高级算法期中作业报告

18340059 黄飞阳 18340060 黄涵

一、主要任务

在这个中期项目中，你将得到一个双 DQN 代理的完整实现，任务包括：

与本课程注册的另一名学员合作；

通读实现，并在团队报告中详细解释每个组件负责什么以及组件是如何连接在一起的；

使用优先回放来提高训练速度；

在 GitHub、Gitee 或任何其他类似平台上开源项目，并在团队报告中包含一个参考链接；

用英语撰写报告。

二、代码分析

1.对于 main.py 部分的分析

Main.py 的主体部分如下：

```
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
```

开头的 progressive 是一个 tqdm 类，tqdm 代表进展，是一个进度条工具库，用户只需要封装任意的迭代器 tqdm(iterator)，常用例子有实时显示下载进度。

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

接下来的内容是判断是否要 training，根据环境读取状态，然后 agent 采取某种动作，接着动作产生状态和奖励，把这个状态和奖励加进 memory。
下面的三条 if 语句分别是间隔 POLICY_UPDATE 步，agent 进行然后更新；间隔 TARGET_UPDATE 步，将 agent 赋值给 target 网络；间隔 EVALUATE_FREQ 步，保存运行结果，测试效果，并在下次训练时重置环境。
在 main.py 中使用的 agent.run()在 utils_drl.py 文件中有写明，代码如下：

```
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)
```

当 training 为真时，self._eps 是不断衰减的。之前 training 的由 len(memory) > WARM_STEPS 控制，可以随机选择 action，获得多样信息。
utils_drl.py 文件中的 agent.learn()代码如下：

```
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()
```

先是 memory.sample()采样，然后是计算 policy 和 target 输出值 values 部分，同时 smooth_l1_loss 计算 loss 损失。
Main.py 中的 memory.push()和 utils_drl.py 中的 memory.sample()在 utils_memory.py 文件中定义，代码如下：

```
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
```

其中，push 函数是在循环数组中寻找下一个值并覆盖掉。Sample 函数是 batch_size 的一个表并返回。

对于 utils_model.py 文件，代码如下：

```
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")
```

这里运用了卷积神经网络函数，接口定义如下：

```
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`

之后是激活函数 relu()，输出的 action_dim 是 action 的得分。

utils_types.py 文件的部分代码如下：

```
# 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
```

定义很多变量类型，并将其赋为 Any 类。Any 类可以执行各种各样的操作以及各种赋值。

utils_env.py 文件中部分代码如下：

```
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
```

初始化使用了 atari 的方法。

```
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
```

Reset 和 step 都调用了 gym 算法。Reset 中预先加载一个步骤，但并没产生任何操作，却使得观测队列非空。

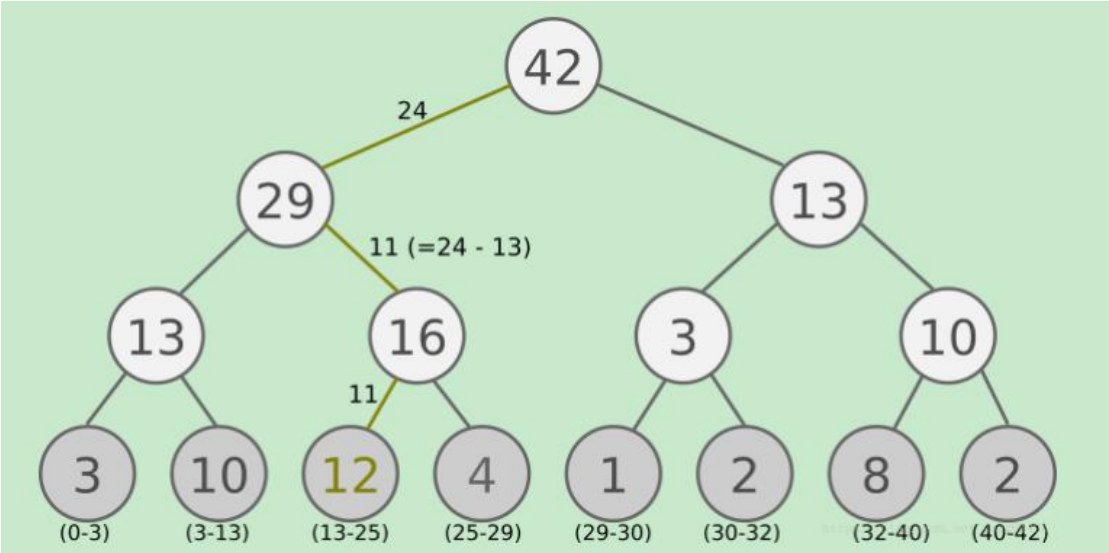
get_frame()渲染当前游戏帧。

Evaluate()使用给定的代理运行游戏，并返回平均奖励和捕获的帧。

三、优先回放来提高训练速度

TD 误差是目标 Q 网络计算的目标 Q 值和当前 Q 网络计算的 Q 值间的差距。在经验回放池样本的 TD 误差不同，导致对程序的反向传播作用不同。TD 误差增大，反向传播作用增强；反之，TD 误差减小，反向传播作用减弱。如果 TD 误差大的样本容易被采样，就可以优化算法。

优先回放算法包含了经验回放的优先级，所以需要更新经验回放池，使用 sumtree 结构来作为样本存储，结构如下：



末尾的叶子节点将存储经验回放样本，同时代表着优先级。内部节点则保存儿子节点的优先级之和。例如根节点的 42 代表在【0,42】之间的均匀采样，如果采样到 26，它属于（25~29）的区间，所以代表它的第 4 个节点被采样到。

算法的主要流程是，输入迭代轮数 T，状态特征维度 n，动作集 A，步长α，采样权重系数β，衰减因子γ，探索率ϵ，当前 Q 网络 Q，目标 Q 网络 Q′，批量梯度下降的样本数 m,目标 Q 网络参数更新频率 C, SumTree 的叶子节点数 S。

之后输出 Q 网络参数。

这里由于程序编写和运行时间太长，没能写出代码并跑出结果。

理论上优先回放算法可以避免一些没有价值的迭代过程，可以给原程序提供一个不错的优化。

四．开元链接

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

五．

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