Technische Hochschule Rosenheim

Modul - Unsupervised and Reinforcement Learning (URL)

Bachelor Programme AAI

11 - DQN

Prof. Dr. Marcel Tilly

Faculty of Computer Science, Cloud Computing

© Technische Hochschule Rosenheim

Agenda

> Code can be found in
>
> - [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

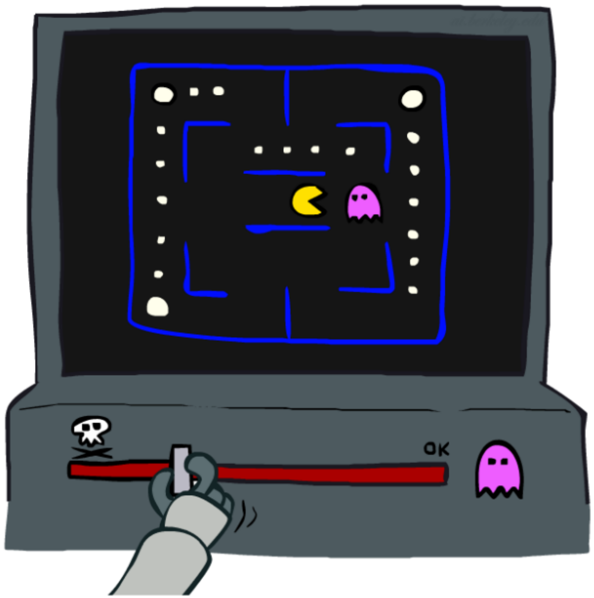
Unknown MDP: Model-Free

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

Deep Reinforcement Learning

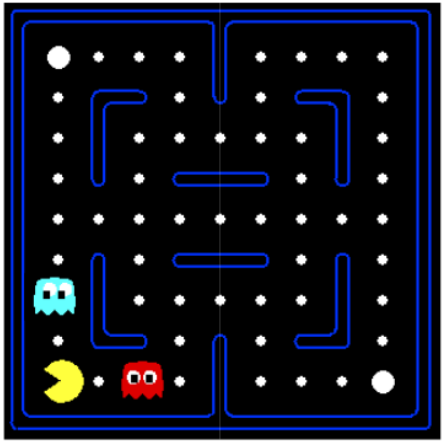
"Deep reinforcement learning = Deep learning+ Reinforcement learning"

Approximate Q-Learning



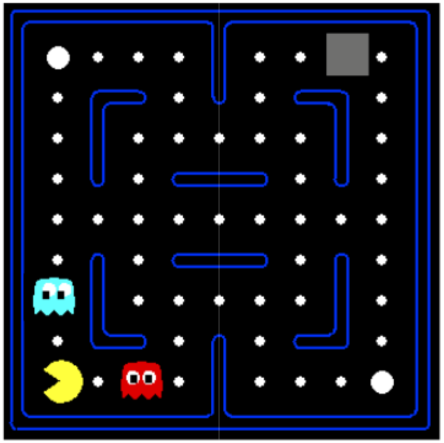
Example: Pacman

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

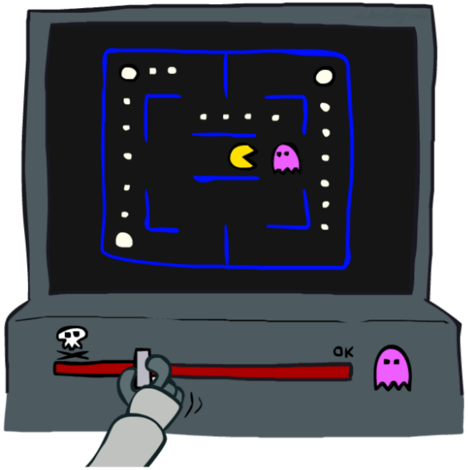
- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training 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?



