Technische Hochschule Rosenheim

Modul - Unsupervised and Reinforcement Learning (URL)

Bachelor Programme AAI

10 - Q-Learning

Prof. Dr. Marcel Tilly

Faculty of Computer Science, Cloud Computing

© Technische Hochschule Rosenheim

Agenda

Technische Hochschule Rosenheim

> Code can be found in
>
> - Q-Learning.ipynb
> - DQN.ipynb
> - or on GitHub or hosted on myBinder

On the menu for today:

- Temporal Difference Learning
  - SARSA
  - Q-Learning
- DQN

The Story So Far: MDPs and RL

Technische Hochschule Rosenheim

Known MDP: Offline Solution

Goal	Technique
Compute V*, Q*, π*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

Goal	Technique
Compute V*, Q*, π*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a fixed policy π	Value Learning

Model-Based Learning

Technische Hochschule Rosenheim

Goal

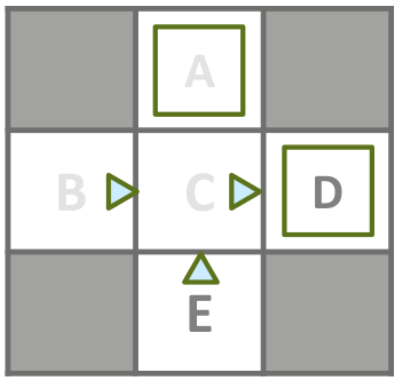
- Learn an approximate model based on experiences
- Solve for values as if the learned model were correct

- Step 1: Learn empirical MDP model
  - Count outcomes s' for each s, a
  - Normalize to give an estimate of P
  - Discover each when we experience (s, a, s')
- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

Example: Model-Based Learning

Technische Hochschule Rosenheim

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$\hat{T}(s, a, s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
...

$\hat{R}(s, a, s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
...

Direct Evaluation

Technische Hochschule Rosenheim

Goal

- Compute values for each state under π

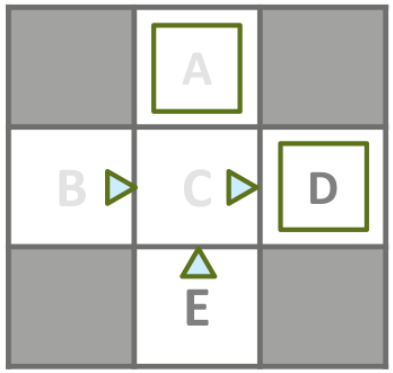
- Idea: Average together observed sample values
  - Act according to π
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples
- This is called **direct evaluation**

Example: n-armed bandit

Example: Direct Evaluation

Technische Hochschule Rosenheim

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

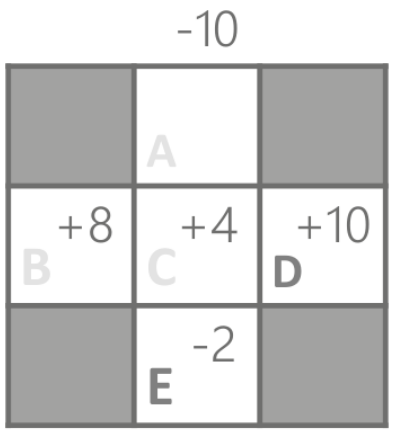
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

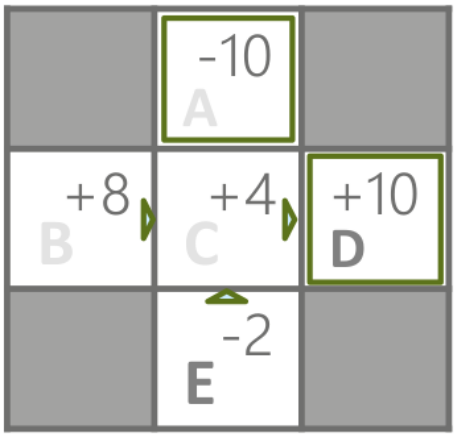
Output Values



Problems with Direct Evaluation

- What's good about *direct evaluation*?
  - It's easy to understand
  - It doesn't require any knowledge of P,R
  - It eventually computes the correct average values, using just sample transitions
- What's bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

Output Values



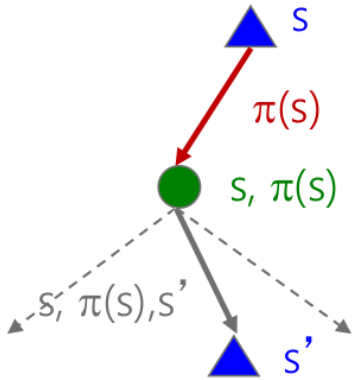
If B and E both go to C under this policy, how can their values be different?

