Autonomous Learning

Assignment 2

Simon Reichhuber

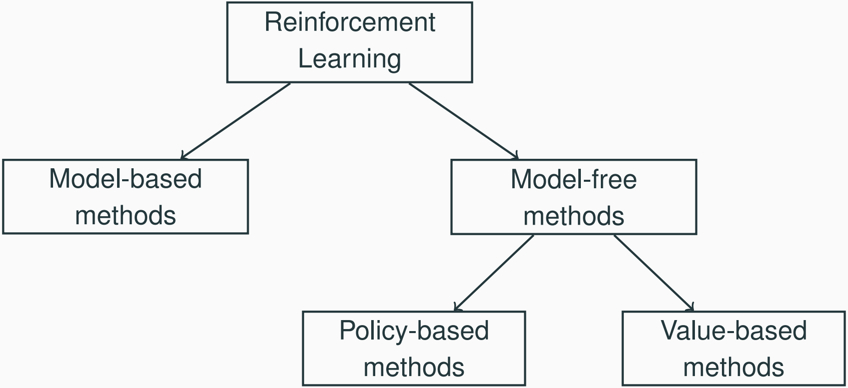
June 15, 2020

University of Kiel, Summer Term 2019

Table of Contents

Reinforcement Learning

Overview


Reinforcement Learning
Model-based methods
Model-free methods
Policy-based methods
Value-based methods

Model-based methods

- Agent learns a model of the environment
- Action a_1 in state s_1 $\rightarrow$ state s_2 and reward r_2

  $\Rightarrow$ improvement of the estimates of $T(s_2|s_1, a_1)$ and $R(s_1, a_1)$
- As soon as the model is sufficient $\rightarrow$ policy

Example: Value Iteration and Policy Extraction

Model-free methods

Value-based methods

... are based on *temporal difference learning*: The learn the value function V^{π} or V^{*} or the Q function Q^{π} or Q^{*}

Policy-based methods

... directly learn the optimal policy π^{*} (or try to approximate the optimal policy in case the real optimal policy is not reachable)

VALUE-BASED METHODS

Christian-Albrechts-Universität zu Kiel
Technische Fakultät

Value-based methods

on-policy learning

off-policy learning

learns the values of policy π

learns the values of a different policy

ON-POLICY LEARNING

- Agents learn the value of the policy in use to make decisions
- The estimated value function is updated using the results of actions chosen by policy π

Example: SARSA

- The estimated value function can be updated by hypothetical actions: actions that are not explicitly explored

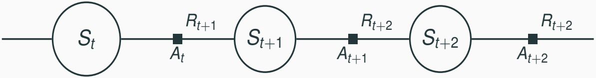
$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)]$$

- The agent learns strategies that he not necessarily explored during training.

Example: Q-Learning

SARSA

- On-policy
- An episode consists of an alternating sequence of states and state-action-pairs:



- Learning the state-action-pairs:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

Quintuple: $\big(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}\big) \rightarrow$ SARSA

SARSA learns $Q^{\pi}(S_t, A_t)$

$$Q^{\pi}(S_t, A_t) \leftarrow Q^{\pi}(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q^{\pi}(S_{t+1}, A_{t+1}) - Q^{\pi}(S_t, A_t)\Big]$$

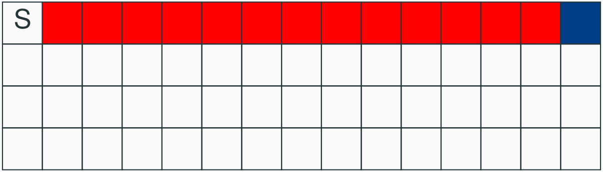
Q-Learning learns $Q^{*}(S_t, A_t)$

$$Q^{*}(S_t, A_t) \leftarrow Q^{*}(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q^{*}(S_{t+1}, a) - Q^{*}(S_t, A_t)\right]$$

Both converge to Q^{π} resp. Q^{*} if enough samples for each state-action-pair are given.

