Jogando RTS com Aprendizado de Reforço Profundo

Victor Aliende da Matta

Supervisor: Prof. Dr. Denis Deratani Mauá

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

Introdução

Aprendizado de Reforço (AR) é uma das grandes áreas que compõe Aprendizado de Máquina [2], onde são estudados algoritmos que descrevem o comportamento de *agentes* em um *ambiente*, buscando o maior *retorno* por suas ações.

Pesquisa na área de Aprendizado de Reforço comumente adota jogos eletrônicos como o ambiente para testar seus algoritmos [3], já que estes proporcionam desafios complexos e bem definidos para a análise e comparação de algoritmos, em um ambiente totalmente controlável. Dentro destes, os jogos de estratégia em tempo real (RTS) se destacam por poder envolver um espaço de ação grande e mutável, grande conjunto de estados, temas de planejamento, informação imperfeita e recompensas distantes para as ações dos agentes. Conforme essa tendência, nesse trabalho utilizaremos como estudo de caso um jogo RTS criado para a pesquisa de algoritmos de AR.

Como resultado principal, seguimos a abordagem de [3], modelamos nossa agente como uma Rede Neural Convolucional e a otimizamos com subida de gradiente, calculando este com o algoritmo A3C (*Asynchronous Advantage Actor-Critic*). Testamos nossos agentes contra duas estratégias estabelecidas no artigo.

Aprendizado de Reforço

Geralmente, os problemas da área são formulados como Processos de Decisão de Markov (MDP) finitos. Um Processo de Decisão de Markov é uma quíntupla (S, A, T, R, γ) [2], onde:

- S é o conjunto de estados (que representa o ambiente)
- A é o conjunto de ações que o agente pode tomar
- $T : S \times S \times A \to [0, 1]$ representa a probabilidade de um novo estado s' dados o estado atual s e ação a
- $R : S \times A \to \mathbb{R}$ a função de retorno
- $\gamma \in [0, 1]$ é um fator que determina o a importância de retornos futuros comparados com retornos imediatos.

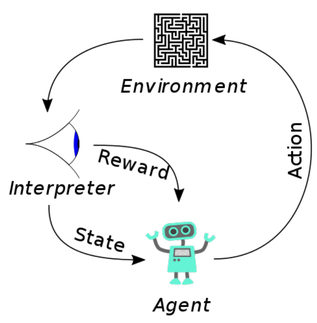

Figura 1: A operação de convolução

O agente define uma *política* $\pi : A \times S \to [0, 1]$ que representa a distribuição de probabilidade da escolha de uma ação em um dado estado $\pi(a|s)$.

Assim temos um mecanismo para gerar amostras, partindo de um estado S_0, selecionamos uma ação $A_0 \sim \pi(a|S_0)$, deixamos o agente observar $R_0 = R(S_0, A_0)$ e selecionamos um estado $S_1 \sim T(s|S_0, A_0)$, a partir do qual o processo se repete, até alcançar um estado S_T que seja terminal. Ao conjunto $S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2, ..., S_T$, damos o nome de *episódio*.

O objetivo do agente é, através de suas observações dos episódios, aprender uma política que maximiza o *retorno descontado* G_0 onde: $G_t := R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... = R_t + \gamma G_{t+1}$.

Será útil para nossa discussão definir a função *valor* $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$ de um estado s, para a política π em um tempo t qualquer.

Ambiente

O ambiente utilizado no trabalho foi o ELF [3].

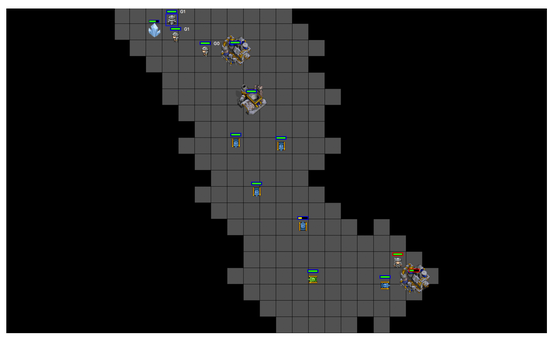

Figura 2: A operação de convolução

O jogo consiste de dois jogadores disputando por recursos, construindo unidades e as controlando com o objetivo final de destruir a base do adversário. Mais precisamente, cada jogador começa com sua base em cantos opostos de um mapa quadricular, que contém fontes de recursos. É importante ressaltar que os jogadores só tem visão do que está suficiente próximo de suas unidades (e conhecimento da localização da base inimiga), isso significa que se trata de um problema de *informação parcial*, ilustrado na figura. Assim, a formulação do problema como um MDP é uma aproximação.

Algoritmo

Atacaremos o problema definindo uma *parametrização* da política, ou seja, uma família de funções da forma $\pi_\theta(a|s) = \mathbb{P}(A_t = a|S_t = s, \theta_t = \theta)$ e uma função objetivo $\mathcal{J}(\theta)$ de tal forma que nosso problema se reduz a encontrar um vetor $\theta \in \mathbb{R}^d$ que maximiza a função $\mathcal{J}$. Nossa função objetivo será escolhida como $\mathcal{J}(\theta) \doteq v_{\pi_\theta}(S_0)$, ou seja, o valor do estado inicial na política definida por θ.

