人工智能实验期末 Project
通过强化学习实现黑白棋自动对弈模型

冯大纬 邓明昱

18340040 18340034

2021 年 1 月 7 日

摘要

我们基于 DQN (Deep Q Network) 实现了三种变种算法。一是通过对抗训练的 DQN 算法。二是使用了 EVA (Ephemeral Value Adjusments) 优化的 DQN。三是使用了蒙特卡洛树搜索 (Monte Carlo Tree Search) 的 DQN。

1 实验原理

1.1 强化学习 (Reinforcement Learning, RL)

马尔科夫决策过程 (Markov Decision processes, MDP) 是对强化学习环境的环境的正式表述。大部分的强化学习问题都可以规约到 MDP 模型上，甚至包括最优控制问题和不完全信息问题。MDP 是一个马尔科夫过程，每个状态只由上一个状态决定。

1.1.1 马尔科夫收益过程 (Markov Reward Processes, MRP)

在 MDP 模型中有一个概念为 MRP，其定义是一个具有价值衡量的马尔科夫链。其形式化定义为 $\langle S, P, R, \gamma \rangle$，其中 S 是一个有限状态集，P 是状态转移概率矩阵 $P_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$，$R$ 是奖励函数 $R_s = \mathbb{E}[R_{t+1|S_t=s}]$，$\gamma$ 是衰减系数 $\gamma \in [0, 1]$。而 MRP 的返回值 G_t 定义为 $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$。从状态 s 出发的价值函数 $V(s) = \mathbb{E}[G_t | S_t = s]$ 可以衡量这个状态的长期收益。将期望展开可以得到 MDP 中的一个重要概念贝尔曼方程 (Bellman Equation):

$$v(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s) v(s') \tag{1}$$

重写为矩阵形式为：

$$v = R + \gamma P v \tag{2}$$

可以发现 (2) 式实际上是一个线性方程，解为 $v = (I - \gamma P)^{-1} R$，求解需要 $O(n^3)$ 的时间复杂度 (n 个状态)。直接求解对于小规模的 MRP 问题是可行的。也有很多其他迭代做法可以处理这个问题，如动态规划、蒙特卡洛方法、时序差分学习。

1.1.2 马尔科夫决策过程 (Markov Decision processes, MDP)

MDP 是一个带有决策的 MRP，它是一个所有状态都具有马尔科夫性的环境。其形式化定义为 $\langle S, A, P, R, \gamma \rangle$，与 MRP 的形式化定义的差别仅在与引入了有限动作集 A。P 和 R 的定义由于 A 的引入而有少许修改：$P^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a], R^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$。在这个模型下，就可以通过一个 MDP 策略来定义一个智能体的行为：$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$。

在引入了策略后，这个 MDP 问题就可以建模为一个马尔科夫过程 $\langle S, P^\pi \rangle$，状态和奖励序列是一个由 $P^\pi_{ss'} = \sum_{a \in A} \pi(a|s) P^a_{ss'}$ 和 $R^\pi_s = \sum_{a \in A} \pi(a|s) R^a_s$ 确定的 $\langle S, P^\pi, R^\pi, \gamma \rangle$ MRP 过程。

1.1.3 MDP 中的状态值函数与动作值函数

状态值函数 (state-value function) 的定义为在状态 s 下使用策略 π 的期望收益，形式化为 $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$。动作值函数 (action-value function) 的定义为在状态 s 下使用动作 a 并基于策略 π 的期望收益，形式化为 $q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$。

将 (1) 贝尔曼方程带入这两条式子，可以得到：

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \tag{3}$$

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \tag{4}$$

1.1.4 策略迭代与值迭代

在求解大规模的 MDP 问题时，难以得到最大化 (3) 或 (4) 的策略的准确解。因此考虑通过迭代的方式进行计算。策略迭代 (Policy Iteration) 通过下式进行迭代：

$$q^{\pi_i}(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^{\pi_i}(s')$$

$$\pi_{i+1}(s) = \arg\max_a q^{\pi_i}(s, a)$$

值迭代 (Value Iteration) 通过下式进行迭代：

$$v(s) \leftarrow \max_{a \in A} [R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v(s')]$$

$$\pi(s) \leftarrow \arg\max_{a \in A} [R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v(s')]$$

