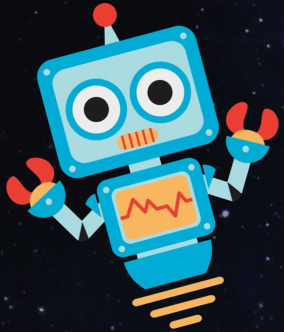
fun-ai-talk

AI 遇见斗地主 2023

DouZero + PerfectDou

Starting soon

[Music: 房東的貓 - New Boy(朴树)]



AI 遇见斗地主 2023

DouZero 2021 + PerfectDou 2022

hululu.zhu@gmail.com

05/2023

How are you today?

今日安排

- 斗地主【实战】入门
- 强化学习RL和斗地主
- 为何斗地主是很难的AI问题？
- DouZero 算法 快手
- PerfectDou 算法 NetEase Games
- 讨论

斗地主实战入门

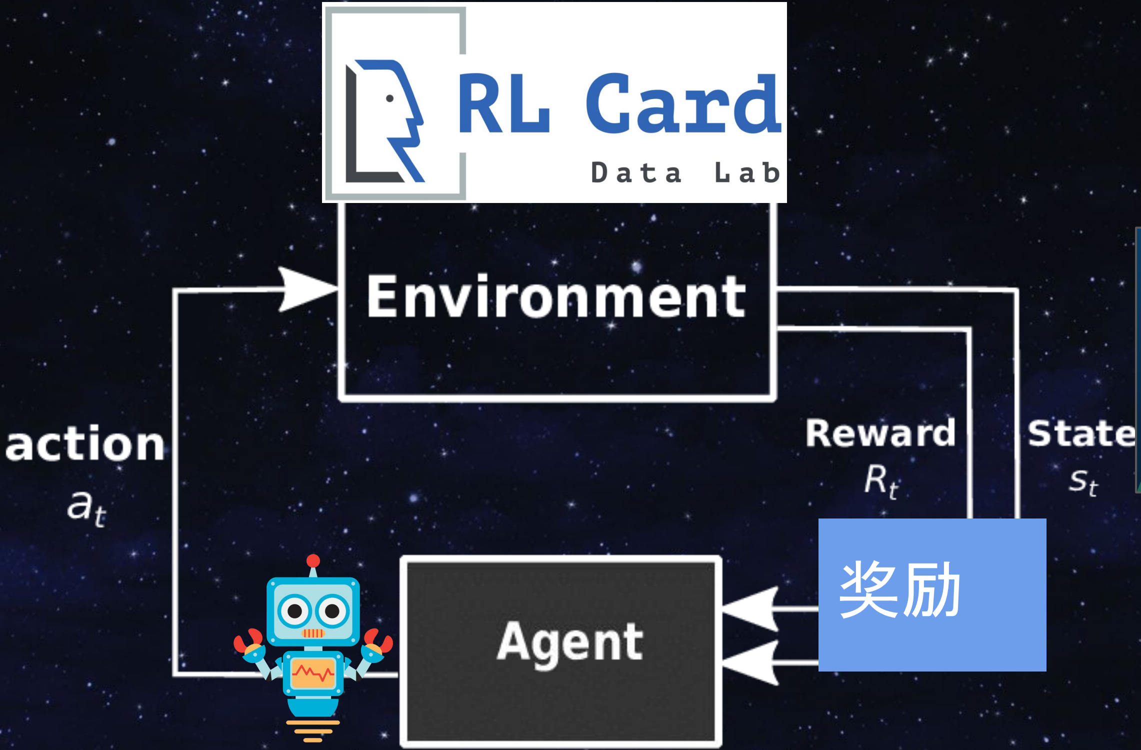
www.douzero.org/

perfectDou demo

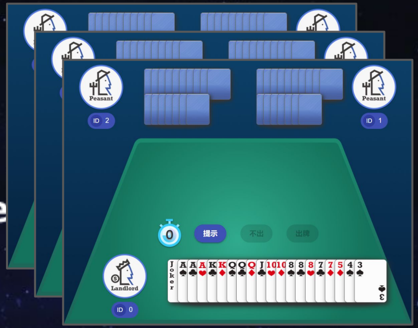
强化学习 (RL)

