FAKULTÄT FÜR WIRTSCHAFTSINFORMATIK
UND WIRTSCHAFTSMATHEMATIK

logo_uni_neu.pc

Prof. Dr. Leif Döring
Daniel Schmidt, Benedikt Wille

Programming task:
Bandits

Reinforcement Learning
16.03.2026

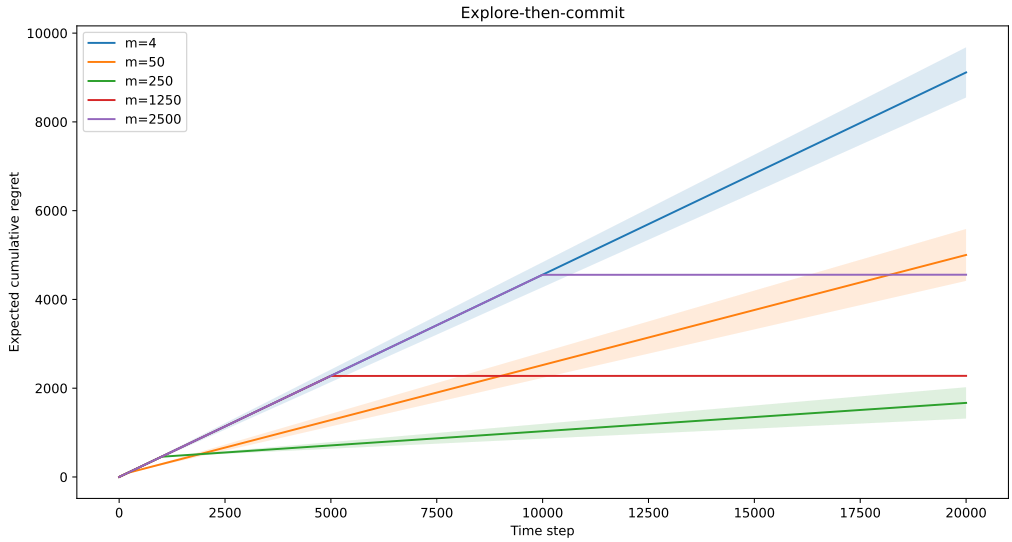
All experiments, if not stated otherwise, were performed on a 4-armed gaussian bandit with

a	0	1	2	3
μ_a	0.1257302210933933	-0.1321048632913019	0.6404226504432821	0.10490011715303971
σ_a^2	2.1327161474381335	7.961083366591781	1.826461730187002	1.9691338123491897
Δ_a	0.5146924293498888	0.7725269137345839	0	0.5355225332902424

and optimal arm 2. The sum over the reward gaps is 1.8227418730747152, its average is 0.4556854682686788. The algorithms were run for $n = 20000$ time steps and averaged over 500 seeds. The confidence intervals in the plots are 95% confidence intervals.

a) What are the typical regret rates observed for each algorithm? Based on this, which algorithm performs best? Provide an intuitive explanation of how these rates arise and what constants should appear in front of them.

Explore-then-commit:



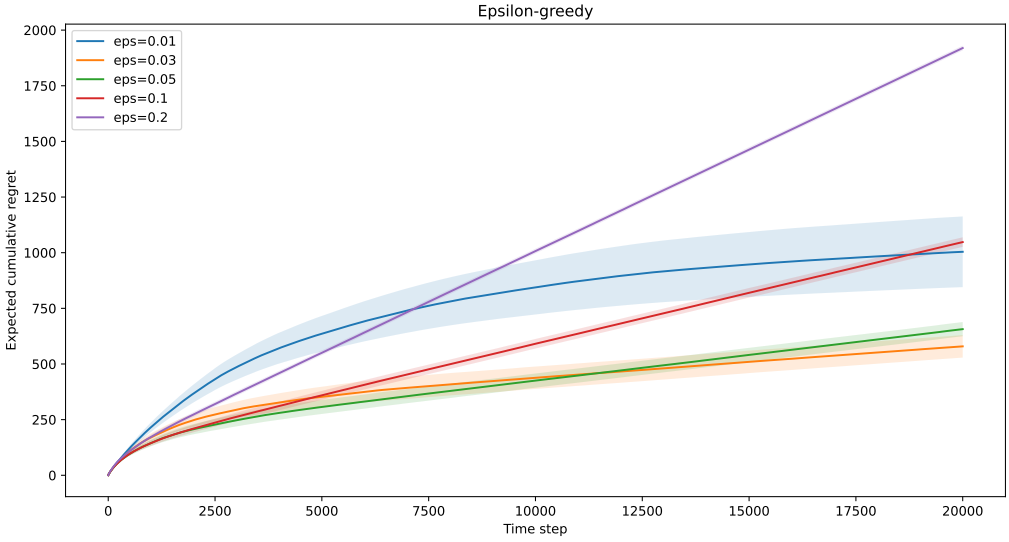
The explore-then-commit algorithm has a regret bound of

$$R_n(\pi) \leq m \sum_{a \in \mathcal{A}} \Delta_a + (n - mK) \sum_{a \in \mathcal{A}} \Delta_a \exp\left(-\frac{m\Delta_a^2}{4\sigma^2}\right)$$

and thus linear regret in n for any fixed m. The first term represents the exploration phase, where the algorithm accumulates regret proportional to the reward gaps because it explores all arms. This can be seen in the plot, where the different curves branch off a common diagonal after m steps whose steepness corresponds to the average reward gap 0.4556854682686788*. The second term represents the linear regret rate after branching off. The higher m, the smaller the*

