Deep Reinforcement Learning

2 - Multi-Armed Bandits to Sequences of Decisions

Prof. Dr. Malte Schilling

Autonomous Intelligent Systems Group

WWU MÜNSTER

Overview Lecture

- Exploration-Exploitation Tradeoff
- Decision Making: Multi-Armed Bandit
  - Strategies
    - ε-greedy
    - UCB
    - Gradient-based Action Selection
- Sequences of States – towards Sequential Decision Making

Admin - Exercises

- Exercise slot is Friday morning, 10:15 AM to 11:45, M5.
- Solutions (for programming exercises in python) should be submitted by small teams of two or three persons. Everybody has to be able to present the solution during meetings.
- First exercise sheet out today – due next Wednesday (upload in learnweb).

Register groups: Send an email to malte.schilling@uni-muenster.de with name and account information for each group member until Tuesday, 25.10.2022!

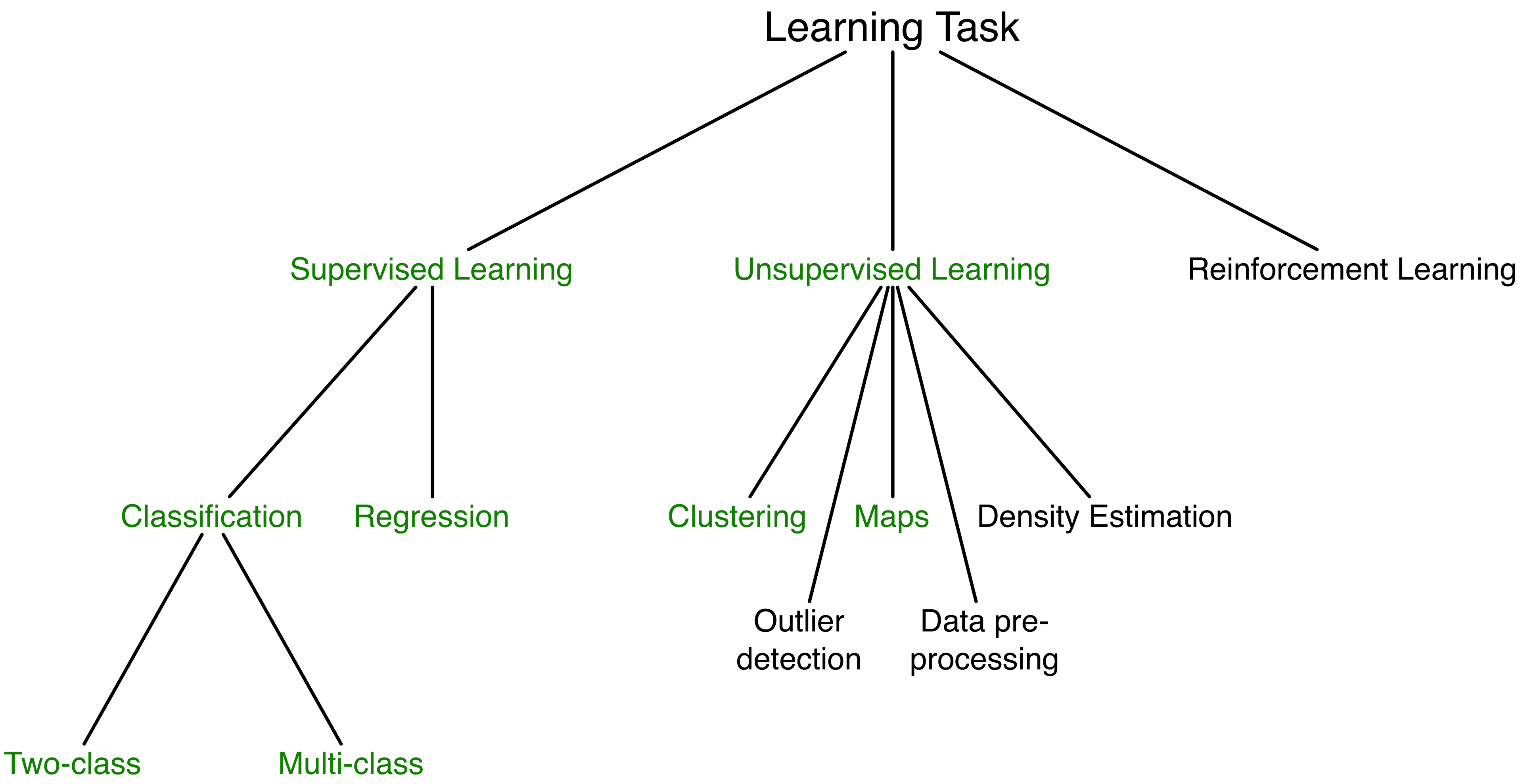
Join the learnweb course!

Übungstermin Freitag, 21.10.2022

Inhalte:

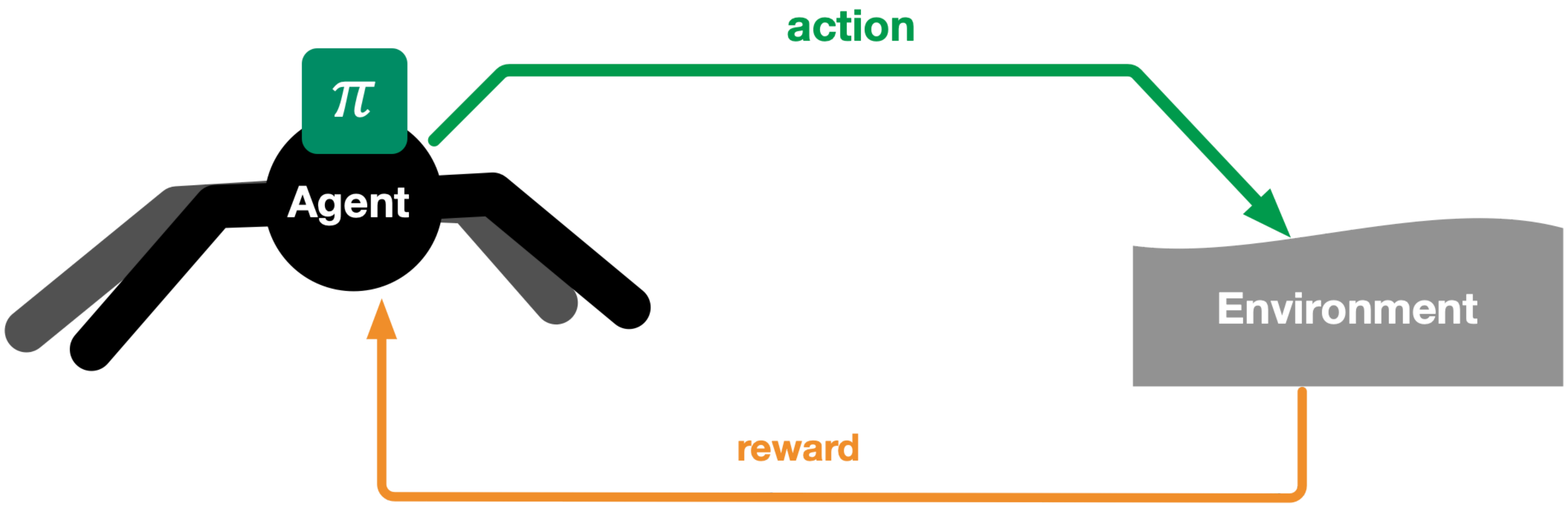
- Troubleshooting für Python Aufgaben
- Basierend auf dem Übungszettel: zusammenstellen der notwendigen numpy Funktionen (arrays, grundlegende statistische Funktionen, …) und plot Funktionen (für eine random policy).

