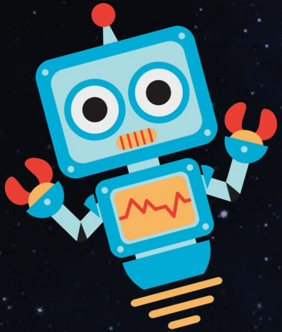


When AI meets 斗地主

A Deep Dive into KuaiShou "DouZero" AI

hululu.zhu@gmail.com

03/2022

How are you today?

About me, 10+ years at Google...

My ML work experience starts at 2016 at Google Fiber

fiber

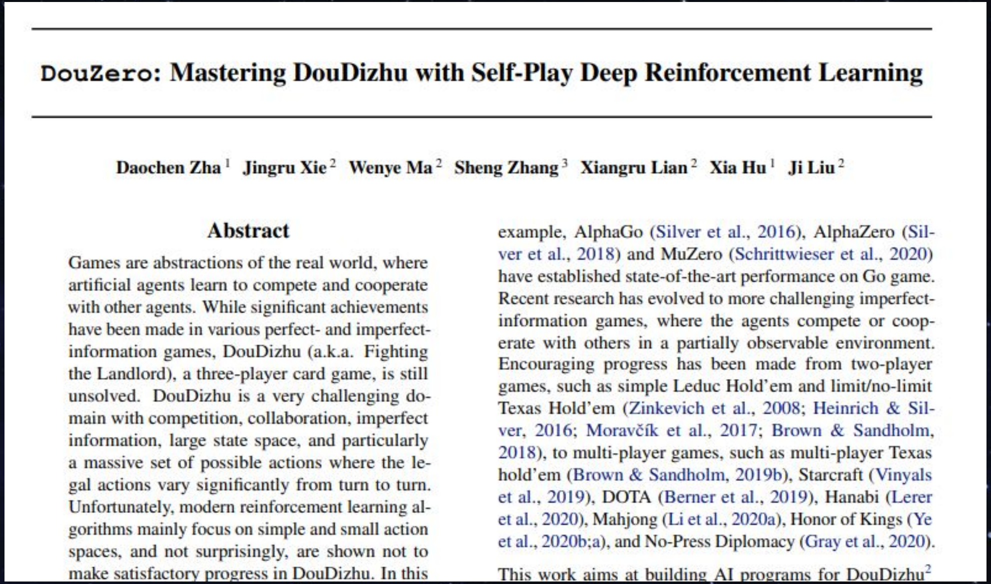
intern

SWE

Agenda

- 斗地主 Problem Statement
- DouZero Algorithm
- DouZero Code Review
- Live Group Challenge to DouZero AI
- Discussion and Q&A

by 快手



DouZero: Mastering DouDizhu with Self-Play Deep Reinforcement Learning

Daochen Zha[1] Jingru Xie[2] Wenye Ma[2] Sheng Zhang[3] Xiangru Lian[2] Xia Hu[1] Ji Liu[2]

Abstract

Games are abstractions of the real world, where artificial agents learn to compete and cooperate with other agents. While significant achievements have been made in various perfect- and imperfect-information games, DouDizhu (a.k.a. Fighting the Landlord), a three-player card game, is still unsolved. DouDizhu is a very challenging domain with competition, collaboration, imperfect information, large state space, and particularly a massive set of possible actions where the legal actions vary significantly from turn to turn. Unfortunately, modern reinforcement learning algorithms mainly focus on simple and small action spaces, and not surprisingly, are shown not to make satisfactory progress in DouDizhu. In this

example, AlphaGo (Silver et al., 2016), AlphaZero (Silver et al., 2018) and MuZero (Schrittwieser et al., 2020) have established state-of-the-art performance on Go game. Recent research has evolved to more challenging imperfect-information games, where the agents compete or cooperate with others in a partially observable environment. Encouraging progress has been made from two-player games, such as simple Leduc Hold'em and limit/no-limit Texas Hold'em (Zinkevich et al., 2008; Heinrich & Silver, 2016; Moravčík et al., 2017; Brown & Sandholm, 2018), to multi-player games, such as multi-player Texas hold'em (Brown & Sandholm, 2019b), Starcraft (Vinyals et al., 2019), DOTA (Berner et al., 2019), Hanabi (Lerer et al., 2020), Mahjong (Li et al., 2020a), Honor of Kings (Ye et al., 2020b;a), and No-Press Diplomacy (Gray et al., 2020).