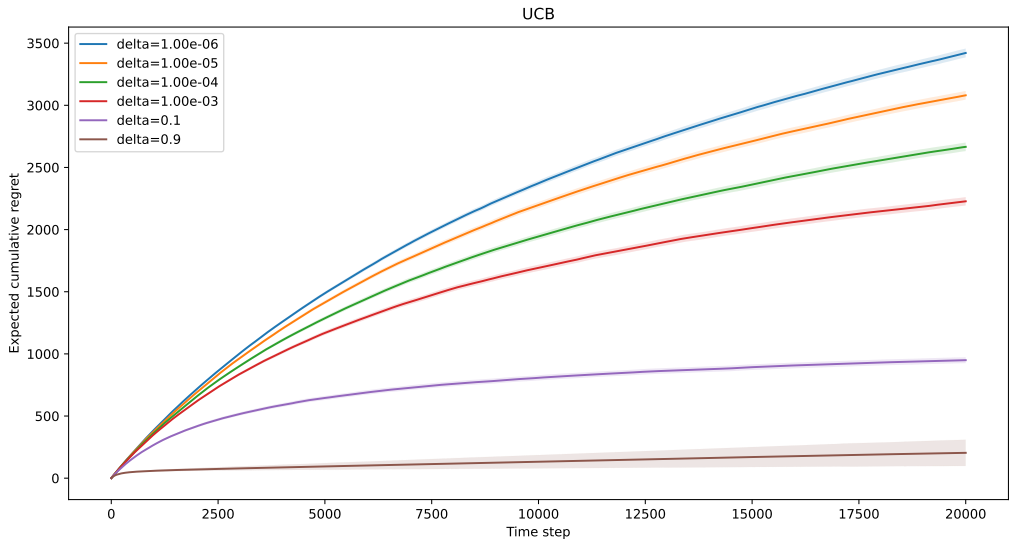
factor in front of the linear term, representing how likely the algorithm is to know the optimal arm after m exploratory steps. This can again be seen in the plot, as the steepness of the curves decreases with increasing m. We can see the exploration-exploitation tradeoff clearly, as on the plotted timescale, $m = 250$ is the best choice. Smaller m incurs a higher regret due to the second term and the exploitation phase occuring too early. Higher m incurs a higher regret due to the first term and the exploration phase being too long.

ϵ-greedy:



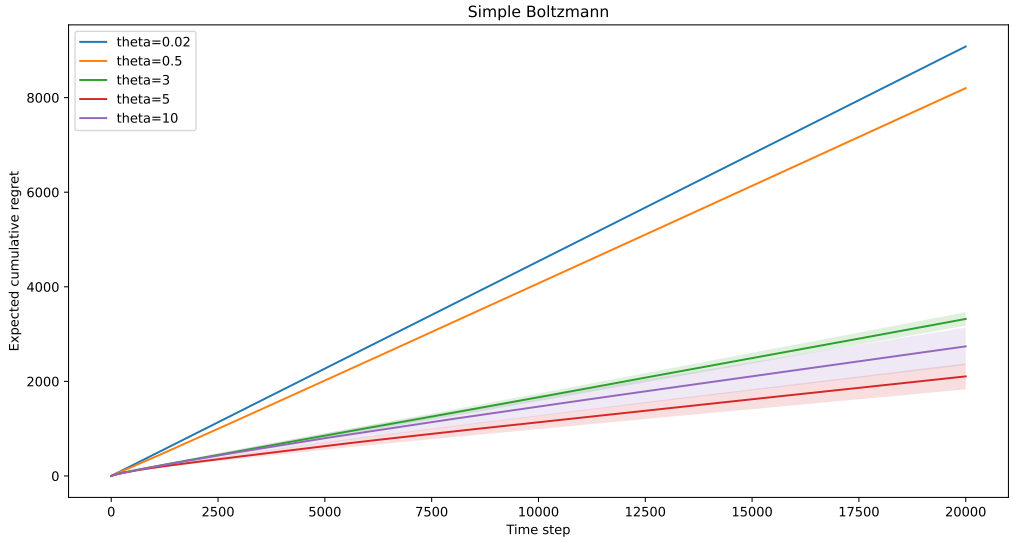
In the lecture we only derived an asymptotic regret bound for decreasing exploration rates that was logarithmic. The plots seem to hint that some of the fixed choices invoke a logarithmic regret rate while some invoke a linear regret rate. The higher the choice of ϵ, the more it looks like a linear rate, the lower, the more it looks like a logarithmic one. In reality, there are two contributions to the regret. One of them is linear and proportional to the average reward gap times ϵ and can be deduced by simply accounting for the average regret occuring during exploration. This can be seen best with the curve corresponding to $\epsilon = 0.2$, where it almost matches a linear function with this slope. The other contribution is logarithmic in nature and corresponds to the regret incurred during the exploitation phase. In fact, if we use the asymptotic bound from the lecture with $K = 4$, $C = 2$, and $d = 0.5$ we recognize that for $\epsilon = 0.03$ the curve seems to approximate the logarithmic curve with constant $\frac{(K-1)C}{d^2}$ quite well. In general, the higher ϵ, the more the linear contribution dominates, and thus the more the curve looks like a linear function. The lower ϵ, the more the logarithmic contribution dominates, and thus the more the curve looks like a logarithmic function. If we want to get rid of the logarithmic term, naturally, we need to send ϵ to zero. Another interesting effect is that the constant in front of the logarithmic term grows for smaller fixed values of ϵ. This is due to the algorithm committing on average longer on a suboptimal arm if the probability of choosing other arms at random and thus revealing the subobtimality of the committed arm is too low.

