Relatório de aprendizado por reforço usando PPO(otimizado) e DQN no ambiente KAS com observação em imagem

Aluno: Victor Gabriel Tenório Oliveira

Algoritmos, CNN, ambiente e wrappers usados

Algoritmos

👉 **Foi utilizado a implementação da biblioteca RAY[rllib] dos algoritmos DQN e PPO.**

O algoritmo DQN executou 887_000 passos em 10h, 17min e 54s.

O algoritmo PPO otimizado rodou apenas 100_352 passos em 22hr, 2min e 57s (2º experimento abaixo).

1 passo = turno em que todos os agentes fazem 1 ação no ambiente KAS.

> Ambos os algoritmos não chegaram a 1_000_000 timesteps devido a erros e a dificuldade de continuar o experimento sem bugs.
>
> No caso do DQN, o experimento de 887_000 passos foi continuado até 1_000_000 em 1h, 17min e 46s, mas o resultado do `tune.run` no arquivo progress.csv estava com mais colunas do que o progress.csv anterior e muitas colunas estavam com um valor da coluna ao lado, deixando difícil de corrigir e entender o que estava acontecendo.
>
> No caso do PPO, o 1º experimento deveria ter sido continuado várias vezes, mas recomeçou a partir do zero devido a um “error to load checkpoint”. Ao total, foram 6 experimento resumidos abaixo:

- 1º experimento com ID 50058_00000, 57_856 timesteps em 12hr, 43min e 27s, usando 6cpus sem overclock
- 2º experimento com ID 5a3bd_00000, 100_352 timesteps em 22hr, 2min e 57s, usando 6cpus sem overclock
- 3º experimento com ID 74e74_00000, 56_832 timesteps em 12hr, 28min e 52s, usando 12cpus com overclock
- 4º experimento com ID bf209_00000, 64_512 timesteps em 14hr, 23min e 58s, usando 12cpus com overclock
- 5º experimento com ID cbe8a_00000, 94_720 timesteps em 21hr, 36min e 41s, usando 12cpu com overclock
- 6º experimento com ID ace0c_00000, 60_416 timesteps em 15hr, 17min e 7s, usando 12cpus sem overclock

Total de timesteps: 434_688

Total de tempo: 98hr 33min 2s

👉 Todo o projeto foi executado na máquina local (não no Google Colab) e precisou de um ambiente virtual criado com anaconda3 versão 2023.07-2 (latest, mas provavelmente pode ser qualquer versão) para acessar o source code das bibliotecas PettingZoo, RAY[rllib] e SuperSuit. O acesso ao código fonte é necessário para consertar um erro da versão antiga dos ambientes do PettingZoo. Os erros e correções estão explicados em um passo a passo no README do projeto. Também existem informação relevante para o uso futuro da RAY[rllib] no README.

```
# Parâmetros de input do DQN
.training(
    n_step = 10,
    lr = 1e-3,
    gamma = 0.95
)
config.exploration_config.update({ # Decaying epsylon-greedy
    "initial_epsilon": 1.5,
    "final_epsilon": 0.01,
    "epsilon_timesteps": 1_000_000,
})
################################
# Parâmetros de input do PPO
# (não usou decayinng epsilon-greedy, pois dava erro)
.training(
    train_batch_size=512,
    lr=3e-5,    # <----- Otimizado com optuna
    gamma=0.95, # <----- Otimizado com optuna
    lambda_=0.9,
    use_gae=True,
    # clip_param=0.4,
    grad_clip=None,
    entropy_coeff=0.1, # <----- Otimizado com optuna
    vf_loss_coeff=0.25,
    sgd_minibatch_size=64,
    num_sgd_iter=10,
)
```

Mais detalhes do código nos comentários dos arquivos `src/DQN.py` e `src/PPO.py`

CNN

Para processar a imagem, foi utilizada uma arquitetura CNN encontrada no tutorial da documentação do PettingZoo.
Para saber mais detalhes do código, ver código em `src/cnn.py`.

Environments — Ray 2.6.1

Ray

https://docs.ray.io/en/latest/rllib/rllib-env.html#pettingzoo-multi-agent-environments

Link do tutorial

Ambiente e wrappers

> O ambiente usado com observação em imagem, ações discretas e multi-agente é Knights Archers Zombies (‘KAZ’) - PettingZoo Documentation (farama.org).
>
> A definição do ambiente com os wrappers usados está no arquivo `src/env_setup.py`.
>
> Os wrappers usados e o que eles fazem estão descritos no pedaço de código abaixo:

```
env = ss.color_reduction_v0(env, mode="B")           # mode="B" reduz observação RGB para Gray Scale
env = ss.dtype_v0(env, "float32")                    # Coverte uint8 (original) -> float32
env = ss.resize_v1(env, x_size=84, y_size=84)        # Reduz a imagem do tamanho 512x512 original para 84x84
env = ss.normalize_obs_v0(env, env_min=0, env_max=1) # Transforma valores da imagem de [0, 255] para [0, 1]
env = ss.frame_stack_v1(env, stack_size=3)           # Aplica frame stack com 3 frames
```

👉 A biblioteca SuperSuit possui wrappers feitos para o ambiente PattingZoo e, segundo a página do github do projeto e mensagens no Discord da Farama, essa biblioteca será completamente reescrita, descontinuada e integrada na bilbioteca PettingZoo no futuro. Para observar o impacto de cada wrapper, é possível criar um ambiente manualmente, chamar o método reset e printar a observação (modificada pelo wrapper). O código que observa o impacto dos wrappers está no aquivo `manual examples of env/KAS.py` do projeto.

