人工智能实验

Artificial Intelligence Laboratory

实验 10 强化学习

计算机学院

2020.12.31

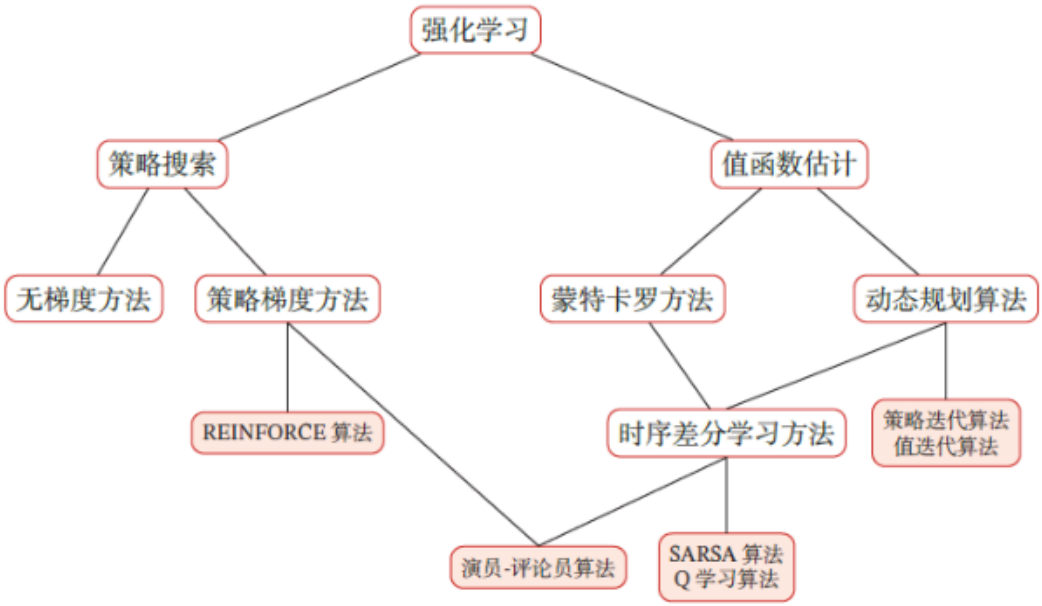
一、算法原理

1、背景

强化学习（Reinforcement Learning，RL），也叫增强学习，是指一类从（与环境）交互中不断学习的问题以及解决这类问题的方法。强化学习问题可以描述为一个智能体从与环境的交互中不断学习以完成特定目标（比如取得最大奖励值）。

一般的监督学习一般需要一定数量的带标签的数据。在很多的应用场景中，通过人工标注的方式来给数据打标签的方式往往行不通。比如在棋类游戏中，训练一个好的模型就需要收集大量的不同棋盘状态以及对应动作。这种做法实践起来比较困难，一是对于每一种棋盘状态，即使是专家也很难给出“正确”的动作，二是获取大量数据的成本往往比较高。对于下棋这类任务，虽然我们很难知道每一步的“正确”动作，但是其最后的结果（即赢输）却很容易判断。因此，这类问题可以通过强化学习解决，通过让智能体在与环境的交互过程中不断地试错，并从中学习到使期望回报最大的行动策略。

强化学习的方法可以大致分为两个方向，我将在介绍完相关概念之后，分别介绍这些方法。



2、相关概念

状态（state）：Agent 当前的所处的环境

动作（action）：Agent 根据当前 state 采取的行动。

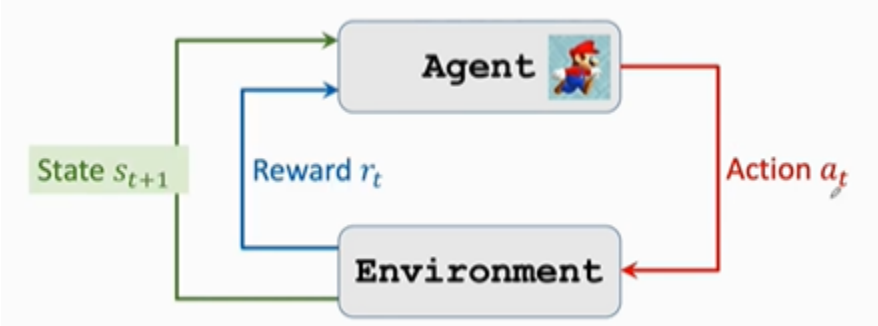
策略（policy）：Agent 根据某一状态采取动作的方式。Policy 往往有随机性。

$$\pi(a \mid s) = \mathbb{P}(A = a \mid S = s).$$

及时奖励（reward）：Agent 采取某个行动的所得到的结果。

状态转移：Agent 在执行某个动作后会产生新的状态。状态转移也会有随机性。这种随机性来自系统。

- $p(s' \mid s, a) = \mathbb{P}(S' = s' \mid S = s, A = a).$



轨迹（trajectory）：轨迹是智能体与环境交互的整个过程，可以用三元组（state、action、reward）叠加表示。

回报（return）：未来的累计奖励。

- $U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \cdots$

折扣回报（discounted return）：折扣后的未来累计奖励。由于未来具有很大的不确定性，我们不能保证能够得到未来的奖励，因此乘以一个折扣率，使得未来奖励变低，而且未来越远，期望也就越低。折扣率是一个超参数，介于 0-1 之间。

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

At time step t, the return U_t is random.