强化学习 (RL) 概念在斗地主的体现

单张
对子
三条
不出
…



奖励



牌局信息，非图像

斗地主对于AI非常难！！

- 超多状态(state, 手牌)
  - 17 or 20 cards out of 54 (4.7e13 - 3.2e14)
- 超大"动作"(action 出什么牌)空间
  - 理论上超过 27k 种出牌方式！
  - PerfectDou 简化到600+ (代码)
- 不完美信息(inprefect info 看不到对手牌)
  - 相比AlphaGo看到所有信息
  - 类似AI打starcraft或者王者荣耀，但是不处理图片
- 同时竞争competition与合作collaboration！
  - 地主要1v2
  - 农民要合作2v1

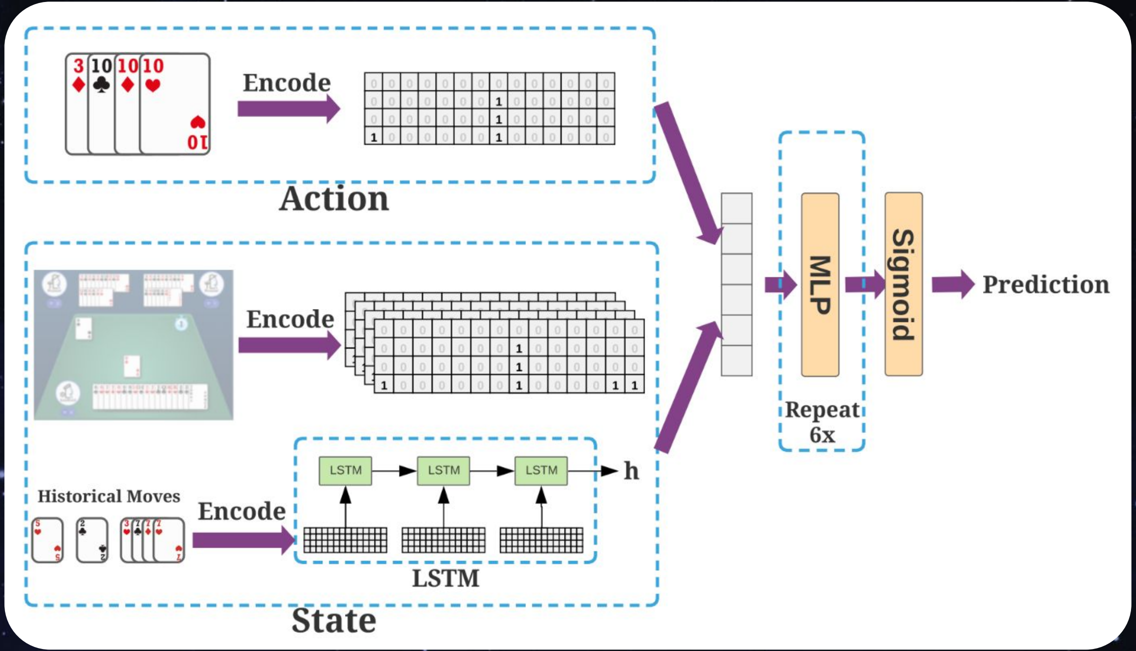
Action Type	Number of Actions
Solo	15
Pair	13
Trio	13
Trio with Solo	182
Trio with Pair	156
Chain of Solo	36
Chain of Pair	52
Chain of Trio	45
Plane with Solo	21,822
Plane with Pair	2,939
Quad with Solo	1,326
Quad with Pair	858
Bomb	13
Rocket	1
Pass	1
Total	27,472

DouZero 算法简述: Deep Monte-Carlo 深度蒙特卡洛

- 用一个现有的k时刻的actor模型
    - 每一次出牌一定几率按照模型分布(exploitation)
    - 剩余的纪律按照随机分布(exploration)
- 玩到游戏结束
    - AlphaGo因为围棋步数太多只能做MTCS(蒙特卡洛树搜索)
- 重复玩很多把
- 把所有k时刻模型的牌局(replay buffer)传给learner
    - 学习胜利的案例
    - 远离失败的案例
    - 更新模型至k+1时刻模型，更新actor模型
- 重复以上步骤

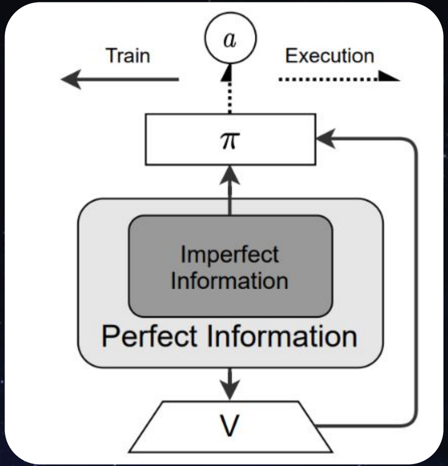
DouZero Shared Model Architecture

- Concat action and state features
- LSTM used to encode the sequence history moves
- "Deep" fully connected layers (MLP)
- "Sigmoid" to control output between (0, 1)
- Mean square error used as loss function