Recap – Types of Learning



Distinction of different types of learning

Parts of Decision Making



Agent

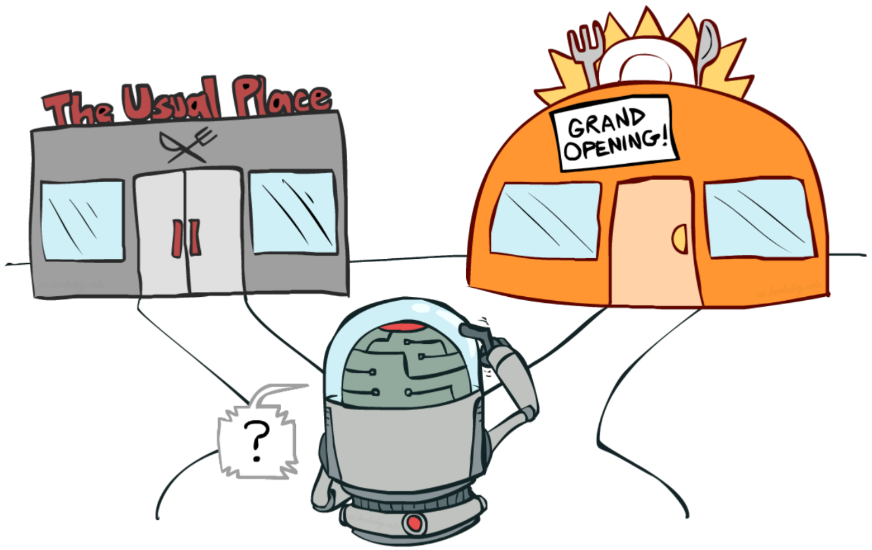
- A **policy** *is* the agent's behavior – chooses an action.

Environment

- Provides reward

Examples of Rewards

- Fly stunt manoeuvres in a helicopter
  - positive reward: if following a desired trajectory
  - negative reward: when crashing
- Manage an investment portfolio – reward is given as the money in that account
- playing computer games – reward directly given as score
- Locomotion of a robot
  - positive reward for movement in the correct direction
  - negative reward / cost: falling over (not maintaining hight); energy consumed



Decision Making: sticking to a good past experience might make you miss out on even better options, but at least you can be confident to get something good.

- Homework answers: What is your example of an exploration-exploitation tradeoff?
- What would be the reward?

(Klein und Abbeel 2014)

Exploitation-Exploration Tradeoff

As information of a novel environment is incomplete, we need to gather information for good decisions and want to keep the risk under control.

Exploitation

Taking advantage of the best known option.

Exploration

Take some risk to collect information about unknown options.

An optimal long-term strategy may involve short-term sacrifices, e.g. learning from failure during exploration helps us avoid a certain action.

Examples for Exploitation-Exploration Tradeoff

- Restaurant Selection
  - Exploitation: Go to your favourite restaurant
  - Exploration: Try a new restaurant
- Oil Drilling
  - Exploitation: Drill at the best known location
  - Exploration: Drill at a new location
- Game Playing
  - Exploitation: Play the move you believe is best
  - Exploration: Play an experimental move

(Silver 2015)

Rewards

- A **reward** R_t is a scalar feedback given to the agent from the environment
- that indicates how well the agent is doing (at time t).
- The agent aims to maximise the cumulative reward over time.

Reward Hypothesis

All goals can be represented as maximization of a scalar reward (an expected cumulative reward).

Return (preliminary Def.)

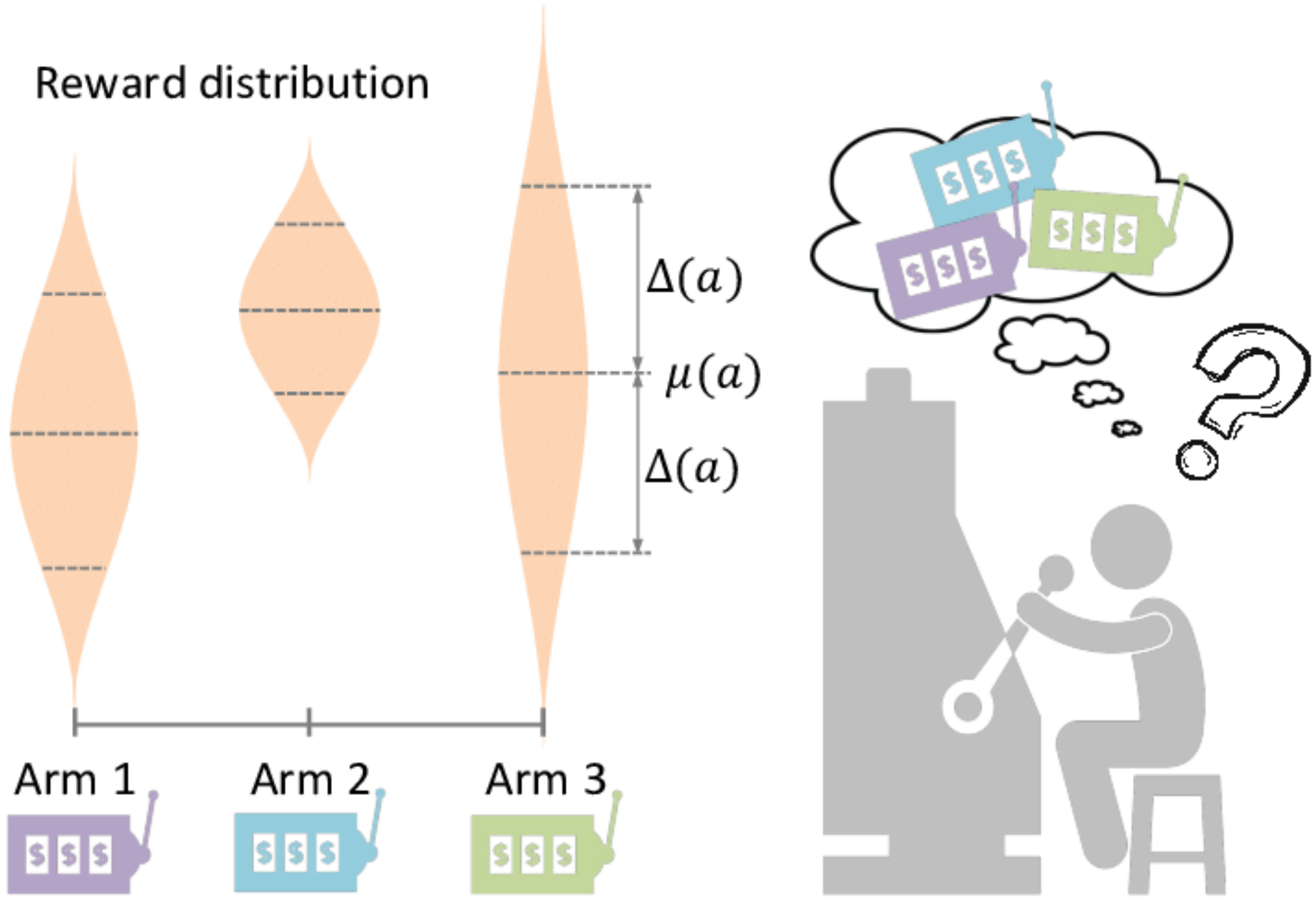
The accumulated reward over time is called the return G_t.