- Two sources of randomness:
  1. Action can be random: $\mathbb{P}[A = a \mid S = s] = \pi(a \mid s)$.
  2. New state can be random: $\mathbb{P}[S' = s' \mid S = s, A = a] = p(s' \mid s, a)$.

动作价值函数（action-value function）：因为回报是一个随机变量，依赖于之后的状态和行动，因此定义动作价值函数，它是回报的期望值。该函数可以评价 Agent 在当前环境 st 采取动作 at 的好坏。

- $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$.

最优动作价值函数（optimal action-value function）：由于我们的期望回报值是跟某一个特定策略有关的，因此我们可以定义其中最大的为

- $Q^*(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t)$.

值得注意的是，动作价值函数满足贝尔曼（Bellman）方程

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s', a') | s, a\right]$$

状态价值函数（state-value function）：该函数用于评估使用某个策略 pi 的情况下当前状态的好坏。

- $V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$. (Actions are discrete.)
- $V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)] = \int \pi(a|s_t) \cdot Q_\pi(s_t, a)\, da$. (Actions are continuous.)

3、价值学习（Value-Based Reinforcement Learning，Q-learning）

3.1 动机

由于最优动作价值函数$Q^*(s_t, a_t)$是当前状态 st 下采取动作 at 所能获得的最大期望回报，因此根据强化学习的马尔科夫性质，我们只要每次选取使得期望回报最大的动作即可。所以，能否准确地估计出当前状态 st 下采取动作 at 所能获得的最大期望回报，决定了智能体采取的动作的好坏。因此，价值学习就是为了学习如何估计好期望回报。

3.2 Deep Q-network(DQN)

由于往往整个状态空间是很大的，而且动作空间也可能很大，因此对于如此一个复杂的映射关系，我们可以用神经网络去拟合，由此也就产生了 DQN 算法。

DQN: Approximate $Q^*(s, a)$ using a neural network (DQN).

- $Q(s, a; \mathbf{w})$ is a neural network parameterized by $\mathbf{w}$.
- Input: observed state s.
- Output: scores for every action $a \in \mathcal{A}$.

3.3 Temporal Difference（TD，时序差分学习）

对于上述的基于深度学习的算法，其学习过程需要真实样本的标签，而获取

每一个状态以及对应采取的某一行动最后所产生的回报的真实值需要完整地完成整个游戏，这样的策略显然是效率低下的。由于我们知道最优动作价值函数$Q^*(s_t, a_t)$满足贝尔曼方程，因此，我们可以利用每一步环境给予的及时奖励（reward）来更新网络。及时奖励即为真实标签的一部分，包含着正确的信息。

我们首先定义 TD target 为

$$y_t = r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w}_t)$$
$$= r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t).$$

神经网络的损失函数为：

$$\text{Loss: } L_t = \frac{1}{2}[Q(s_t, a_t; \mathbf{w}) - y_t]^2.$$

我们通过对损失函数求导，利用梯度下降法，即可更新网络。

$$\text{Gradient descent: } \mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L_t}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}.$$

3.4 DQN+TD 算法

Temporal Difference (TD) Learning

Algorithm: One iteration of TD learning.

1. Observe state $S_t = s_t$ and action $A_t = a_t$.
2. Predict the value: $q_t = Q(s_t, a_t; \mathbf{w}_t)$.
3. Differentiate the value network: $\mathbf{d}_t = \frac{\partial Q(s_t, a_t; \mathbf{w})}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.
4. Environment provides new state s_{t+1} and reward r_t.
5. Compute TD target: $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t)$.
6. Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot (q_t - y_t) \cdot \mathbf{d}_t$.

7.重复以上过程直到模型收敛。

3.5 ε贪心策略

$$\pi^{\epsilon}(s) = \begin{cases} \pi(s), & \text{按概率} 1-\epsilon, \\ \text{随机选择} \mathcal{A} \text{中的动作}, & \text{按概率} \epsilon. \end{cases}$$

如果采用确定性策略π，每次试验得到的轨迹是一样的，只能计算出$Q\pi(s, \pi(s))$，而无法计算其他动作a'的Q 函数，因此也无法进一步改进策略。这样情况仅仅是对当前策略的利用（exploitation），而缺失了对环境的探索（exploration），即试验的轨迹应该尽可能覆盖所有的状态和动作，以找到更好的策略。

3.6 DQN+TD+Experience replay 算法

上述引进时序差分学习之后可能会有以下两个问题：

• 由于时序差分学习是在同一条轨迹上采样并进行更新的，这样样本之间的相关性会更高，导致模型学习会很差。

• 同时，如果模型在往某一方面优化之后，很可能会朝着这方面一直学习。比如说模型学习到往左移动的回报很高，这样模型就会倾向于向左移动，导致学习样本中大多数都是向左移动的。

