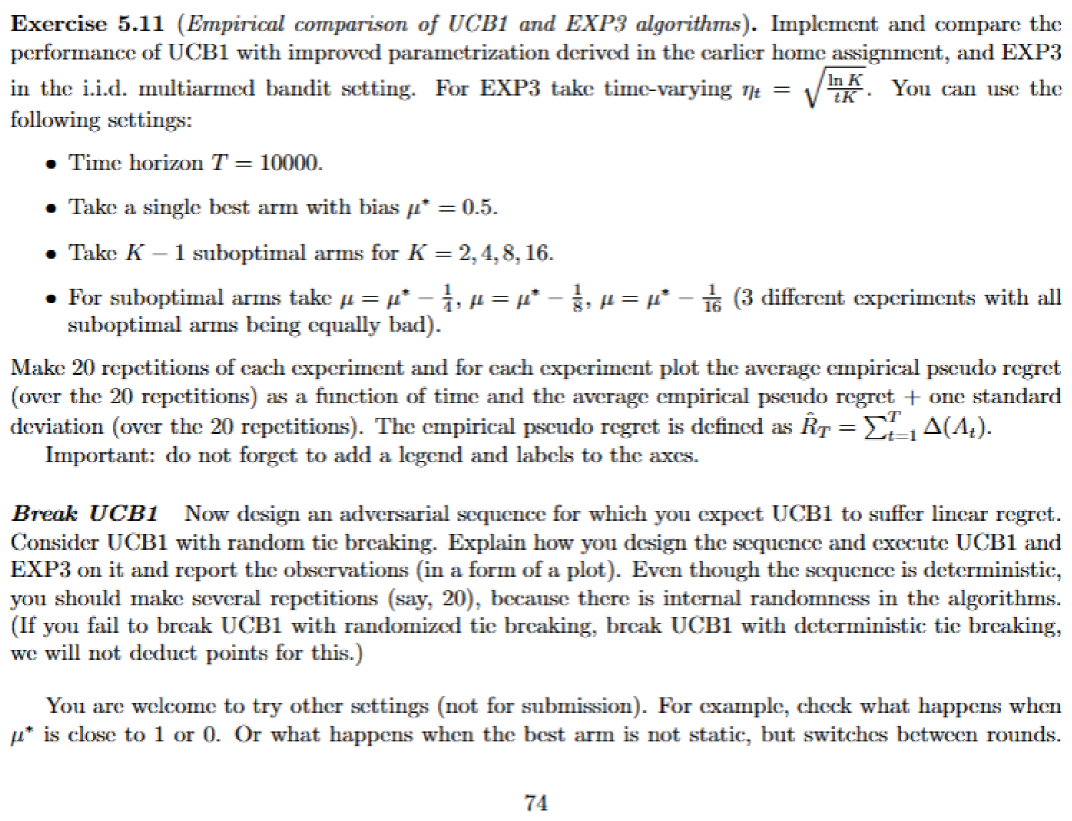


**Exercise 5.11** (*Empirical comparison of UCB1 and EXP3 algorithms*). Implement and compare the performance of UCB1 with improved parametrization derived in the earlier home assignment, and EXP3 in the i.i.d. multiarmed bandit setting. For EXP3 take time-varying $\eta_t = \sqrt{\frac{\ln K}{tK}}$. You can use the following settings:

- Time horizon $T = 10000$.
- Take a single best arm with bias $\mu^* = 0.5$.
- Take $K - 1$ suboptimal arms for $K = 2, 4, 8, 16$.
- For suboptimal arms take $\mu = \mu^* - \frac{1}{4}$, $\mu = \mu^* - \frac{1}{8}$, $\mu = \mu^* - \frac{1}{16}$ (3 different experiments with all suboptimal arms being equally bad).

Make 20 repetitions of each experiment and for each experiment plot the average empirical pseudo regret (over the 20 repetitions) as a function of time and the average empirical pseudo regret + one standard deviation (over the 20 repetitions). The empirical pseudo regret is defined as $\hat{R}_T = \sum_{t=1}^{T} \Delta(A_t)$.

Important: do not forget to add a legend and labels to the axes.

***Break UCB1*** Now design an adversarial sequence for which you expect UCB1 to suffer linear regret. Consider UCB1 with random tie breaking. Explain how you design the sequence and execute UCB1 and EXP3 on it and report the observations (in a form of a plot). Even though the sequence is deterministic, you should make several repetitions (say, 20), because there is internal randomness in the algorithms. (If you fail to break UCB1 with randomized tie breaking, break UCB1 with deterministic tie breaking, we will not deduct points for this.)

You are welcome to try other settings (not for submission). For example, check what happens when $\mu^*$ is close to 1 or 0. Or what happens when the best arm is not static, but switches between rounds.