This work aims at building AI programs for DouDizhu[2]

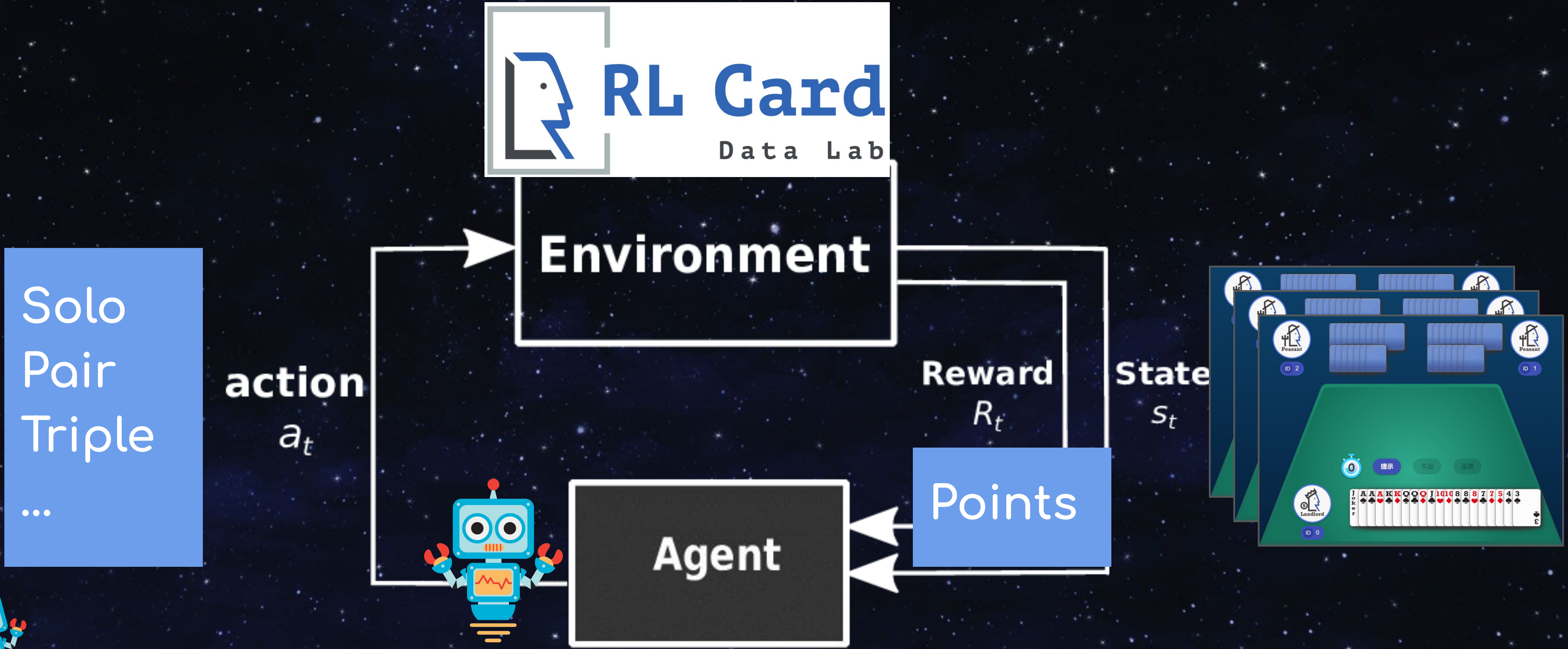
斗地主 is a tough problem!