因此，我们可以设置一个经验池，其中保存着历史的 (s_t, a_t, r_t, s_{t+1}) 四元组，我们可以在训练过程中通过从经验池里面采样学习，这样学习到的样本更加丰富。

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

4、策略学习（Policy-Based Reinforcement Learning）

4.1 动机

由于强化学习的目标是为了让智能体学习到使得回报最大的行动策略，因此我们可以直接学习最优的策略。

同时，值函数的空间可能会非常大，可能的映射关系会非常复杂，即使采用了神经网络，可能拟合的效果都不会很好。但是，可能的动作会很简单，比如向左或者向右移动。因此有时候直接学习策略也是一种不错的方法。

4.2 REINFORCE algorithm

同样，根据当前状态采取某一个行动很可能也是一个复杂的映射关系，我们也用一个神经网络$\pi(a|s,\theta)$来拟合策略函数$\pi(a|s)$，其中θ是神经网络的参数。

我们定义J(θ)

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t|\pi_\theta\right] = \mathbb{E}_S[V(S;\boldsymbol{\theta})] = \int_\tau r(\tau)p(\tau;\theta)\mathrm{d}\tau$$

其中，τ是一条轨迹，r(τ)代表某条轨迹的回报，p(τ; ϑ)是某条轨迹的概率。因此，J(θ)是对状态价值函数的所有状态求期望，其取值大小只与所采用的的策略有关，因此一个策略的好坏可以通过J(θ)来判断。我们希望找到使得J(θ)的那个神经网络。

$$\theta^* = \arg\max_\theta J(\theta)$$

为了得到最优的参数，我们可以使用梯度上升法，但是J(θ)是某个策略下很多轨迹的回报的期望值，根据多变量求导的原则，我们需要穷举所有的轨迹，这样显然是不可能的。因此，我们采用蒙特卡罗抽样的方法，用抽样的轨迹所算出来的梯度来近似真实梯度。

$$\begin{aligned}\nabla_\theta J(\theta) &= \int_\tau r(\tau)\nabla_\theta p(\tau;\theta)\mathrm{d}\tau \\ &= \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau \\ &= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]\end{aligned}$$

其中

$$\text{We have: } p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)$$

$$\text{Thus: } \log p(\tau;\theta) = \sum_{t\geq 0}\log p(s_{t+1}|s_t,a_t) + \log\pi_\theta(a_t|s_t)$$

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0}\nabla_\theta \log\pi_\theta(a_t|s_t)$$

Therefore when sampling a trajectory τ, we can estimate J(θ) with

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log\pi_\theta(a_t|s_t)$$

算法 14.6　REINFORCE 算法

输入: 状态空间 $\mathcal{S}$,动作空间 $\mathcal{A}$,可微分的策略函数 $\pi_\theta(a|s)$,折扣率 γ,学习率 α;

随机初始化参数 θ;

repeat

　根据策略 $\pi_\theta(a|s)$ 生成一条轨迹:$\tau = s_0, a_0, s_1, a_1, \cdots, s_{T-1}, a_{T-1}, s_T$;

　for t=0 **to** T **do**

　　计算 $G(\tau_{t:T})$;

　　$\theta \leftarrow \theta + \alpha\gamma^t G(\tau_{t:T}) \frac{\partial}{\partial\theta} \log \pi_\theta(a_t|s_t)$;　　// 更新策略函数参数

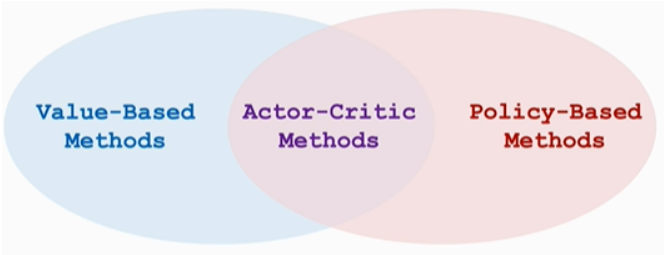
　end

until π_θ 收敛;

输出: 策略 π_θ

同样，为了加快训练速度，我们不把整个游戏全部运行完，而是只运行一步，然后通过某种方法估计出$Q(s_t, a_t) = r(\tau)$，我们可以借鉴前面基于值函数的方法，利用一个神经网络去拟合$Q(s_t, a_t)$，这样就是下面的演员-评论员算法。

5、演员评论员算法（Actor-Critic Methods）



由状态价值函数的定义，它可以分解为两个部分：π(a|s)和$Q(s,a)$。

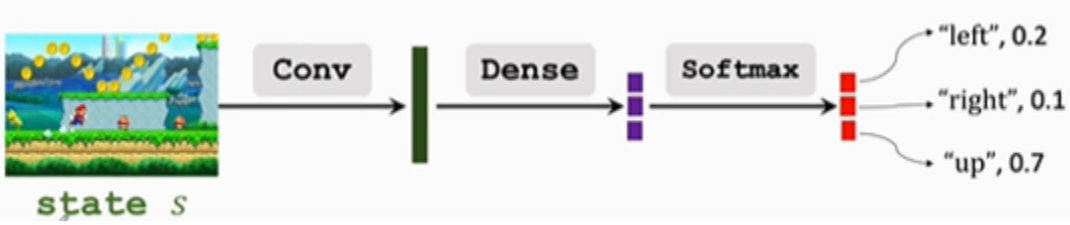
$$V_\pi(s) = \sum_a \pi(a|s) \cdot Q_\pi(s,a).$$

因此我们可以用两个网络分别近似这两个映射。

Policy network (actor):

- Use neural net $\pi(a|s;\boldsymbol{\theta})$ to approximate $\pi(a|s)$.
- $\boldsymbol{\theta}$: trainable parameters of the neural net.