$$G_t = R_{t+1} + R_{t+2} + \dots$$

(Sutton und Barto 2018)

Multi-Armed Bandit

Multi-Armed Bandit Overview

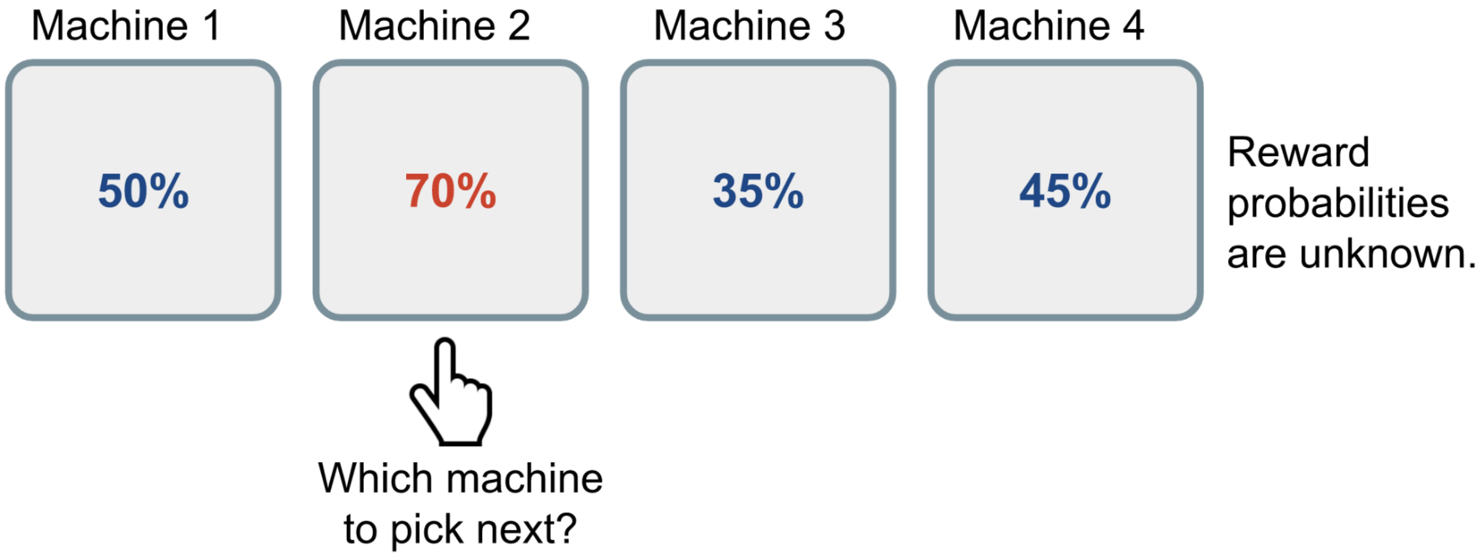


(Gao u. a. 2021)

Multi-Armed Bandit

Idea: in a casino with multiple slot machines of unknown probabilities.

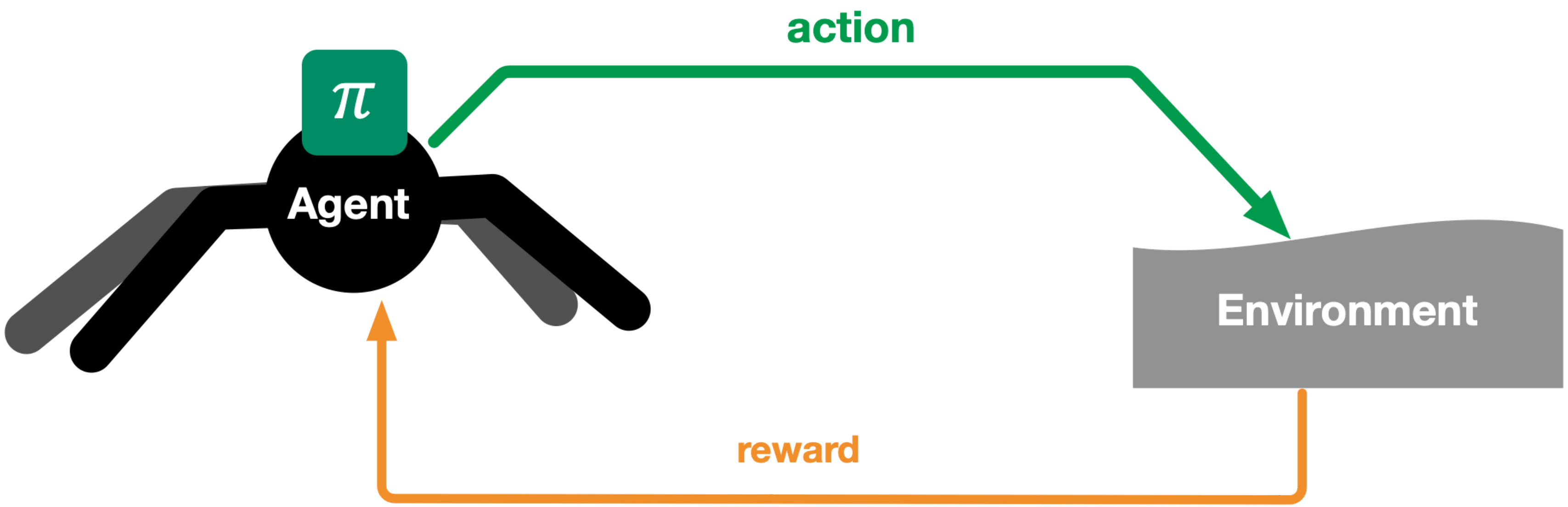
Which action (slot machine) should you choose for optimal reward?



