高级算法期中作业报告

1.Main tasks

In this interim project, you will get a complete implementation of a dual dqn agent. The tasks include:

Work with another student registered for this course；

Read through the implementation and explain in detail in the team report what each component is responsible for and how the components are connected together;

Use priority playback to improve training speed；

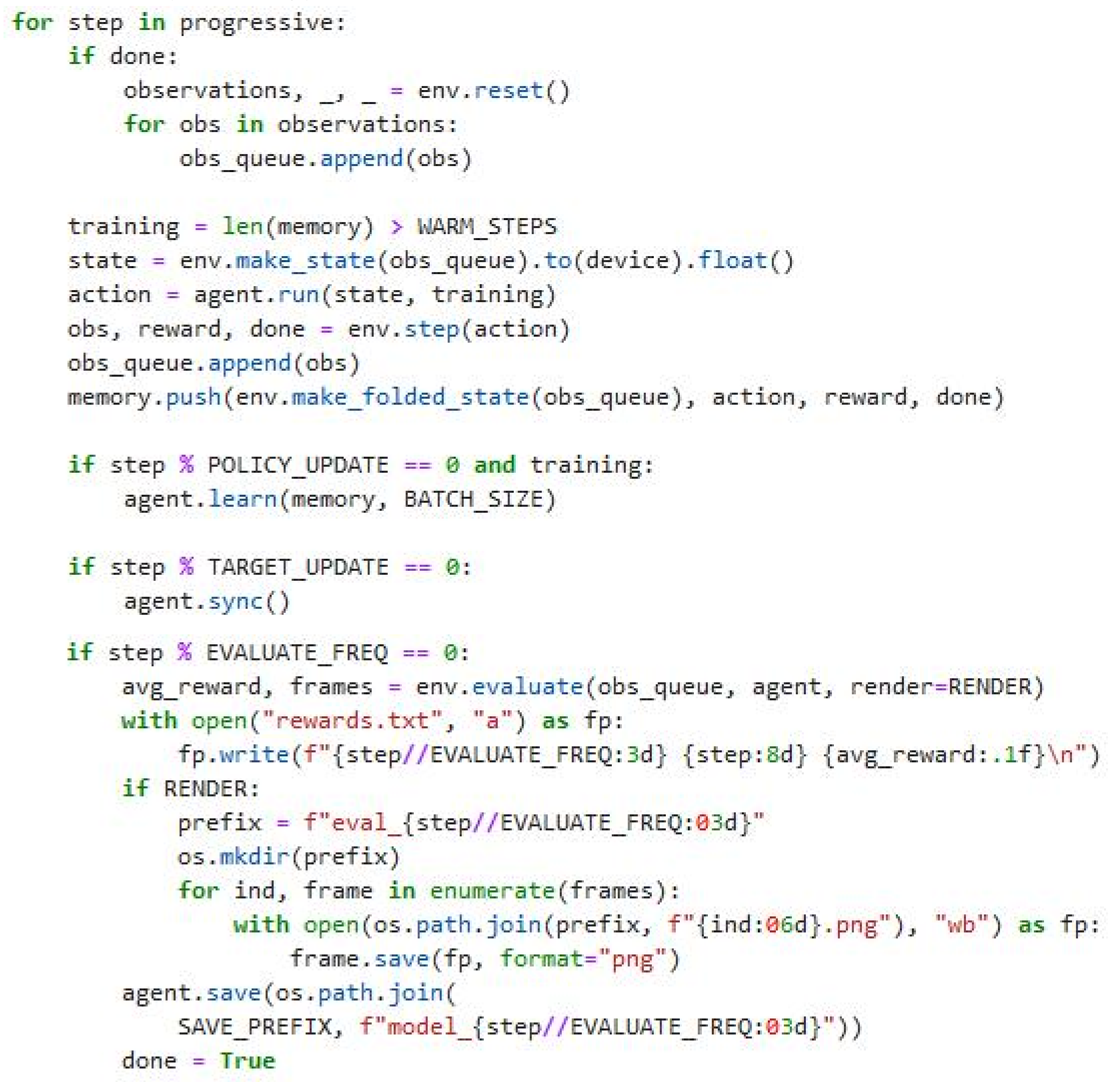
Open source projects on GitHub, gitee or any other similar platform, and include a reference link in the team report;

Write the report in English.