Cliff Scenario

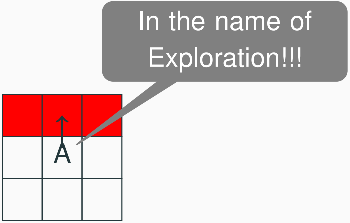
- The agent starts in S
- The target is blue final state (positive reward)
- But there are cliffs on the way (negative reward and restart!)



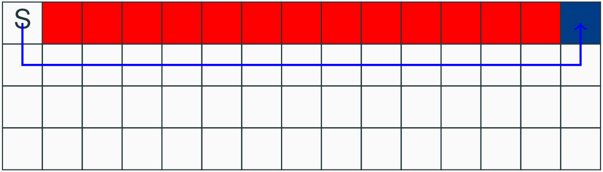
Cliff Scenario

Q-Learning → highest action value

BUT: Exploration → random action

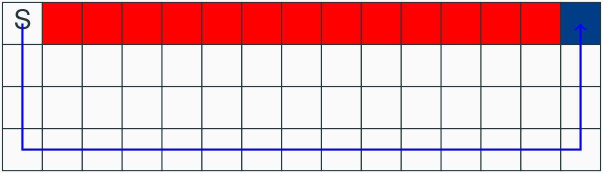


Cliff Scenario with Q-Learning



- optimal (fastest) route
- not save

Cliff Scenario with SARSA



- longer route
- save
- good online performance

Algorithm 1: SARSA

Initialize $Q(s, a)$ arbitrarily and the $Q(terminal, \cdot) = 0$; **repeat**

  Initialize s;

  Choose a from s using policy derived from Q (e.g. ϵ-greedy);

  repeat

    Take action a, observe r, s';

    Choose a' from s' using policy derived from Q (e.g. ϵ-greedy);

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

    $s \leftarrow s'$;

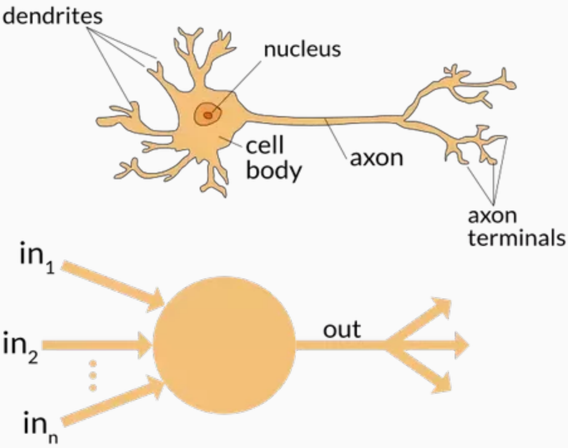
    $a \leftarrow a'$;

  until *s is terminal*;

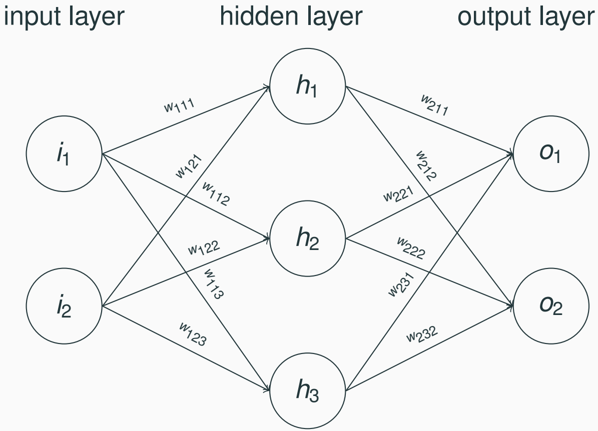
until *Q is converged*;