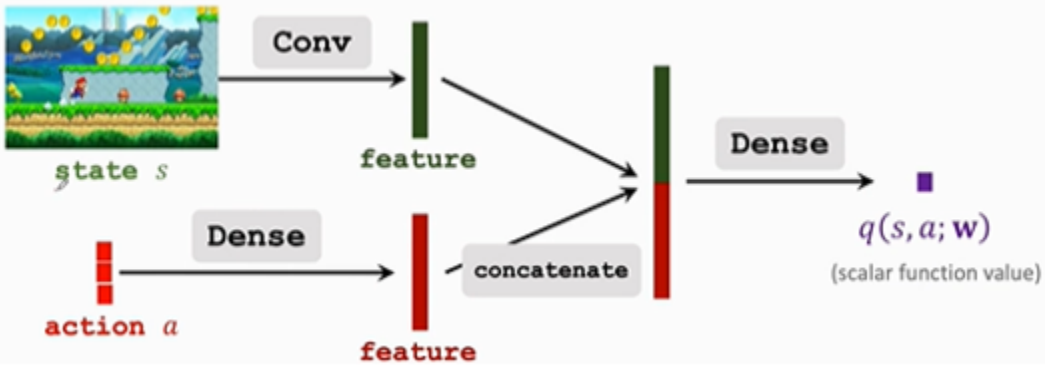
- Input: state s, e.g., a screenshot of Super Mario.
- Output: probability distribution over the actions.



Value network (critic):

- Use neural net $q(s, a; \mathbf{w})$ to approximate $Q_\pi(s, a)$.
- $\mathbf{w}$: trainable parameters of the neural net.

- Inputs: state s and action a.
- Output: approximate action-value (scalar).



Training: Update the parameters $\boldsymbol{\theta}$ and $\mathbf{w}$.

- Update policy network $\pi(a|s; \boldsymbol{\theta})$ to increase the state-value $V(s; \boldsymbol{\theta}, \mathbf{w})$.
  - Actor gradually performs better.
  - Supervision is purely from the value network (critic).
- Update value network $q(s, a; \mathbf{w})$ to better estimate the return.
  - Critic's judgement becomes more accurate.
  - Supervision is purely from the rewards.

Summary of Algorithm

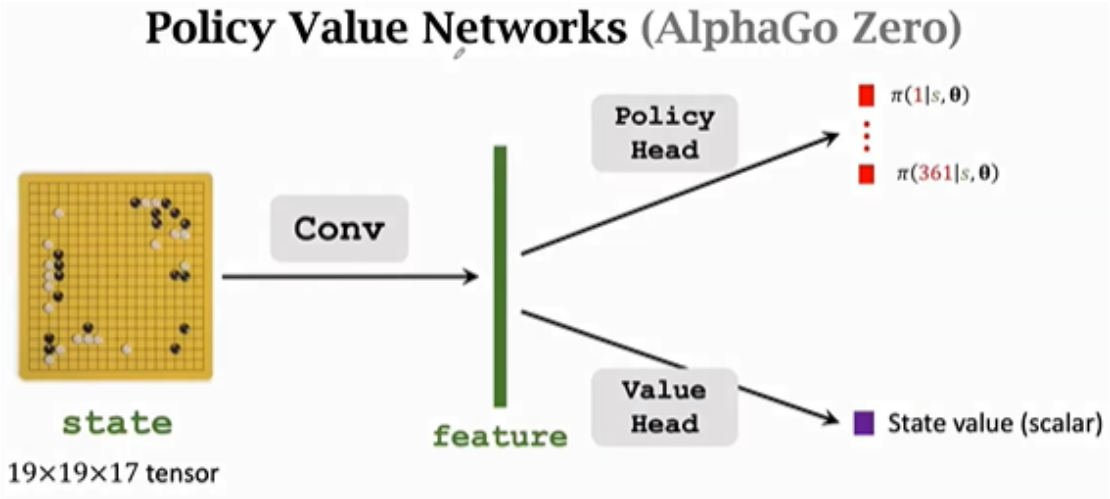
1. Observe state s_t and randomly sample $a_t \sim \pi(\cdot\,|s_t; \boldsymbol{\theta}_t)$.
2. Perform a_t; then environment gives new state s_{t+1} and reward r_t.
3. Randomly sample $\tilde{a}_{t+1} \sim \pi(\cdot\,|s_{t+1}; \boldsymbol{\theta}_t)$. (Do not perform $\tilde{a}_{t+1}$!)
4. Evaluate value network: $q_t = q(s_t, a_t; \mathbf{w}_t)$ and $q_{t+1} = q(s_{t+1}, \tilde{a}_{t+1}; \mathbf{w}_t)$.
5. Compute TD error: $\delta_t = q_t - (r_t + \gamma \cdot q_{t+1})$.
6. Differentiate value network: $\mathbf{d}_{w,t} = \frac{\partial q(s_t, a_t; \mathbf{w})}{\partial\, \mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}_t}$.
7. Update value network: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \delta_t \cdot \mathbf{d}_{w,t}$.
8. Differentiate policy network: $\mathbf{d}_{\theta,t} = \frac{\partial \log \pi(a_t|s_t, \boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$.
9. Update policy network: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \beta \cdot q_t \cdot \mathbf{d}_{\theta,t}$.