2.Code analysis

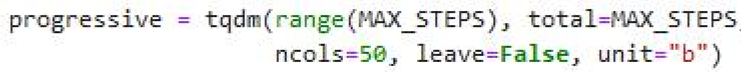
a.Analysis of part main.py

The main body of main.py is as follows：



```python
for step in progressive:
    if done:
        observations, _, _ = env.reset()
        for obs in observations:
            obs_queue.append(obs)

    training = len(memory) > WARM_STEPS
    state = env.make_state(obs_queue).to(device).float()
    action = agent.run(state, training)
    obs, reward, done = env.step(action)
    obs_queue.append(obs)
    memory.push(env.make_folded_state(obs_queue), action, reward, done)

    if step % POLICY_UPDATE == 0 and training:
        agent.learn(memory, BATCH_SIZE)

    if step % TARGET_UPDATE == 0:
        agent.sync()

    if step % EVALUATE_FREQ == 0:
        avg_reward, frames = env.evaluate(obs_queue, agent, render=RENDER)
        with open("rewards.txt", "a") as fp:
            fp.write(f"{step//EVALUATE_FREQ:3d} {step:8d} {avg_reward:.1f}\n")
        if RENDER:
            prefix = f"eval_{step//EVALUATE_FREQ:03d}"
            os.mkdir(prefix)
            for ind, frame in enumerate(frames):
                with open(os.path.join(prefix, f"{ind:06d}.png"), "wb") as fp:
                    frame.save(fp, format="png")
        agent.save(os.path.join(
            SAVE_PREFIX, f"model_{step//EVALUATE_FREQ:03d}"))
        done = True
```

‘’Progressive’ at the beginning is a tqdm class. Tqdm stands for progress. It is a progress bar tool library. Users only need to encapsulate any iterator tqdm (iterator). Common examples include real-time display of download progress.

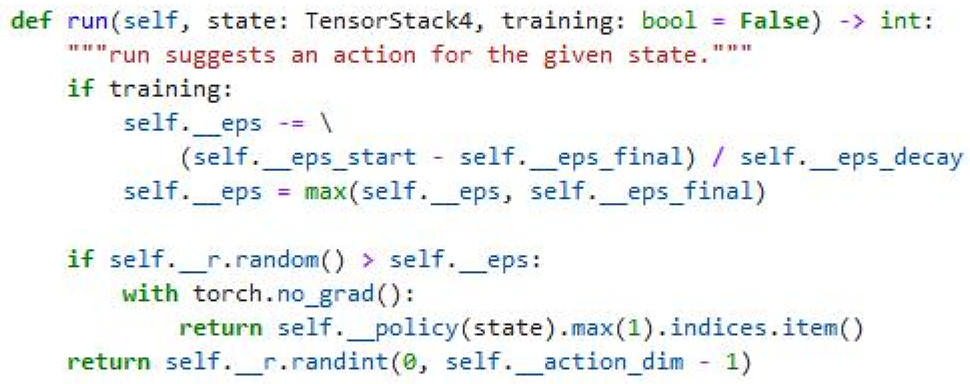


```
progressive = tqdm(range(MAX_STEPS), total=MAX_STEPS,
                   ncols=50, leave=False, unit="b")
```

The next step is to determine whether to train, read the state according to the environment, and then the agent takes some action, and then the action generates the state and reward, and adds the state and reward to the memory.

The following three if statements are interval policy_ In the update step, the agent performs the update step, and then updates it_ Update step, assign the agent to the target network; interval evaluate_ Freq step, save the running results, test the effect, and reset the environment in the next training.