Following a naive approach, one would gather information over a long time to get a true estimate of each of the probabilities. But as a consequence one will spend too much time on suboptimal actions.

(Weng 2018)

Parts of Decision Making



Agent

- A **policy** *is* the agent's behavior – chooses an action.
- **Value function**: Keep track of the value of an action.

Environment

- Provides reward

Action values

Action value

The action value for action a is the expected reward

$$Q_t(a) = \mathbb{E}\left[R_t \mid A_t = a\right]$$

A simple estimate is the average of the sampled rewards:

$$\begin{aligned} Q_t(a) &= \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} \\ &= \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \end{aligned}$$

with $\mathbb{1}$ being the indicator function, therefore we count choosing action a as

$\sum_{i=1}^{t} \mathbb{1}(A_i = a)$

Maximization of cumulative reward

Goal is to maximize **cumulative reward** $\sum_{t=1}^{T} r_t$ (the return G).

The optimal action produces the maximal reward. Deviating from that action leads to a potential loss or **regret**.

The probability for the optimal reward θ^* of the optimal action a^* is

$$\theta^* = Q(a^*) = \max_{a \in \mathcal{A}} Q(a) = \max_{1 \leq i \leq K} \theta_i$$

Regret

The regret of an action a is given as

$$\Delta_a = v_* - q(a)$$

and the regret for choosing the optimal action is zero.

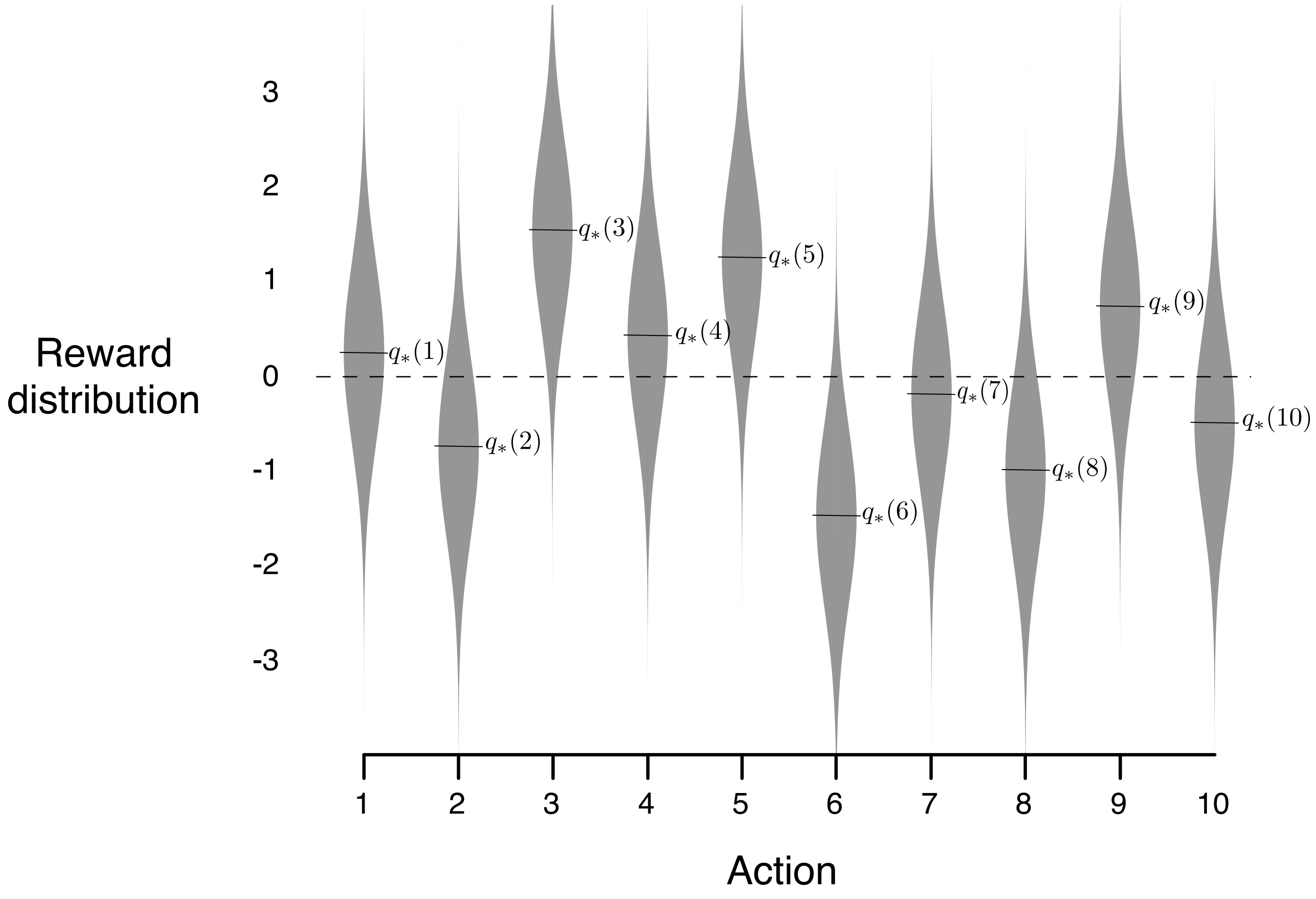
Total regret as a loss

The loss function is the total regret for not selecting the optimal action up to the time step T

$$L_T = \mathbb{E}\Big[\sum_{t=1}^{T}\big(\theta^* - Q(a_t)\big)\Big] = \sum_{t=1}^{T}\Delta_{A_t}$$

(Weng 2018)

An example bandit problem



Random action values $q_*(a), a = 1, \ldots, 10$ (selected from a normal distribution, zero mean, unit std. dev.).

(Sutton und Barto 2018)

Task – Maximize the return

- Come up with an algorithm that maximizes the accumulated reward and improves over time.
- Consider advantages and disadvantages of your approach.
- Why is the experimental procedure given in the python example problematic?