1.1.5 无建模 (model-free) 的强化学习算法

在前文提到的 R, P 并不总是能完全暴露给智能体，因此需要一个基于与环境交互得到信息的无建模强化学习算法。在不能采用 MDP 模型时，蒙特卡洛策略评估 (Monte Carlo policy evaluation) 和时序差分学习 (Temporal Difference Learning, TD Learning) 依旧是可以使用的。蒙特卡洛方法通过随机采样结果来对真实的的 $v(s)$ 进行估计，当采样次数趋于正无穷时，显然是可以收敛到正确结果的。蒙特卡洛方法可以通过 $v(S_t) \leftarrow v(S_t) + (G_t - v(S_t))/N(S_t)$ 或者 running-mean 方法 $v(S_t) \leftarrow v(S_t) + \alpha(G_t - v(S_t))$ 来对 $v(s)$ 进行估计。

时序差分学习则是直接在不完整的经验集上通过反复抽样 (bootstrapping) 进行学习。一个简单的 TD 算法为通过下式进行迭代：

$$v(S_t) \leftarrow v(S_t) + \alpha[R_{t+1} + \gamma v(S_{t+1}) - v(S_t)] \tag{5}$$

其中 $R_{t+1} + \gamma v(S_{t+1})$ 被称为 TD target, $\delta_t = R_{t+1} + \gamma v(S_{t+1}) - v(S_t)$ 被称为 TD error。
类似地，可以使用 q 进行迭代：

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t)] \tag{6}$$

或者

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a q(S_{t+1}, a) - q(S_t, A_t)] \tag{7}$$

(6) 式的迭代方法被称为 SARSA (State Action Reward State Action), (7) 式被称为 Q Learning。

1.1.6 N-step Q Learning

值得一提的是，Q Learning 可以使用更长的轨迹进行迭代，相应地有下面的计算式：

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha[\sum_{i=1}^{N} R_{t+i} + \gamma \max_a q(S_{t+N}, a) - q(S_t, A_t)] \tag{8}$$

1.1.7 Deep Q Network

在有了 (7) 式以后，引入参数网络 Q 对 q 函数进行拟合也就变得顺理成章了。将 (7) 式改写为：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{9}$$

同理，前文提到的其他非建模学习方法也可以通过这种方法进行拟合，然后通过反复抽样学习对 Q 网络进行更新以拟合 q 值，这就是 DQN。

1.2 多智能体学习

在多智能体学习 (Multi-agent Learning, MAL) 中将其他玩家考虑为环境的一部分，使用上一个子节推导的单智能体 Q Learning 算法进行迭代会违背其一开始对环境做出的理论假设，学习过程会不稳定。

黑白棋是一个二人零和马尔科夫游戏。在这个模型下，双方策略是对立的，有 $\pi(\sigma, \sigma^-) = -\pi(\sigma^-, \sigma)$。那么可以定义一个新的学习模型 Minimax-Q Learning。双方的优化目标可以转化为：

$$v(s) = \max_{\sigma} \left\{ \min_{a^-} \sum_{a} \sigma(a) Q(s, a, a^-) \right\} \tag{10}$$

这个问题可以通过双方各自优化自己的策略来解决。因此在这个特殊的模型下，依然可以使用单智能体 Q Learning 算法。在黑白棋中，因为双方完全对立，且都具有对整个游戏环境的完全观测，所以可以使用同一个 Q 网络对环境进行估值。拆分成两个完全独立的网络训练也是可行的。

1.3 DQN 的优化

1.3.1 TargetNet

如果边更新网络边采样，可能会导致策略频繁变化缺乏稳定性。因此有一个技巧就是将参数暂时冻结进行采样，而当 Q 网络有了一定量的更新时，将采样的网络与 Q 网络进行同步。这个技巧可以提高 DQN 训练的稳定性。

1.3.2 Dueling DQN

将 (8) 式中的状态值函数与动作值函数分离，在 Q 网络中通过不同的网络分支对两个函数分别进行拟合。这个技巧对 DQN 的训练有普遍的加速。

1.3.3 优先经验重放 (Prioritized Experience Replay)

优先经验重放机制的出现是为了更好地使用经验池中的一些高价值数据。普通经验池在采样时，各个经验会以相同的概率被抽取。而优先经验重放方法则对经验以 TD error 加权进行抽取。加权方法和经验池维护方法是多样的，不同的设计会在优化效果和运行速度方面有极大的不同。

