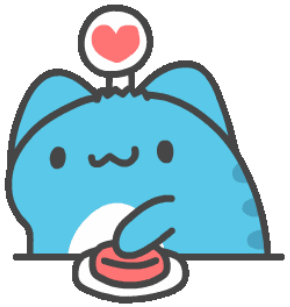
深度Q网络 DQN

Deep Q-Learning

东川路第一可爱猫猫虫



nature

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

深度Q网络 DQN

Deep Q-Learning

东川路第一可爱猫猫虫

nature

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

主要内容

- Q-learning遇到的问题
- 解决办法：神经网络近似Q值
- 神经网络近似Q值遇到的问题
- 解决办法：DQN
- DQN的两个重要模块

  经验回放 固定target
- DQN的全套流程
- DQN的实战代码

Q-learning遇到的问题

- Q-learning是一种表格型方法

  通过更新Q表来得到最优Q值

- 表格能存储的值是有限的

  只适用于状态和动作离散、空间较小的情况

- 在状态空间巨大的游戏里就需要新方法

  比如Atari游戏Space Invaders

  19岁的乔布斯工作过的游戏企业

Space Invaders

巨大的状态空间

- 单个帧由 210x160 像素的图像组成
- 图像是彩色 （RGB）

  有 3 个通道
- 有$256^{210\times160\times3} = 256^{100800}$个可能的观测值
- 这种时候用Q表不太可能了

  一个思路是用神经网络等非线性函数来近似Q值

预处理

- 用神经网络之前先进行预处理

  降低状态的复杂性，以减少训练所需的计算时间

- 四步

  消除闪烁

  提取亮度通道

  缩放图像

  帧堆叠

1 消除闪烁

- 取当前帧与前一帧的像素最大值

  因为游戏里的某些图像会闪烁

  只在奇数帧或偶数帧显示

- 取最大值可以合并物体

2 提取亮度通道