74

Even though you can break UCB1, it is actually pretty robust, unless you design adversarial sequences that exploit the knowledge of the algorithm.

## 1 Deep Q-Learning (50 points) [Christian]

We (re-)consider the *Lunar Lander* environment from the Gymnasium framework, see `https://gymnasium.farama.org/environments/box2d/lunar_lander/` for details of the RL task. You may need to install Gymnasium via `pip install gymnasium[box2d]` and perhaps some other packages.

The notebook `LunarLanderDQN2025Assignment.ipynb` implements a simple deep Q-learning except for one line. If you have problems with the visualization, you can comment out the corresponding lines (try to get it running, it is fun).

### 1.1 Neural architecture

The policy network definition contains the lines:

```
    def forward(self, x_input):
        x = F.tanh(self.fc1(x_input))
        x = F.tanh(self.fc2(x))
        x = torch.cat((x_input, x), dim=1)
        x = self.output_layer(x)
        return x
```

#### 1.1.1 Structure (5 points)

What is `x = torch.cat((x_input, x), dim=1)` doing? What is this neural network structure concept called?

#### 1.1.2 Activation function (5 points)

Why has the tanh activation function been chosen instead of, say, the standard logistic function?

### 1.2 Adding the Q-learning (15 points)

Add the missing line to compute the variable `y` setting the actual Q-learning targets. The line should be added to the notebook after the comment line `# Compute targets`.

After that, the notebook should reliably find a policy solving the task with a return larger than 200 (of course, there might be bad trials that do not find a sufficiently good solution).

Report the line in the report and *briefly* explain what it is doing. Describe the equation it implements using the notation introduced in the lecture.

The notebook saves a plot of the learning process in the file `deepQ.pdf`. Show a successful learning process in the report by including the corresponding plot (or a beautified version of it).

### 1.3 Epsilon (5 points)

Explain what the line

```
    explore_p = explore_stop + (explore_start - explore_stop)*np.exp(-decay_rate*ep)
```

is doing.

### 1.4 Gather (5 points)

Explain what the line

```
    Q_tensor = torch.gather(output_tensor, 1, actions_tensor.unsqueeze(-1)).squeeze()}
```

is doing and why it is necessary (it is for a data structure/programming reason, not a theoretical one).

### 1.5 Target network (15 points)

Introduce a delayed target $Q$ update via a target network `targetQN`. Smoothly blend the parameters:

$$\theta_{\text{target}} \leftarrow \tau\theta_{\text{main}} + (1-\tau)\theta_{\text{target}}$$

for $\tau \in ]0,1[$ (sometimes referred to as Polyak averaging), where $\theta_{\text{main}}$ and $\theta_{\text{target}}$ are the parameters of the networks `mainQN` and `targetQN`, respectively.

The code for updating the target network can be added directly after the gradient-based update of the main network. You can get the parameters of network like this:

```
        sdTargetQN = targetQN.state_dict()
```

You can change the values in this data structure as follows:

```
        for key in sdTargetQN:
            sdTargetQN[key] = 42.
```

You can overwrite the parameters using `load_state_dict`.

Report the code you added in the report and *briefly* explain what it is doing.

Keeping all other hyperparameters from the notebook unchanged and the same for both approaches, empirically compare the performance with and without target network. Do you observe a difference?

