# RL4Trading

Anonymous
*École Polytechnique*

This project is an attempt to train a reinforcement learning-based trading bot using a deep Q-network. The agent learns to make trading decisions through interaction with a simulated stock trading environment, employing strategies like epsilon-greedy for action selection and using a replay buffer to facilitate learning from a diverse set of experiences. The goal is to maximize the total wealth over time through effective trading strategies.

## The Environment

Our environment is inspired by the OpenAI Gymnasium package. It simulates a simplified stock exchange market for trading. Currently, we only consider one stock at a time, but our project aims to consider several stocks simultaneously.

In this environment, the agent can buy new shares with its own money, hold its shares for the future, or sell its shares. The amount of shares exchanged is also chosen by the agent. All the choices among the possible actions are based on the observed state. This observed state includes the closing prices of the considered stock for a predefined number of previous days ($T$), the account balance (available money), and the floating number of shares held. We chose to normalize the closing prices.

The environment handles the action taken and updates the state accordingly, providing rewards based on the change in total wealth. The reward is equal to 0 if the current episode is over, which happens when the agent goes bankrupt without available cash nor shares or if the agent explored the complete range of data initially provided. Otherwise, the reward is equal to the account balance plus the value of held shares, which is equal to the floating number of held shares times the current value of one share.

Let's delve deeper into the code:

- `_next_observation`: This function returns the observed state as described previously.
- `_take_action`: This function takes as input the action to operate as well as a proportion of shares. If the chosen action is buy or sell, this proportion represents the percentage of possible shares to exchange. The function then updates the account balance and the floating number of shares held.
- `step`: This function computes the wealth possessed by the agent, calls the function `_take_action` with the parameters of the action and the proportion also received as input. Then, if the episode does not end, it computes the new wealth possessed by the agent and computes the reward.
- `reset`: This function reinitializes all the parameters of the environment.

## Our Agent

Our agent interacts with the environment, making decisions based on the current state and learning from the outcomes. It employs a model for decision-making, a target model for stable Q-value estimation, an epsilon-greedy strategy for action selection, and a replay buffer to store and sample experiences. The agent trains over a specified number of episodes, updating its models to maximize rewards, which correspond to effective trading strategies.

- Implements an epsilon-greedy policy for action selection, balancing exploration and exploitation. With a probability of $\epsilon$, the agent takes a random action; otherwise, it chooses the best action according to its current policy. Epsilon decays over time to shift from exploration to exploitation.
- A crucial component for stable and efficient learning, the replay buffer stores experiences that the agent encounters. Each experience includes the state, action, reward, next state, and done flag. The agent samples from this buffer to learn, helping to break the correlation between consecutive experiences.

## The Network

The core neural network model used for approximating the Q-function consists of an LSTM layer to process sequences of stock prices, followed by fully connected layers to integrate additional information such as account balance and shares held. The network outputs a value for each possible action, guiding the agent's choices.

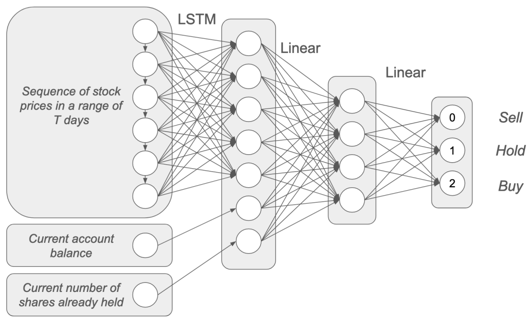
LSTM
Linear
Linear
Sequence of stock prices in a range of T days
Current account balance
Current number of shares already held
0
1
2
Sell
Hold
Buy

FIG. 1. Structure of the DQN

## Conclusion

In conclusion, our project aims to develop a reinforcement learning-based trading bot, termed RL4Trading, capable of making effective decisions in a simulated stock trading environment. We have presented the design and implementation details of the environment, agent, and neural network model. Our agent employs advanced techniques such as epsilon-greedy policy, replay buffer, and deep Q-network to learn and optimize trading strategies over time.

Moving forward, we plan to evaluate the performance of RL4Trading on historical stock data and further refine its learning algorithms. Additionally, we aim to extend the capabilities of our project to handle multiple stocks simultaneously, thereby enhancing its applicability and robustness in real-world trading scenarios.