- 游戏在变色，RGB增加了计算复杂度
  但是色彩对游戏决策并无影响
- 因此对图像进行灰度处理
  把RGB三个通道减少到1

3 缩放图像

- 调整像素密度

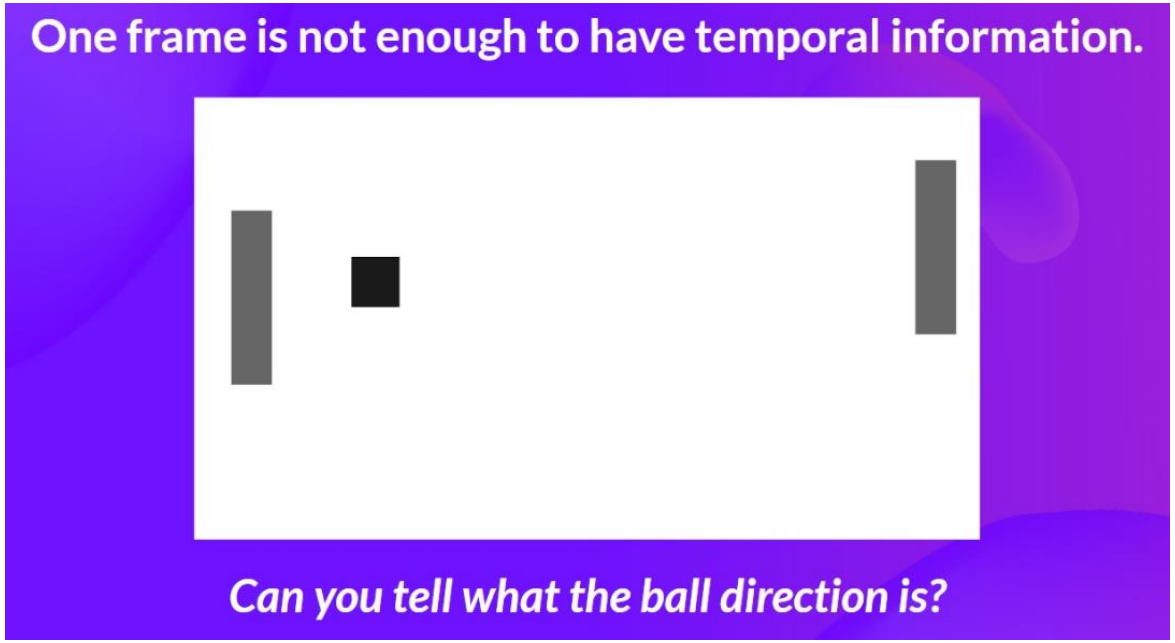
  降低分辨率

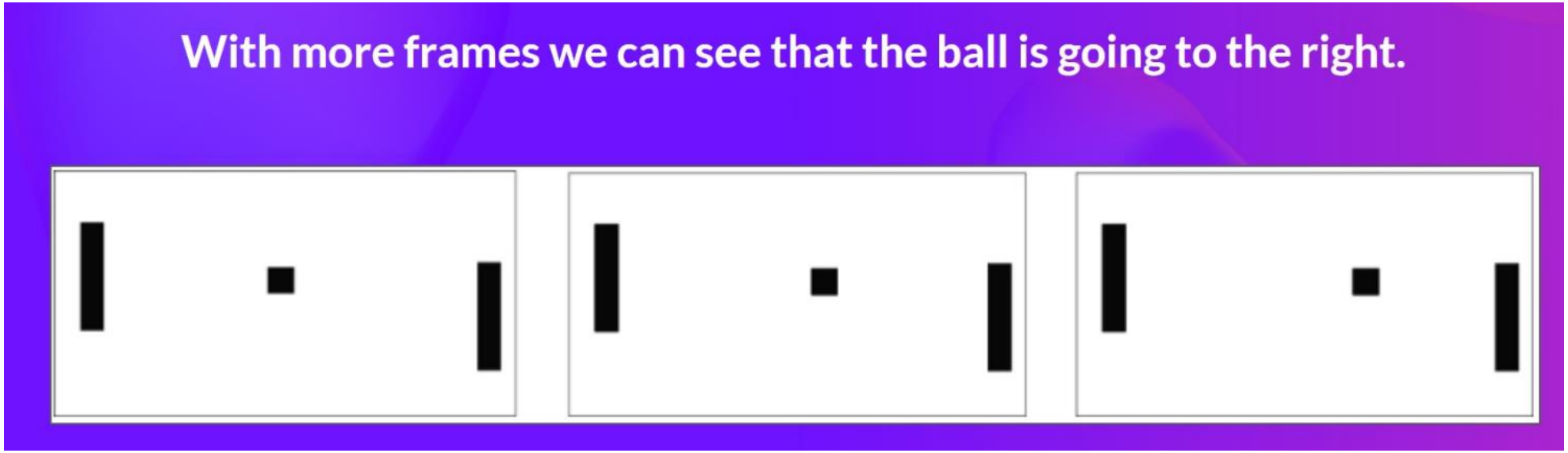
  210×160

  变成84×84

4 帧堆叠

- 将四个帧堆叠在一起
- 因单帧无法反映运动信息





损失函数

- 传统Q-learning

$$\mathrm{Q(s,a)} \leftarrow \mathrm{Q(s,a)} + \alpha(\mathrm{R(s,a)} + \gamma \mathrm{maxQ(S',a)} - \mathrm{Q(s,a)})$$

- DQN用参数为w的神经网络近似Q函数

$$\mathrm{Q}_\omega(s,a)$$

- 我们要让$\mathrm{Q}_\omega(s,a)$逼近$\mathrm{r} + \gamma \mathrm{maxQ}(\mathrm{s}',\mathrm{a}')$

  用均方误差作为损失函数

$$\frac{1}{2N}\Sigma_{i=1}^{N}\left[Q_\omega(s_i,a_i) - \left(r_i + \gamma max_{a'}Q_\omega(s',a')\right)\right]^2$$

用神经网络来近似Q值遇到的问题

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function[20]. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values (Q) and the target values $r + \gamma \max_{a'} Q(s', a')$.