- Huge number of states
  - 17 or 20 cards out of 54 (4.7e13 - 3.2e14)
- Big number of action spaces
  - More than 27k actions! See the chart right
- Imperfect information
  - AI "sees" full board like Chess/Go, e.g. deepblue, alphago
  - But 斗地主 AI cannot see others' cards
- Compete & Collaborate at the same time!
  - Landlord wants to beat 2 peasants
  - 2 peasant collaborate to try to beat the landlord

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

Reinforcement Learning (RL) for 斗地主

Reinforcement Learning (RL) for 斗地主



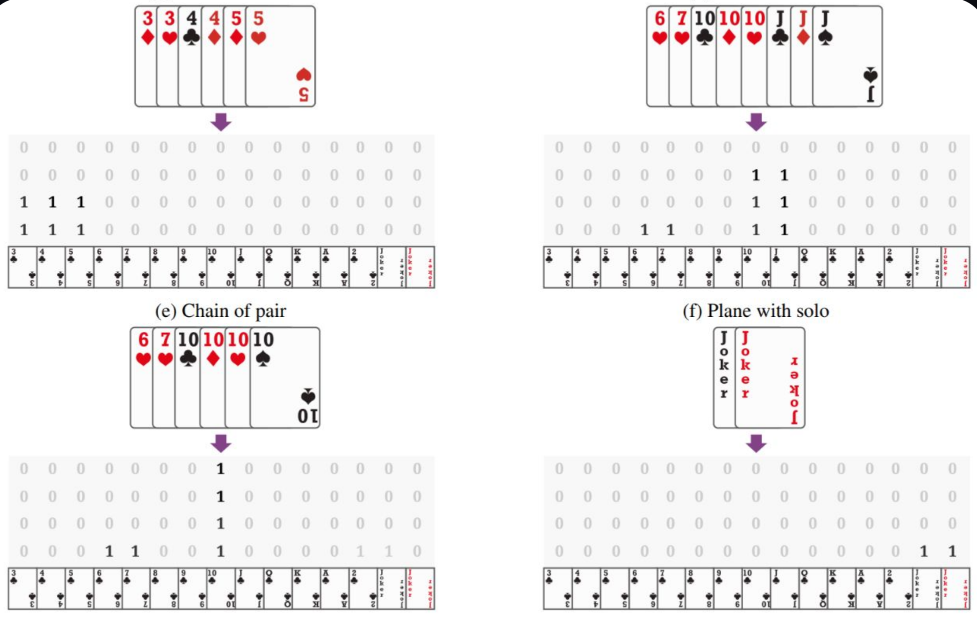
DouZero Algorithm: Deep Monte-Carlo (DMC)

- Reinforcement Concepts
  - **State**: Describe the environment, e.g. cards at hand and history cards
  - **Action**: Describe the action AI can take, e.g. pass or solo or pairs or…
  - **Reward**: Quantitatively describe the outcome

- Deep Monte-Carlo (more details later)
  - Deep Neural Network to estimate the Q value
    - Given state, with action, estimated Q value
  - Use Monte-carlo to iterate estimation
  - Similar to Deep Q Network, but simpler and less "overestimating"
  - More efficient than Policy Gradient methods, because with <=17/20 cards, only a few options out of 27k action spaces are necessary

DouZero Algorithm: Deep Monte-Carlo (DMC) cont.

- How to encode the "state" and "action"? Use size-54 vectors
  - Suit is not important except for Joker
  - So 3->2, plus 2 Joker = 15 *4 = size 60 vector
  - Jokers do not need 4 spaces, so 60-6=54 is sufficient for any action or cards at hand



(e) Chain of pair

(f) Plane with solo

(g) Quad with solo

(h) Rocket

DouZero Algorithm: Deep Monte-Carlo (DMC) cont.

What info used for “state” and “action”?

- Note peasant has more “state” info to differentiate landlord and other peasant’s most recent move

Landlord

	Feature	Size
Action	Card matrix of the action	54
State	Card matrix of hand cards	54
	Card matrix of the union of the other two players’ hand cards	54
	Card matrix of the most recent move	54
	Card matrix of the the played cards of the first Peasant	54
	Card matrix of the the played cards of the second Peasant	54
	One-hot vector representing the number cards left of the first Peasant	17
	One-hot vector representing the number cards left of the second Peasant	17
	One-hot vector representing the number bombs in the current state	15
	Concatenated matrix of the most recent 15 moves	5×162