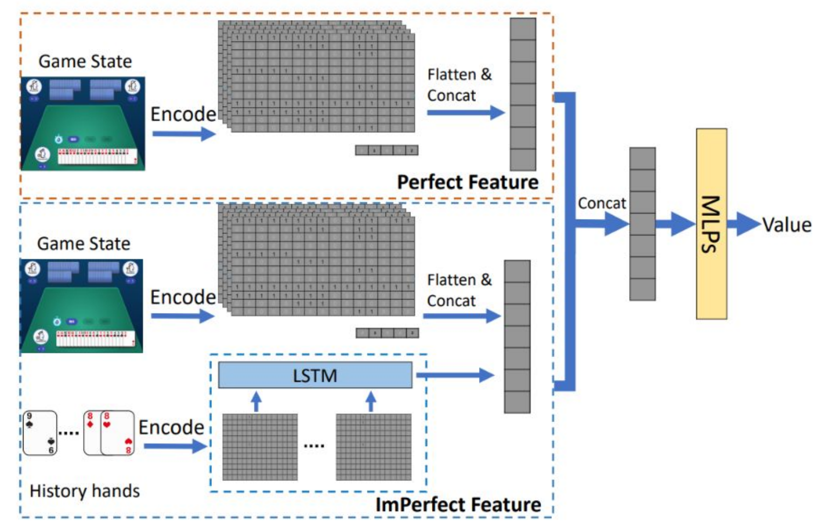
【更强】PerfectDou有啥高明

- 用actor critic方式训练
  - critic来给出牌局当前信息的评估
  - 此评估来指导actor策略网络的行为
- **Perfect-training-imperfect-execution**
  - 训练时候使用"完美信息"
    - "Calculate Minimum Step to Play Out All Cards"
  - 实战时候还是用"非完美信息"
  - 我的理解和不理解
    - 所以这里的critic就看到了所有的手牌，作弊了
    - 作者说这个看了所有人手牌的critic可以更好得通过distill（蒸馏）来指导actor的策略网络
- 用PPO（GAE，A means Advantage）优化
  - 优化代码没有给！有点鸡贼哈哈

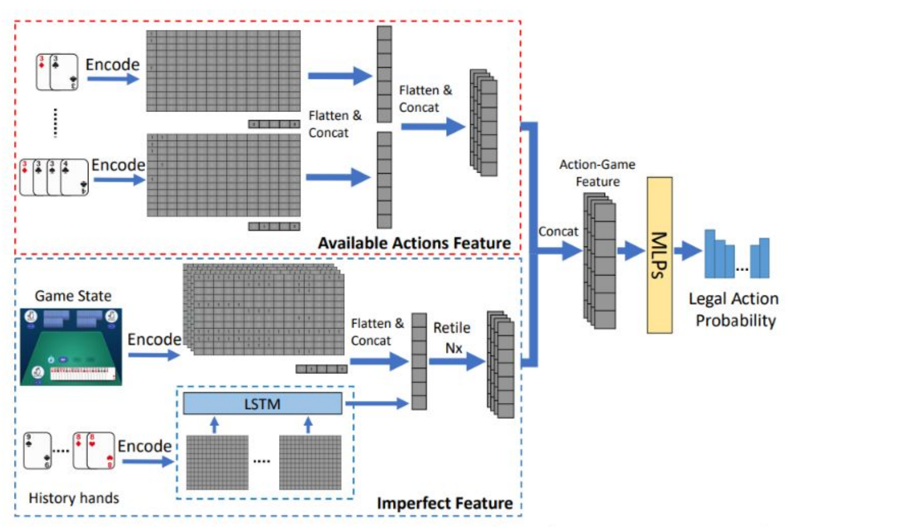


PerfectDou Perfect-training-imperfect-execution

价值网络(oracle指挥)