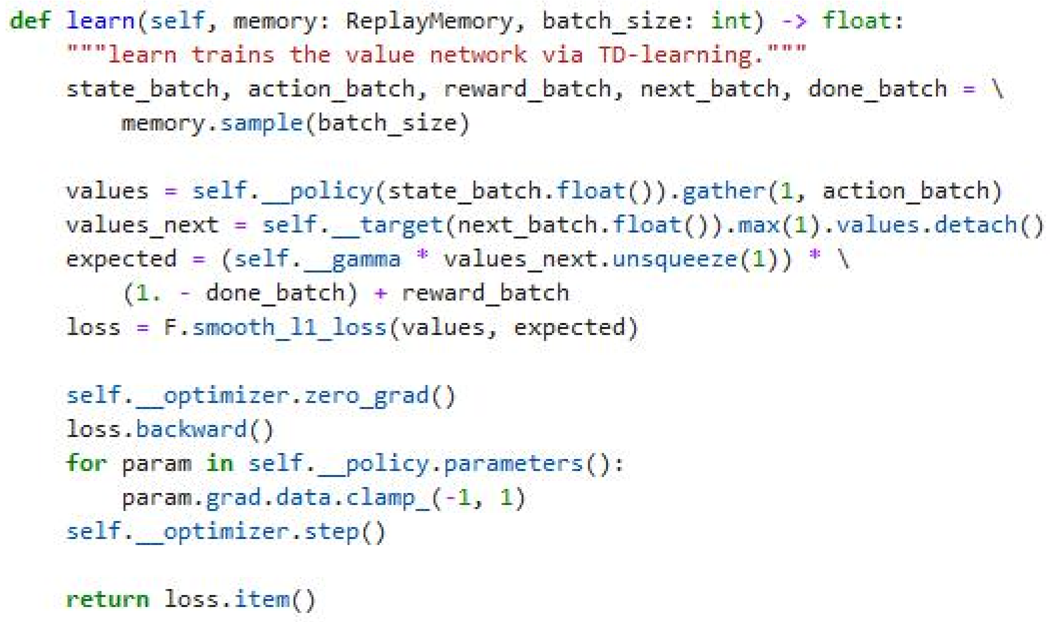
The "agent.run()" used in main.py is indicated in the utils_drl.py file, and the code is as follows:



```
def run(self, state: TensorStack4, training: bool = False) -> int:
    """run suggests an action for the given state."""
    if training:
        self.__eps -= \
            (self.__eps_start - self.__eps_final) / self.__eps_decay
        self.__eps = max(self.__eps, self.__eps_final)

    if self.__r.random() > self.__eps:
        with torch.no_grad():
            return self.__policy(state).max(1).indices.item()
    return self.__r.randint(0, self.__action_dim - 1)
```

When training is true, self_ EPS is constantly decreasing. Before training, len (memory) > warm_ Step control, action can be selected randomly to obtain various information.