UCB:



Although in the lecture we only derived a regret bound for the UCB algorithm with a special choice of δ, the plot hints at a logarithmic regret growth even for other choices (which is indeed the case). We can see that with decreasing δ, the regret increases. Delta controls the confidence level of the upper confidence bounds, and thus how much the algorithm distrusts its estimation. The higher the delta, the likelier it will explore other options than the one currently deemed best. If it does so too often, it can choose suboptimal arms too often, while if delta is too small, it might take a longer time to realize if the algorithm is stuck on a suboptimal arm. However, in this case, probably because the reward gaps are fairly high and we set the sigma parameter to 7, while most arms only have a relatively small variance, quick exploration in the beginning followed by more exploitation seems to work very well.

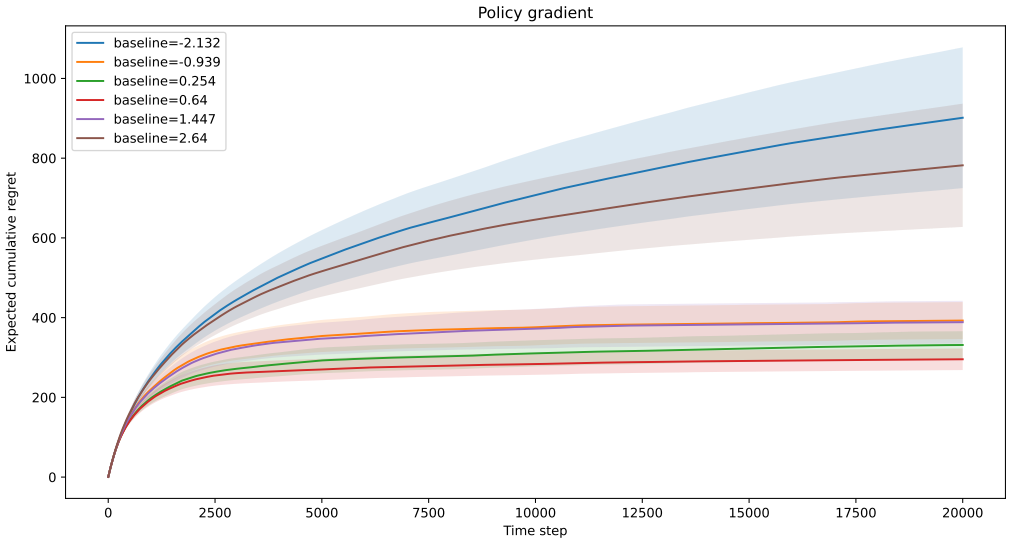
Boltzmann Exploration:



The Boltzmann exploration can either be seen as using Gumbel distributed exploration bonuses similar to UCB or as always using softmax instead of (ϵ-)greedy action selection. The temperature in both cases controls how much the algorithm favors actions with higher estimated

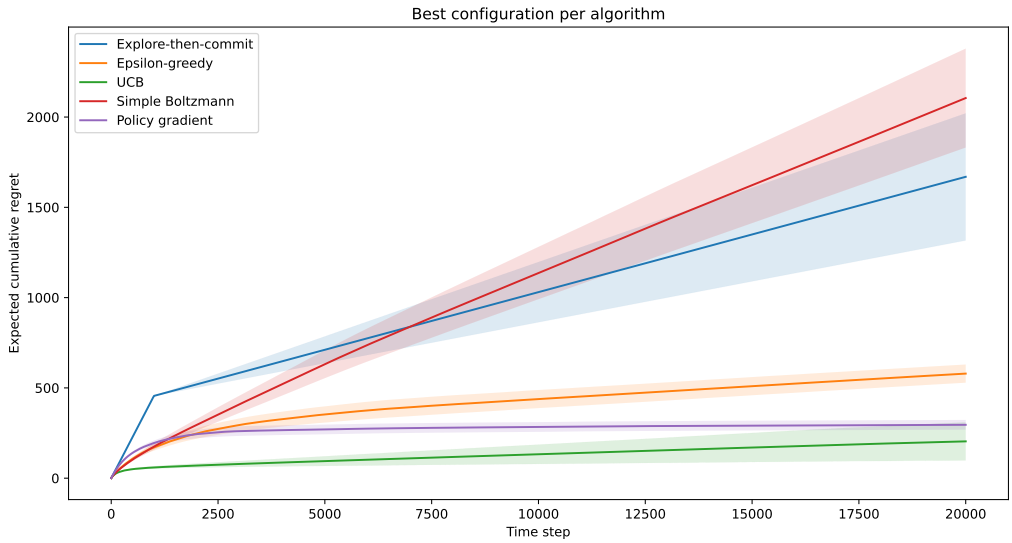
rewards. The higher the temperature, the more it favors actions with higher estimated rewards, and thus the more it exploits. The lower the temperature, the more it favors actions with lower estimated rewards, and thus the more it explores. The plots indicate that the Boltzmann exploration incurs linear regret. We can indeed see that for very small temperatures, where action selection is basically uniformly random, the regret is linearly growing with a slope given by the average reward gap. However, too high of a temperature is also suboptimal, as it can lead to the algorithm committing on a suboptimal arm for a long time. In this case, however, the slope constant is unclear.

Policy Gradient:



For the plots, a constant stepsize of 0.1 *was used. Later in this course and in following courses, we will learn which stepsizes are suggested by theory and which are used in practice. For the moment, the choice of stepsize may remain arbitrary though. The plot indicates that the policy gradient algorithm may incur logarithmic regret, but not much is known to this end. We can observe that a good baseline seems to be to to use the highest value of the arms. This is not entirely in line with the interpretation that it should be an estimate of the average reward in order to reduce variance, but as you may see in following courses is, what practicioners commonly use in more advanced settings.*

Which algorithm perfoms best?