Why Not Use Policy Evaluation?

Technische Hochschule Rosenheim

- Simplified Bellman updates calculate V for a fixed policy:
  - Each round, replace V with a one-step-look-ahead layer over V
- This approach fully exploited the connections between the states
- Unfortunately, we need P and R to do it!

- **Key question**: How can we do this update to V without knowing P and R?
- In other words, how do we take a weighted average without knowing the weights



Temporal Difference Learning

© Technische Hochschule Rosenheim

Temporal Difference Learning

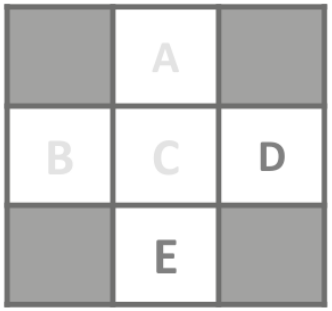
Technische Hochschule Rosenheim

- Temporal difference learning (TD learning) is an agent learning from an environment through episodes with no prior knowledge of the environment.
- Compared to Dynamic Programming, TD does not need to wait until the end of the episode to make an estimate of the value function.
- At time t + 1, TD methods immediately form a target $R_{t+1} + \gamma V(S_{t+1})$ and make a useful update with step size α using the observed reward R_{t+1} and the estimate $V(S_{t+1})$

Temporal Difference Learning

Technische Hochschule Rosenheim

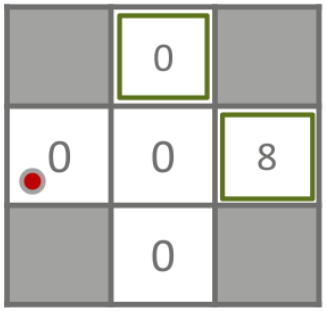
States



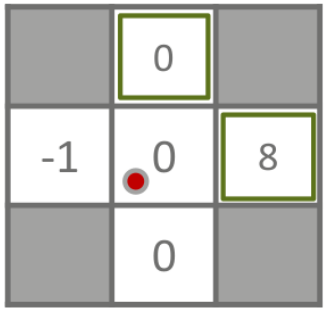
Assume: $\gamma = 1$, $\alpha = 1/2$

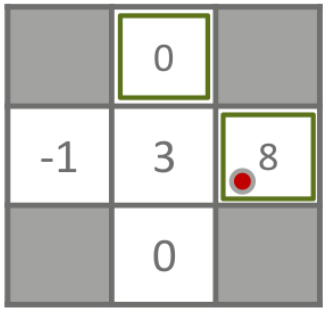
Observed Transitions

B, east, C, -2



C, east, D, -2





$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

Temporal Difference Learning

Technische Hochschule Rosenheim

- TD approximates the current estimate based on the previously learned estimate (bootstrapping)

$$V(s) = V(s) + \alpha(R + \gamma V(s') - V(s))$$

> The *new* value of a previous state = *old* value of the previous state + learning_rate(reward + discount(value of current state) – value of previous state)

- The difference between the actual reward $(R + \gamma V(s'))$ and the expected reward $V(s) = R + \gamma V(s') - V(s)$ is called the TD error

TD Learning

Technische Hochschule Rosenheim

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

SARSA – On-policy TD Control

Technische Hochschule Rosenheim

- SARSA = State-Action-Reward-State-Action
- Learn an action-value function instead of a state-value function
- Q is the action-value function for policy π
- Q-values are the values $Q_\pi(s, a)$ for s in S, a in A
- SARSA experiences are used to update Q-values
- Use TD methods for the prediction problem

SARSA Update Rule

Technische Hochschule Rosenheim

- We want to estimate $Q_\pi(s, a)$ for the current policy π, and for all states s and action a
- The update rule is similar to that for TD(0) but we transition from state-action pair to state-action pair, and learn the values of state-action pairs
- The update is performed after every transition from a non-terminal state s_t
- If S_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1})$ is zero
- The update rule uses $(s_t, a_t, R_{t+1}, s_{t+1}, a_{t+1})$ = SARSA

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma(s_{t+1}, a_a t + 1) - Q(s_t, a_t))$$

SARSA Algorithm

Technische Hochschule Rosenheim

Sarsa: An on-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

SARSA takes into account the current exploration policy which, for example, may be greedy with random steps.