Beispiel in Python

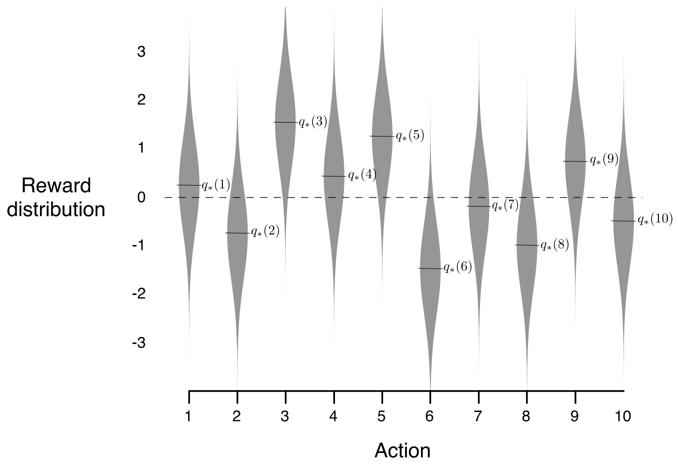
Generating 10 bandits.

```python
import time
import numpy as np
# Bandit class is taken from https://github.com/lilianweng/multi-armed-bandit
class BernoulliBandit(object):

    def __init__(self, n, probas=None):
        assert probas is None or len(probas) == n
        self.n = n
        if probas is None:
            np.random.seed(int(time.time()))
            self.probas = [np.random.random() for _ in range(self.n)]
        else:
            self.probas = probas

        self.best_proba = max(self.probas)

    def generate_reward(self, i):
        # The player selected the i-th machine.
```

Can be realized in different ways and using different forms of distributions.

Normal Distribution

In the simplest case, we assume the reward as following a normal distribution. Such a bandit can be represented through a mean and standard deviation. Rewards are sampled from this distribution.



Bernoulli Distribution

In the case of discrete reward events, a Bernoulli Distribution can be used. The probability describes how often a fixed reward is given (otherwise zero would be given as a reward).

The probability distribution of the expected reward should be described using a beta distribution.

Multi-Armed Bernoulli Bandit

A Bernoulli multi-armed bandit can be described as a tuple of $\langle \mathcal{A}, \mathcal{R} \rangle$, where:

- K machines with reward probabilities, $\theta_1, \ldots, \theta_K$
- for each time step t, take an action a on one slot machine and receive a reward r.
- $\mathcal{A}$ is a set of actions (one for each slot machine) – the value of action a is the expected reward $Q(a) = \mathbb{E}[r_t|a] = \theta$ (when at time t action a_t is choosing the i-th machine $Q(a_t) = \theta_i$).
- $\mathcal{R}$ is the reward function (a distribution on rewards). For a Bernoulli bandit,we observe a reward in a stochastic fashion ($r_t = \mathcal{R}(a_t)$ may return 1 with probability $Q(a_t)$).

This is a simplified version of a Markov decision process (there is no state $\mathcal{S}$).

(Weng 2018)

The greedy policy

- Produce an estimate for action values
- Select action with highest value $A_t = \arg\max_{a \in \mathcal{A}} \hat{Q}_t(a)$
- This is exploiting the currently available information.

ε-Greedy Strategy

- Take the best action most of the time: $\hat{a}_t^* = \arg\max_{a \in \mathcal{A}} \hat{Q}_t(a)$
- But with $p = \varepsilon$ do random exploration.

Best action is estimated from the collected action values from past experience (averaging the rewards for that action):

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{i=1}^{t} r_i \cdot \mathbb{1}(a_i = a)$$

$\mathbb{1}$ – binary indicator function for selecting an action

$N_t(a) = \sum_{i=1}^{t} \mathbb{1}(a_i = a)$ – counting how many times an action was selected

(Weng 2018)

ε-greedy Algorithm

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

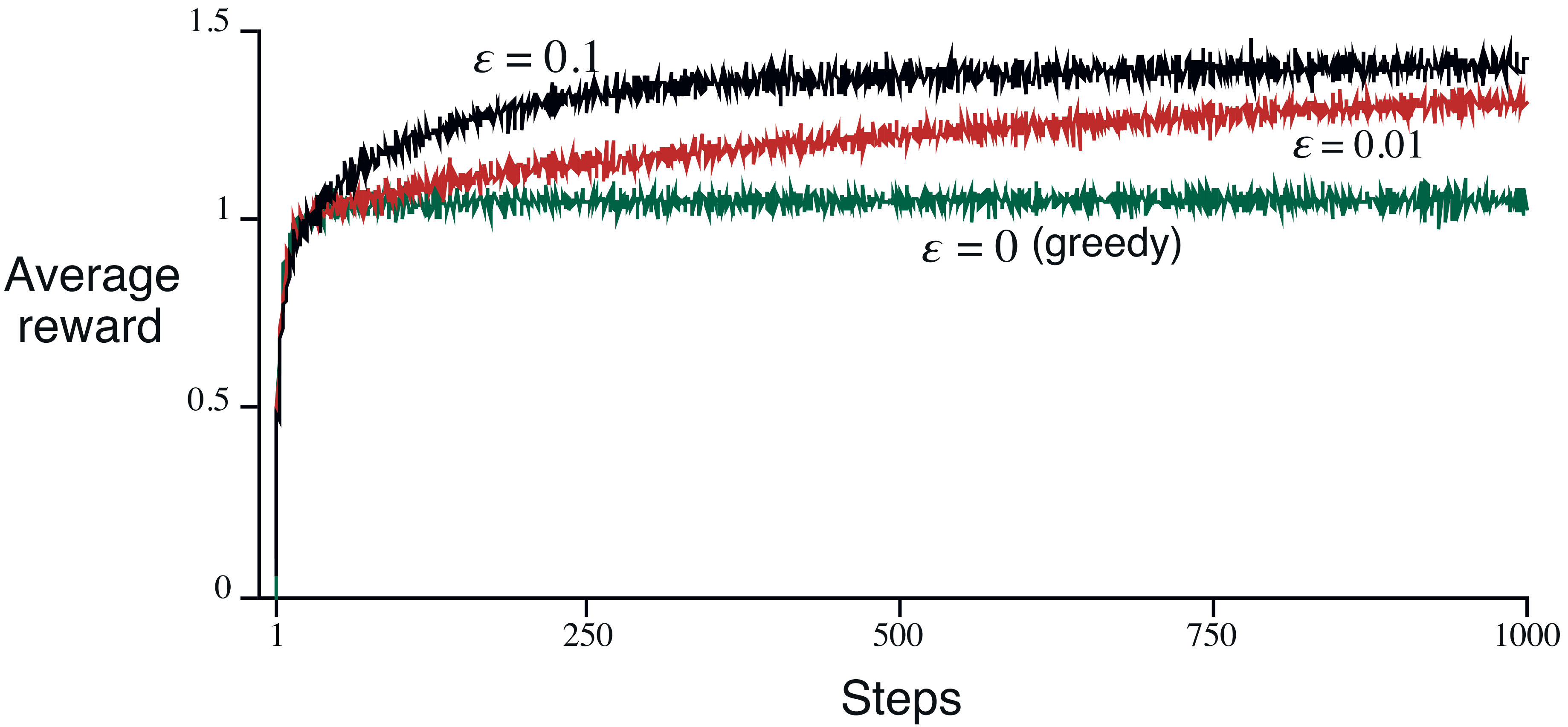
$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Note: Incremental implementation to update estimates of value function is efficient.