- 强化学习不稳定，甚至发散
- 论文给出了三点原因

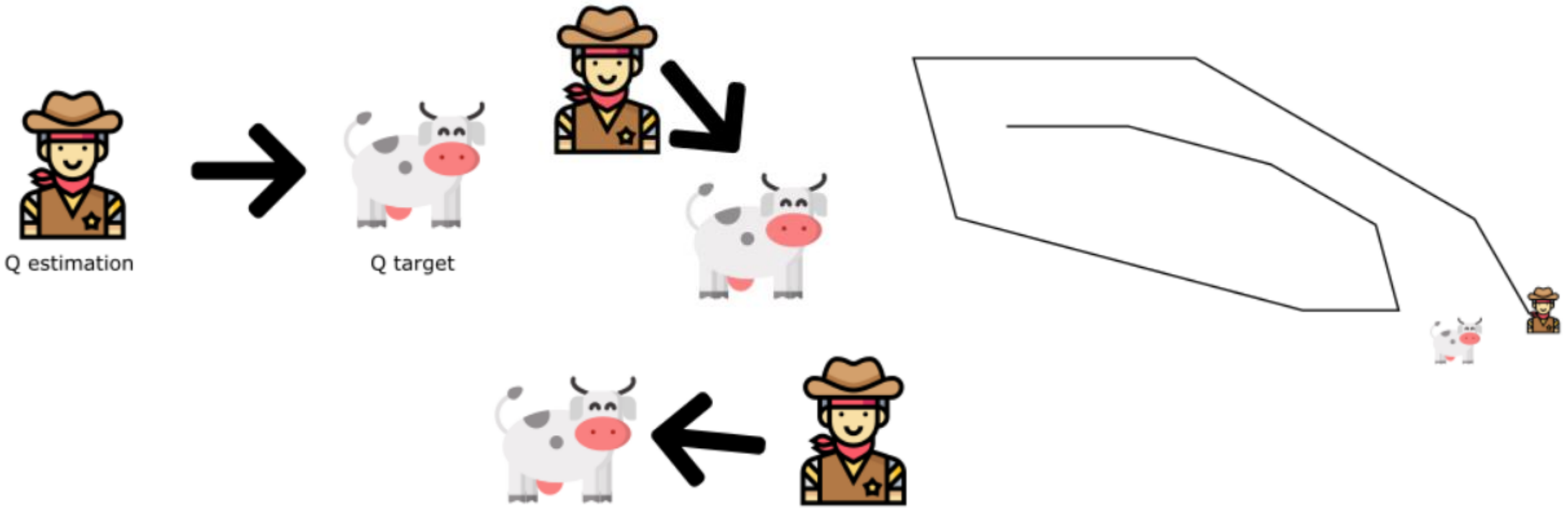
不稳定性的三个来源

instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values (Q) and the target values $r + \gamma \max_{a'} Q(s', a')$.

- 观测序列存在相关性
- 对Q值的微小更新可能显著改变策略，这会改变后续数据分布
- Action value（Q值）与目标值存在相关性

来源三：Q值与目标值的相关性

- Huggingface文档举的一个形象的例子



经验回放

- 设置一个重放内存区replay memory
  - 该区容量为N，N是一个可以定义的超参数
  - 每次将从环境中得到的四元组存放进去
    （状态、动作、奖励、下一状态）
- 好处
  - 通过从存储区里随机抽样，消除序列的相关性
  - 更有效地利用经验
  - 避免遗忘（catastrophic forgetting灾难性遗忘）

目标网络（固定Target）

- 既然目标值会动
  - 就先把目标网络固定住
  - 让训练网络逼近目标值
  - 每隔C步从Q网络复制参数来更新Q网络
- 这样就切断了二者之间的相关性

DQN的流程

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For t = 1,T **do**

With probability ε select a random action a_t

otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

End For

End For

代码 初始化Q网络

- 三个卷积层

  每个后面跟着一个ReLU

```python
class QNetwork(nn.Module):
    def __init__(self, env):
        super().__init__()
        self.network = nn.Sequential(
            nn.Conv2d(4, 32, 8, stride=4),
            nn.ReLU(),
            nn.Conv2d(32, 64, 4, stride=2),
            nn.ReLU(),
            nn.Conv2d(64, 64, 3, stride=1),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(3136, 512),
            nn.ReLU(),
            nn.Linear(512, env.single_action_space.n),
        )

    def forward(self, x):
        return self.network(x / 255.0)
```

代码 初始化目标网络

- 初始参数与主网络一致

```
q_network = QNetwork(envs).to(device)
optimizer = optim.Adam(q_network.parameters(), lr=args.learning_rate)
target_network = QNetwork(envs).to(device)
target_network.load_state_dict(q_network.state_dict())
```

代码 ε-greedy策略

```
epsilon = linear_schedule(args.start_e, args.end_e, args.
exploration_fraction * args.total_timesteps, global_step)
if random.random() < epsilon:
    actions = np.array([envs.single_action_space.sample()
    for _ in range(envs.num_envs)])
else:
    q_values = q_network(torch.Tensor(obs).to(device))
    actions = torch.argmax(q_values, dim=1).cpu().numpy()
```

代码 更新目标网络参数

- 论文里是硬更新
- 实际代码实现用的是软更新

  让目标网络参数缓慢、渐进地跟随主网络参数变化

  用τ的加权平均来实现

```
# update target network
if global_step % args.target_network_frequency == 0:
    for target_network_param, q_network_param in zip(target_network.parameters
    (), q_network.parameters()):
        target_network_param.data.copy_(
            args.tau * q_network_param.data + (1.0 - args.tau) *
            target_network_param.data
        )
```