Q-Learning

Technische Hochschule Rosenheim

- Q-States

$$Q(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a V(s')$$

- This an equation to quantify the quality of a particular action
- We can also say that *V(s) is the maximum of all the possible values of Q(s, a)*.

$$Q(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a max_{a'} Q(s',a')$$

- Temporal Difference is the component that will help the robot to calculate the Q-values with respect to the changes in the environment over time.

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha TD(a,s)$$

with α is the *learning rate* which controls how quickly the robot adopts to the random changes imposed by the environment

Q-Learning

Technische Hochschule Rosenheim

- But what is TD?

$$TD(s,a) = R_s^a + \gamma max_{a'} Q(s',a') - Q(s,a)$$

- This results in

$$Q_t(s,a) = (1-\alpha)Q_{t-1}(s,a) + \alpha(R_s^a + \gamma max_{a'} Q(s',a'))$$

- Similar to SARSA but off-policy updates
- The learned action-value function Q directly approximates the optimal action-value function Q* independent of the policy being followed
- In update rule, choose action a that maximises Q given S_{t+1} and use the resulting Q-value (i.e. estimated value given by optimal action-value function) plus the observed reward as the target

Q-Learning Algorithm

Technische Hochschule Rosenheim

Tabular TD(0) for estimating v_π

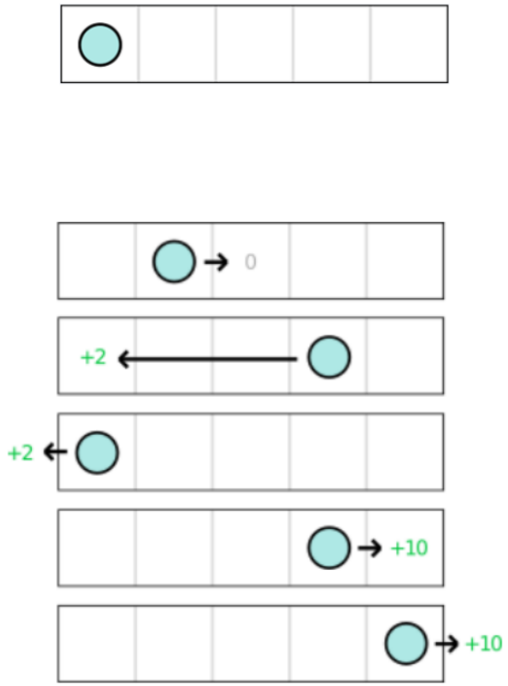
Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

This method is off-policy because we do not have a fixed policy that maps from states to actions. This is why a_{t+1} is not used in the update rule.