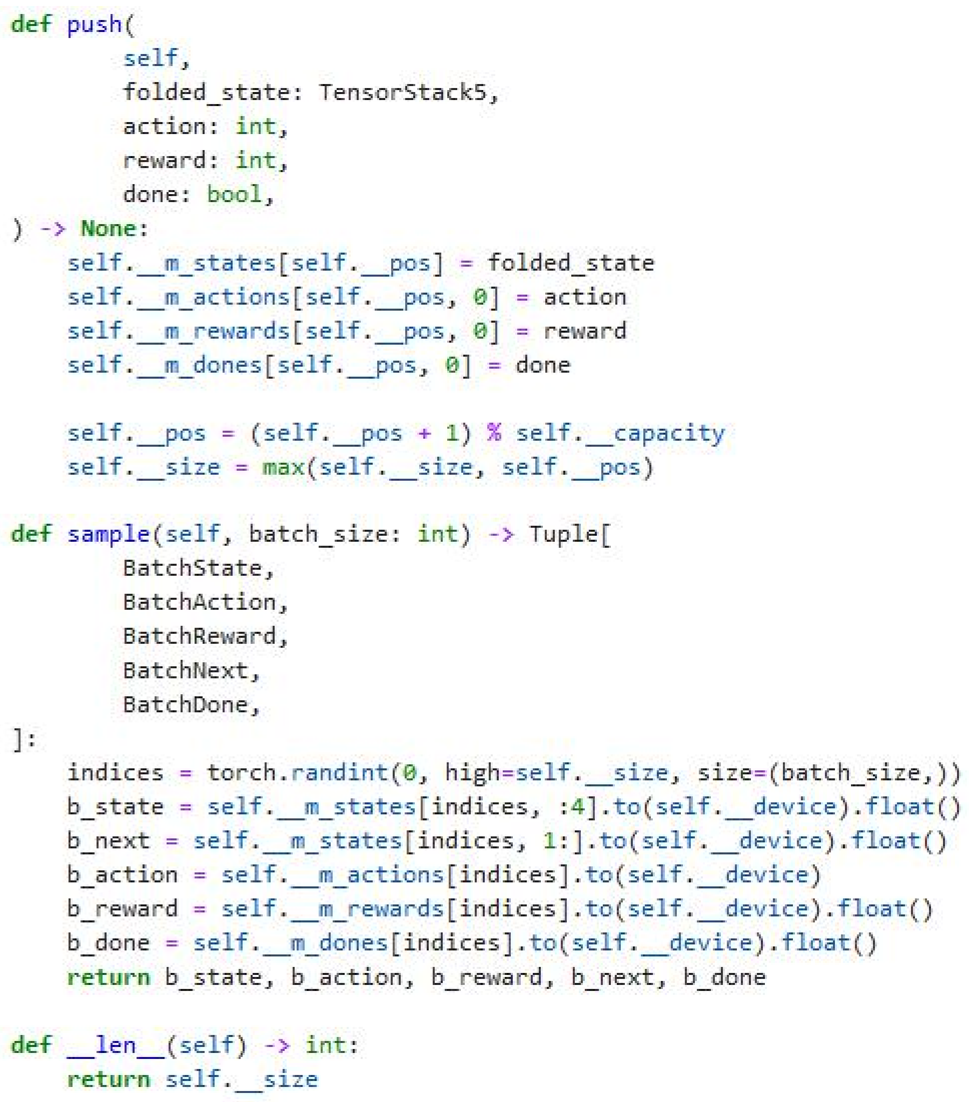
The code of agent.learn() in the utils_drl.py file is as follows:



```
def learn(self, memory: ReplayMemory, batch_size: int) -> float:
    """learn trains the value network via TD-learning."""
    state_batch, action_batch, reward_batch, next_batch, done_batch = \
        memory.sample(batch_size)

    values = self.__policy(state_batch.float()).gather(1, action_batch)
    values_next = self.__target(next_batch.float()).max(1).values.detach()
    expected = (self.__gamma * values_next.unsqueeze(1)) * \
        (1. - done_batch) + reward_batch
    loss = F.smooth_l1_loss(values, expected)

    self.__optimizer.zero_grad()
    loss.backward()
    for param in self.__policy.parameters():
        param.grad.data.clamp_(-1, 1)
    self.__optimizer.step()

    return loss.item()
```

First, memory.sample() is sampled, and then the policy and target output values are calculated. At the same time, smooth_l1_loss is used to calculate the loss of loss.