Artificial Neural Networks

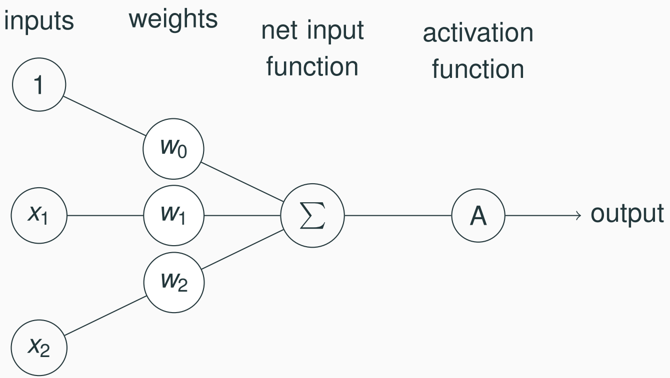
NEURONS


dendrites
nucleus
cell body
axon
axon terminals
in1
in2
inn
out

Neuronal Network



Nodes


inputs
weights
net input function
activation function
1
w_0
x_1
w_1
Σ
A
output
w_2
x_2

Nodes

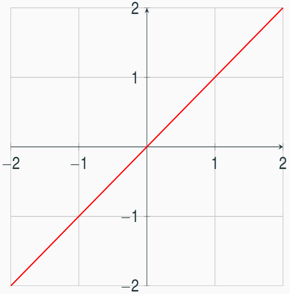
- A node is fired when the inputs meet certain requirements
- The input function sums up all inputs (*input* $\times$ *weight*)
- The activation function decides if and how intense a signal is sent

- The optimisation function adapts the weights according to the error they produce.
- Gradient denotes the relationship between a single weight and the error of the whole network.
- Slow adaption of many weights $\rightarrow$ Which input has what significance?
- Chain Rule:

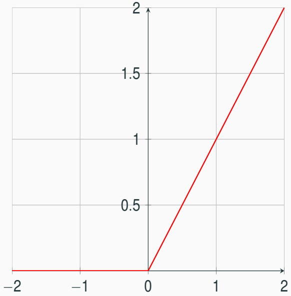
$$\frac{dError}{dweight} = \frac{dError}{dactivation} * \frac{dactivation}{dweight}$$

- Learning: Adaption of the weights until the error cannot be minimised any further.

linear:



ReLu (Rectifier Linear Unit):



Error and Loss Function

- The error for trainable weights θ is generally defined as the difference between the predicted and the actual output.

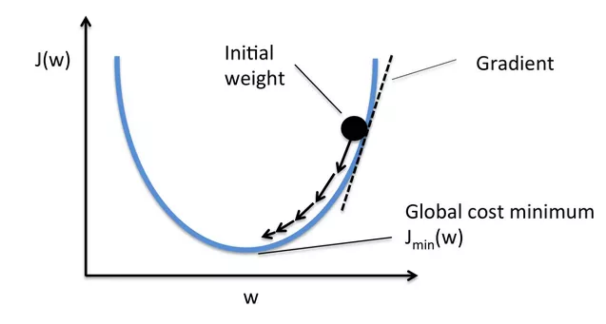
$$J(\theta) = p - \hat{p}$$

- The function for calculating the error is called Loss Function J (or cost function).
- Different Loss Functions $\rightarrow$ different errors
- Frequently used Loss Function:

  mean square error (MSE)

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (p_i - \hat{p}_i)^2$$

- $J(\theta) = J(w) =$ function of the internal parameters (weights and bias)
- The error is passed through the layers from the back to front.



KERAS

- A high-level neural networks API.
- Different libraries can work in the background (e. g. Tensorflow, CNTK, Theano)
- www.keras.io

KERAS - A SIMPLE NET


```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import RMSprop

model = Sequential()

# input and first hidden layer
model.add(Dense(units=128, activation'relu', input_dim=2))

# second hidden layer
model.add(Dense(units=256, activation='relu'))

# output layer
model.add(Dense(units=8, activation='linear'))

model.compile(loss='mse',
              optimizer=RMSprop(lr=0.00025))

# train network
model.fit(x_train, y_train, batch_size=32)

# predict on trained network
prediction = model.predict(x_test)
```

Deep Q-Network

Limits of standard Q-Learning

- Large state space and/or action space $\rightarrow$ very large Q-table
  $\Rightarrow$ approximation of the Q-table using a neural net
- A neural net can generalize its knowledge of visited states for non-visited states
- Abstraction of patterns and understanding actions on the basis of already seen patterns.