Peasant

	Feature	Size
Action	Card matrix of the action	54
State	Card matrix of hand cards	54
	Card matrix of the union of the other two players’ hand cards	54
	Card matrix of the most recent move	54
	Card matrix of the most recent move performed by the Landlord	54
	Card matrix of the most recent move performed by the other Peasant	54
	Card matrix of the the played cards of the Landlord	54
	Card matrix of the the played cards of the other Peasant	54
	One-hot vector representing the number cards left of the Landlord	20
	One-hot vector representing the number cards left of the other Peasant	17
	One-hot vector representing the number bombs in the current state	15
	Concatenated matrix of the most recent 15 moves	5×162

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 $Q_{\mathrm{L}}, Q_{\mathrm{U}}, Q_{\mathrm{D}}$ 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

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 $Q_{\mathrm{L}}, Q_{\mathrm{U}}, Q_{\mathrm{D}}$ 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

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** ▷ *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 $\mathcal{B}_L$, $\mathcal{B}_U$ 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

Model Training loop

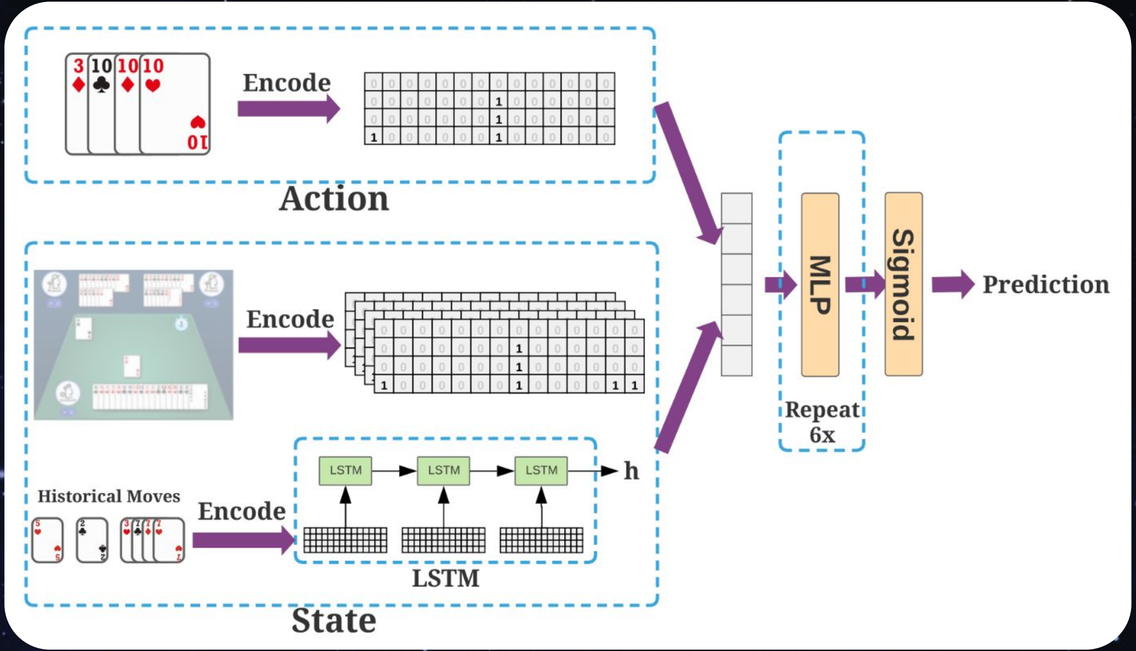
DouZero Algorithm (Learner)

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** $\triangleright$ *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

Mini-batch training with Mean Square Error loss

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



Let's dive into the code now!

- github.com/kwai/DouZero

Let's challenge DouZero together!

- www.douzero.org

Some thoughts

- DouZero is developed by a smart intern at KuaiShou! 2400+ github stars in 8 months!
- DouZero is simple but effective, 4 1080Ti GPU beat 1k+ GPU powered DeltaDou ! Simply "elegant" !
  - In the domain of deep learning AI, experience+compute power << creativity
  - Some Zhihu comments are pretty mean, shame on them
- Monte Carlo is powerful for Reinforcement Learning!
  - AlphaGo uses "Monte Carlo Tree Search" (MCTS) during inference time
  - AlphaGo Zero uses "Monte Carlo Tree Search" (MCTS) to train policy and value networks