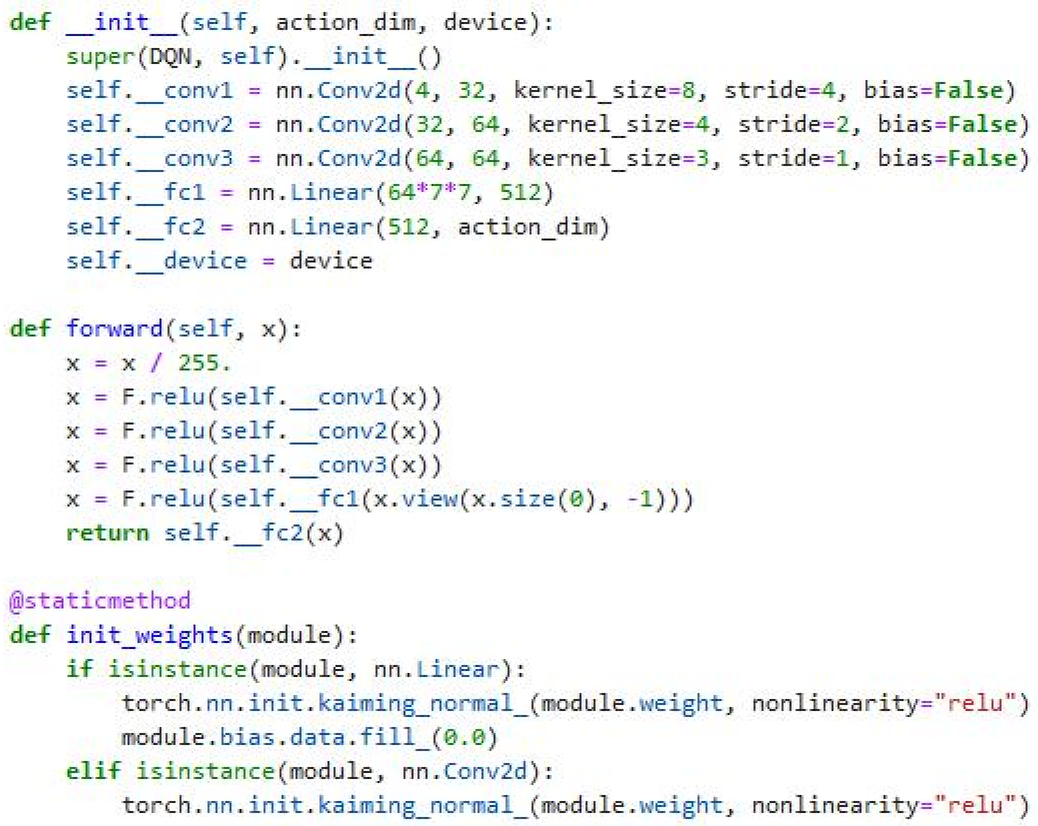
Memory.push() in main.py and memory.sample() in utils_dr.lpy are defined in the utils_memory.py file. The codes are as follows:：



```python
def push(
        self,
        folded_state: TensorStack5,
        action: int,
        reward: int,
        done: bool,
) -> None:
    self.__m_states[self.__pos] = folded_state
    self.__m_actions[self.__pos, 0] = action
    self.__m_rewards[self.__pos, 0] = reward
    self.__m_dones[self.__pos, 0] = done

    self.__pos = (self.__pos + 1) % self.__capacity
    self.__size = max(self.__size, self.__pos)

def sample(self, batch_size: int) -> Tuple[
        BatchState,
        BatchAction,
        BatchReward,
        BatchNext,
        BatchDone,
]:
    indices = torch.randint(0, high=self.__size, size=(batch_size,))
    b_state = self.__m_states[indices, :4].to(self.__device).float()
    b_next = self.__m_states[indices, 1:].to(self.__device).float()
    b_action = self.__m_actions[indices].to(self.__device)
    b_reward = self.__m_rewards[indices].to(self.__device).float()
    b_done = self.__m_dones[indices].to(self.__device).float()
    return b_state, b_action, b_reward, b_next, b_done

def __len__(self) -> int:
    return self.__size
```

The push function is to find the next value in the loop array and override it. The sample function is a table of batch_size and returns。

