Zurich University of Applied Sciences
zhaw School of Engineering

Datalab Seminar

Reinforcement Learning - Deep Q-Learning

29. November 2017

Agenda

- Q-Learning with Neural Networks
- Experience Replay
- Target Networks
- DQN

Q-learning with Neural Networks

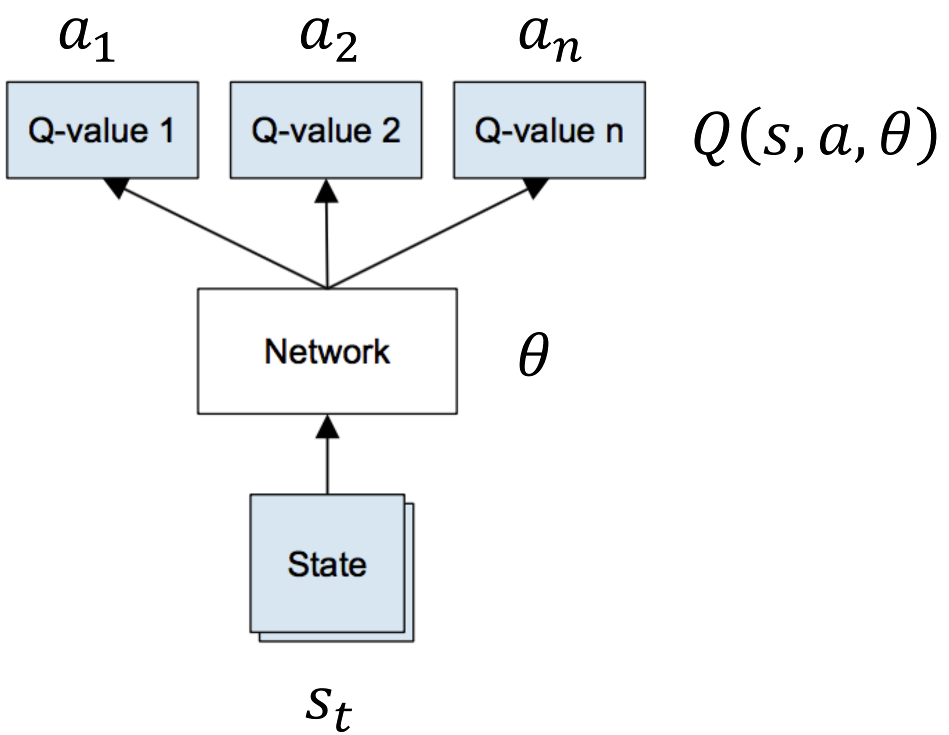
- Recap
  - Tabular Q-Learning update rule

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \overset{\text{learning rate}}{\alpha} \, [\underbrace{r_t}_{\text{reward}} + \overset{\text{discount factor}}{\gamma} \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future reward}}]$$

- $Q(s_t, a_t)$ doesn't have to be a lookup table
- It can also be a neural network or any other type of function approximator

Q-learning with Neural Networks

- However a neural network has a bunch of weights θ
- Q-function looks like this: $Q(s_t, a_t, \theta)$