© Technische Hochschule Rosenheim

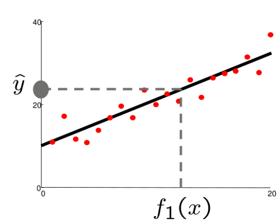
Linear Value Functions

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

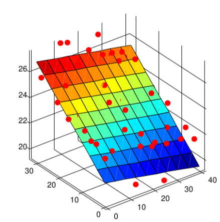
$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

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

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

© Technische Hochschule Rosenheim

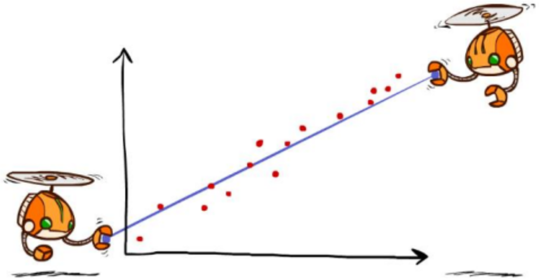
Minimizing the Error

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:

$$\text{transition} \;= (s, a, r, s')$$

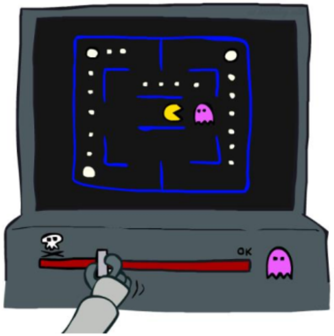
$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$$

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

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

Exact Q's

Approximate Q's



- 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

Example: Pacman

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

s

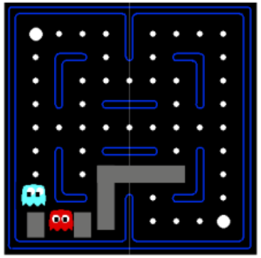
$$f_{DOT}(s, \text{NORTH}) = 0.5$$

$$f_{GST}(s, \text{NORTH}) = 1.0$$

$a = \text{NORTH}$

$r = -500$

s'



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

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

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

$$\text{difference} = -501$$

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

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

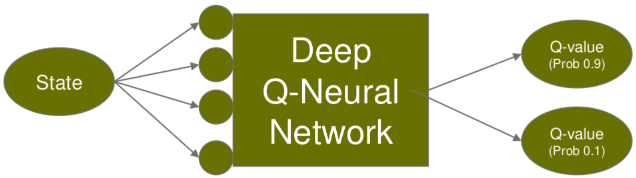
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

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

- 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

- **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!

