Task Sheet 2
Building the World & Memory

Context. Supervised learning is done. This week you turn your script into an *interactive environment* and add an *experience replay buffer.* No gradient updates yet — we first need a place to *store* experience!

Learning goals

- Model a Markov Decision Process (MDP) around daily SPY returns.
- Implement a simple long/flat trading environment with transaction costs in R.
- Understand why and how a replay buffer breaks correlation.

Introduction

Assume, we are in the same trading scenario as in task 1. However, this scenario was not very realistic. Hence, we will now consider trading costs and relevant future consequences. Each trade will therefore induce costs of 0.0005 (or 0.05 percent) of trading volume. Also, a previously bought stock must be cleared (sold) before being able to go flat again. Coinciding, a previously bought stock does not need to be bought again, when the current action indicates long, requiring us to essentially hold the position instead of buying more. Assume, that the investor starts with an equity of 1.

Exercises

1. (**State 2.0**) Extend `make_state(t, pos)` such that the last element encodes the *current position* (0 = flat, 1 = long). Why does the network need to know *pos*? (Glossary: "State", "Action")
2. (**Environment Theory**) Make yourself familiar with the term environment. What critical information must the environment in our case contain or keep track of?
3. (**Environment Application**) Create two helper functions:
   (a) `env_reset()` returning a list `list(t, pos, equity, done)` with `t = window_size` and `equity = 1`.
   (b) `env_step(env, action)` that
      i. advances time by one bar,
      ii. determines the reward,
      iii. applies (possible) transaction costs to the reward,
      iv. creates the next environment `next_env`,
      v. stores the next state in `obs`,
      vi. returns `list(next_env, reward, obs)`.
4. (**Environment Test**) Start with a single long action at $t = 11$ and an equity of 1. Stay long for 10 periods by executing the environment step by step. Display the outcome of that policy by retrieving the equity.
5. (**Replay buffer initialization**) Implement a buffer with capacity 5 000 that stores $(s, a, r, s', \text{done})$. s is the current state, a the action, r the reward and s' the next state. For that:
   (a) Make yourself familiar with the R basic function `new.env()`. Use it to initialize the replay.
   (b) The replay must hold information on $(s, a, r, s', \text{done})$. Initilaize each of these elements with the best suitable R object type, e.g. integer, matrix etc.
   (c) Build an index `idx` to see where the next record for $(s, a, r, s', \text{done})$ needs to be stored at.
   (d) Create an indication to check whether the replay buffer is full.

6. (**Replay buffer functions**) The replay buffer needs to be equipped with important functionality. Therefore:
   (a) Provide the function `store_transition`, which simply records the current $(s, a, r, s', \text{done})$ in the replay buffer at index `idx`.
   (b) Create the function `sample_batch(n)`, which samples randomly from past entries to the replay buffer.
   (c) Demonstrate that consecutive calls to `sample_batch` yield *uncorrelated* t indices.
7. (**Sanity check**) Use the stock market data from task 1, then start a new environment, make a state, and loop through 10 state transitions for a person who constantly buys. Record those trades in the replay buffer. Print the investors equity at the end of the loop and check if your replay buffer contains all 10 transitions. You can take the following code chunk to give yourself some orientation:

```
env   <- env_reset()
state <- make_state(env$t, env$pos)

for(steps in 1:10){
action <- 1

res   <- env_step(env, action)
store_transition(state, action, res$reward, res$obs, res$next_env$done)

state <- res$obs
env   <- res$next_env

cat("Number of transitions in Replay Buffer:",replay$idx - 1,"\n")

cat("Equity:", res$next_env$equity, "\n")
}
```

Reflection questions

- Figure out, what the option `done` is used for in an environment.
- What does `equity = 1` practically assume?
- Why did we choose `t = window_size` when resetting the environment, and not e.g. `t = 1`?
- If transaction costs doubled, how should the reward function change?
- Why could it be important to sample uncorrelated replay buffer samples in our specific setting?