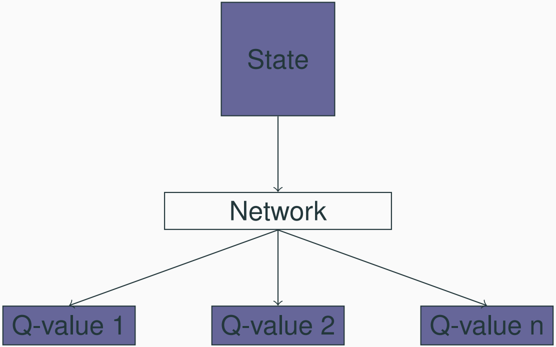
- State space represented as vector
- Loss Function:

$$J(\theta) = \sum (Q - Q_{target})^2$$

- Approximating function:

$$Q(s_t, a_t; \theta) \leftarrow Q(s_t, a_t; \theta) + \alpha \Big[\underbrace{\underbrace{r_t + \gamma \max_a Q(s_{t+1}, a; \theta)}_{target} - Q(s_t, a_t; \theta)}_{TD-error} \Big]$$

MODELING OF $Q(s, a)$



Q-TARGET

$$Q_{target}(s_t, a_1) = Q(s_t, a_1; \theta)$$

$$Q_{target}(s_t) \xleftarrow{a_2 == a_t} Q_{target}(s_t, a_2) = r_{t+1} + \gamma * \max_a Q(s_{t+1}, a; \theta)$$

$$Q_{target}(s_t, a_3) = Q(s_t, a_3; \theta)$$

Unstable learning

- Non-linear approximation function (ANN) $\rightarrow$ unstable learning
- Reasons:
  - Correlation between some observations
  - Correlation between action and target values
  - Small adaptions can lead to significant changes of the policy and subsequently also the distribution of the data.

Experience Replay

- Remember the last N transitions $(s_t, a_t, r_{t+1}, s_{t+1}, done)$
- Instead of learning from just the last transition: For each step draw a random minibatch (of size 32) from the experience replay.
- Q-Learning Updates based on this minibatch
- FiFo

$$\Downarrow$$

- This removes correlations between individual observations and smoothes changes in the data distribution.
- Transitions are used more often for learning $\rightarrow$ data efficiency

Target Network

- In every step the values of the Q-network shift

  $\Rightarrow$ feedback loops between target values and predicted q-values
- Target network: A second neural network used during training
- Calculates the target Q-values for the Loss Function
- Is periodically updated (every C steps)

$$\Downarrow$$

- Reduces the correlation between action and target values

- Defining a minimal and maximal error:

$$\left[r + \gamma \max_{a'} Q(s', a'; \theta^{-}) - Q(s, a; \theta)\right] \in [-1, 1]$$

$\Rightarrow$ more stable learning

Target Network - Loss Function

θ = weights of the Q-network
θ^- = weights of the target network

Loss Function:

$$J(\theta) = \sum \left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2$$

Algorithm 2: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
for *episode=1,M* **do**
    for *t=1,T* **do**
        With probability ϵ select random action a_t
        otherwise select $a_t = \mathrm{argmax}_a\, Q(s_t, a; \theta)$
        Execute action a_t and observe reward r_{t+1} and state s_{t+1}
        Store transition $(s_t, a_t, r_{t+1}, s_{t+1}, done)$ in D
        Sample random minibatch of transitions $(s_j, a_j, r_{j+1}, s_{j+1}, done)$ from D
        Set $y_j = \begin{cases} r_j & \text{if } done \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
        perform a gradient descent step on $\left(y_j - Q(s_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
        Every C steps reset $\hat{Q} = Q$

Bibliography

PICTURES

- https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network
- https://www.quora.com/In-neural-networks-how-important-is-back-propagation-What-is-its-significance