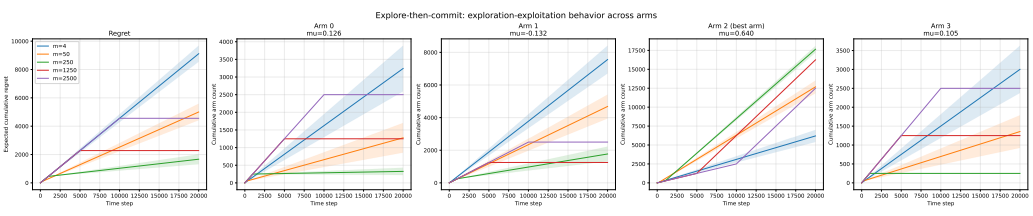
Looking at the plot of the best tested parameter for each algorithm and comparing them, we can say that in this case, the UCB algorithm with $\delta = 0.9$ performs best with respect to the cumulative regret. At the same time, there are some heavy limitations:

- *We did not test all possible parameters for each algorithm, so it could be that there are better configurations for some of the algorithms that we did not test.*
- *We only tested a single bandit instance, so it could be that the results are not generalizable to other bandit instances. In particular, the performance of the algorithms can depend heavily on the reward gaps.*
- *We could consider best to include other things such as robustness over different hyperparameter choices (especially if you combine this with the previous points).*
- *For some algorithms, for example policy gradient, we do not have a theoretical understanding of the regret rates, so it could be that the rates look logarithmic but are in fact not logarithmic.*

If we wanted to make a more general statement about which algorithm performs best, we would need to test more configurations and more bandit instances. However, generally algorithms with logarithmic regret growth are always to be preferred over algorithms with linear regret growth, so we can say that in general one would probably prefer the UCB algorithm and possibly the Policy Gradient if it turned out to show logarithmic regret growth in general.

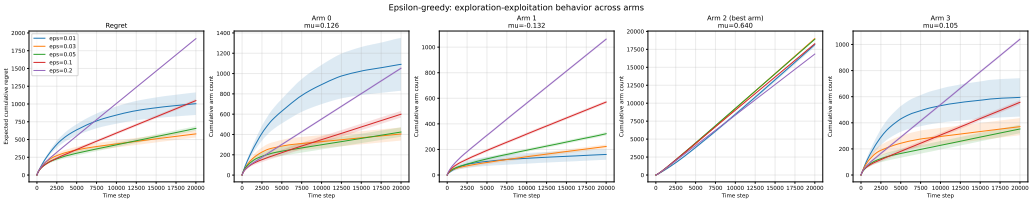
b) Explain the effects of the exploration-exploitation tradeoff and committal behavior. Use appropriate graphs to illustrate these effects in your data. Discuss the differences between the algorithms in this context.

Explore-then-commit:



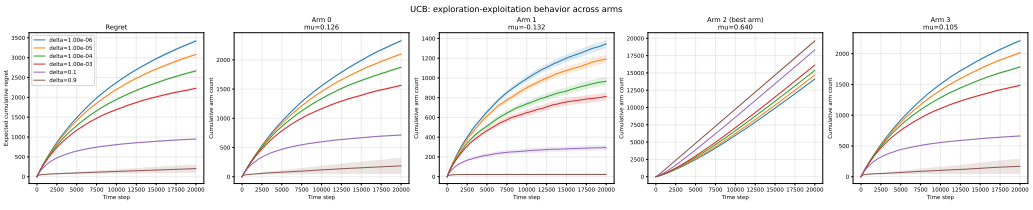
Both effects can be illustrated best using the explore-then-commit algorithm since it separates the exploration from the exploitation phase cleanly. First, a number of exploration steps m is executed, after which the algorithm commits to the arm with the highest estimated reward. The higher m, the more exploration takes place and the more confident we can be that the algorithm finds the best arm. Indeed, the arm counts of the plot reveal that after the exploration phase, the higher m is, the steeper the cumulative arm count increases, meaning that the algorithm committed to the optimal arm with a higher probability. This is reflected in the slope of the regret curve after the exploration phase as well. However, the smaller m is, the less time we spend exploring suboptimal arms, so that the cumulative arm count until the time of committal for each of the suboptimal arms is much higher for larger m. In the plots, we can see that for $m = 2000$ for some of the suboptimal arms it takes time for the cumulative arm counts of the other choices to overtake it, meaning that during exploration we have spent a lot of time on those suboptimal choices. Since we know that arm count of the optimal arm increases faster the higher m is we know that eventually the regret for this choice will be smaller than for any other choice. However, in the bandit setting we always consider a fixed number of timesteps n, meaning we need to consider both the negative effects of too much exploration and too early exploitation. This is what is called the exploration-exploitation tradeoff. Committal behaviour is the phenomenon that an algorithm commits to a suboptimal arm too early. We can see that the explore-then-commit algorithm is particularly prone to this behavior the smaller m is, as we can clearly see that even for the suboptimal arms the average arm count can increase quite drastically.