Besides games, [Deep] Reinforcement Learning is quite useful if proper “Environment”s are created. e.g. nuclear reaction control, air combat control, molecule optimization

nature > articles > article

Article | Open Access | Published: 16 February 2022

Magnetic control of tokamak plasmas through deep reinforcement learning

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis & Martin Riedmiller — Show fewer authors

Nature 602, 414–419 (2022) | Cite this article

Abstract

Nuclear fusion using magnetic confinement, in particular in the tokamak configuration, is a promising path towards sustainable energy. A core challenge is to shape and maintain a high-temperature plasma within the tokamak vessel. This requires high-dimensional, high-frequency, closed-loop control using magnetic actuator coils, further complicated by the diverse

Hierarchical Reinforcement Learning for Air-to-Air Combat

Adrian P. Pope*, Jaime S. Ide*, Daria Mićović, Henry Diaz, David Rosenbluth, Lee Ritholtz, Jason C. Twedt, Thayne T. Walker, Kevin Alcedo and Daniel Javorsek II†
Applied AI Team, Lockheed Martin, Connecticut, USA
†U.S. Airforce, Virginia, USA

Abstract—**Artificial Intelligence (AI) is becoming a critical component in the defense industry, as recently demonstrated by DARPA's AlphaDogfight Trials (ADT). ADT sought to vet the feasibility of AI algorithms capable of piloting an F-16 in simulated air-to-air combat. As a participant in ADT, Lockheed Martin's (LM) approach combines a hierarchical architecture with maximum-entropy reinforcement learning (RL), integrates expert knowledge through reward shaping, and supports modularity of policies. This approach achieved a 2^{nd} place finish in the final ADT event (among eight total competitors) and defeated a graduate of the US Air Force's (USAF) F-16 Weapons Instructor Course in match play.**

Index Terms—**hierarchical reinforcement learning, air combat, flight simulation**

I. Introduction

The Air Combat Evolution (ACE) program, formed by DARPA, seeks to advance and build trust in air-to-air combat

Our approach uses hierarchical reinforcement learning (RL) and leverages an array of specialized policies that are dynamically selected given the current context of the engagement. Our agent achieved 2^{nd} place in the final tournament and defeated a graduate of the USAF F-16 Weapons Instructor Course in match play (5W - 0L).

II. Related Work

Since the 1950s, research has been done on how to build algorithms that can autonomously perform air combat [1]. Some have approached the problem with rule-based methods, using expert knowledge to formulate counter maneuvers to employ in different positional contexts [2]. Other explorations have codified the air-to-air scenario in various ways as an optimization problem to be solved computationally [2] [3] [4] [5] [6].

Optimization of Molecules via Deep Reinforcement Learning

Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N. Zare & Patrick Riley

Scientific Reports 9, Article number: 10752 (2019) | Cite this article

An Author Correction to this article was published on 23 June 2020

This article has been updated

Abstract

We present a framework, which we call Molecule Deep Q-Networks (MolDQN), for molecule optimizatio[n] combining domain knowledge of chemistry and state-of-the-art reinforcement learning techniques (dou

References

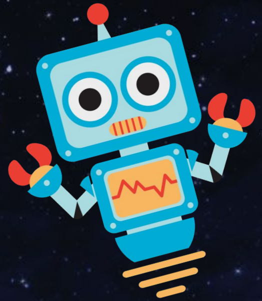
- DouZero Paper: Mastering DouDizhu with Self-Play Deep Reinforcement Learning
- Github DouZero: Mastering DouDizhu with Self-Play Deep Reinforcement Learning
- AlphaGo Paper: Mastering the game of Go with deep neural networks and tree search
- AlphaGo Zero Paper: Mastering the game of Go without human 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

My fun AI talks, join me to share knowledge!

- 写诗写对联模型 go/fun-ai-course-chinese-poem
- 强化学习|游戏 go/fun-ai-course-super-mario
- 强化学习|斗地主 go/fun-ai-course-chinese-douzero
- 模型训练技巧 go/fun-ai-course-ml-alchemist-skills
- NLP 经典论文浅谈 go/fun-ai-course-nlp-papers
- BERT分类器入门 go/bert-classifier-colab-101

Discussion Q&A Time

Please mark attendance at go/iamhere, thank you!



When AI meets 斗地主

A Deep Dive into KuaiShou "DouZero" AI

03/2022