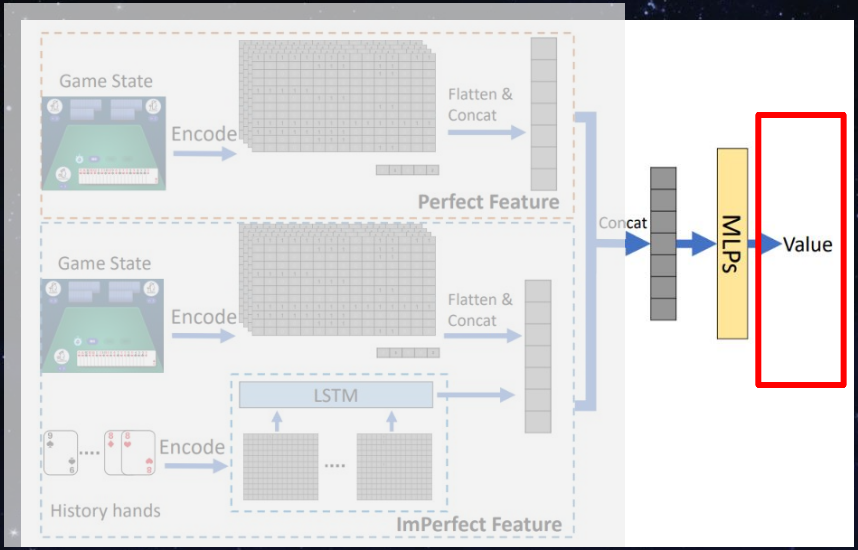


策略网络(被指导和实战)