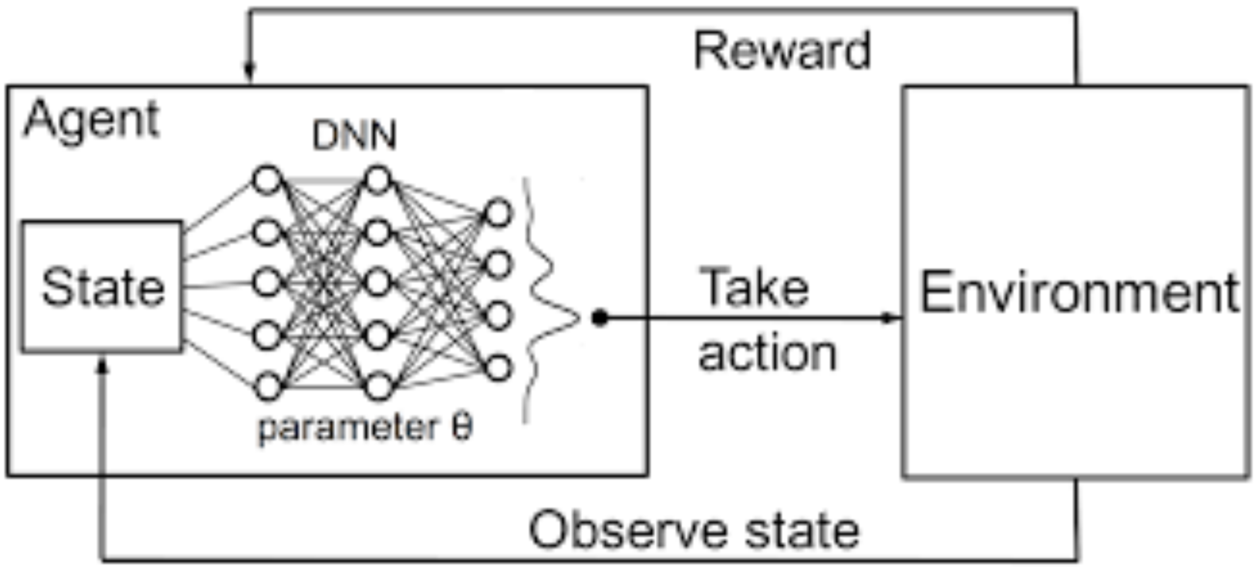


Q-learning with Neural Networks

- For neural networks, we usually an input vector and a target vector
- For deep q-learning with neural networks the target is

$$r_t + \gamma \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future reward}}$$