1.3.4 Ephemeral Value Adjusments (EVA)

在原有的参数网络估计的基础上，再引入一个状态中心的非参数估计算法，以应用更多的轨迹中心的信息。这就是 EVA 优化的核心思想。其更新策略为：

$$Q(s,a) = \lambda Q_\theta(s,a) + (1-\lambda) Q_{NP}(s,a) \tag{11}$$

Q_θ 是参数网络，Q_{NP} 是非参数算法。EVA 优化目前提出了三种 Q_{NP} 的计算方法，分别是 n-step, trajectory-centric planning (TCP) 和 kernel-based RL (KBRL)。三种算法的计算方法如下：

n-step:

$$V_{NP} = \begin{cases} \max\limits_a Q_\theta(s_t,a) & if\ t = T \\ r_t + \gamma V_{NP}(s_{t+1}) & otherwise \end{cases} \tag{12}$$

TCP:

$$Q_{NP} = \begin{cases} r_t + \gamma V_{NP}(s_{t+1}) & if\ a_t = a \\ Q_\theta(s_t,a) & otherwise \end{cases} \tag{13}$$

$$V_{NP} = \begin{cases} \max\limits_a Q_\theta(s_t,a) & if\ t = T \\ \max\limits_a Q_{NP}(s_t,a) & otherwise \end{cases} \tag{14}$$

KBRL:

$$Q_{NP}(s_t,a_t) = \sum_{(s,r,s')\in S_a} \kappa(s_t,s)[r + \gamma \max_{a'} Q_{NP}(s',a')] \tag{15}$$

其中 κ 是一个高斯核函数，应用这个优化需要从网络中产生一个状态的嵌入 (embedding)，这个值通常可以从 Q 网络的中间层引出。

2 评价函数、伪代码

2.1 评价函数

中间状态无奖励值，终局估值为黑子数减去白子数。一开始还使用了一种在后来被认为不合理的评价函数，即只使用对局是否胜利作为评判。

2.2 对抗训练 DQN

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

图 1: 对抗训练 DQN 伪代码

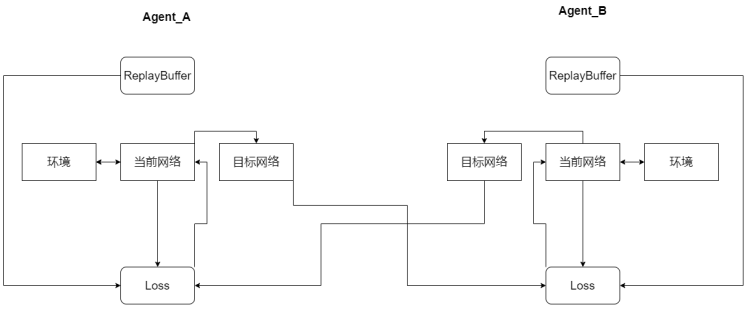

图 2: 对抗训练 DQN 流程图

2.3 EVA

Input : Replay buffer $\mathcal{D}$
Value buffer $\mathcal{L}$
Mixing hyper-parameter λ
Maximum roll-out hyper-parameter τ
for $e := 1, \infty$ **do**
  for $t := 1, T$ **do**
    Receive observation s_t from environment with embedding h_t
    Collect trace computed values from k nearest neighbours
    $Q_{\text{NP}}(s_k, \cdot) | h(s_k) \in \text{KNN}(h(s_t), \mathcal{L})$
    $Q_{\text{EVA}}(s_t, \cdot) := \lambda Q_\theta(\hat{s}, \cdot) + (1 - \lambda) \frac{\sum_{k=0}^{K} Q_{\text{NP}}(s_k, \cdot)}{K}$
    $a_t \leftarrow \epsilon$-greedy policy based on $Q_{\text{EVA}}(s_t, \cdot)$
    Take action a_t, receive reward r_{t+1}
    Append $(s_t, a_t, r_{t+1}, h_t, e)$ to $\mathcal{D}$
    $\mathcal{T}_m := (s_{t:t+\tau}, a_{t:t+\tau}, r_{t+1:t+\tau+1}, h_{t:t+\tau}, e_{t:t+\tau}) | h(s_m) \in \text{KNN}(h(s_t), \mathcal{D}))$
    $Q_{\text{NP}} \leftarrow$ using $\mathcal{T}_m$ via the TCP algorithm
    Append (h_t, Q_{NP}) to $\mathcal{L}$
  end
end