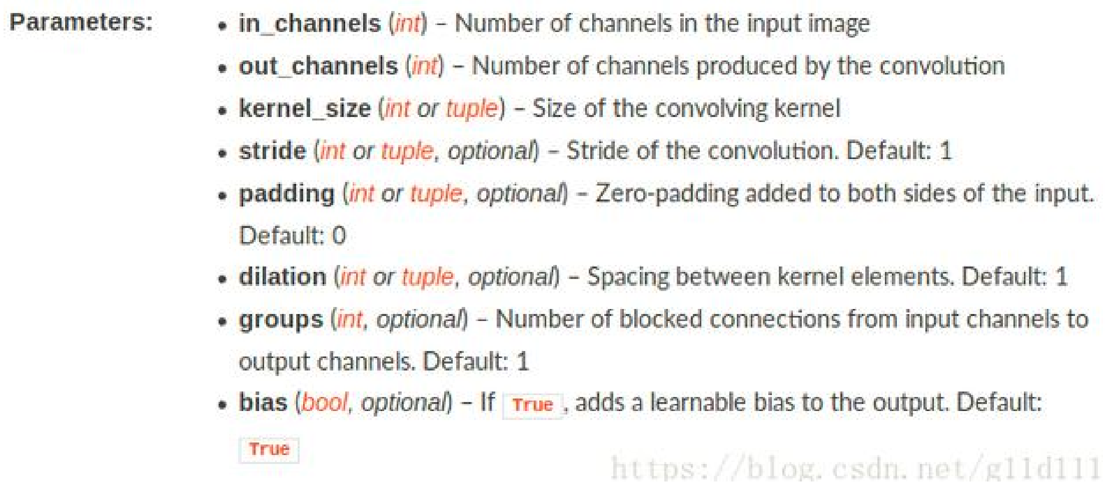
For the utils _model.py file, the code is as follows:



```python
def __init__(self, action_dim, device):
    super(DQN, self).__init__()
    self.__conv1 = nn.Conv2d(4, 32, kernel_size=8, stride=4, bias=False)
    self.__conv2 = nn.Conv2d(32, 64, kernel_size=4, stride=2, bias=False)
    self.__conv3 = nn.Conv2d(64, 64, kernel_size=3, stride=1, bias=False)
    self.__fc1 = nn.Linear(64*7*7, 512)
    self.__fc2 = nn.Linear(512, action_dim)
    self.__device = device

def forward(self, x):
    x = x / 255.
    x = F.relu(self.__conv1(x))
    x = F.relu(self.__conv2(x))
    x = F.relu(self.__conv3(x))
    x = F.relu(self.__fc1(x.view(x.size(0), -1)))
    return self.__fc2(x)

@staticmethod
def init_weights(module):
    if isinstance(module, nn.Linear):
        torch.nn.init.kaiming_normal_(module.weight, nonlinearity="relu")
        module.bias.data.fill_(0.0)
    elif isinstance(module, nn.Conv2d):
        torch.nn.init.kaiming_normal_(module.weight, nonlinearity="relu")
```

The convolution neural network function is used here. The interface is defined as follows:

```
class torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True)
```



Parameters:

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) – Zero-padding added to both sides of the input. Default: 0
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`

After that is the activation function relu (), and the output action_dim is the score of action.

Part of the code for the utils_types.py file is as follows:

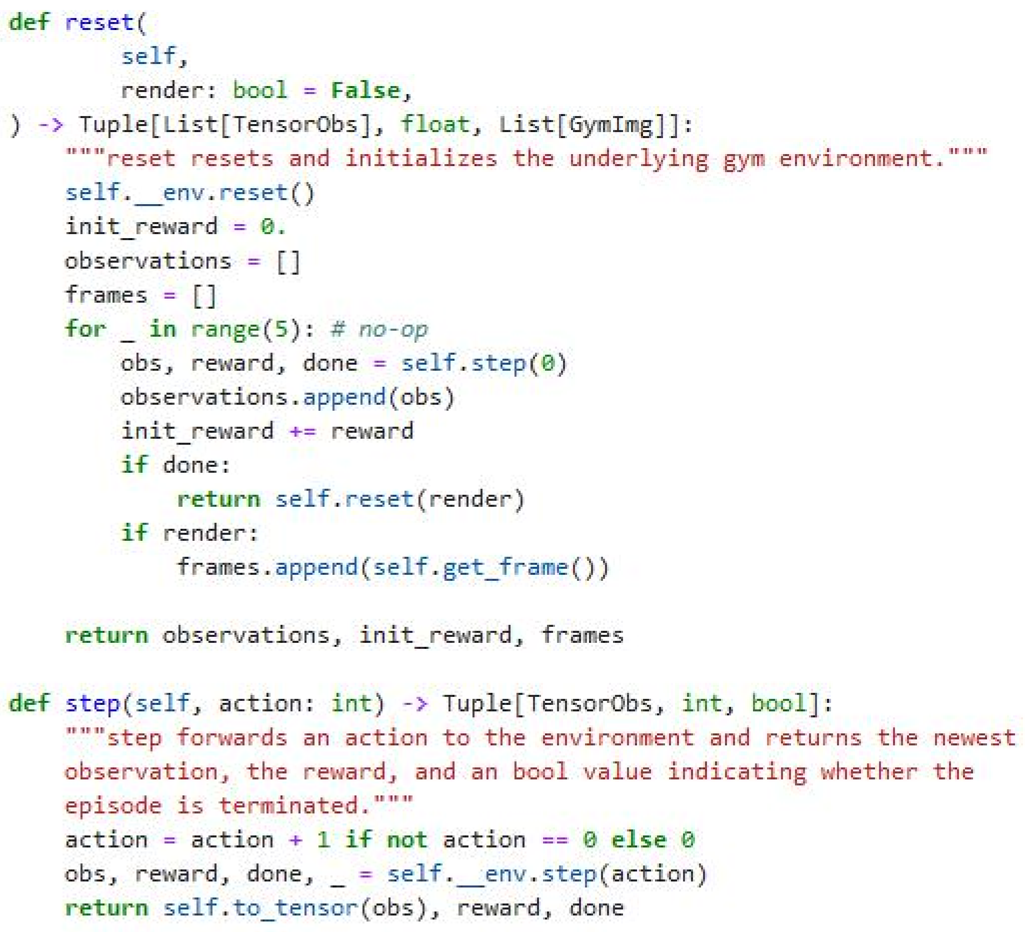
```python
# Tensor with shape (None, 4, 84, 84)
BatchState = Any
# Tensor with shape (None, 1)
BatchAction = Any
# Tensor with shape (None, 1)
BatchReward = Any
# Tensor with shape (None, 4, 84, 84)
BatchNext = Any
# Tensor with shape (None, 1)
BatchDone = Any
# NDArray with shape (210, 160, 3)
GymImg = Any
# NDArray with shape (84, 84, 1)
GymObs = Any
# Tensor with shape (N, 1)
TensorN1 = Any
# Tensor with shape (1, 84, 84)
TensorObs = Any
# A stack with 4 GymObs, with shape (1, 4, 84, 84)
TensorStack4 = Any
# A stack with 5 GymObs, with shape (1, 5, 84, 84)
TensorStack5 = Any
# torch.device("cpu") or torch.device("cuda"), can be conditional on
# torch.cuda.is_available()
TorchDevice = Any
```

Define many variable types and assign them to any class. Any class can perform a variety of operations and assignments.

Part of the code in the utils_env.py file is as follows

```python
def __init__(self, device: TorchDevice) -> None:
    env_raw = make_atari("BreakoutNoFrameskip-v4")
    self.__env_train = wrap_deepmind(env_raw, episode_life=True)
    env_raw = make_atari("BreakoutNoFrameskip-v4")
    self.__env_eval = wrap_deepmind(env_raw, episode_life=True)
    self.__env = self.__env_train
    self.__device = device
```

Initialization uses atari's method



```python
def reset(
        self,
        render: bool = False,
) -> Tuple[List[TensorObs], float, List[GymImg]]:
    """reset resets and initializes the underlying gym environment."""
    self.__env.reset()
    init_reward = 0.
    observations = []
    frames = []
    for _ in range(5): # no-op
        obs, reward, done = self.step(0)
        observations.append(obs)
        init_reward += reward
        if done:
            return self.reset(render)
        if render:
            frames.append(self.get_frame())

    return observations, init_reward, frames

def step(self, action: int) -> Tuple[TensorObs, int, bool]:
    """step forwards an action to the environment and returns the newest
    observation, the reward, and an bool value indicating whether the
    episode is terminated."""
    action = action + 1 if not action == 0 else 0
    obs, reward, done, _ = self.__env.step(action)
    return self.to_tensor(obs), reward, done
```

Both reset and step call the gym algorithm. A step is preloaded in reset, but no operation is generated, which makes the observation queue non empty.