(Sutton und Barto 2018)

Performance of ε-Greedy Strategy



Average performance of ε-greedy method (averaged over 2000 runs) for 10-arm bandit problem.

(Sutton und Barto 2018)

Adaptation of ε-Greedy Strategy

Drawback of ε-Greedy Strategy:

- during exploration we are randomly selecting actions — even though we might already have established bad actions

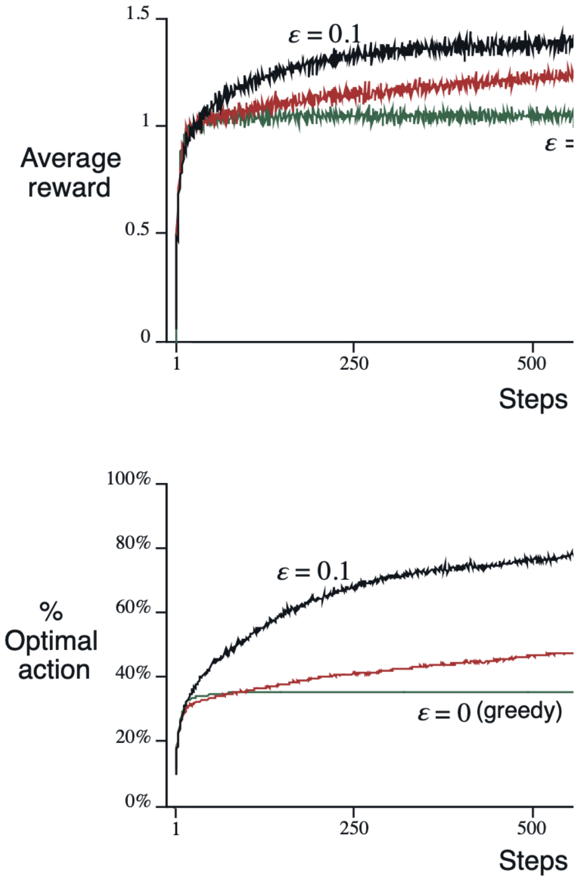
Possible Solutions

- decrease ε over time
- keep track of actions – of an estimate of how uncertain we are about this action (addressing the exploitation-exploration tradeoff)

Übungszettel Hinweise

Aufgabe 1.1: Implementation Multi-Armed Bandit

- Implementierung in Python eines $k = 4$-Bandit-Problems
- Abbildungen zu return über die Zeit und Anteil an Auswahl der optimalen Option (mehrere runs durchführen und auswerten)
- Einfluss des Parameters ε
- und der Initialbedingungen.
- ε über die Zeit verringern und Beobachtungen erklären.



(*Sutton und Barto 2018*)

Aufgabe 2: Stationarität von Multi-Armed Bandits

Voraussetzung der untersuchten Verfahren: **Stationäre Probleme** – die Wahrscheinlichkeiten für das Vergeben von rewards (und später Zustandsübergängen) bleibt gleich.

Aufgabe:

- Warum ist dies ein Problem für die vorgestellten Verfahren?
- Stellen sie dies dar durch eine Anpassung des Mult-Armed-Bandit-Problems.

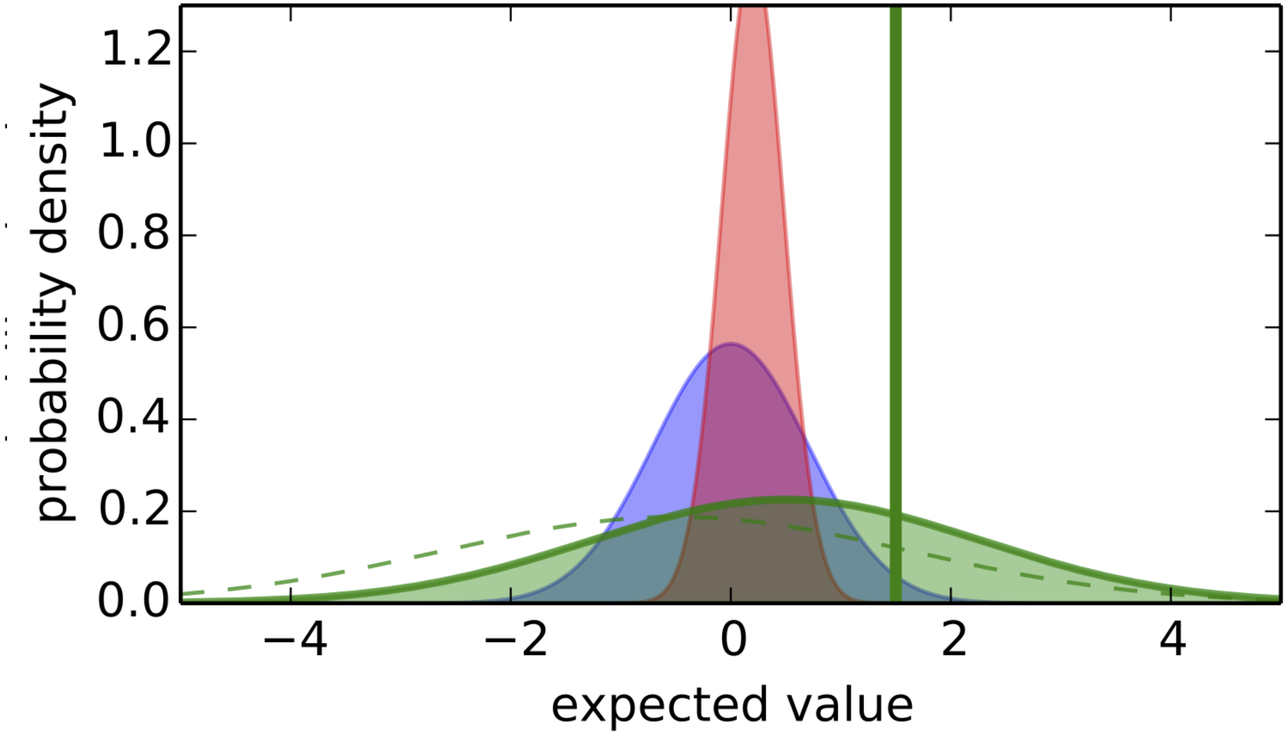
(als Zusatzpunkte!)