ϵ-greedy:



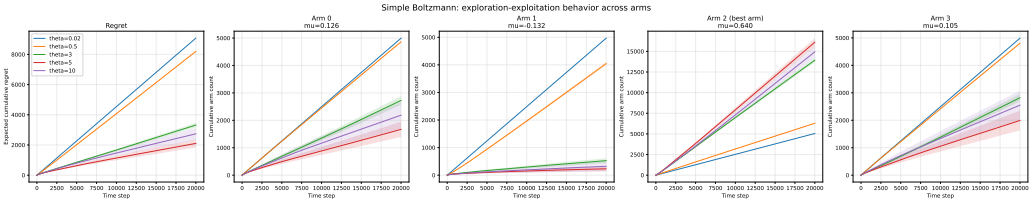
The ϵ-greedy algorithm mixes exploration and exploitation by choosing to explore with a fixed probability ϵ while exploiting with probability $1 - \epsilon$. Thus, we can see that this parameter directly controls the exploration-exploitation tradeoff. In all cases the cumulative arm count of the best arm increases linearly save for the beginning, where the algorithm can get falsely stuck on suboptimal arms. At the suboptimal arms, the arm counts exhibit linear behavour as well that is proportional to ϵ, but perturbed in the beginning, since we can get falsely stuck with explotin those arms as well. Committal behaviour rarely occurs outside of the beginning, but we still explore for a fixed fraction until the end, clearly defining the magnitude of the exploration-exploitation tradeoff.

UCB:



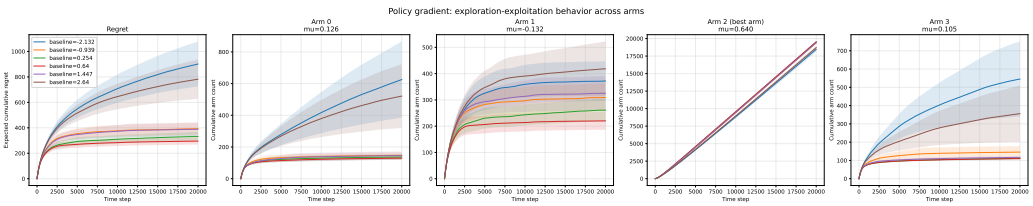
The UCB Algorithm is the first of the algorithms that exhibits ever growing rates on the arm counts for the optimal arm and ever decreasing ones for the rest of the arms. This is because it is uncertainty-aware in the sense that it always adds an exploration bonus proportional to the amount of times it visited the arms. Additionally, for arms with lower values, it is more certain to have the correct estimate sooner, so that the total arm count is much smaller for arm 1 than the rest of the suboptimal arms at all times. Thus, it can not only identify the best arm, but plays the arms less and less the worse they get. Committal behavor rarely occurs outside of the beginning and gets increasingly more unlikely. The exploration-exploitation tradeoff happens automatically and is steered by δ, where we lean more toward exploitation the more steps the algorithm took.

Boltzmann Exploration:



We can clearly see both effects at play in the Boltzmann exploration as well. Here, the temperature controls how much the algorithm leans towards exploring with a softmax. The higher the temperature, the more exploitation occurs, while very small temperatures lead to almost uniform exploration. We can see that for very small temperatures, the arm counts of all arms increase at a similar rate, while for larger temperatures, the arm counts increase much faster for the optimal arm. However, the higher the temperature, the more we lean towards a purely greedy algorithm, meaning that we can get stuck on suboptimal arms for a long time if they have positive average reward. This can be seen in the plots as well, as for $\theta = 20$, the arm counts at arms 0 and 3 increase much faster than for $\theta = 5$. All in all, again, we need to balance the exploration with the exploitation in order to learn well while avoiding commital behavior.

Policy Gradient:



The shape of the arm counts for the policy gradient algorithm is similar to the one of the UCB algorithm. Over time, the probability of choosing the optimal arm is reinforced more

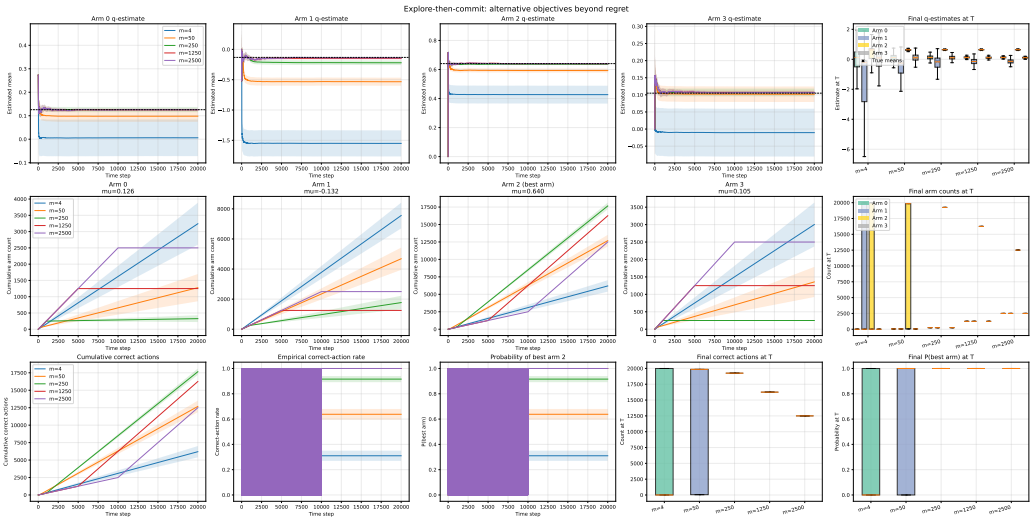
and more, while the smaller the real mean values, the less they are reinforced, leading to the same exploration-exploitation tradeoff and ever decreasing committal behaviour as for UCB. The role of the UCB's δ is here played by a combination of the stepsize and the temperature in the softmax parametrization, where higher stepsizes and temperatures lead to more exploration.

c) In the bandit setting, what objectives besides regret minimization could be relevant (especially in the RL context)? Compare the algorithms based on these alternative objectives using appropriate metrics. Provide intuitive explanations of how these metrics arise and determine which algorithm performs best according to them. Consider the number of correctly chosen actions, estimates of arm means, and the probabilities of choosing the optimal arms over time as well as at the end of the time horizon.