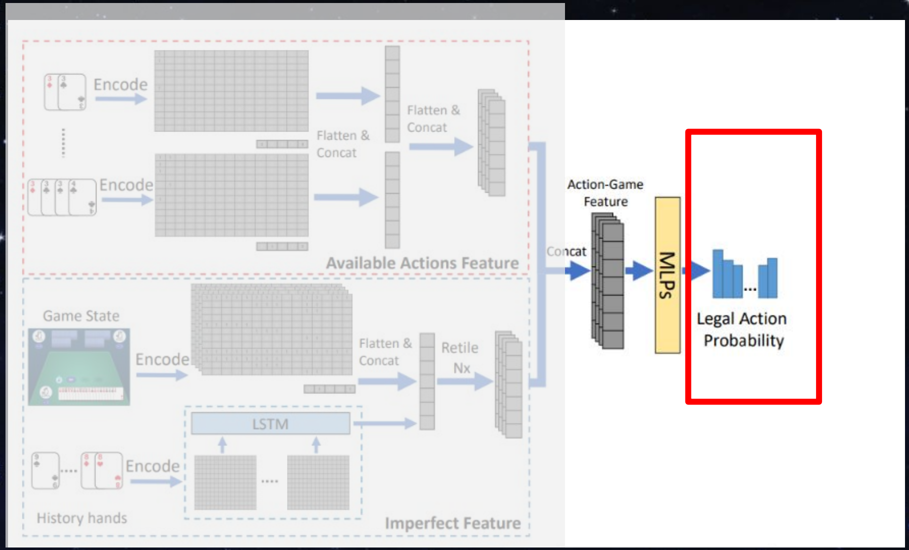


PerfectDou Perfect-training-imperfect-execution

价值网络（oracle指挥）

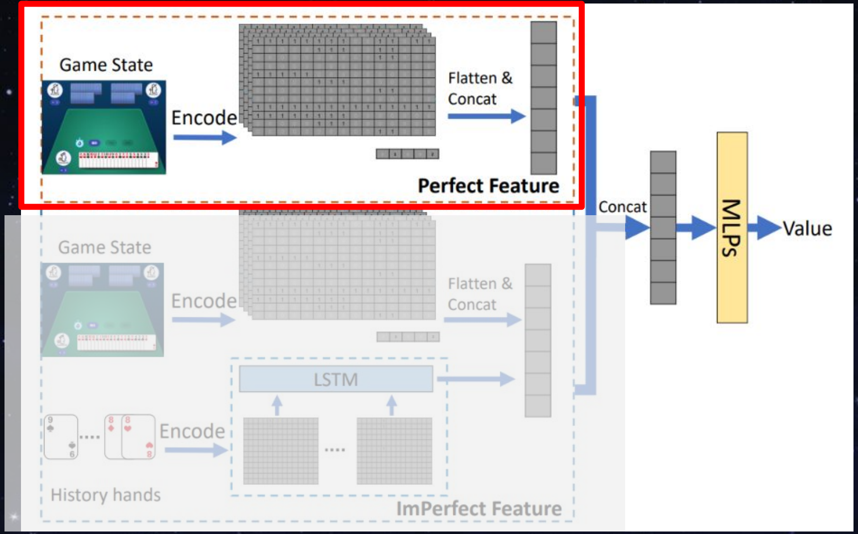


策略网络（被指导和实战）

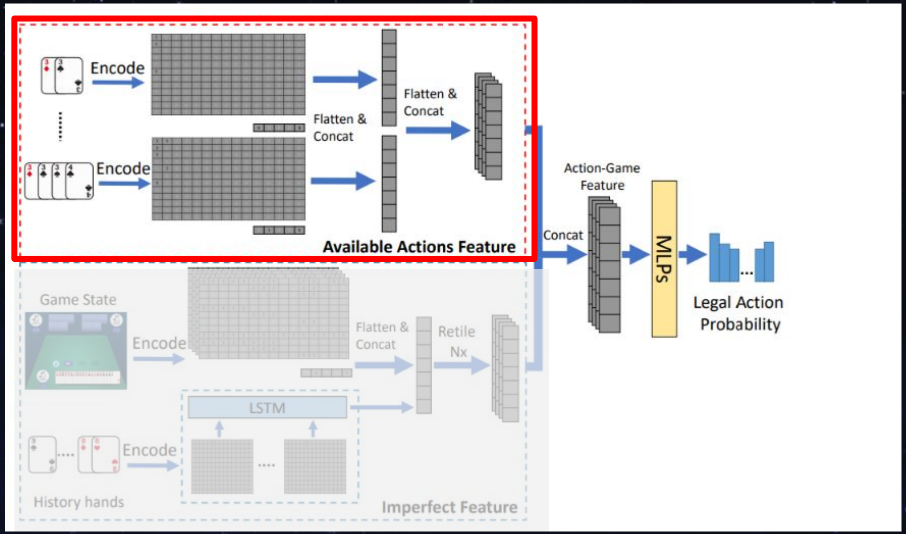


PerfectDou Perfect-training-imperfect-execution

价值网络（oracle指挥）



策略网络（被指导和实战）



有空的话，看一下代码

https://github.com/kwai/DouZero

https://github.com/Netease-Games-AI-Lab-Guangzhou/PerfectDou

DouZero Algorithm (actor)

Input: Shared buffers $\mathcal{B}_L$, $\mathcal{B}_U$ and $\mathcal{B}_D$ with B entries and size S for each entry, exploration hyperparameter ϵ, discount factor γ

Initialize local Q-networks Q_{L}, Q_{U} and Q_{D}, and local buffers $\mathcal{D}_L$, $\mathcal{D}_U$ and $\mathcal{D}_D$

for iteration = 1, 2, ... **do**

    Synchronize Q_{L}, Q_{U} and Q_{D} with the learner process

    for t = 1, 2, ... T **do** ▷ *Generate an episode*

        $Q \leftarrow$ one of $Q_{\mathrm{L}}, Q_{\mathrm{U}}, Q_{\mathrm{D}}$ based on position

        $a_t \leftarrow \begin{cases} \arg\max_a Q(s_t, a) & \text{with prob } (1-\epsilon) \\ \text{random action} & \text{with prob } \epsilon \end{cases}$

        Perform a_t, observe s_{t+1} and reward r_t

        Store $\{s_t, a_t, r_t\}$ to $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$ accordingly

    end for

    for t = T-1, T-2, ... 1 **do** ▷ *Obtain cumulative reward*

        $r_t \leftarrow r_t + \gamma r_{t+1}$ and update r_t in $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$

    end for

    for $p \in \{L, U, D\}$ **do** ▷ *Optimized by multi-thread*

        if $\mathcal{D}_p$.length $\geq L$ **then**

            Request and wait for an empty entry in $\mathcal{B}_p$

            Move $\{s_t, a_t, r_t\}$ of size L from $\mathcal{D}_p$ to $\mathcal{B}_p$

        end if

    end for

end for

DouZero Algorithm (actor)

Input: Shared buffers $\mathcal{B}_L$, $\mathcal{B}_U$ and $\mathcal{B}_D$ with B entries and size S for each entry, exploration hyperparameter ϵ, discount factor γ

Initialize local Q-networks Q_{L}, Q_{U} and Q_{D}, and local buffers $\mathcal{D}_L$, $\mathcal{D}_U$ and $\mathcal{D}_D$

for iteration = 1, 2, ... **do**

  Synchronize Q_{L}, Q_{U} and Q_{D} with the learner process

  for t = 1, 2, ... T **do** ▷ *Generate an episode*

    $Q \leftarrow$ one of $Q_{\text{L}}, Q_{\text{U}}, Q_{\text{D}}$ based on position

    $$a_t \leftarrow \begin{cases} \arg\max_a Q(s_t, a) & \text{with prob } (1-\epsilon) \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

