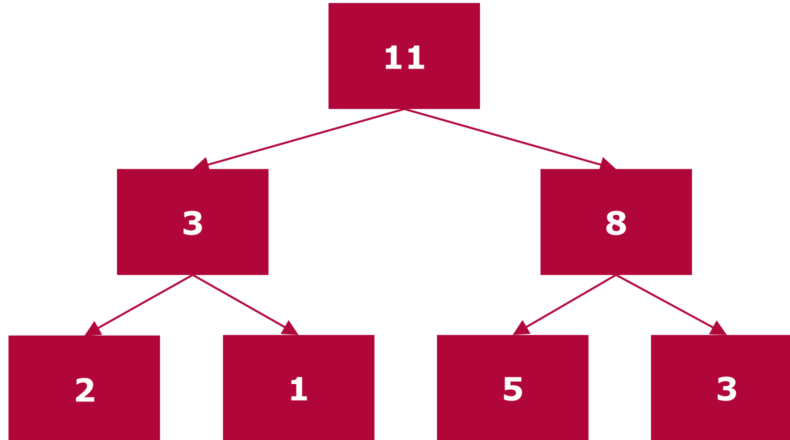
Now that we‵ve got everything together a last problem arises: We need a data structure for the buffer that performs two tasks: Efficiently storing the TD-errors and allowing efficient sampling from it. Solution: **Sum Tree**



Searching for something at position n: Check left-hand element, if it is larger than the searched value, go left, otherwise subtract left and go right.

Prioritized Experience Replay – Data Structure

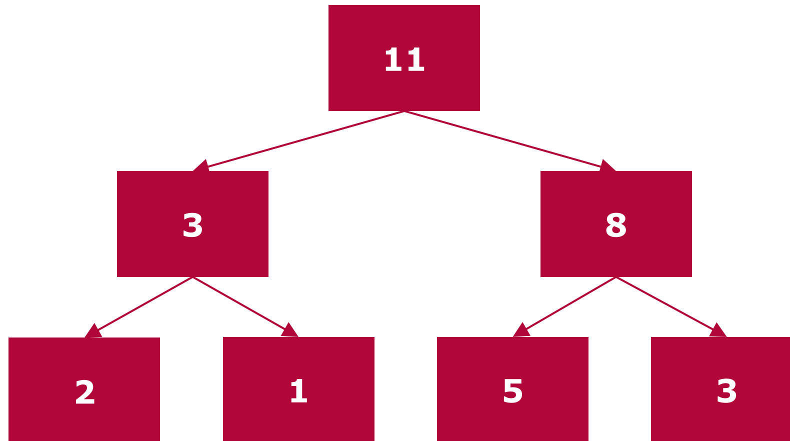
Now we‛ve got everything together a last problem arises: We need a data structure for the buffer that performs two tasks: Efficiently storing the TD-errors and allowing efficient sampling from it. Solution: **Sum Tree**



Replacing an element: Replace value at the leaf, iterate tree upwards and recompute sums.

Prioritized Experience Replay – Data Structure

Now we‘ve got everything together a last problem arises: We need a data structure for the buffer that performs two tasks: Efficiently storing the TD-errors and allowing efficient sampling from it. Solution: **Sum Tree**