图 **3:** EVA 伪代码

3 关键代码

3.1 Q 网络

```
class QNet(nn.Module):
    def __init__(self, embedding_dim=512):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 64, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 128, 3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.conv4 = nn.Conv2d(128, 128, 3, padding=1)
        self.bn4 = nn.BatchNorm2d(128)
        self.fc1 = nn.Linear(8 * 8 * 128, embedding_dim)
        self.fc2 = nn.Linear(embedding_dim, 65)

    def forward(self, x):
        x = x.view(-1, 1, 8, 8)
        x = torch.relu(self.bn1(self.conv1(x)))
        x = torch.relu(self.bn2(self.conv2(x)))
        x = torch.relu(self.bn3(self.conv3(x)))
        x = torch.relu(self.bn4(self.conv4(x)))
        x = torch.sigmoid(self.fc1(x.view(-1, 8 * 8 * 128)))
        # embedding = x
        x = torch.sigmoid(self.fc2(x))
        return x  # , embedding
```

3.2 关键接口

```
s = game_state.Get_State() # 获取局面
a = agent_a.eps_greedy(s, game_state, 1) # 做出决策
game_state.Add(1, a) # 局面更新
r = game_state.Gameover() * 100.0 # 计算reward
s_ = game_state.Get_State() # 获取下一局面
agent_a.Store_transition(s, a, r, s_) # 加入经验池
agent_a.Learn(agent_b.Q_) # DQN学习
```

4 实验结果

我们实现了对抗学习 DQN、EVA 优化自对弈 DQN、基于蒙特卡洛方法的自对弈 DQN。但是因为资源有限，我们只对第一种算法进行了大量训练。第三种做法采用了别人的预训练网络参数并进行了少量自对弈训练。预训练参数的棋力是测试中最高的。因为棋类游戏没有可供参考的得分，所以只能与一些固定模型对弈作为参考。这里采用了纯随机、Minimax 和 AlphaBeta 剪枝的三种模型作为参考。可以发现，在越过了某些临界点后，对弈胜率会迅速提高，符合常理。

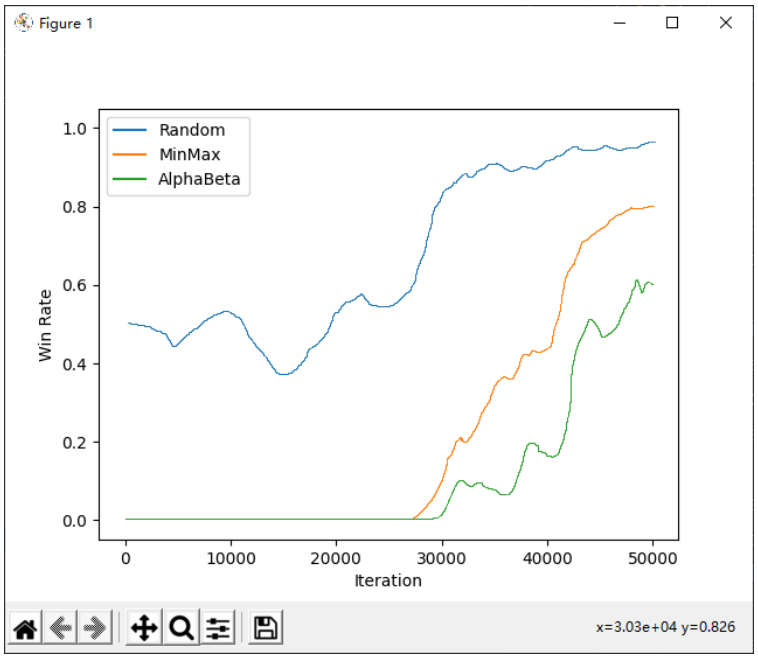

图 4: 对抗学习对三种简单 AI 的战绩

表 1: 对局胜率

算法	胜率
Random	0.929
MiniMax	0.794
AlphaBeta	0.585

5 总结与感想

感觉挺可惜的一点是手上有很多已经实现的算法但没有资源去训练。另一方面，因为两个人都对黑白棋毫无了解，导致缺乏一些非常重要的先验经验，同时也没有办法对对弈过程中的行为和估值进行分析。在临近提交时，和一位对黑白棋颇有研究的同学交流，得知了许多可用于算法设计的知识。

如果想要进一步优化效果，可以通过棋谱进行初期训练，然后进行大量的自对弈。同时添加终局搜索。在距离结束状态 20 步时，带剪枝的搜索就已经可以在能接受的时间范围内完成了。

References

Hansen, Steven, Alexander Pritzel, Pablo Sprechmann, André Barreto, and Charles Blundell, "Fast deep reinforcement learning using online adjustments from the past," in Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, eds., *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018, pp. 10590–10600.

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver, "Prioritized Experience Replay," in Yoshua Bengio and Yann LeCun, eds., *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.