Q-Learning Examples

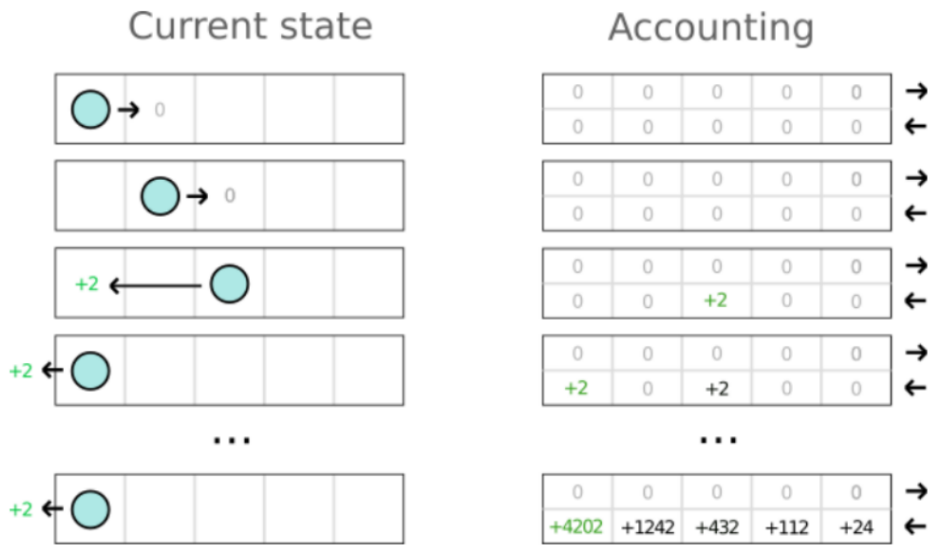
A person in dark dungeon

- What the person knows:
  - The dungeon is 5 tiles long
  - The possible actions are FORWARD and BACKWARD
  - FORWARD is always 1 step, except on last tile it bumps into a wall
  - BACKWARD always takes you back to the start
  - Sometimes there is a wind that flips your action to the opposite direction
- What the person doesn't know:
  - Entering the last tile gives you +10
  - Entering the first tile gives you +2
  - Other tiles have no reward



Naïve Strategy

- The person is going to follow a safe (but naive) strategy:
  - Always choosing the most lucrative action based on your accounting
  - If the accounting shows zero for all options, choose a random action



Naïve Strategy Result

Technische Hochschule Rosenheim

- The person seems to always prefer going BACKWARD even though we know that the optimal strategy is probably always going FORWARD. The best rewards (+10) are at the end of the dungeon after all. Because there is a random element that sometimes flips our action to the opposite, the person actually sometimes reaches the other end unwillingly, but based on the spreadsheet is still hesitant to choose FORWARD.

WHY?

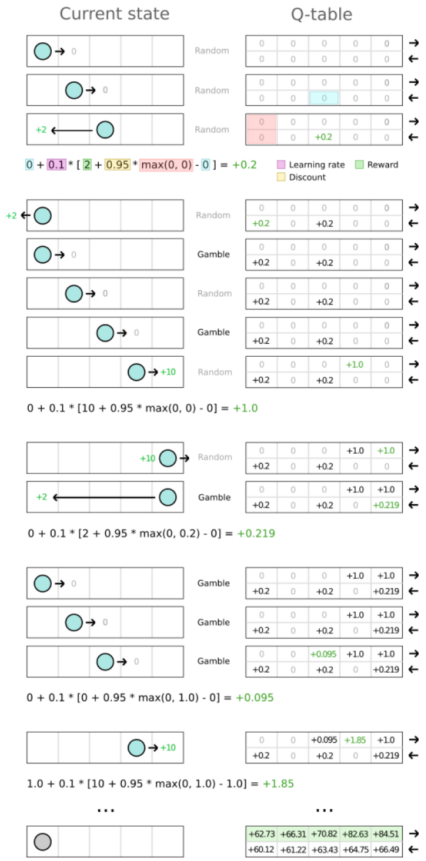
- The agent has chosen a very greedy strategy. During the very short initial randomness, the person might willingly choose to go FORWARD once or twice, but as soon as he chooses BACKWARD once, he is doomed to choose that forever. Playing this dungeon requires long term planning and declining smaller immediate awards to reap the bigger ones later on.

Q-Learning at Work

Technische Hochschule Rosenheim

Our strategy:

- Choose the most lucrative action from our spreadsheet by default
- Sometimes gamble and choose a random action
- If the accounting shows zero for all options, choose a random action
- Start with 100% gambling (exploration), move slowly toward 0%
  - Use discount = 0.95
  - Use learning_rate = 0.1



Q-Learning vs. SARSA

Technische Hochschule Rosenheim

- The reason that Q-learning is *off-policy* is that it updates its Q-values using the Q-value of the next state s' and the greedy action a'.
  - In other words, it estimates the return (total discounted future reward) for state-action pairs assuming a greedy policy were followed despite the fact that it's not following a greedy policy.
  - The update function for the off-policy Q-learning algorithm:

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

    , where a' are all actions, that were probed in state s'
- The reason that SARSA is on-policy is that it updates its Q-values using the Q-value of the next state s' and the current policy's action a'.
  - It estimates the return for state-action pairs assuming the current policy continues.
  - This is the update function for the on-policy SARSA algorithm:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma(s_{t+1}, a_a t + 1) - Q(s_t, a_t))$$

On-policy vs Off-policy

Technische Hochschule Rosenheim

- Q-learning learns an optimal policy no matter what the agent does, as long as it explores enough.
  - There may be cases ignoring what the agent does is dangerous (there will be large negative rewards).
  - An alternative is to learn the value of the policy the agent is actually carrying out so that it can be iteratively improved.
  - As a result, the learner can consider the costs associated with exploration.
- An **off-policy** learner learns the value of the optimal policy independently of the agent's actions. Q-learning is an off-policy learner.
- An **on-policy** learner learns the value of the policy being carried out by the agent, including the exploration steps.