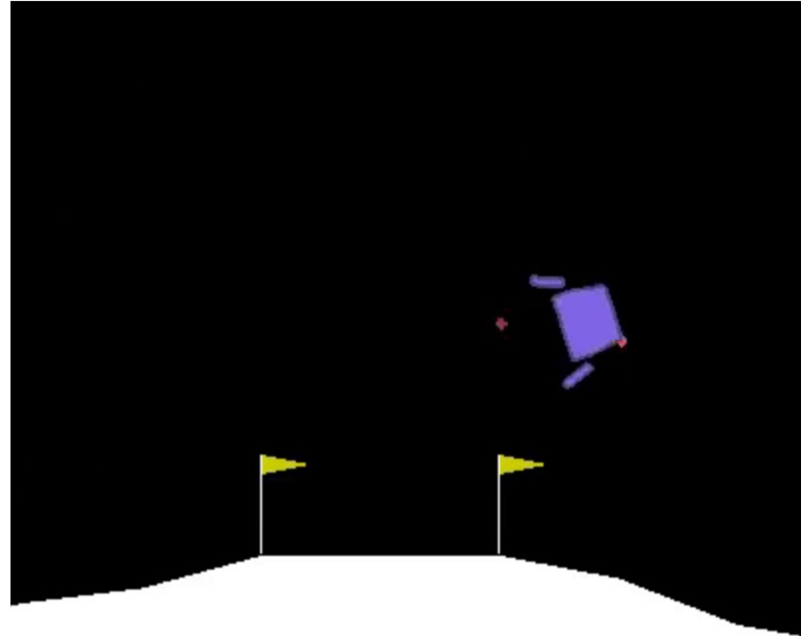
Implementation: The whole tree can be stored in an array of fixed length if the number of proposed elements is known. Pointer based implementation is possible as well.

Assignment: Bringing it all together.

Hasso Plattner Institut

Assignment, work in groups of 3.

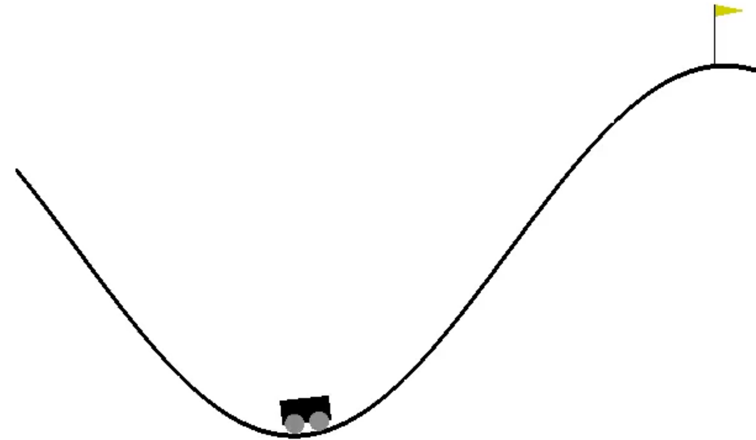
1. Implement Deep Q-Networks in its most basic version using the ML library of your choice. If any implementation issues arise, don’t hesitate to contact us.
2. Add Double Learning, Dueling Networks and Prioritized Experience Replay.
3. Measure the difference in performance, for example on the LunarLander-v2 environment provided by the Open AI gym: https://www.gymlibrary.ml/environments/box2d/lunar_lander/



OpenAI Gym Example

Every Gym has a standardized interface that you can use.

- The .reset() methods returns it to its initial state.
- The .step() method takes an action choice, a vector of length 1 that contains either a 0 or a 1 in the case of the MountainCar environment.
- The return value of .step() consists, similarly to the environments provided by us, 4 values: The new state, the reward, a variable that indicates the end of an episode (we reached a sink state) and a dictionary full of debug variables.
- The observation_space and action_space variables indicate the dimensionality of the state and the action space.



Schedule

Week	Dates	Topic
1	April 21	Introduction
2	April 25/28	Finite + Infinite Time MDPs
3	May 2/5	Approximate Dynamic Programming (ADP) + DP Exercise
4	May 12	Q-Learning (QL) (not Mon May 9)
5	May 16/19	**Q-Learning Extensions and Deep Q-Networks**
6	May 23	DQN Extensions (not Thu May 26 “Himmelfahrt”)
7	May 30/June 2	Policy Gradient Algorithms
8	June 9	Project Assignments (not Mon June 6 “Pfingstmontag”)
…		

Dueling Networks - Advantage

Hasso Plattner Institut

What is the meaning of

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

?

$A(s_t, a_t)$ is called the advantage.

The maximum of $Q(s_t, a_t)$ is always the same as the maximum of the corresponding $A(s_t, a_t)$.