    Perform a_t, observe s_{t+1} and reward r_t

    Store $\{s_t, a_t, r_t\}$ to $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$ accordingly

  end for

  for t = T-1, T-2, ... 1 **do** ▷ *Obtain cumulative reward*

    $r_t \leftarrow r_t + \gamma r_{t+1}$ and update r_t in $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$

  end for

  for $p \in \{L, U, D\}$ **do** ▷ *Optimized by multi-thread*

    if $\mathcal{D}_p$.length $\geq L$ **then**

      Request and wait for an empty entry in $\mathcal{B}_p$

      Move $\{s_t, a_t, r_t\}$ of size L from $\mathcal{D}_p$ to $\mathcal{B}_p$

    end if

  end for

end for

Initialize

DouZero Algorithm (actor)

Input: Shared buffers $\mathcal{B}_L$, $\mathcal{B}_U$ and $\mathcal{B}_D$ with B entries and size S for each entry, exploration hyperparameter ϵ, discount factor γ

Initialize local Q-networks Q_{L}, Q_{U} and Q_{D}, and local buffers $\mathcal{D}_L$, $\mathcal{D}_U$ and $\mathcal{D}_D$

for iteration = 1, 2, ... **do**

  Synchronize Q_{L}, Q_{U} and Q_{D} with the learner process

  for t = 1, 2, ... T **do** ▷ *Generate an episode*

    $Q \leftarrow$ one of $Q_{\mathrm{L}}, Q_{\mathrm{U}}, Q_{\mathrm{D}}$ based on position

    $$a_t \leftarrow \begin{cases} \arg\max_a Q(s_t, a) & \text{with prob } (1-\epsilon) \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

    Perform a_t, observe s_{t+1} and reward r_t

    Store $\{s_t, a_t, r_t\}$ to $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$ accordingly

  end for

  for t = T-1, T-2, ... 1 **do** ▷ *Obtain cumulative reward*

    $r_t \leftarrow r_t + \gamma r_{t+1}$ and update r_t in $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$

  end for

  for $p \in \{L, U, D\}$ **do** ▷ *Optimized by multi-thread*

    if $\mathcal{D}_p$.length $\geq L$ **then**

      Request and wait for an empty entry in $\mathcal{B}_p$

      Move $\{s_t, a_t, r_t\}$ of size L from $\mathcal{D}_p$ to $\mathcal{B}_p$

    end if

  end for

end for

Monte Carlo Method

DouZero Algorithm (actor)

Input: Shared buffers $\mathcal{B}_L$, $\mathcal{B}_U$ and $\mathcal{B}_D$ with B entries and size S for each entry, exploration hyperparameter ϵ, discount factor γ
Initialize local Q-networks Q_{L}, Q_{U} and Q_{D}, and local buffers $\mathcal{D}_L$, $\mathcal{D}_U$ and $\mathcal{D}_D$
for iteration = 1, 2, ... **do**
  Synchronize Q_{L}, Q_{U} and Q_{D} with the learner process
  for t = 1, 2, ... T **do** ▷ *Generate an episode*
    $Q \leftarrow$ one of $Q_{\mathrm{L}}, Q_{\mathrm{U}}, Q_{\mathrm{D}}$ based on position
    $a_t \leftarrow \begin{cases} \arg\max_a Q(s_t, a) & \text{with prob } (1-\epsilon) \\ \text{random action} & \text{with prob } \epsilon \end{cases}$
    Perform a_t, observe s_{t+1} and reward r_t
    Store $\{s_t, a_t, r_t\}$ to $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$ accordingly
  end for
  for t = T-1, T-2, ... 1 **do** ▷ *Obtain cumulative reward*
    $r_t \leftarrow r_t + \gamma r_{t+1}$ and update r_t in $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$
  end for
  for $p \in \{L, U, D\}$ **do** ▷ *Optimized by multi-thread*
    if $\mathcal{D}_p$.length $\geq L$ **then**
      Request and wait for an empty entry in $\mathcal{B}_p$
      Move $\{s_t, a_t, r_t\}$ of size L from $\mathcal{D}_p$ to $\mathcal{B}_p$
    end if
  end for
end for

Start playing each episode until "done"

DouZero Algorithm (actor)

Input: Shared buffers $\mathcal{B}_L$, $\mathcal{B}_U$ and $\mathcal{B}_D$ with B entries and size S for each entry, exploration hyperparameter ϵ, discount factor γ