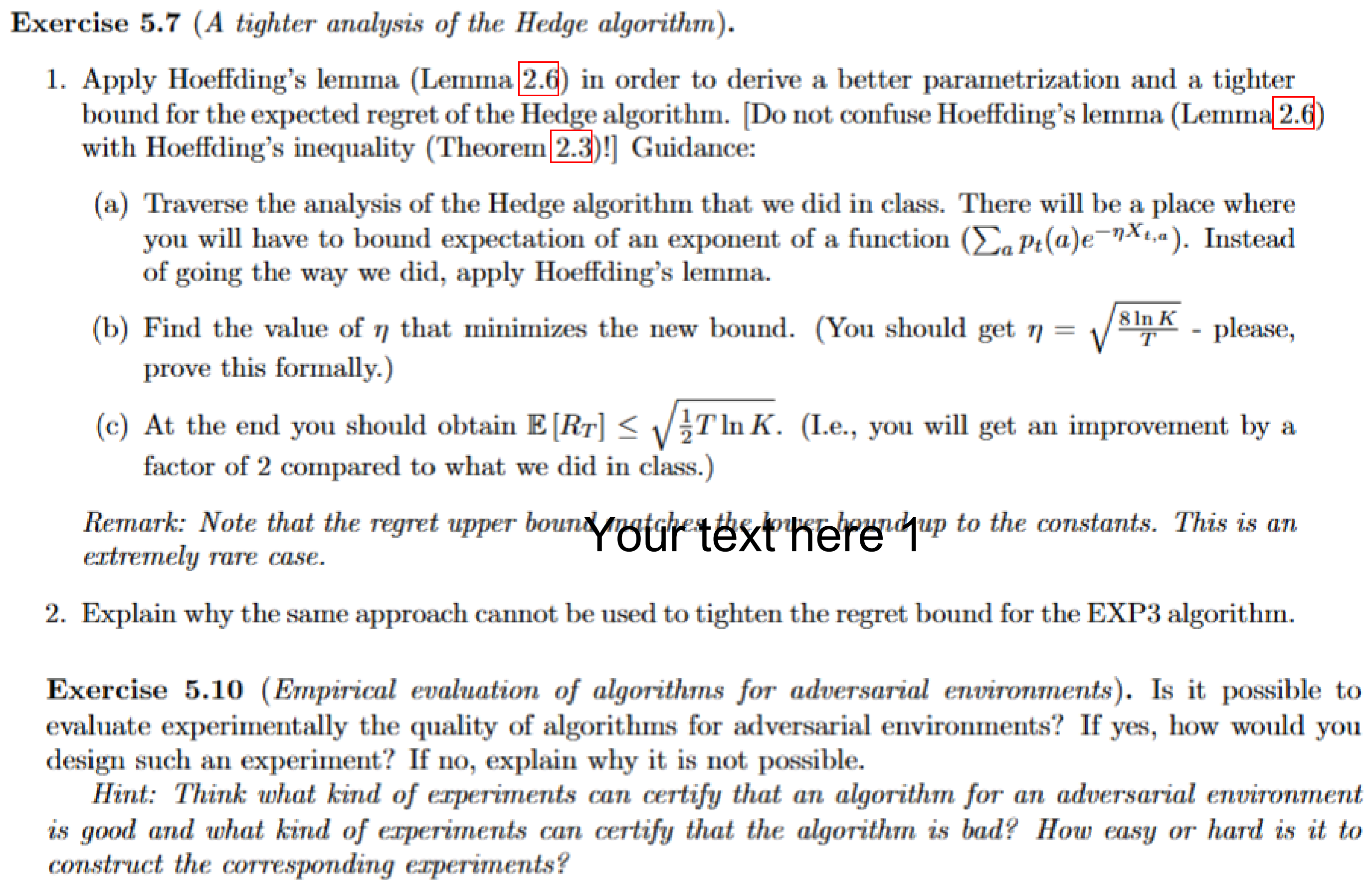
## 2 A tighter analysis of the Hedge algorithm (20 points) [Yevgeny]

Solve Exercise 5.7 in Yevgeny's lecture notes.

## 3 Empirical evaluation of algorithms for adversarial environments (5 points) [Yevgeny]

Solve Exercise 5.10 in Yevgeny's lecture notes.

## 4 Empirical comparison of UCB1 and EXP3 algorithms (25 points) [Yevgeny]



**Exercise 5.7** (*A tighter analysis of the Hedge algorithm*).

1. Apply Hoeffding's lemma (Lemma 2.6) in order to derive a better parametrization and a tighter bound for the expected regret of the Hedge algorithm. [Do not confuse Hoeffding's lemma (Lemma 2.6) with Hoeffding's inequality (Theorem 2.3)!] Guidance:

   (a) Traverse the analysis of the Hedge algorithm that we did in class. There will be a place where you will have to bound expectation of an exponent of a function ($\sum_a p_t(a)e^{-\eta X_{t,a}}$). Instead of going the way we did, apply Hoeffding's lemma.

   (b) Find the value of $\eta$ that minimizes the new bound. (You should get $\eta = \sqrt{\frac{8\ln K}{T}}$ - please, prove this formally.)

   (c) At the end you should obtain $\mathbb{E}[R_T] \leq \sqrt{\frac{1}{2}T\ln K}$. (I.e., you will get an improvement by a factor of 2 compared to what we did in class.)

   *Remark: Note that the regret upper bound matches the lower bound up to the constants. This is an extremely rare case.*

Your text here 1

2. Explain why the same approach cannot be used to tighten the regret bound for the EXP3 algorithm.

**Exercise 5.10** (*Empirical evaluation of algorithms for adversarial environments*). Is it possible to evaluate experimentally the quality of algorithms for adversarial environments? If yes, how would you design such an experiment? If no, explain why it is not possible.

*Hint: Think what kind of experiments can certify that an algorithm for an adversarial environment is good and what kind of experiments can certify that the algorithm is bad? How easy or hard is it to construct the corresponding experiments?*