Otimização

👉 Foi otimizado 3 parâmetros (era pra ser 2, mas não prestei atenção) do algoritmo PPO com 3 valores possíveis, totalizando 27 possibilidades:

- Leraning Rate → `[ 1e-5,  2e-5,  3e-5 ]`
- Gamma → `[ 0.90,  0.95,  0.99 ]`
- Entropy Coefficient → `[ 0.05,  0.1,  0.15 ]`

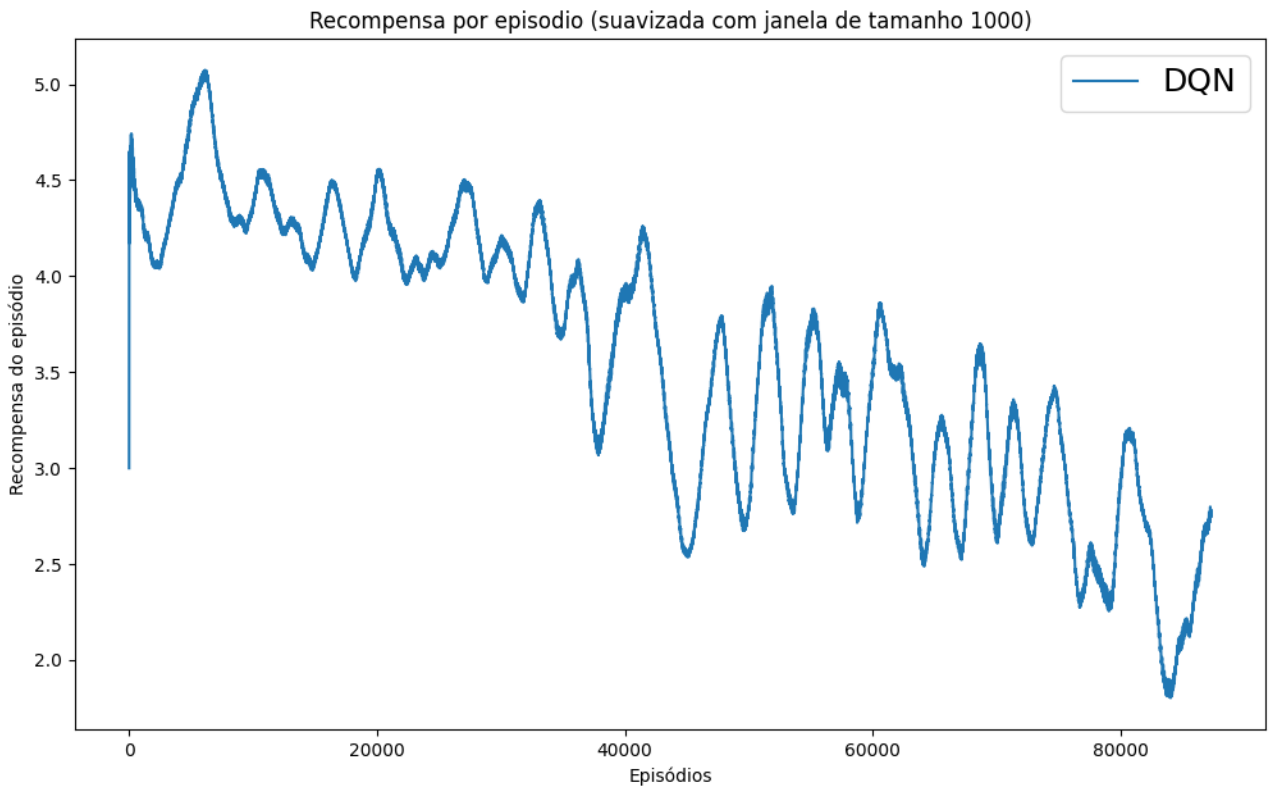
> Foi realizado busca em grid com 27 trials usando um critério de parada de 45 minutos para cada `tune.run`. Cada `tune.run` demorou por volta de 47 minutos para finalizar, totalizando 22 horas de execução.
> Cada trial tinha uma pontuação calculada a partir da média das 20 últimas recompensas do episódio e a melhor pontuação foi obtida com os parâmetros abaixo.
>
> Para saber mais detalhes do código, ver arquivo `src/optuna_PPO.py`

```
# Melhor pontuação
{'lr': 3e-05, 'gamma': 0.95, 'entropy_coeff': 0.1}
```