Q-Learning Tutorial

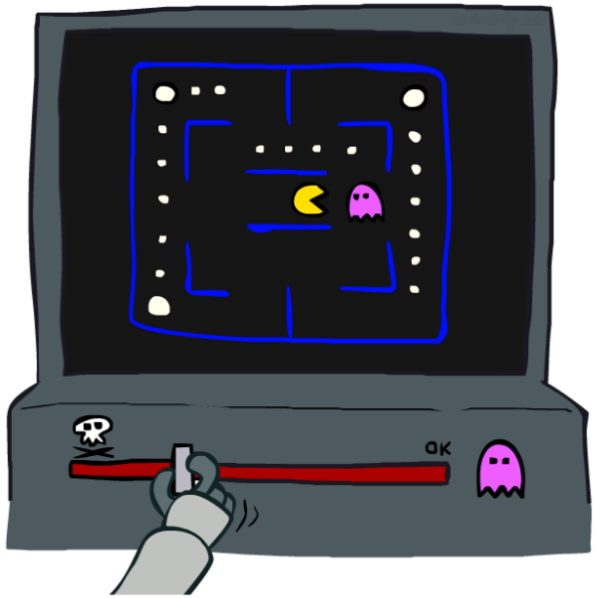
Technische Hochschule Rosenheim

Video of Demo Q-learning – Manual Exploration – Gridworld

Deep Reinforcement Learning

“Deep reinforcement learning = Deep learning+ Reinforcement learning”

Approximate Q-Learning



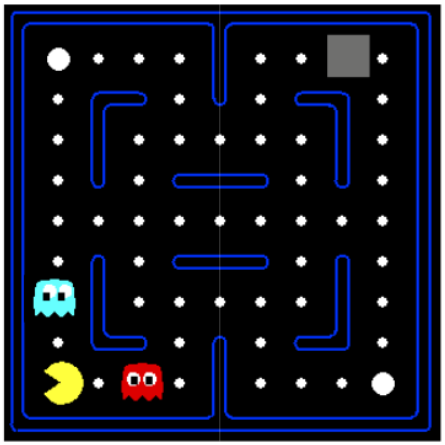
Example: Pacman

Technische Hochschule Rosenheim

Let's say we discover through experience that this state is bad:

In naïve q-learning, we know nothing about this state:

Or even this one!



Generalizing Across States

Technische Hochschule Rosenheim

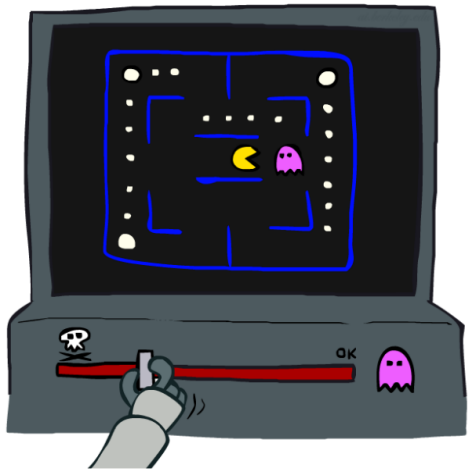
- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again

Feature-Based Representation

Technische Hochschule Rosenheim

- **Solution**: Describe a state using a vector of features (properties)

- Features are functions from states to real numbers (often 0/1) that capture important properties of the state
- Example features f :
  - Distance to closest ghost
  - Distance to closest dot
  - Number of ghosts
  - 1 / (dist to dot)2
  - Is Pacman in a tunnel? (0/1)
  - ..... etc.
  - What is it the exact state on this slide?



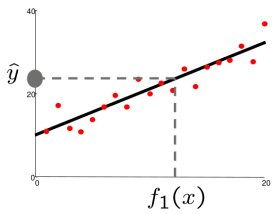
Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

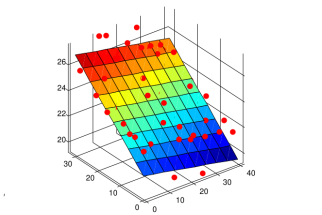
$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but actually be very different in value!