Übungstermin Freitag, 21.10.2022

Inhalte:

- Troubleshooting für Python Aufgaben
- Basierend auf dem Übungszettel: zusammenstellen der notwendigen numpy Funktionen (arrays, grundlegende statistische Funktionen, …) und plot Funktionen
- für eine random policy

Optimism in the Face of Uncertainty



The more uncertain we are about an action-value, the more important it is to explore that action as it could turn out to be the best action.

One approach: Keep an (over-)optimistic estimate for each action.

(Hasselt und Borsa 2021)

Upper Confidence Bounds (UCB)

Idea: favor exploration of actions that still have a strong potential to have an optimal value.

This potential is measured as an upper confidence bound of the reward value $\hat{U}_t(a)$. It depends on how often we have tried an action ($N_t(a)$).

Therefore, the true reward value is bound to:

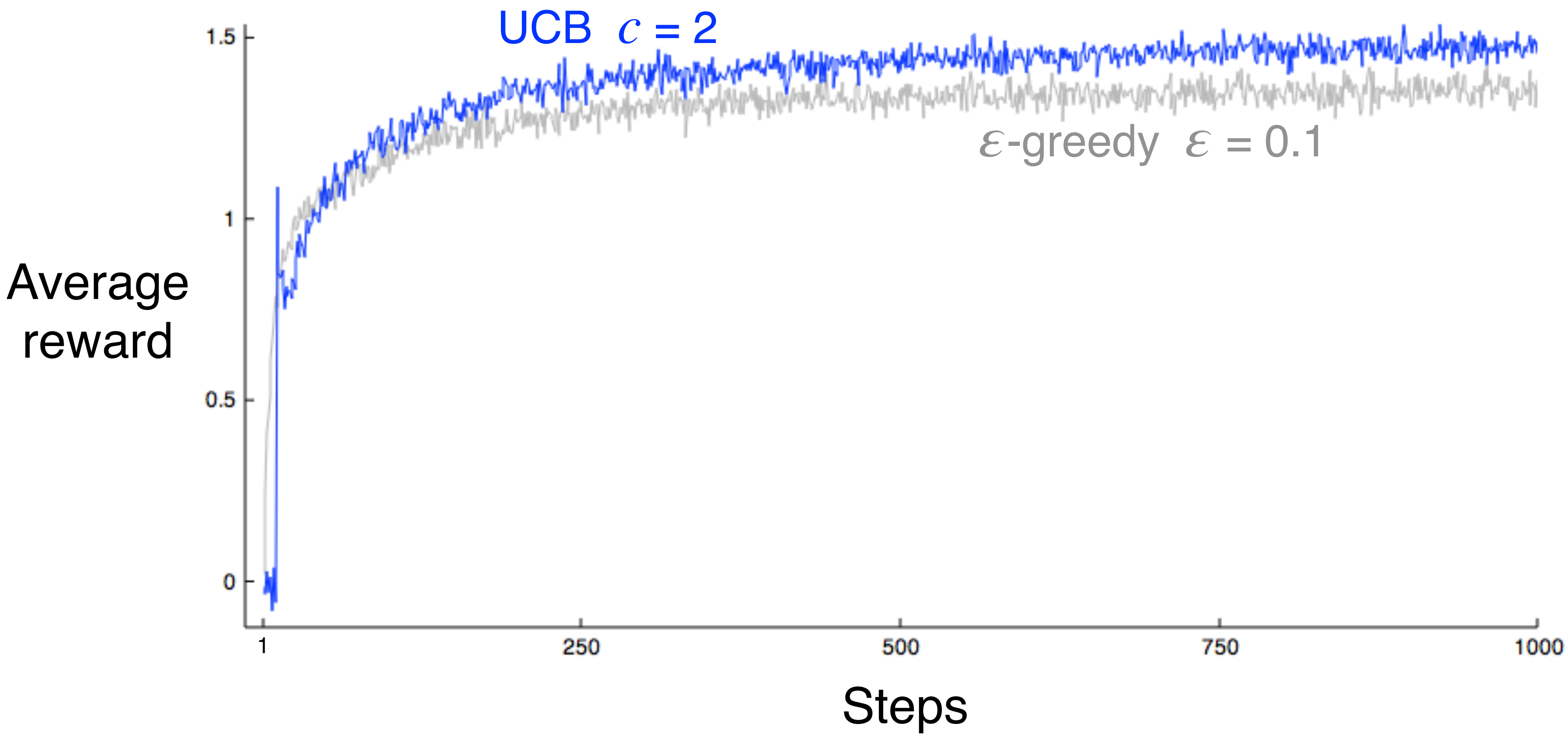
$$Q(a) \leq \hat{Q}_t(a) + \hat{U}_t(a)$$

In UCB algorithm, actions are selected greedily in order to maximize the upper confidence bound:

(Weng 2018)

$$a_t^{UCB} = \arg\max_{a \in \mathcal{A}} \hat{Q}_t(a) + \hat{U}_t(a) = \arg\max_{a \in \mathcal{A}} \hat{Q}_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}$$

Average Performance of UCB



Average performance of UCB (averaged over 2000 runs) for 10-arm bandit problem.

UCB outperforms ε-greedy.

(Sutton und Barto 2018)

Considering regret

Regret was defined as $\Delta_a = v_* - q(a)$.

It depends on

- the action count – how often each action was selected
- and on the overall collected regret $L_T = \mathbb{E}\Big[\sum_{t=1}^{T}\big(\theta^* - Q(a_t)\big)\Big]$.

Therefore, we should aim for algorithm that avoid large regrets.

Regret in the case of UCB

Selection of a_t:

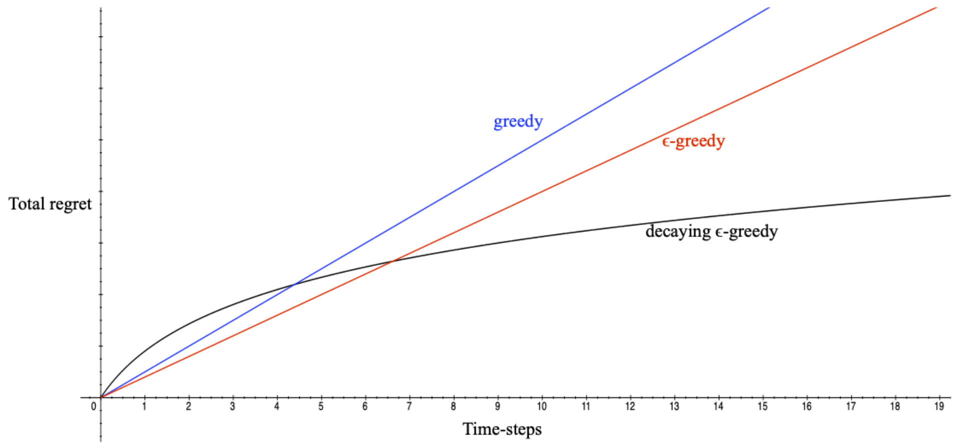
$$a_t = \arg\max_{a \in \mathcal{A}} Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}}$$

Intuitively,

- When Δ_a is large, the action will not be selected
- unless $N_t(a)$ is (comparatively) small.