Idea: Use this information to learn the advantages explicitly, and only learn the actual Q-values because we are required to do so.

A specialized network architecture can do this task for us.

Dueling Networks - Normalization

Hasso Plattner Institut

When using dueling networks the Q-value is computed according to:

$$Q(s_t, a_t) = A(s_t, a_t) - \max_{a \in \mathbf{A}} A(s_t, a) + V(s_t)$$

Notice: A Advantage $\mathbf{A}$ Action set

We do this according to our off-policy approach. If we want to learn the optimal policy, it is reasonable to assume that $Q(s_t, a_t) = V(s_t)$ for the optimal action.

Furthermore, this solves the problem of identifiability. If we update a value, we need to change the gradients either in A, if the advantage was estimated poorly or in V if the value was off.

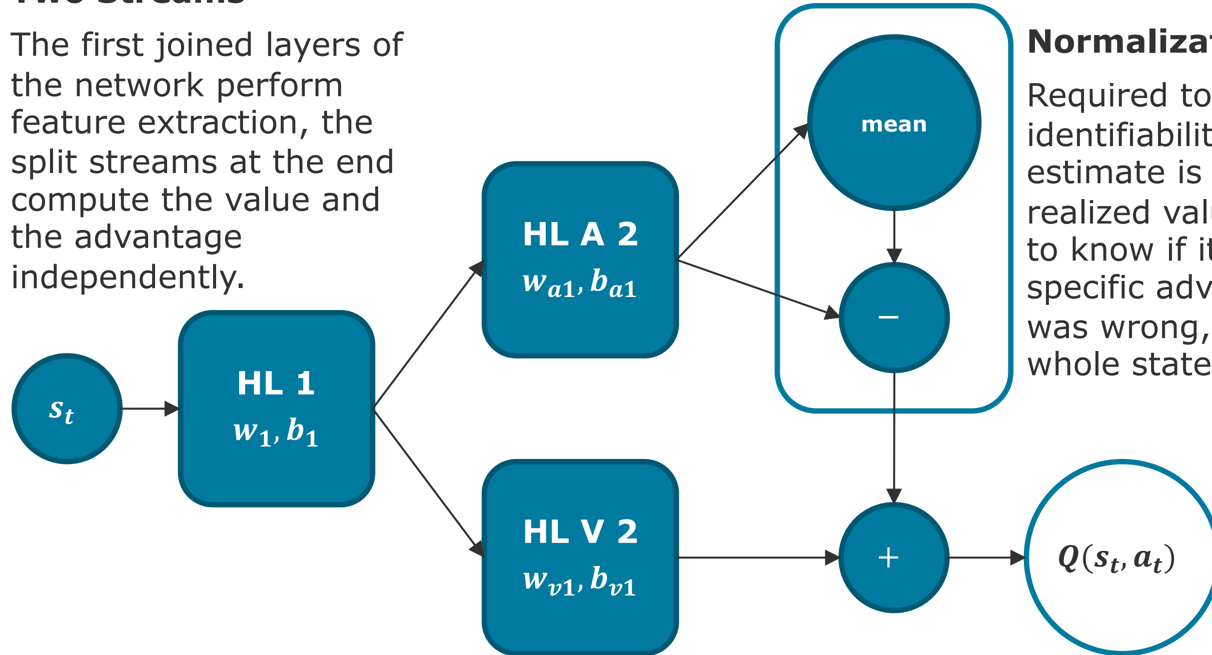
A more stable version of the same idea computes Q according to:

$$Q(s_t, a_t) = A(s_t, a_t) - \frac{1}{|\mathbf{A}|} \sum_{i=1}^{|\mathbf{A}|} A(s_t, a_i) + V(s_t)$$

Dueling Networks - Implementation

Two Streams

The first joined layers of the network perform feature extraction, the split streams at the end compute the value and the advantage independently.



Normalization

Required to ensure identifiability. If an estimate is off the realized value, we need to know if it was only the specific advantage which was wrong, or if the the whole state is misvalued.