9. Update policy network: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \beta \cdot \delta_t \cdot \mathbf{d}_{\theta,t}$.（with Baseline）

6、AlphaGo Zero

6.1 概述

在 AlphaGo 击败李世石之后，又推出了 AlphaGo Zero，AlphaGo Zero 不再采用人类棋谱，但是以碾压地优势击败 AlphaGo，也为其他棋类游戏提供了借鉴思路。因此，本次实验致力于复现 AlphaGo Zero，并将其应用于黑白棋中。



AlphaGo Zero 可以理解为演员评论家算法+蒙特卡罗树搜索。如上图所示，价值网络（评论家）和策略网络（演员）会共用一个卷积神经网络，并在卷积神经网络末尾接上不同的输出头来实现不同的功能。该网络的输出会指导蒙特卡罗树搜索，蒙特卡罗树搜索的结果也会反过来更新该神经网络的参数。值得注意的是，由于一盘棋局的中间状态不好设置即时奖励（中间状态太多，并且并不能保证谁一定会赢），因此 AlphaGo Zero 没有采用时序差分学习，而是把完整的一盘游戏玩完，利用最后的输赢，作为每一个时刻的预计得分，即

$$V_\pi(s_t) = V_\pi(s_{t+1}) = \cdots = V_\pi(s_T) = \mathrm{z} \qquad (\mathrm{z} = 1/-1)$$

AlphaGo Zero 训练过程主要分为 3 个阶段：自我对战学习阶段、训练神经网络阶段和评估网络阶段。

6.2 自我对战学习阶段

自我对战学习阶段主要是 AlphaGo Zero 自我对弈，产生大量棋局样本的过程，由于 AlphaGo Zero 并不使用围棋大师的棋局来学习，因此需要自我对弈得到训练数据用于后续神经网络的训练。在自我对战学习阶段，每一步的落子是由蒙

特卡罗树搜索（MCTS）来完成的。在 MCTS 搜索的过程中，遇到不在树中的状态，则使用神经网络的结果来更新 MCTS 树结构上保存的内容。在每一次迭代过程中，在每个棋局当前状态 s 下，每一次移动使用 1600 次 MCTS 搜索模拟。最终 MCTS 给出最优的落子策略 p，这个策略 p 和神经网络的输出 π 是不一样的。当每一局对战结束后，我们可以得到最终的胜负奖励 z（1 或者-1）。这样我们可以得到非常多的样本(s,p,z),这些数据可以训练神经网络阶段。

6.3 训练神经网络阶段

在训练神经网络阶段，我们使用自我对战学习阶段得到的样本集合(s,p,z),训练我们神经网络的模型参数。训练的目的是对于每个输入 s，神经网络输出的 π 、v 和我们训练样本中的 p、z 差距尽可能的少。因此我们可以定义如下损失函数：

$$L_{value} = (z - v)^2$$

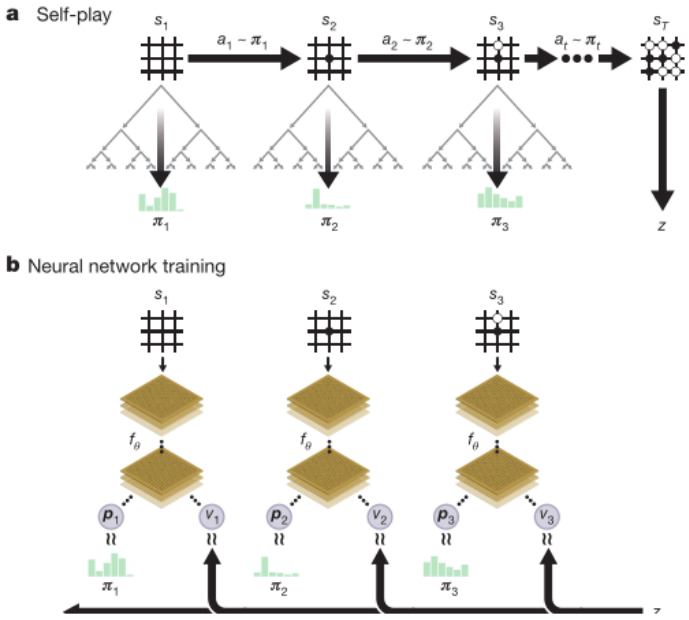
$$L_{policy} = cross\_entropy(\pi, p)$$

$$\mathrm{L} = L_{value} + L_{policy} + c\|\theta\|^2$$

损失函数由三部分组成，第一部分是均方误差损失函数，用于评估神经网络预测的胜负结果和真实结果之间的差异。第二部分是交叉熵损失函数，用于评估神经网络的输出策略和 MCTS 输出的策略的差异。第三部分是 L2 正则化项，减少过拟合程度。