Essa função é diferenciável e além disso temos uma forma conveniente de expressar seu gradiente através do teorema do gradiente de política [2]:

$$\nabla \mathcal{J}(\theta) \propto \mathbb{E}_\pi\Big[q_\pi(S_t, A_t)\frac{\nabla_\theta \pi_\theta(A_t|S_t)}{\pi_\theta(A_t|S_t)}\Big] = \mathbb{E}_\pi\Big[G_t\frac{\nabla_\theta \pi_\theta(A_t|S_t)}{\pi_\theta(A_t|S_t)}\Big] \tag{1}$$

Isso sugere um algoritmo para achar um vetor θ, chamado REINFORCE [2], que pode ser descrito como:

$$\theta = \theta + \alpha\gamma^t G_t \frac{\nabla_\theta \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \tag{2}$$

Esse algoritmo é próximo da teoria porém é relativamente lento na prática, principalmente pela variância no processo de treinamento, dado que as atualizações dependem totalmente dos episódios observados. Daí surge a classe de algoritmos *Actor-Critic* [2], que utilizam uma estimativa do valor do estado ao invés do retorno, diminuindo a variância no treinamento. O algoritmo A3C, de *Asynchronous Advantage Actor-Critic* [1], é uma modificação do Actor-Critic que usa fortemente paralelismo para diversificar a experiência do agente a cada atualização dos parâmetros (o que deixa o treinamento mais estável) e usufruir totalmente de processadores multi-core, por esses motivos, o treinamento com o algoritmo A3C tende a ser significativamente mais rápido do que com Actor-Critic puro.

Modelo

O nosso modelo será uma Rede Neural Convolucional com duas cabeças: uma que nos dá a distribuição $\pi(\cdot|S)$ e a outra nos dá a estimativa do valor $\hat{v}$.

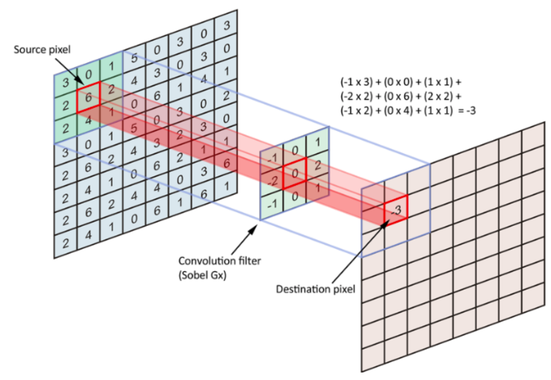

Figura 3: A operação de convolução

A arquitetura da nossa rede consiste de duas camadas convolucionais (utilizando o ReLU como função de ativação) e uma camada de max-pool. Temos então duas operações lineares, uma determinando a política e a outra determinando o valor do estado.

Resultados

Reportamos os resultados do treinamento contra 2 estratégias fixas, que chamamos de AI_SIMPLE e AI_HIT_AND_RUN, conforme [3].

Contra AI_HIT_AND_RUN, os resultados encontrados foram melhores, porém dentro da variação estatística reportada no artigo. É possível que nosso modelo simplificado é capaz de rapidamente convergir para uma solução simples e eficiente contra essa estratégia, superando o resultado do artigo.

Contra AI_SIMPLE, os resultados do artigo foram decisivamente melhores. Podemos atribuir essa diferença ao modelo mais complexo, e a um processo de otimização diferente: utilizando *curriculum learning*, onde o algoritmo primeiro observa o AI_SIMPLE jogando contra si mesmo (aprendendo *off-policy*) e depois começa a ser treinado a partir da observação de sua própria política interagindo com o ambiente (aprendendo *on-policy*). No artigo, o agente também foi treinado a partir de estados aleatórios, aumentado a diversidade do treino.

Abordagem	Acurácia
Agente aleatório vs AI_SIMPLE	0.242
Agente aletório vs AI_HIT_AND_RUN	0.259
vs AI_SIMPLE	0.537
vs AI_HIT_AND_RUN	0.671
Tian et al. vs AI_SIMPLE	0.684
Tian et al. vs AI_HIT_AND_RUN	0.636

Tabela 1: Principais resultados

Conclusões

Passamos pelos principais conceitos e algoritmos essenciais da área de Aprendizado de Reforço Profundo, construindo uma base para se aprofundar no assunto. Muito do trabalho atual na área é uma continuação natural do que é visto aqui.

Os algoritmos aqui estudados já são suficientemente poderosos para atacar vários problemas, podendo lidar com um conjunto de estados grande e complexo, distinguindo-se dos algoritmos tabulares clássicos de Aprendizado de Reforço. Poder que vêm também com desvantagens, pois os algoritmos empregados requerem uma quantidade grande de tempo de treinamento.

Referências

[1] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[2] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[3] Yuandong Tian, Qucheng Gong, Wenling Shang, Yuxin Wu, and C Lawrence Zitnick. Elf: An extensive, lightweight and flexible research platform for real-time strategy games. In *Advances in Neural Information Processing Systems*, pages 2659–2669, 2017.