It can be shown, that $\Delta_a \cdot N_t(a) \leq O(\log t)$ for all a.

Comparison of regret



Asymptotic total regret (Lai und Robbins 1985)

Asymptotic total regret is at least logarithmic in number of steps

$$\lim_{t\to\infty} L_t \geq \log t \sum_{a|\Delta_a>0} \frac{\Delta_a}{KL(\mathcal{R}_a||\mathcal{R}_{a*})}$$

(Silver 2015)

With $KL(\mathcal{R}_a||\mathcal{R}_{a*}) \propto \Delta_a^2$ – note: Hard problems have similar-looking arms with

with $KL(\mathcal{R}_a||\mathcal{R}_{a*}) \propto \Delta_a^2$ note: Hard problems have similar looking arms with different means. See here for explanation on Kullback-Leibler Divergence.

Policy – Directly learning how to act

Consider action selection as a probability distribution:

- For each action: Consider an (estimated) preference $H_t(a)$ of that action which
- can be directly used to express a probability for selecting that action (as a soft-max distribution)

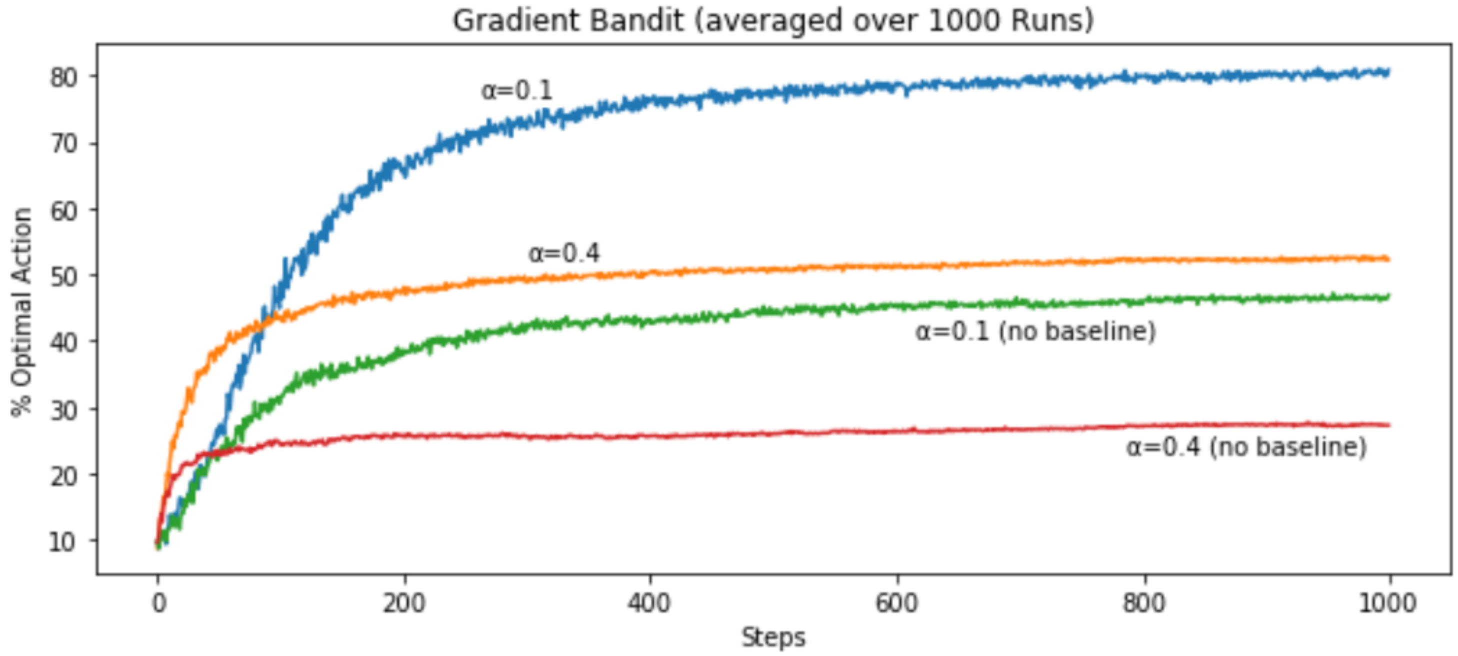
$$p(A_t = a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} = \pi_t(a)$$

Gradient Bandit Algorithm

Learn / adapt the action preference function directly using stochastic gradient ascent:

$$H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)), \qquad \text{and}$$
$$H_{t+1}(a) = H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a) \qquad \text{for all } a \neq A_t$$

Average Performance of Gradient Bandit Algorithm



Shown are variations of the α parameter and the importance of including the mean reward in the update for unbiasing (for brown curve this baseline is removed and the added bias of $+4$ affects learning).

Following (Sutton und Barto 2018).

Further variations for Solving Bandit Problems

- employing an optimistic estimate of the action value instead of a default initialization
- Bayesian UCB: introduce a prior assumption for the reward distribution – as a Gaussian – and use confidence intervals
- Thompson Sampling – formulate action selection as a probabilistic process itself (selecting an action with a probability that estimates it is optimal)

For further explanation see (Sutton und Barto 2018) or (Weng 2018).

References

Gao, Chongming, Wenqiang Lei, Xiangnan He, Maarten Rijke, und Tat-Seng Chua. 2021. „Advances and Challenges in Conversational Recommender Systems: A Survey".

Hasselt, Hado van, und Diana Borsa. 2021. „Reinforcement Learning Lecture Series 2021". https://www.deepmind.com/learning-resources/reinforcement-learning-lecture-series-2021.

Klein, Dan, und Pieter Abbeel. 2014. „UC Berkeley CS188 Intro to AI". http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

Lai, T. L., und H. Robbins. 1985. „Asymptotically Efficient Adaptive Allocation Rules". *Advances in Applied Mathematics* 6: 4–22.

Silver, David. 2015. „UCL Course on RL UCL Course on RL UCL Course on Reinforcement Learning". http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

Sutton, Richard S., und Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. Second. The MIT Press.

Weng, Lilian. 2018. „The Multi-Armed Bandit Problem and Its Solutions". https://lilianweng.github.io/posts/2018-01-23-multi-armed-bandit/.