Initialize local Q-networks Q_{L}, Q_{U} and Q_{D}, and local buffers $\mathcal{D}_L$, $\mathcal{D}_U$ and $\mathcal{D}_D$

for iteration = 1, 2, ... **do**

  Synchronize Q_{L}, Q_{U} and Q_{D} with the learner process

  for t = 1, 2, ... T **do** ▷ *Generate an episode*

    $Q \leftarrow$ one of $Q_{\mathrm{L}}, Q_{\mathrm{U}}, Q_{\mathrm{D}}$ based on position

    $a_t \leftarrow \begin{cases} \arg\max_a Q(s_t, a) & \text{with prob } (1-\epsilon) \\ \text{random action} & \text{with prob } \epsilon \end{cases}$

    Perform a_t, observe s_{t+1} and reward r_t

    Store $\{s_t, a_t, r_t\}$ to $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$ accordingly

  end for

  for t = T-1, T-2, ... 1 **do** ▷ *Obtain cumulative reward*

    $r_t \leftarrow r_t + \gamma r_{t+1}$ and update r_t in $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$

  end for

  for $p \in \{L, U, D\}$ **do** ▷ *Optimized by multi-thread*

    if $\mathcal{D}_p$.length $\geq L$ **then**

      Request and wait for an empty entry in $\mathcal{B}_p$

      Move $\{s_t, a_t, r_t\}$ of size L from $\mathcal{D}_p$ to $\mathcal{B}_p$

    end if

  end for

end for

Exploitation: Apply current model's best suggestion with prob (1-e)

Exploration: Try some random action with prob e

DouZero Algorithm (actor)

Input: Shared buffers $\mathcal{B}_L$, $\mathcal{B}_U$ and $\mathcal{B}_D$ with B entries and size S for each entry, exploration hyperparameter ϵ, discount factor γ

Initialize local Q-networks Q_{L}, Q_{U} and Q_{D}, and local buffers $\mathcal{D}_L$, $\mathcal{D}_U$ and $\mathcal{D}_D$

for iteration = 1, 2, ... **do**

    Synchronize Q_{L}, Q_{U} and Q_{D} with the learner process

    for t = 1, 2, ... T **do** ▷ *Generate an episode*

        $Q \leftarrow$ one of $Q_{\text{L}}, Q_{\text{U}}, Q_{\text{D}}$ based on position

        $a_t \leftarrow \begin{cases} \arg\max_a Q(s_t, a) & \text{with prob } (1-\epsilon) \\ \text{random action} & \text{with prob } \epsilon \end{cases}$

        Perform a_t, observe s_{t+1} and reward r_t

        Store $\{s_t, a_t, r_t\}$ to $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$ accordingly

    end for

    for t = T-1, T-2, ... 1 **do** ▷ *Obtain cumulative reward*

        $r_t \leftarrow r_t + \gamma r_{t+1}$ and update r_t in $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$

    end for

    for $p \in \{L, U, D\}$ **do** ▷ *Optimized by multi-thread*

        if $\mathcal{D}_p$.length $\geq L$ **then**

            Request and wait for an empty entry in $\mathcal{B}_p$

            Move $\{s_t, a_t, r_t\}$ of size L from $\mathcal{D}_p$ to $\mathcal{B}_p$

        end if

    end for

end for

Delayed "credit assignment", the sooner the better

DouZero Algorithm (actor)

Input: Shared buffers $\mathcal{B}_L$, $\mathcal{B}_U$ and $\mathcal{B}_D$ with B entries and size S for each entry, exploration hyperparameter ϵ, discount factor γ

Initialize local Q-networks Q_{L}, Q_{U} and Q_{D}, and local buffers $\mathcal{D}_L$, $\mathcal{D}_U$ and $\mathcal{D}_D$

for iteration = 1, 2, ... **do**

  Synchronize Q_{L}, Q_{U} and Q_{D} with the learner process

  for t = 1, 2, ... T **do** ▷ *Generate an episode*

    $Q \leftarrow$ one of $Q_{\mathrm{L}}, Q_{\mathrm{U}}, Q_{\mathrm{D}}$ based on position

    $$a_t \leftarrow \begin{cases} \arg\max_a Q(s_t, a) & \text{with prob } (1-\epsilon) \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

    Perform a_t, observe s_{t+1} and reward r_t

    Store $\{s_t, a_t, r_t\}$ to $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$ accordingly

  end for

  for t = T-1, T-2, ... 1 **do** ▷ *Obtain cumulative reward*

    $r_t \leftarrow r_t + \gamma r_{t+1}$ and update r_t in $\mathcal{D}_L$, $\mathcal{D}_U$, or $\mathcal{D}_D$

  end for

  for $p \in \{L, U, D\}$ **do** ▷ *Optimized by multi-thread*

    if $\mathcal{D}_p$.length $\geq L$ **then**

      Request and wait for an empty entry in $\mathcal{B}_p$

      Move $\{s_t, a_t, r_t\}$ of size L from $\mathcal{D}_p$ to $\mathcal{B}_p$

    end if

  end for

end for

Sync with "learner" processor by moving episodes to "train the AI"