There are a lot of alternative metrics for bandits from which we can derive alternative goals. Most of them get much more relevant in the context of the standard Reinforcement Learning problem, so it makes sense to already think about them in the much easier context of bandits:

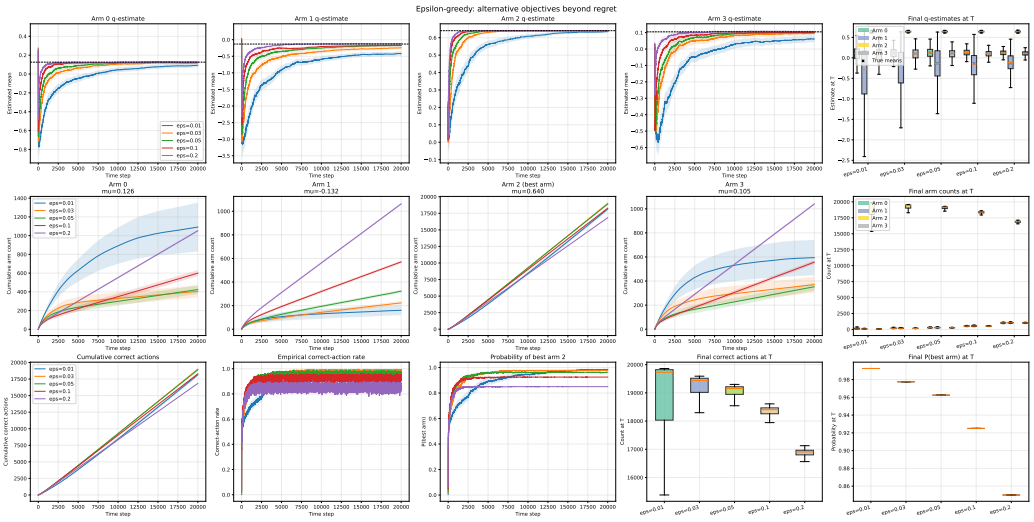
- ***Q-estimates**: These are the estimates of the average reward for each arm. They are relevant because they can be used to determine which arm is best. The more accurate the estimates are, the better the algorithm will choose the optimal arm. We can track these for each arm individually over time or look at the algorithm's final estimates at the end of the time horizon.*
- ***Arm counts**: These are the number of times each arm was chosen. They are relevant because they can indicate how much the algorithm explored each arm and how much it exploited the best arm, revealing insights into its exploration-exploitation balance. We can track these for each arm individually over time or look at the final arm counts at the end of the time horizon.*
- ***Correct action rates**: These are the rates at which the algorithm chose the optimal arm. They are relevant because they directly indicate how well the algorithm is performing in terms of actual action selection. We can track these as cumulative rates over time, as averages over time, or look at the final correct action rate at the end of the time horizon.*
- ***Probability of best arm**: This is the probability that the algorithm will choose the optimal arm. It is relevant because it indicates how likely the algorithm is to make the correct decision in the next step. We can track this over time or look at the final probability at the end of the time horizon.*

Explore-then-commit:



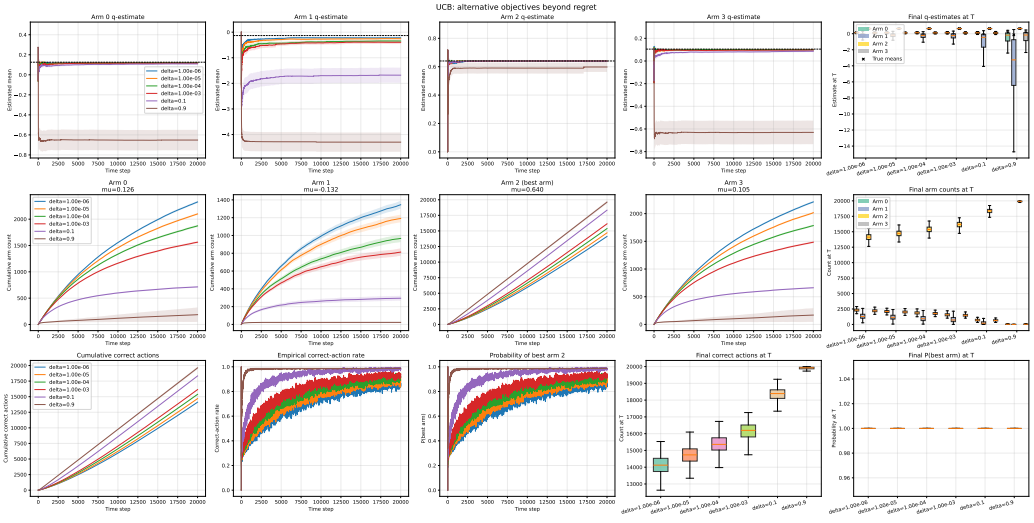
The explore-then-commit algorithm first explores all arms for a fixed number of steps m and then commits to the arm with the highest estimated reward. This means that during the exploration phase, the Q-Values will be estimated by a Monte Carlo estimator of the mean with sample size corresponding to m. During the committal phase, only the chosen arm gets further approximated, so the estimates of the other arms remain fixed. We can see that for higher values of m the estimates become more and more accurate. As is standard for Monte Carlo estimators, the variance also plays a role in how fast the estimator converges. Overall, we can see, that at the end of the time horizon, the optimal arm's estimated value is the most accurate. The intuition behind the arm counts was already explained in the second question, but we can see even clearer in the final arm count, that with growing m, the arm count for the optimal arm decreases while the arm count for suboptimal arms increases. The correct action rates again reflect, that until the exploration phase is over, it is 1 every four exploration steps (we used the round-robin scheme) and after that the higher m is, the higher is the correct action rate as well as the probability of choosing the correct arm. The number of correctly chosen actions at the end, however, does not necessarily increase, as during the exploration phase, suboptimal arms are chosen frequently, while the probability of choosing the correct arm in the end increases.