通过训练神经网络，我们可以优化神经网络的参数 θ，用于后续指导我们的 MCTS 搜索过程。

整个训练过程可以用下面的图展示出来。



6.4 评估网络阶段

这个阶段主要用于确认神经网络的参数是否得到了优化。这个过程中，博弈双方为当前训练轮次的神经网络，以及之前训练轮次保存的最好的网络。自我对战的双方各自使用自己的神经网络指导 MCTS 搜索，并对战若干局，检验 AlphaGo Zero 在新神经网络参数下棋力是否得到了提高。除了神经网络的参数不同，这个过程和第一阶段的自我对战学习阶段过程是类似的。

6.5 上限置信区间算法 UCT

在棋类游戏中，经常有这样的问题，我们发现在某种棋的状态下，有 2 个可选动作，第一个动作历史棋局中是 0 胜 1 负，第二个动作历史棋局中是 8 胜 10 负，那么我们应该选择哪个动作好呢？如果按ϵ−贪婪策略，则第二个动作非常容易被选择到。但是其实虽然第一个动作胜利 0%，但是很可能是因为这个动作的历史棋局少，数据不够导致的，很可能该动作也是一个不错的动作。那么我们如何在最优策略和探索度达到一个选择平衡呢？ϵ−贪婪策略可以用，但是 UCT 是一个更不错的选择。

首先为每一个节点定义如下 4 个变量：

N(s,a)：表示在状态 s 下选择动作 a 的次数。

W(s,a)：表示在状态 s 下选择动作 a 的总行动价值。

Q(s,a)：表示在状态 s 下选择动作 a 的平均行动价值。

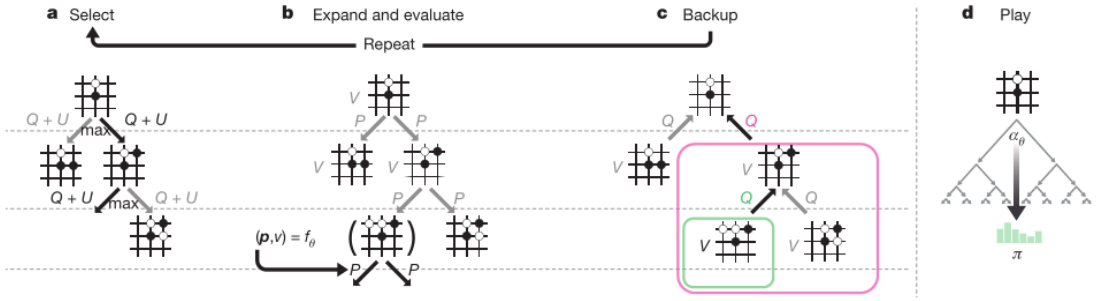
P(s,a)：表示在状态 s 下选择动作 a 的先验概率。

UCT 首先计算每一个可选动作节点对应的分数，这个分数考虑了历史最优策略和探索度，然后 UCT 会选择分数最高的动作执行。

$$U(s,a) = c_{puct} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

$$a_t = \arg\max_a (Q(s_t,a) + U(s_t,a))$$

6.6 Monte Carlo Tree Search



第 1 步：select

对于所有可能的动作，我们 UCT 给每个动作打分（公式如上），然后选择分数最高的动作。

第 2 步：expand and evaluate

由于一个好的智能体应该要知道自己下了一步棋之后会有什么结果，因此它需要模拟对手下棋。但是对手的选择是不可预知的，因此我们用“推己及人”的想法，从自己的策略函数中随机采样来作为对手的行动，这样一直扩展直到叶子节点。在扩展的过程中会产生很多新的中间节点，我们把它按照下面的公式初始化，其中 pa 是从策略网络输出的概率。

$$\{N(s_L,a)=0, \dot{W}(s_L,a)=0, Q(s_L,a)=0, \dot{P}(s_L,a)=p_a\}$$

而对于叶子节点，如果一局已经玩完了，则令 v=±1，否则用价值网络估计一个 v。

第 3 步：backup

做完了扩展和仿真后，我们需要进行回溯，将新叶子节点分支的信息回溯累加到祖先节点分支上去。这个回溯的逻辑也是很简单的，从每个叶子节点 L 依次向根节点回溯，并依次更新上层分支数据结构如下：

$$N(s_t, a_t) = N(s_t, a_t) + 1$$
$$W(s_t, a_t) = W(s_t, a_t) + v$$
$$Q(s_t, a_t) = \frac{W(s_t, a_t)}{N(s_t, a_t)}$$