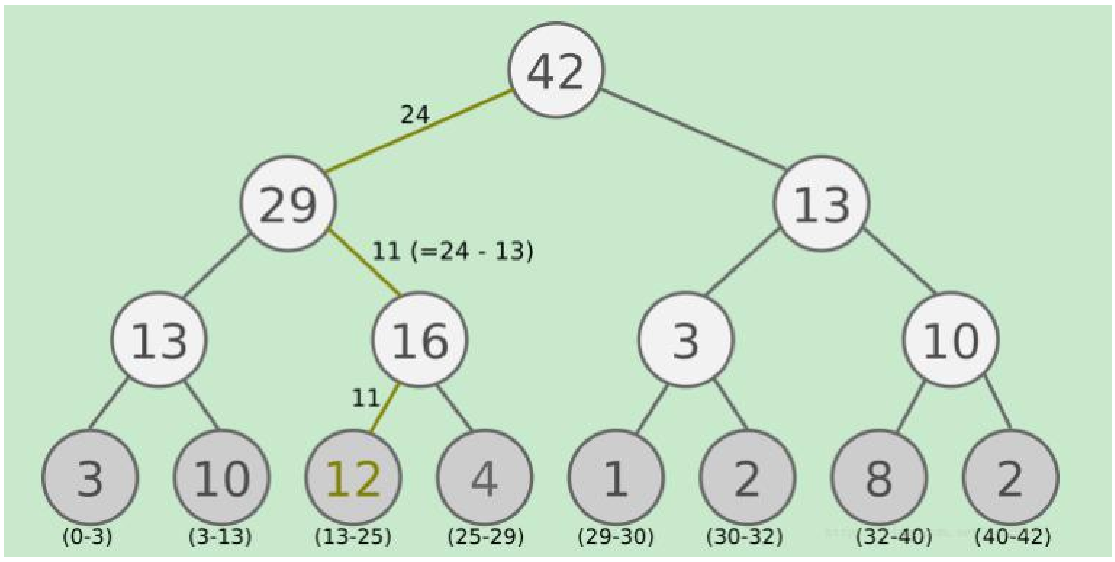
get_frame() renders the current game frame.。

Evaluate() uses the given agent to run the game and returns the average reward and frames captured.

3. Priority playback to improve training speed

TD error is the difference between the target Q value calculated by the target Q network and the Q value calculated by the current Q network. The TD error of the samples in the experience playback pool is different, which leads to the different back propagation effect on the program. With the increase of TD error, the effect of back propagation is enhanced; on the contrary, the effect of back propagation decreases with the decrease of TD error. If the sample with large TD error is easy to be sampled, the algorithm can be optimized.

Priority playback algorithm contains the priority of experience playback, so it is necessary to update the experience playback pool and use sumtree structure to store samples. The structure is as follows:



The leaf node at the end will store the experience playback samples and represent the priority. The internal node keeps the sum of the priorities of the child nodes. For example, 42 of the root node represents the uniform sampling between [0,42]. If the sample reaches 26, it belongs to the interval of (25 ~ 29), so it represents that the fourth node is sampled.

The main flow of the algorithm is as follows: input iteration round number T, state characteristic dimension N, action set a, step size α, sampling weight coefficient β, attenuation factor γ, exploration rate ϵ, current Q network Q, target Q network Q ′, sample number of batch gradient descent m, target Q network parameter update frequency C, leaf node number s of sumtree.
Then output Q network parameters.

Because of the long time of programming and running, I can't write the code and run the result.

It's a good idea to avoid some iterative process in theory.

4. Open Source link

https://github.com/ming990818/gjsf.mid

5. Division of labor

member	ideas	Coding	writing
Huang han	60%	50%	30%
Huang feiyang	40%	50%	70%