ϵ-greedy:



The ϵ-greedy algorithm plays each arm randomly with probability ϵ, so that on average, the Monte-Carlo estimator of the Q-values contains at least $\epsilon n/K$ samples for each arm, meaning eventually the Q-values will converge for every value of epsilon. Higher values mean faster convergence. Since the algorithm plays the arm it deems best most of the time, the estimate of the best arm converges the fastest and is the most accurate in the end. For the arm counts, we observe the same effect as for the explore-then-commit algorithm, where ϵ again steers the exploration-exploitation tradeoff. The correct action rates demonstrate that with all values of epsilon the algorithm will find the optimal arm, but due to the added random exploration, the correct action rate will always be lower than 1 and decreasing as ϵ increases. The same applies to the probability of choosing the optimal arm.

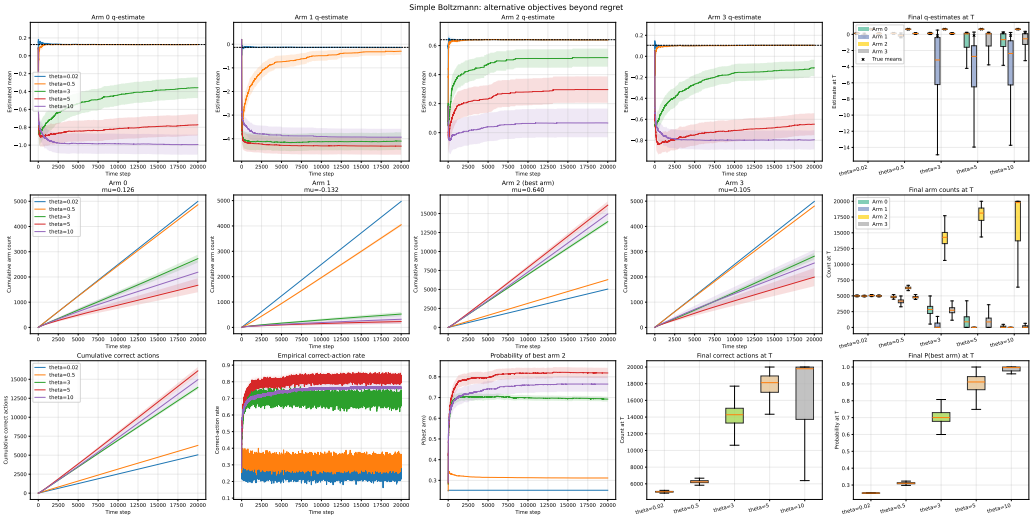
UCB:



The UCB algorithm uses an upper confidence bound to balance exploration and exploitation. Due to this behavior, it will play arms that have high uncertainty until the Monte-Carlo estimator is sufficiently accurate in order to see which arm is optimal. Thus, we can observe that the closer the Q-value is to the optimal one, the faster the Q-estimates converge, meaning the very negative arm 1 seems to be played a lot less. This has also been discussed regarding the arm

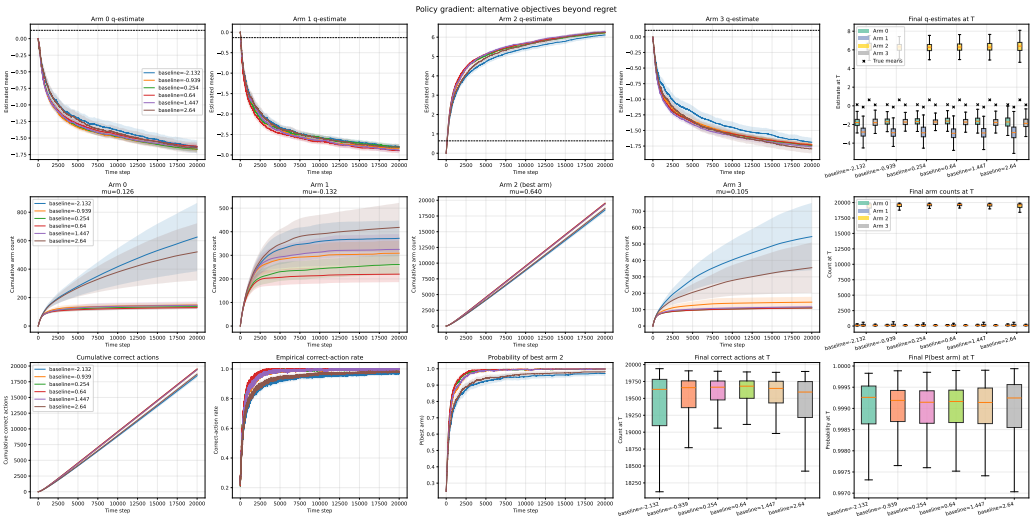
counts. We can furthermore see, that the correct action rates and the probability of choosing the correct action keep increasing to 1.