第 4 步：play

再执行完上述的 3 步大约 1600 次之后，AlphaGo Zero 才真正执行一步 a。a 为下面的 pi 值最大那个。

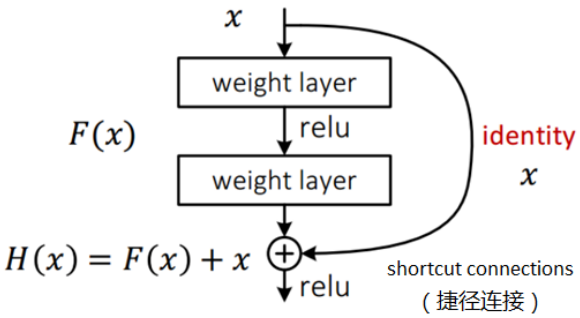
$$\pi(a|s) = \frac{N(s,a)^{1/\tau}}{\sum_b N(s,b)^{1/\tau}}$$

其中，τ 为温度参数，控制探索的程度，τ 越大，不同走法间差异变小，探索比例增大，反之，则更多选择当前最优操作。

值得注意的是，在 AlphaGo Zero 在真正走了一步之后，其搜索树不会被全部丢弃，它选择的那个子节点变成新的树根，这个子树的所有值将被保存下来。

7、ResNet

自深度学习发展以来，deeper learning 是人们追求的目标，但是人们发现越深的网络越难优化，模型性能甚至不如浅层网络，因此 ResNet 作者何恺明提出残差的思想，使得网络可以很轻松的扩展到上百层，同时模型性能可以随着模型的深度的加深而提高。



残差思想不同于直接学习特征图，而是学习输入与特征图之间的残差。假设特征图为 H(x)，残差网络试图学习 F(x)=H(x)-x，所以最后的特征图应该是 F(x)+x。这种残差的思想不会增加任何的计算量，反而会使得模型更容易训练。残差结构如上图所示，直接把原始输入与输出相加，即为需要的特征图。这种残差跳跃式的结构，打破了传统的神经网络 n-1 层的输出只能给 n 层作为输入的惯例，使某

一层的输出可以直接跨过几层作为后面某一层的输入，称为“shortcut connections”。

这种“shortcut connections”有两种情况：

• shortcut connections 前后通道数相同。由于通道数相同，所以可以直接相加，采用计算方式为 H(x)=F(x)+x。

• shortcut connections 前后通道数不同。通道不同，采用的计算方式为 H(x)=F(x)+Wx，其中 W 是卷积操作，用来调整 x 维度的。

为什么残差结构有作用，作者给出了一种解释：解决梯度消失问题。

考虑任意两层网络 l1 和 l2，递归展开表达式：

$$\begin{aligned} a^{(l2)} &= a^{(l2-1)} + f\left(a^{(l2-1)}\right) \\ &= (a^{(l2-2)} + f\left(a^{(l2-2)}\right)) + f\left(a^{(l2-1)}\right) \\ &= a^{(l1)} + \sum_{i=l1}^{l2-1} f\left(a^{(i)}\right) \end{aligned}$$

所以在前向传播时，输入信号可以从任意低层直接传播到高层。由于包含了一个天然的恒等映射，一定程度上可以解决网络退化问题。同时，反向传播

$$\frac{\partial L}{\partial a^{(l1)}} = \frac{\partial L}{\partial a^{(l2)}} \times \frac{\partial a^{(l2)}}{\partial a^{(l1)}} = \frac{\partial L}{\partial a^{(l2)}}\left(1 + \frac{\partial}{\partial a^{(l1)}} \sum_{i=l1}^{l2-1} f\left(a^{(i)}\right)\right)$$

根据该公式，反向传播时，梯度可以顺着 shortcut connections 直接传播到底层，从而缓解了梯度消失的问题。综上，可以认为残差连接使得信息前后向传播更加顺畅。

二、实验设置

本次实验致力于复现 AlphaGo Zero，并把它应用于黑白棋中。同时，为了验证 AlphaGo Zero 算法的优越性，我们同时用 DQN+Experience replay+ε贪心的方法训练另一个网络，让这两个网络进行对抗。

