Sergeev Nikita BS20-AI-01 Reinforsment Learning Assignment 2 report

---

Choice of the environment representation for model training:

For the enviroment representation I implemented class *Environment* with the following fields:

- *world_h*, *world_w* - represents total world height and width
- *cargos* - represents dict with *keys* - label of the cargo in the world map. And values are lists with coordinates of the cargo cells.
- *desired space* - list of coordinates of cells in wich situated desired space of the world.
- *world* - numpy array simillar to the given world map, but desired space represented with **-1** instead of **r** in the initial world. It's made so to have number respresentation of the world to make it perceived to the algorithm.
  Also my *Environment* class have following methods:
- *move_cargo* - method for moving particular cargo along the enviroment. It takes 2 argiments - cargo which will be moved by this method and action (direction in which cargo will be moved)
- *check_move* - method that return if move of the cargo in the particular direction legal or by this move cargo will crush into the wall.
- *reset* - roll back the environment into it initial state. In other words, if after several moves of cargo we will call this method, positions of all cargos will be returned to the initial.
- *get_cargo_overlaps* - return the number of cells in which cargo overlap with each other.
- *get_cargo_overlaps_with_desired* - method, which returns number of cargo cells placed in the desired space in the current state of the environment.
- *is_done* - method which returns is game other, according to the rules given in the assignment description (all cargo should be fully in the desired space and don't overlaps).

Training environment generation:

For training my DQN model I create function *generate random world* and it performs following steps to generate random world:

- Generate random world size (height and width) in range from 5 to 20
- Initially create world full of zeros with the generated size
- Generate size of the desired zone in range from 2 to correspondig world size
- Then I randomly choice cell which will be left up corner of the desired zone.
- And from this chousen cell I fill world with desired zone

- Then I choice number of cargos in the env randomly from 1 to 5
- Number of cells in the cargo is also chousen randomly
- Then I generate cargo by expand cargo by one of the adjacent cells.
- Also while generating cargos I check what cargo doesnt intercept with the desired zone and doesnt go out of bounds.

Model architecture choice

For the given task I implemented Deep Q Network with the following architecture:

```
class DQN(nn.Module):
    def __init__(self, output_size):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.global_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc1 = nn.Linear(64, 64)
        self.fc2 = nn.Linear(64, output_size)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = self.global_pool(x)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

This network have

- 3 convolutional layers with `stride=1` and `padding=1` to not reduce size of the input matrix.
- Global average poolling to convert given input matrix to tensor with size equal to number of layers in the final convolutional layer.
- 2 Fully connected layers on the flatten output of the global average pooling
- And after fully connected layers we have 4 neurons as an output, which represent 1 of 4 actions, which cargo may perform.

As an input DQN get current state of the environment. And this state will be different for any cargo in the environment.
For a given cargo state will be the world map of the enviroment with the following changes:

- Desired zone will be represented as zone with *-1*
- Other cargos will be represented as *-2* (like walls with which cargo don't want to overlap)
- And cargo which will be moved in the current step is represented as *1*

Process of the optimization of the DQN look very simmilar to the code given in the 4'th lab:

```
def optimize_model():
    if len(memory) < BATCH_SIZE:
        return
    transitions = memory.sample(BATCH_SIZE)
    batch = Transition(*zip(*transitions))

    non_final_mask = torch.tensor(
        tuple(map(lambda s: s is not None, batch.next_state)), dtype=torch.bool
    )
    non_final_next_states = torch.cat([s for s in batch.next_state if s is not
None])

    state_batch = torch.cat(batch.state).to(device)
    action_batch = torch.cat(batch.action).to(device)
    reward_batch = torch.cat(batch.reward).to(device)

    state_action_values = policy_net(state_batch).gather(1, action_batch)

    next_state_values = torch.zeros(BATCH_SIZE).to(device)
    next_state_values[non_final_mask] = (
        target_net(non_final_next_states.to(device)).max(1)[0].detach()
    )

    expected_state_action_values = (next_state_values * GAMMA) + reward_batch

    loss = F.smooth_l1_loss(
        state_action_values, expected_state_action_values.unsqueeze(1)
    )

    optimizer.zero_grad()
    loss.backward()
    for param in policy_net.parameters():
        param.grad.data.clamp_(-1, 1)
    optimizer.step()
```

Conclusion:

After **LOTS** of experiments with model architecture, model optimization, state of the enviroment, etc.. I didn't find a way to make solution of the environment for generall random world, which will converge at the end of training. So, i decided to traing my model for every particular environment. After ~100 episodes my solution with ϵ greedy policy gonverges and give $\pm$ optimal answer on testing.