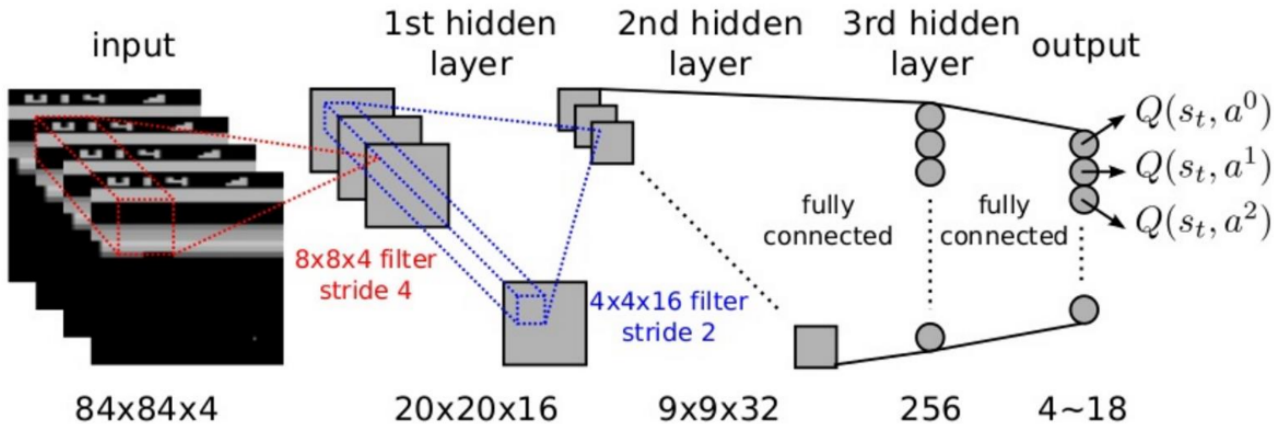
Deep Q learning for Atari

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

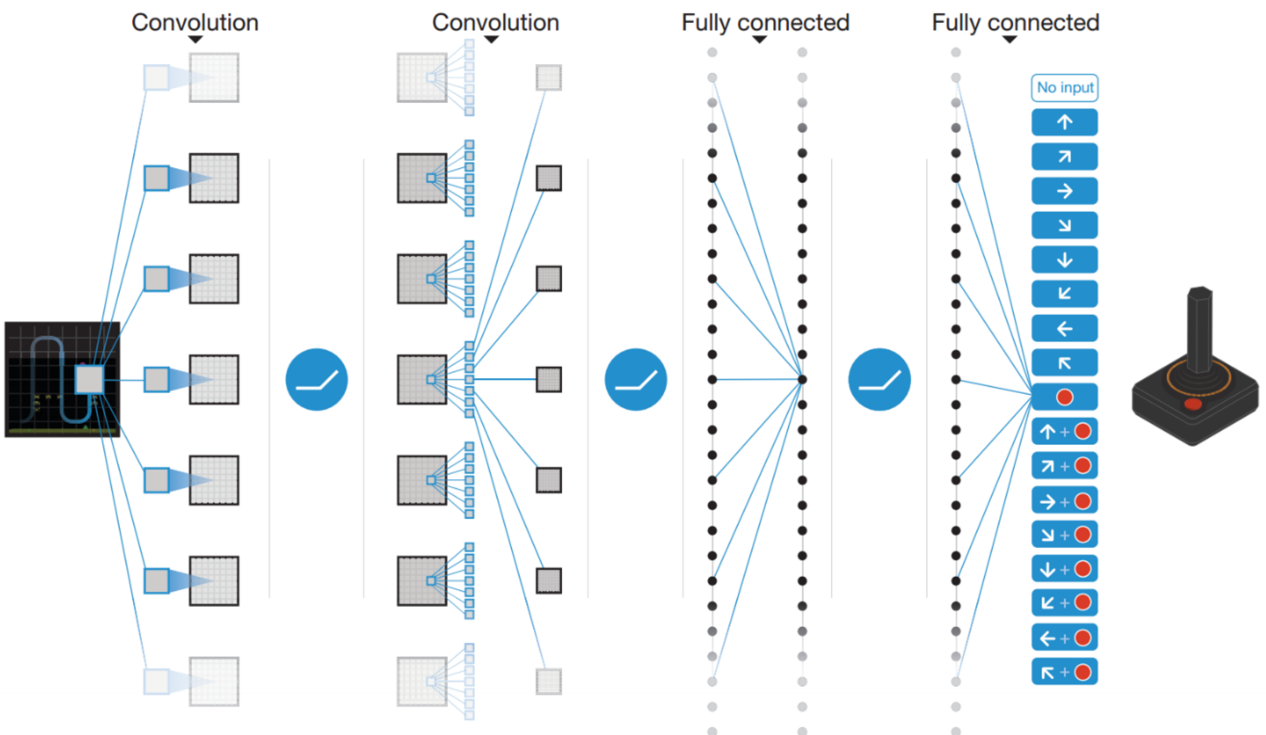
How it works

In simple terms

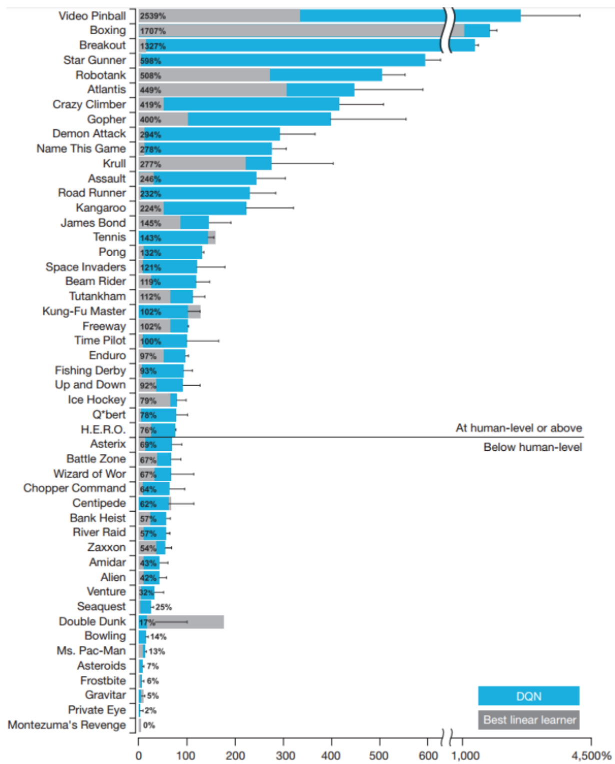
- Take the image, turn it to a gray scale image and crop the necessary image part.
- Apply some convolution filters and full connected layers with the output layers.
- Give that image (we call “state”) , calculate the Q values , find the error and backpropagate.
- Repeat this process as long as you want.



DQN - PLaying Atari



Comparison DQN vs. Human



Summary

Technische Hochschule Rosenheim

Lessons learned today:

- DQN

> Code can be found in
>
> - 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?

© Technische Hochschule Rosenheim

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