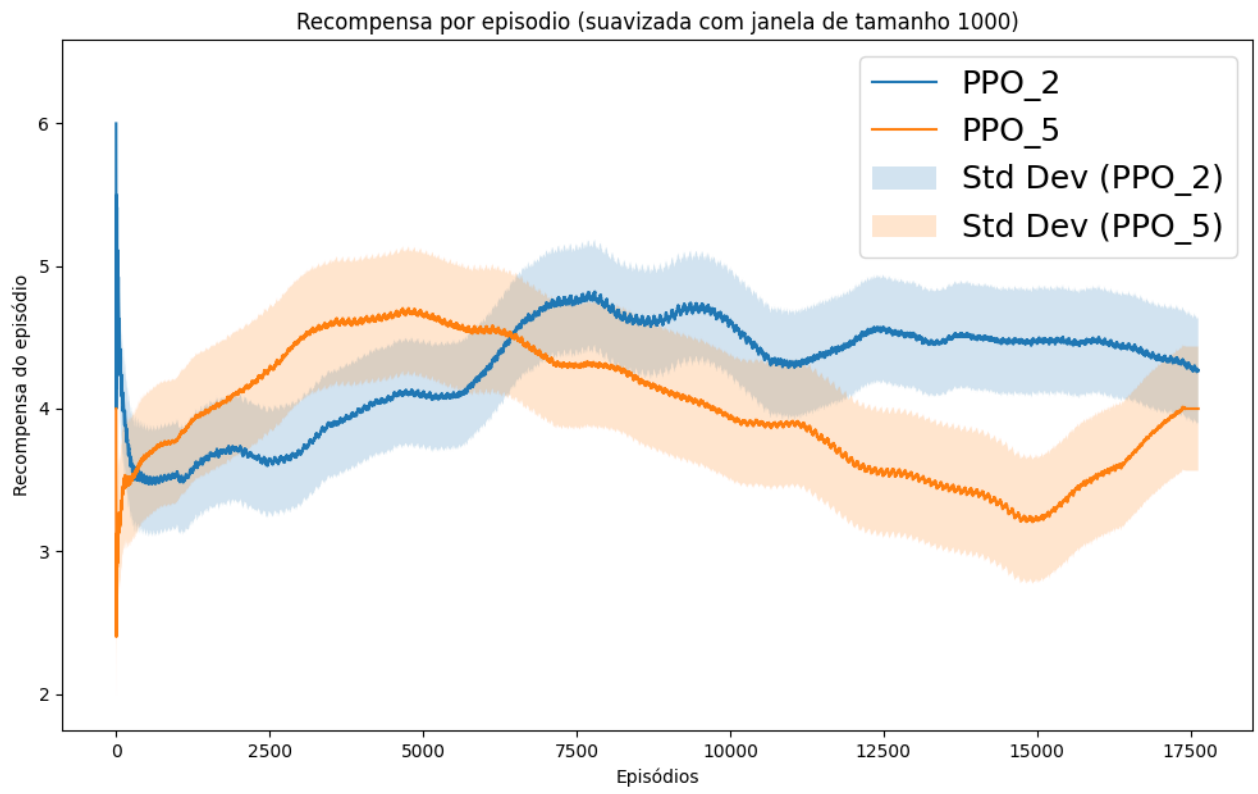
Como ver o agente jogando

1. Crie um ambiente virtual, execute `conda activate “path_do_ambiente”`, `pip install -r requirements.txt`
2. Siga os passos do README para corrigir os erros de typo.
3. Execute `pytho --version` para verificar se está na versão 3.9 e, então, execute python `src/visualize_agent_playing.py`
4. Modifique `PLAY_WITH_DQN` dentro de `src/visualize_agent_playing.py` para escolher entre DQN e PPO jogando
5. Se rodar sem erro, deve existir um novo `DQN.gif` ou `PPO.gif` gerado na pasta do projeto

Resultados e comparação



Recompensa por episódio, DQN, 887_000 passos



Recompensa por episódio, PPO_2 azul (experimento 2), 100_352 passos
Recompensa por episódio, PPO_5 laranja (experimento 5), 94_720 passos

> No caso do DQN, o algoritmo desaprendeu ao longo do tempo e isso indica que precisa de parâmetros melhores ou mais tempo de treino.
>
> No caso do PPO_2 (experimento 2) e PPO_5 (experimento 5), o algoritmo finalizou com uma recompensa melhor que DQN e aparenta estar aprendendo em vez de desaprender.
>
> Para plotar o gráfico do PPO, foi acrescentado um padding de recompensas no final do PPO_5, repetindo a última recompensa até igualar o tamanho do eixo horizontal.
>
> Nenhum dos 2 agoritmos aparenta ter convergido e, observando o agente jogando, ele não aprendeu nenhuma estratégia inteligente.