Prediction:
$\hat{y} = w_0 + w_1 f_1(x)$



Prediction:
$\hat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x)$

Minimizing the Error

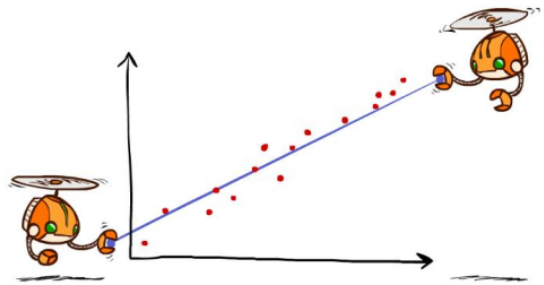
Technische Hochschule Rosenheim

Imagine we had only one point x, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\ \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$



Approximate q update explained:

$$w_m \leftarrow w_m + \alpha\left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

"target" "prediction"

Approximate Q-Learning

Technische Hochschule Rosenheim

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Q-learning with linear Q-functions:

  transition $= (s, a, r, s')$

  difference $= \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$

  $Q(s,a) \leftarrow Q(s,a) + \alpha$ [difference]

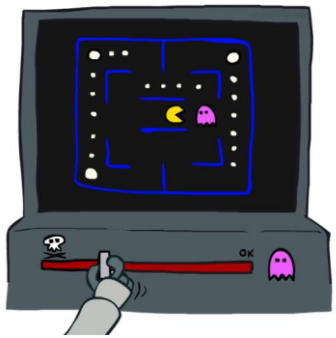
  $w_i \leftarrow w_i + \alpha$ [difference] $f_i(s,a)$

- Intuitive interpretation:
  - Adjust weights of active features
  - e.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

- Formal justification: online least squares

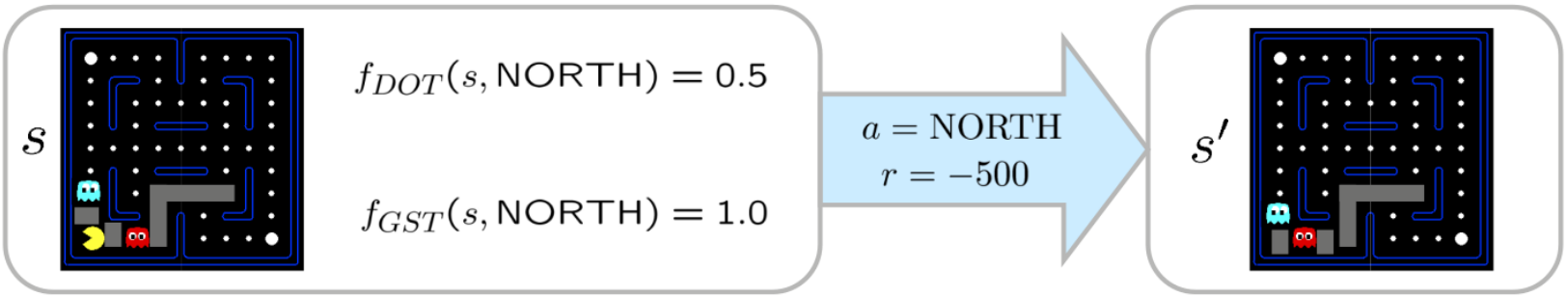
Exact Q's

Approximate Q's



Example: Pacman

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$



$Q(s, \text{NORTH}) = +1$

$Q(s', \cdot) = 0$

$$r + \gamma \max_{a'} Q(s', a') = -500 + 0$$

difference $= -501$

$$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$$
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

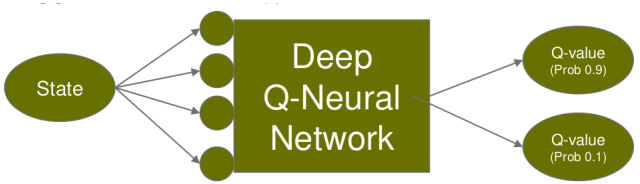
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

Adding *Deep* to Q-Learning

Technische Hochschule Rosenheim

- Q-Learning function helps to find the maximum expected future reward of an action given a current state

- If there is a gigantic set of states this approach is not scalable. We can use a neural network to predict the Q-value



The Prediction Part

Technische Hochschule Rosenheim

- The neural network job is to learn the parameters so assume that the training is done and we got the final network ready so.
- At the time of prediction, we use this trained network to predict the next best action to take in the environment so we give a input state and the network gives the Q values for all actions then we take the max Q-value to take the corresponding action in the environment.

best_action = arg max(NN predicted Q-values)

- Which means → for this state , this is the best action to take in the env.