DouZero Algorithm (Learner)

Input: Shared buffers $\mathcal{B}_L$, $\mathcal{B}_U$ and $\mathcal{B}_D$ with B entries and size S for each entry, batch size M, learning rate ψ

Initialize global Q-networks Q^g_{L}, Q^g_{U} and Q^g_{D}

for iteration = 1, 2, ... until convergence **do**

  for $p \in \{L, U, D\}$ **do** $\triangleright$ *Optimized by multi-thread*

    if the number of full entries in $\mathcal{B}_p \geq M$ **then**

      Sample a batch of $\{s_t, a_t, r_t\}$ with $M \times S$ instances from $\mathcal{B}_p$ and free the entries

      Update Q^g_{p} with MSE loss and learning rate ψ

    end if

  end for

end for

DouZero Algorithm (Learner)

Input: Shared buffers $\mathcal{B}_L$, $\mathcal{B}_U$ and $\mathcal{B}_D$ with B entries and size S for each entry, batch size M, learning rate ψ
Initialize global Q-networks Q^g_{L}, Q^g_{U} and Q^g_{D}
for iteration = 1, 2, ... until convergence **do**
  for $p \in \{L, U, D\}$ **do** $\triangleright$ *Optimized by multi-thread*
    if the number of full entries in $\mathcal{B}_p \geq M$ **then**
      Sample a batch of $\{s_t, a_t, r_t\}$ with $M \times S$ instances from $\mathcal{B}_p$ and free the entries
      Update Q^g_{p} with MSE loss and learning rate ψ
    end if
  end for
end for

Initialize

DouZero Algorithm (Learner)

```
Input: Shared buffers B_L, B_U and B_D with B entries
  and size S for each entry, batch size M, learning rate ψ
Initialize global Q-networks Q^g_L, Q^g_U and Q^g_D
for iteration = 1, 2, ... until convergence do
  for p ∈ {L, U, D} do    ▷ Optimized by multi-thread
    if the number of full entries in B_p ≥ M then
      Sample a batch of {s_t, a_t, r_t} with M × S in-
        stances from B_p and free the entries
      Update Q^g_p with MSE loss and learning rate ψ
    end if
  end for
end for
```

Model Training loop

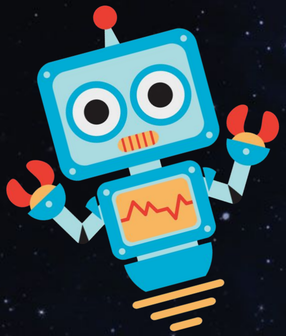
DouZero Algorithm (Learner)

Input: Shared buffers $\mathcal{B}_L$, $\mathcal{B}_U$ and $\mathcal{B}_D$ with B entries and size S for each entry, batch size M, learning rate ψ
Initialize global Q-networks Q^g_{L}, Q^g_{U} and Q^g_{D}
for iteration = 1, 2, ... until convergence **do**
  for $p \in \{L, U, D\}$ **do** ▷ *Optimized by multi-thread*
    if the number of full entries in $\mathcal{B}_p \geq M$ **then**
      Sample a batch of $\{s_t, a_t, r_t\}$ with $M \times S$ instances from $\mathcal{B}_p$ and free the entries
      Update Q^g_{p} with MSE loss and learning rate ψ
    end if
  end for
end for

Mini-batch training with Mean Square Error loss

References

- PerfectDou: Dominating DouDizhu with Perfect Information Distillation
- DouZero Paper: Mastering DouDizhu with Self-Play Deep Reinforcement Learning
- Github DouZero: Mastering DouDizhu with Self-Play Deep Reinforcement Learning
- AlphaGo Paper: Mastering the game of Go with deep neural networks and tree search
- AlphaGo Zero Paper: Mastering the game of Go without human knowledge | Nature
- DeltaDou Paper: Expert-level Doudizhu AI through Self-play
- Zhihu/一堆废纸：DouZero斗地主AI深度解析，以及RLCard工具包介绍
- Wang Susen：深度强化学习(5/5)：AlphaGo & Model-Based RL
- Dou dizhu - Wikipedia



AI 遇见斗地主 2023

DouZero 2021 + PerfectDou 2022

hululu.zhu@gmail.com
05/2023