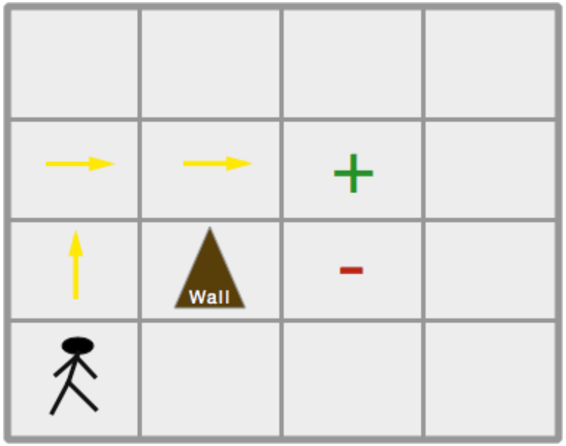
Q-learning with Neural Networks

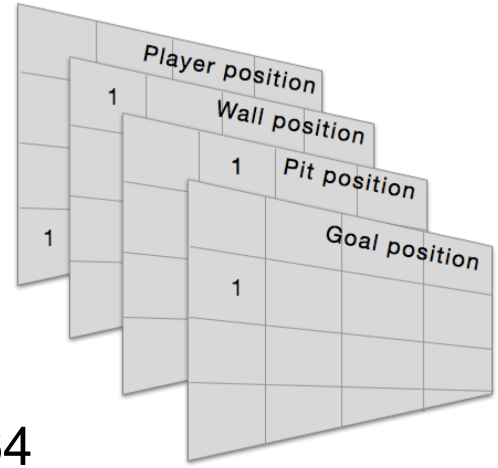


Example / Demo

http://outlace.com/rlpart3.html#Online-Training

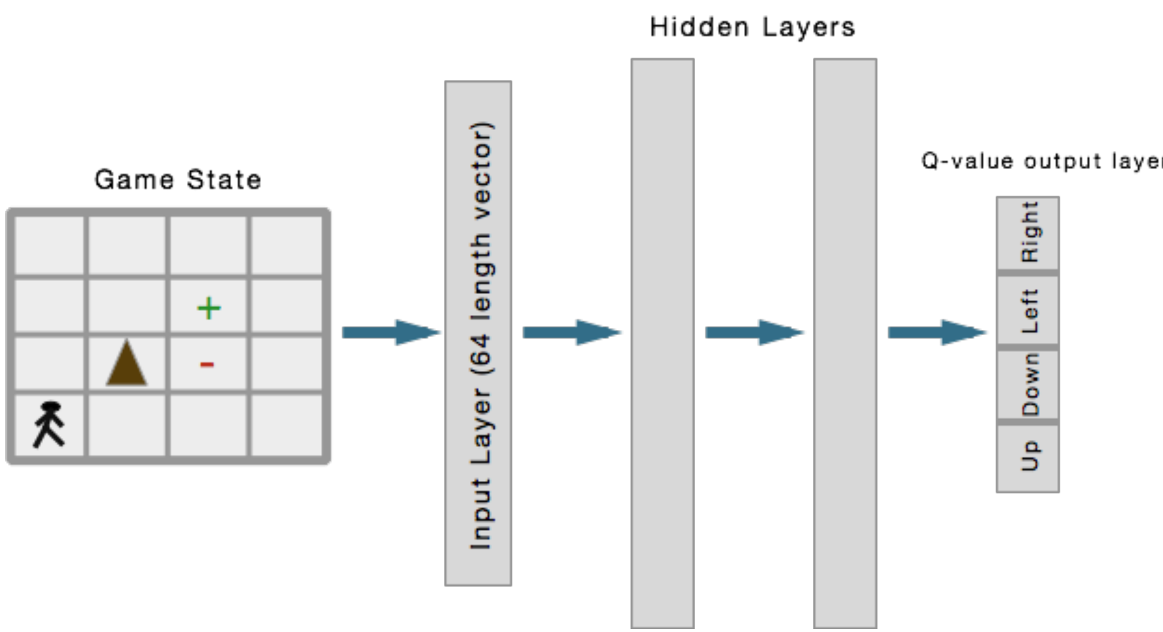
- Gridworld
  - Actions
    - 0 -> up
    - 1 -> down
    - 2 -> left
    - 3 -> right
  - States
    - For every object
      - player, wall, pit, goal
    - Separate grid with
      - 1 at the position of the object
      - 0 everywhere else
    - 4x4x4 values = state vector of length 64





Network

- Input / state vector with 64 nodes
- 2 hidden layers both with 20 nodes
- Output layer with size 4 (number of actions)



Network Code


```
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import RMSprop
```

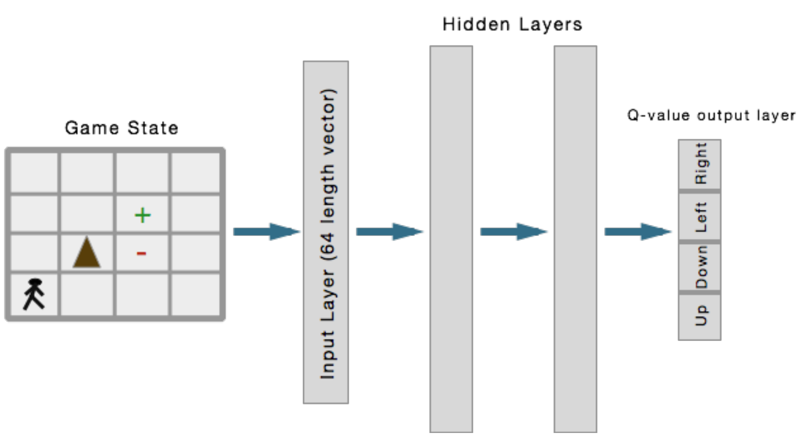
```
Using TensorFlow backend.
```

```
model = Sequential()

# 2 hidden layers both with 20 nodes, 64 input nodes
model.add(Dense(20, activation='relu', kernel_initializer='lecun_uniform', input_shape=(64,)))
model.add(Dense(20, activation='relu', kernel_initializer='lecun_uniform'))

# Use linear activation for q-values
model.add(Dense(4, activation='linear', kernel_initializer='lecun_uniform'))

rms = RMSprop()
model.compile(loss='mse', optimizer=rms)
```



Train Agent

Zurich University
of Applied Sciences
zh
aw
School of
Engineering

for i in range(number of games (epochs))
    while game is in progress
        evaluate network (get q-values for all actions with the given state s)
        if random float < epsilon
            choose random action a
        else
            choose action a with the highest q-value
        Execute action a, collect reward r and get new state s_{t+1}
        Evaluate network with new state s_{t+1} and get maximal q-value (maxQ)
        Get new target values:
            For executed action: reward r + (gamma * maxQ)
            For all other actions: copy q-values with old state s
        Fit the network for state s with the new target values
        State s = new state s_{t+1}
        Update epsilon

Epsilon Greedy (exploration vs exploitation dilemma of rl)

Experience Replay

- Breaks up the correlation of the data
- Advantages
  - More efficient use of previous experiences
  - Better convergence behaviour, partly since the data is more i.i.d.