Boltzmann Exploration:



Since the Boltzmann exploration uses a softmax distribution to select arms, it will not necessarily learn the optimal Q-values fast, but only their relationship with each other. The more uniform we make the distribution, the faster the more accurate the estimates get. Like UCB, the closer the arm is to being optimal, the faster its estimate converges to the real one. As discussed before, we can see that the higher the temperature, and thus, the less exploration occurs, the more do the arm counts gravitate towards the optimal arm. Finally, for sufficiently high temperatures, we can see that the correct action rates as well as the probability of choosing the correct arm keep growing.

Policy Gradient:

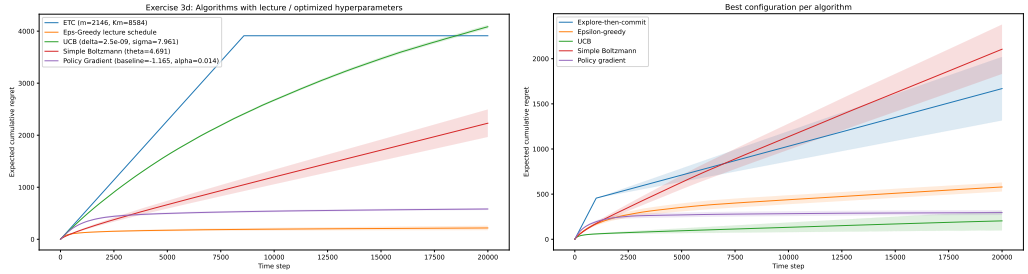


We can see that similar to the Boltzmann exploration, the arm's real Q-Values are not learned. In fact, they keep decreasing for suboptimal arms and increasing for the optimal arm. This is an artifact of the algorithm further reinforcing the probability of playing the right arm. The intuition

for the arm counts has been discussed above. Finally, the correct action rate and the probability of playing the correct action grow to 1 pretty fast.
In general, we may recognize that the optimal parameter in terms of cumulative regret is not always the optimal one for all metrics. For example, the arm counts at the optimal arm are influenced heavily by the exploration-exploitation tradeoff, meaning that even though an algorithm may choose the correct arms less frequently on average, it may still be better in terms of regret. Another example would be the correct action rates, whose upper bound in the case of the ϵ-greedy algorithm only depends on ϵ. So if we want to definte what constitutes an optimal parameter we need to be very careful with respect to what metric we are doing so. Another interesting line of thought would be when do we consider that the algorithm has learned anything? Do we mean it minimizes Regret? Achieves good estimates for the Q-Values? Or should it maximize the correct action rates and the probability of choosing the correct action?
For the estimation of the Q-Values there is an interesting phenomenon that can be observed throughout the algorithms that use mainly Monte-Carlo estimators. We tend to underestimate the Q-Values. This is due to the fact that if we sampled rewards from a symmetric distribution independently, we are equally as likely to be receive a sample higher than the mean as one that is lower. However, if we receive one that is lower, we usually stop playing that arm for a while, so that the estimator remains smaller than the average. This downward drift can only be overcome by sampling an larger amount of times. Conversely, there are algorithms that tend to overestimate the values for a variety of reasons that may become clearer later on in the course. Keep these two types of biases in mind, you may encounter them in Reinforcement Learning examples later on. In general, the estimation on the optimal arm will be the most accurate, as we will play it the most amount of times. Later in Reinforcement Learning, we will see that this metric will become very important and ϵ-greedy performs best at it, since it always comes back to exploring all arms, hence why it will be used in Reinforcement Learning extensively.
Arm Counts are heavily influenced by the exploration exploitation tradeoff and can't be reliably taken as a performance metric on their own. However, in general, if the slope of the arm counts on the optimal arm keeps increasing, we can infer that the regret is sublinear and in this sense we have a good algorithm. Similarly, we can infer that the correct action rates and the probability of choosing the correct arm keep increasing if this is the case.
Intuitively speaking, for good algorithms that learn something, we could postulate, that the correct action rates and the probability of choosing the best arm should go to 1 with time. However, picture an agent that learns via the ϵ-greedy algorithm but remove the exploration at the end of the time horizon. Now the correct action rates will be much higher and arbitrarily close to 1 for good parameters. Indeed, these metrics on their own are typically not a good indicator of how good an algorithm is for this reason.

d) Use the optimal parameters from the lecture and think about how to estimate these parameters numerically for each algorithm. Compare your results in short to your findings in a)–c). For which algorithms does choosing the optimal parameters require „cheating“? How difficult is it to numerically determine the correct parameters? Based on your findings, which algorithm do you conclude is the best?

In the following graph we used two types of optimization. For Explore-then-commit, ϵ-greedy, and UCB we have regret upper bounds, which we can optimize given the fixed time horizon. For the other two algorithms we did a random search over the possible domain and evaluated each parameter choice on 20 random seeds to decide which parameter choice is the best.



We can see that even our naive approach to parameter tuning can be much better than the optimization over the regret upper bounds. This is, because they are upper bounds for all bandit models and thus not always very accurate. In fact, even our best guess from part a) was sometimes better than tuning the parameters given the regret upper bounds. Additionally, all of our regret upper bounds use knowledge about the game that we would normally not assume in the bandit setting. A notable exception seems to be the ϵ-greedy algorithm with decreasing stepsizes, where the performance was made much better even though the assumptions on the bandit were not fulfilled, demonstrating how important it is to consider the constants of the upper bounds and to know in what regime they are typically sharp.

Our random search method was quite primitive, as we will see in further courses, when we learn a bit more about hyperparameter optimization, and thus it yielded no substantial improvement with respect to the previously guessed parameters, demonstrating that a good intuition can still be comparably good to naive brute force.