1、AlphaGo Zero 模型

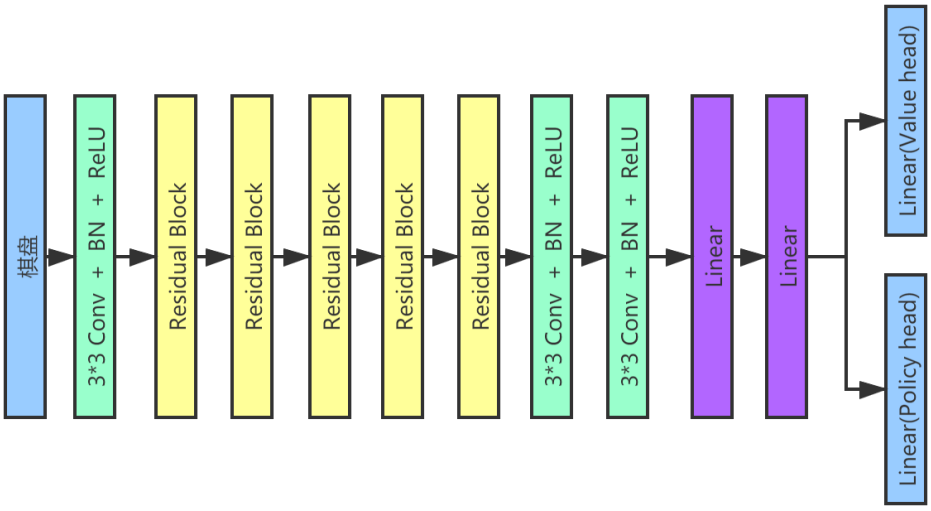
1.1 训练超参数设置

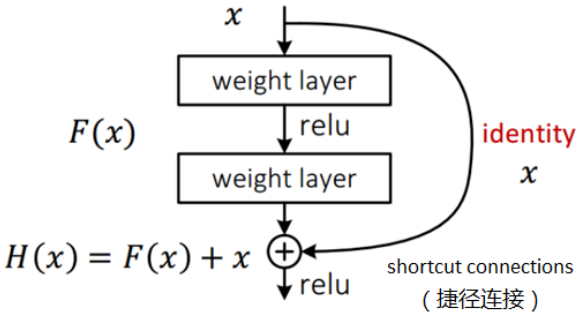
训练总轮数	30
每轮自学习次数	100
搜索树的阈值	15
更新模型的胜率	0.6
每轮保留的训练数据的上限	200000
蒙特卡罗采样次数	25
新旧模型对抗次数	20
UCT 缩放因子$C_{p_{uct}}$	1
经验池长度	3

1.2 神经网络超参数设置

学习率（lr）	0.001
每次学习迭代轮数（epochs）	15
dropout	0.3
batch_size	64
神经网络特征图数量	512

1.3 神经网络模型



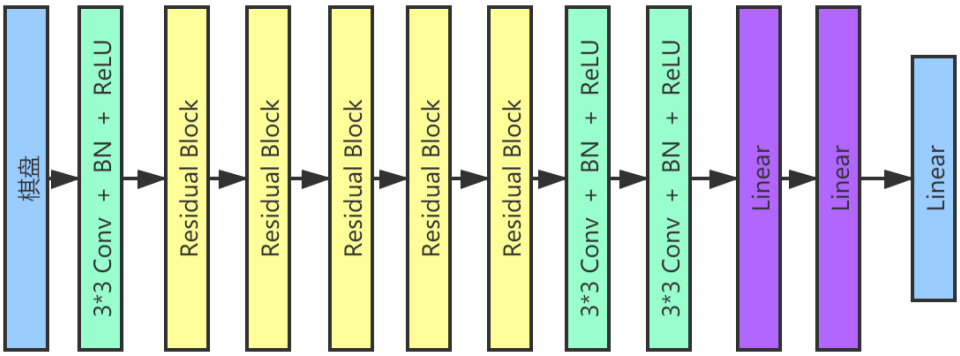


Layer name	Input size	Output size	Filter structure
Conv1	8*8*1	8*8*512	$[3\times 3 \quad 512]$
Residual Block	8*8*512	8*8*512	$\begin{bmatrix}3\times 3 & 512\\ 3\times 3 & 512\end{bmatrix}\times 2$
Conv2	8*8*512	6*6*512	$[3\times 3 \quad 512]$
Conv3	6*6*512	4*4*512	$[3\times 3 \quad 512]$
Conv5_x	8*8*256	4*4*512	$\begin{bmatrix}3\times 3 & 512\\ 3\times 3 & 512\end{bmatrix}\times 2$
FC1	8192	1024	
FC2	1024	512	
FC3	512	65	
FC4	512	1	

2、DQN + Experience replay + ε贪心模型

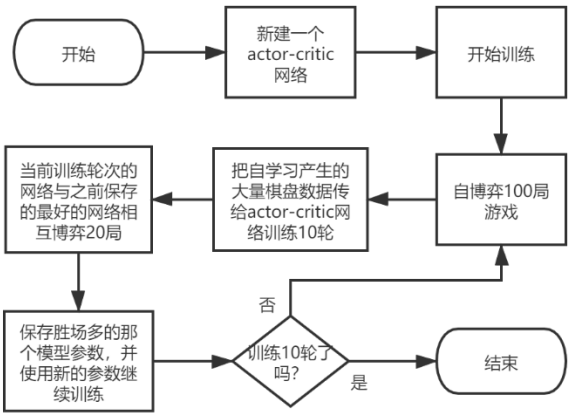
为了体现对比实验的效果，该模型的所有超参数均尽量与上一个模型一致。对于 DQN 及时奖励的设置问题，这里只在最后一步确定模型输赢之后及时奖励才是±1，其余中间步骤均为 0。

神经网络也大致与上述的模型一致。

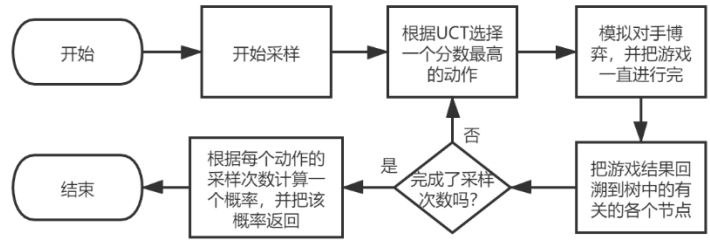


三、流程图

模型训练主流程



蒙特卡罗树搜索流程



与人类玩家博弈流程流程