The Training Part

Technische Hochschule Rosenheim

- **Objective function**: This is a regression problem so we use any regression loss function to minimize the total error of the training data. The neural network loss function for predicting the Q values

$$L = (R_s^a + \gamma max_{a'} Q(s', a', w) - Q(s, a, w))^2$$

- Again Gradient Descent!

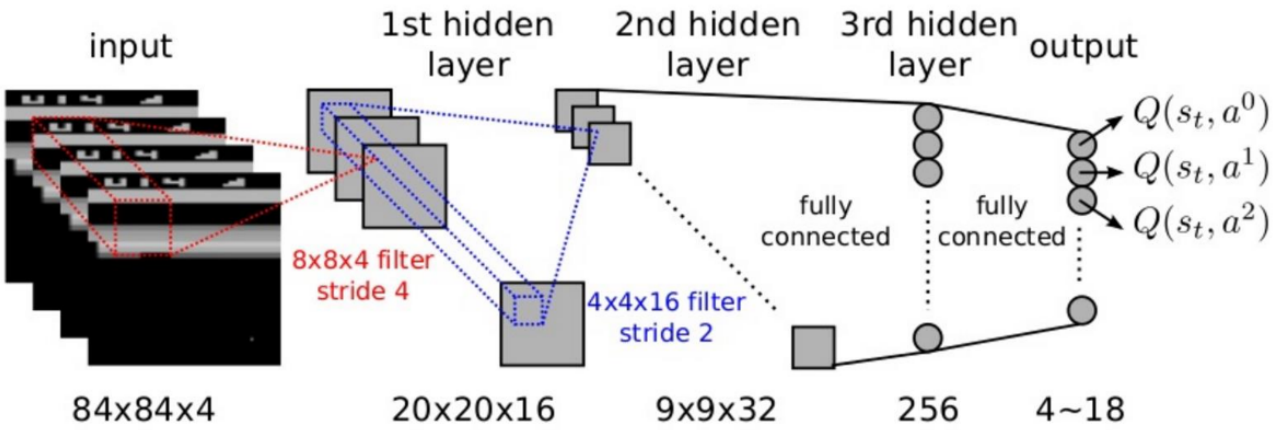
Deep Q learning for Atari

https://www.youtube.com/watch?v=z48JCQZwwzA

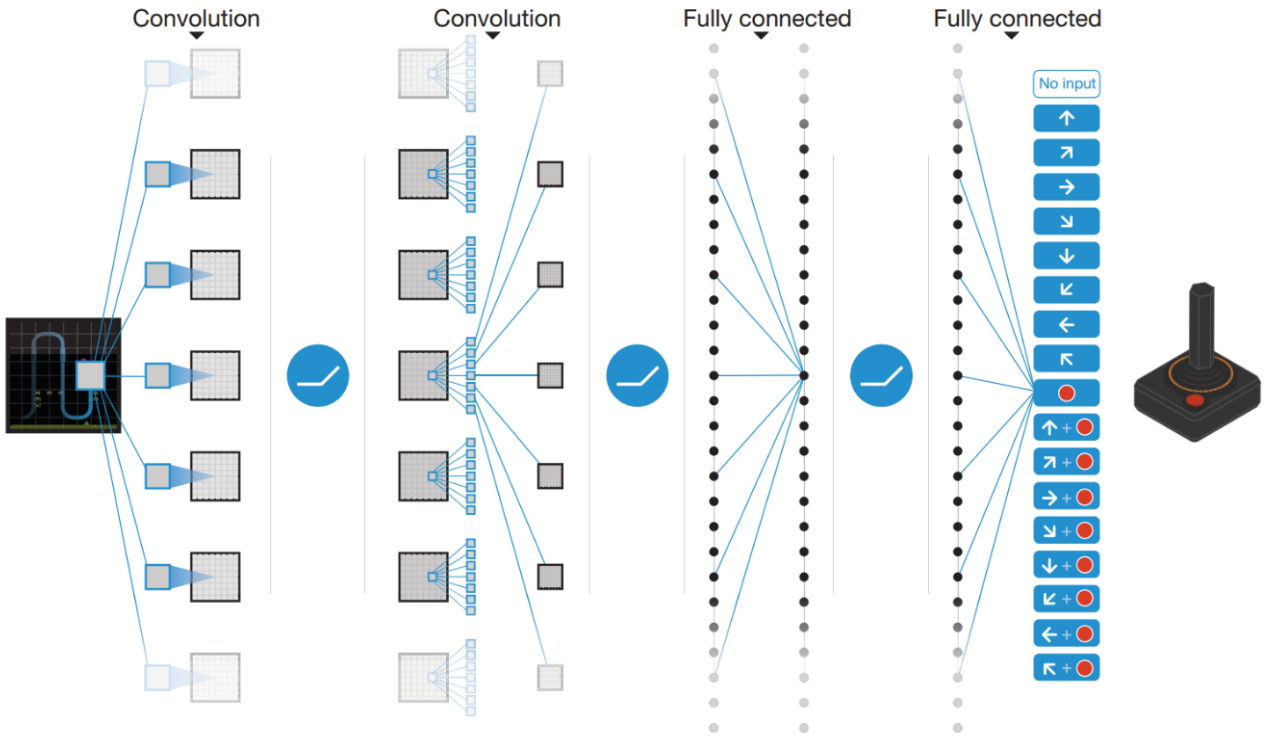
How it works

In simple terms

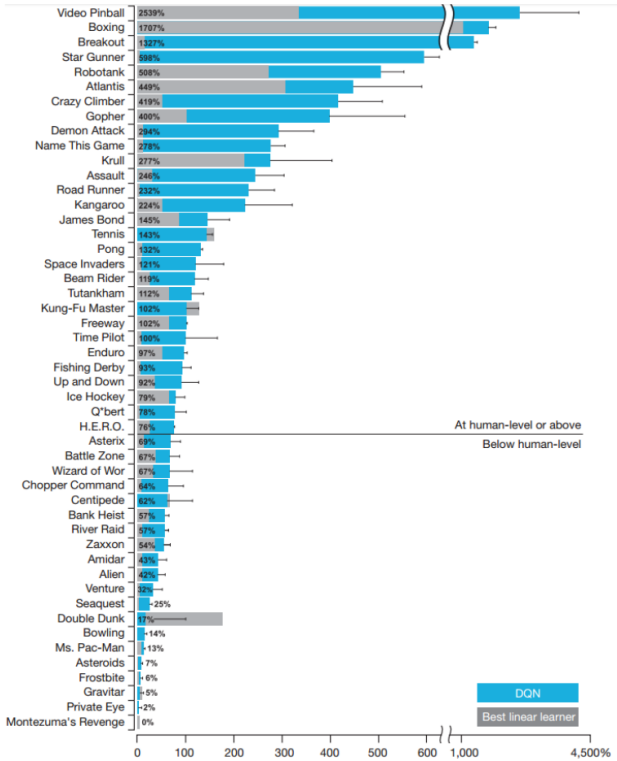
- Take the image, turn it to a gray scale image and crop the necessary image part.
- Apply some convolution filters and full connected layers with the output layers.
- Give that image (we call “state”) , calculate the Q values , find the error and backpropagate.
- Repeat this process as long as you want.



DQN - PLaying Atari



Comparison DQN vs. Human



Summary

Technische Hochschule Rosenheim

Lessons learned today:

- Temporal Difference Learning
  - SARSA
  - Q-Learning
- DQN

> Code can be found in
>
> - Q-Learning.ipynb
> - DQN.ipynb
> - or on GitHub or hosted on myBinder

Exercise

https://inf-git.fh-rosenheim.de/aai-url/10_uebung

- Q-Learning vs SARSA
- Questions?

References

- *Playing Atari with Deep Reinforcement Learning*, Mnih et al, 2013.
- *Deep Recurrent Q-Learning for Partially Observable MDPs*, Hausknecht and Stone, 2015. Algorithm: Deep Recurrent Q-Learning.
- *Dueling Network Architectures for Deep Reinforcement Learning*, Wang et al, 2015. Algorithm: Dueling DQN.
- *Deep Reinforcement Learning with Double Q-learning*, Hasselt et al 2015. Algorithm: Double DQN.
- *Prioritized Experience Replay*, Schaul et al, 2015. Algorithm: Prioritized Experience Replay (PER).
- *Rainbow: Combining Improvements in Deep Reinforcement Learning*, Hessel et al, 2017. Algorithm: Rainbow DQN