Assignment 2. Report

Artem Chernitsa, B20-AI-01

Choice of the environment representation used for the model training

First of all, it was necessary to create a field on which the location of cargos is possible, which means that means are needed to understand their absolute or relative location in the world. Based on the conditions of the task, I decided to present the world in the form of a simple grid, as we did in the tasks earlier, where each cargo has the coordinates of each part of it, and the same applies to the reward zone.

Each cargo was represented as a tuple of two coordinates: the upper-left corner and the lower-right corner. If necessary, it was mentally completed to a rectangle to make it easier to work with it. The same was done for the reward zone. Thus, it was easy to operate by moving cargo to the field and checking the stop condition. Since the number of actions is limited only for each specific cargo, and the number of cargo itself is not limited, I decided to save the state for a specific cargo. In the model, I pass a tuple of coordinates of the reward zone, the same for my own coordinates and the reward received for each part standing in the reward zone and deducted for each intersection with another cargo.

Way of training environment generation

In order to teach the agent to solve the problem, the following cycle was created: first we set the initial conditions, then, before each episode, we get the current state of the environment, which, importantly, returns us not only the coordinates of the reward zone, but also the coordinates relative to the cargo selected for the course and the reward for it. So you can distinguish which cargo is moving at the moment, and try to clarify with the network where it is best to move a particular cargo.

The model returns probabilities for performing an action. If an action is impossible, I just throw it out and redistribute the probabilities to the remaining actions and so on

until a possible action is found. Each step is recorded in history, then the reward is collected exactly as it was on the fifth laboratory work.

The loss function looks like this: `loss = -torch.sum(torch.log(prob_batch) * expected_returns_batch)`. The total reward was considered as follows: `batch_Gvals.append(batch_Gvals[i - 1] * gamma + reward_batch[len(transitions) - i - 1])`. It's a bit like Monte Carlo, but it's not completely it with respect to other parts of realization.

The main problem lies in the fact that when information is incorrectly communicated and rewards are distributed, either a meaningful wandering occurs (he understands that the loss function becomes stable and instead of following to the reward zone, he walks from side to side as if on the spot). It also happens that he goes around some places in a circle, and if he never gets into the reward zone, then he may never get into it again, and it turns out that "standing still" is more profitable than going somewhere, because the loss function converges to a constant and it suits him. The problem was partially solved by a random choice of action, but not completely, and in the case of several cargos, it can work catastrophically for a long time and not converge to an optimal solution. Sure, the random function does not converge, but it gives an acceptable result in speed for small maps.

Architecture choice for the neural network you use for the prediction

The number of input parameters to the network is only 9, so it is possible to organize communication for such a small number of pairs of meters in just a few layers. For sure, I tried and installed without any problems the input and output layers and two hidden ones with the ReLU activation function, and also applied “softmax” on the input layer to give out probabilities. In general, the main problem lay in the algorithm for representing the state of the environment and the learning cycle, so it is difficult to predict the quality of the network, but it gives excellent results for some specific samples, which may mean its suitability.