Experience Replay

- Pseudo code

In state s, take action a, observe new state s_{t+1} and reward r_{t+1}

If replay buffer is not yet full

    Store this as a tuple (s, a, s_{t+1}, r_{t+1}) in the replay buffer

Else

    Overwrite one element in the replay buffer with the new tuple

    Randomly take n samples from the replay buffer → we call this minibatch, n=batchSize

    Iterate through minibatch

        Evaluate network with new state s_{t+1} and get maximal q-value (maxQ)

        Get new target values:

            For executed action: reward r + (gamma * maxQ)

            For all other actions: copy q-values with old state s

        Fit the network for state s with the new target values

Target Networks - Problem

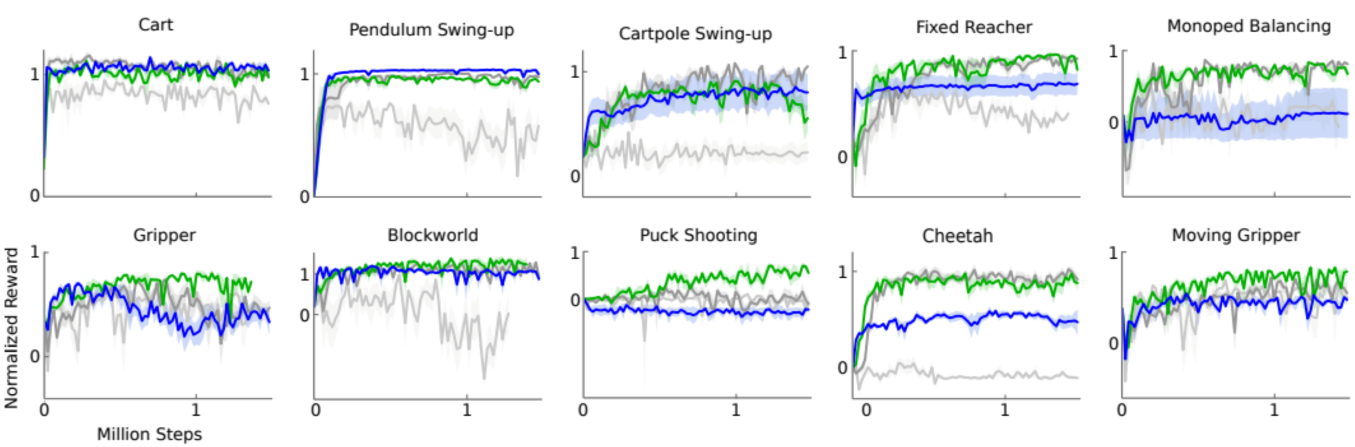
- The target depends on the current network

$$Q(s_t, a) \leftarrow r_t + \gamma \max_a Q(s_{t+1}, a)$$

- But $Q(s_t, a)$ and $Q(s_{t+1}, a)$ are very close together
- Everytime we update the target is likely to shift
  - An infinite loop is generated
  - Leads to instabilities, oscillations or divergence

Target Networks - Solution

- Target Networks
  - 2 networks
    - 1 frozen network to compute the targets $(\hat{Q})$
    - 1 network that we train (Q)
    - Periodically we sync the networks by copying the weights $\hat{Q} \leftarrow Q$



light grey: without target networks

Blue, green, dark grey: target network variants

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING, Timothy P. Lillicrap et. al

DQN

Zurich University of Applied Sciences
zh aw School of Engineering

- DQN is introduced in 2 papers (aka Atari papers)
  - Playing Atari with Deep Reinforcement Learning on NIPS in 2013
  - Human-level control through deep reinforcement learning on Nature in 2015

- Deep Q-Learning with 2 additional techniques
  - Experience Replay
  - Target Network