

Inteligência Artificial 19/20



ZHED and Reinforcement Learning

Trabalho realizado por:

- Matheus Gonçalves
- Miguel Pires
- Ricardo Cardoso

- up201405081
- up201406989
- up201604686

Introdução

O Zhed é um jogo de tabuleiro do tipo solitário que consiste em resolver um puzzle. O tabuleiro é constituído por quadrículas numeradas que representam as peças do jogo e uma casa de destino.

O objectivo do jogo é expandir as peças numeradas (em que o número simboliza quantas casas irá ocupar quando expandida) em quatro direções (cima, esquerda, direita, baixo) de forma que a sua expansão cubra a casa destino. Uma mecânica interessante do jogo é o facto de uma peça já expandida poder ser "atravessada" por outra a expandir.

Neste projeto final o nosso objetivo é colocar os agentes a aprenderem a resolver este puzzle por **Reinforcement Learning**, usando alguns dos seus algoritmos principais.

Descrição do problema

O problema que nos foi proposto é referente à resolução do jogo Zhed com recurso à implementação de dois algoritmos de reinforcement learning, nomeadamente o Q-Learning e Sarsa. O objetivo é criar um agente que com os treinos necessários consiga resolver um nível recorrendo à informação que é guardada durante os treinos.

Para isso criamos um ambiente constituído por um tabuleiro que contém informação sobre o jogo. Esta informação é atualizada a cada iteração realizada por cada um dos algoritmos, sempre com o objetivo de treinar o algoritmo a atingir a solução do problema de uma forma o mais eficaz possível.

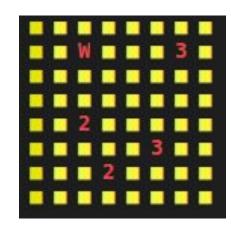


Fig.1. Exemplo de um tabuleiro no seu estado inicial

Abordagem

Começamos por implementar toda a lógica do jogo usando a biblioteca OpenAl Gym no ficheiro **gym_zhed/envs/zhed_env.py** .

Tal lógica foi implementada definindo um espaço de observação bem como um espaço de ação para o nosso agente. O espaço de observação engloba todos os estados que o ambiente poderá assumir, para tal chegamos a uma fórmula que traduz esse mesmo número, $n! + 4^n$, sendo n o número de peças presentes no tabuleiro. Para o espaço de ação definimos 4 ações diferentes por cada peça ou seja, 4 * n.

De seguida implementamos 2 algoritmos:

- Q-Learning
- SARSA

Q-Learning

Para este algoritmo começamos por definir quantos episódios (total de jogos) é que o agente irá concluir. Por cada episódio o agente escolhe uma ação do qual tem duas hipóteses: ou explora o ambiente escolhendo uma ação aleatória do conjunto de ações disponíveis ou então aproveita a ação com o melhor valor de Q para aquele estado da tabela de valores guardados. A escolha de tal decisão (explorar ou aproveitar) é feita através de um valor pré definido que traduz a taxa de exploração (ɛ). Depois de escolhida a ação o agente dá o passo necessário a executá-la retornando um novo estado e uma recompensa por a ter executado. As recompensas são mapeadas da seguinte forma:

Agente seleciona ação inválida: r = -1

Agente chega ao fim do jogo: r = +1000

• Agente sai fora do tabuleiro: r = -(n) * 10, sendo $n \circ 0$ de casas que saiu fora

Agente expande uma peça: r = (E - P) * 50

Sendo E igual ao número de casas que o agente expandiu numa dada direção e P o número associado à peça, deixando assim o nosso agente com uma reward positiva se expandiu mais casas do que aquelas que eventualmente poderia expandir ou negativa caso não aproveite todo o potencial de uma peça.

Depois de ter feito uma ação o agente recalcula o novo valor para ser inserido na tabela seguindo a fórmula de Q-Learning:

$$\underbrace{\operatorname{New}Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \alpha \underbrace{[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)]}_{\text{Reward}}$$

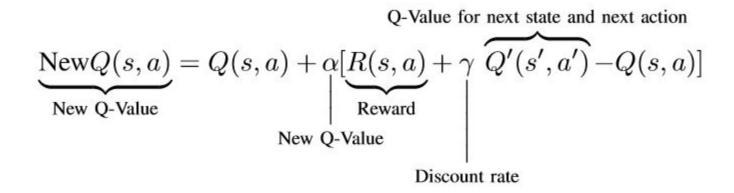
- α Simboliza a taxa de aprendizagem, i.e a importância da recompensa atual (quanto maior for o valor maior será o impacto da recompensa);
- γ Factor de desconto que traduz a importância de recompensas futuras (valor alto significa mais importância em recompensas futuras).

Por fim o algoritmo verifica se conseguiu alcançar o objectivo que é preencher a casa destino, ou se alcançou o limite de ações num dado nível, dando o episódio por terminado em ambos os casos. Com o episódio terminado o tabuleiro volta ao seu estado inicial e o processo repete-se até atingir o número de treinos pretendido.

Finalmente os dados da tabela com os valores finais de Q são guardados num ficheiro de texto dentro da pasta *tables*.

SARSA (State, Action, Reward, State, Action)

O procedimento de treino para este algoritmo é exatamente igual ao de Q-Learning excepto na função de atualização de novos valores para a tabela. A única diferença que é visível é o facto de este algoritmo ter em conta no cálculo do novo valor de Q para uma determinada ação o estado futuro bem como a futura ação que irá desempenhar/executar.



Avaliação experimental

Como exemplo prático consideramos o nível 4:

Um jogador humano facilmente conseguiria resolver este nível sem fazer muito esforço, teríamos então de mover a peça 3 na horizontal para a esquerda, a peça 1 de cima para baixo, e a peça 1 da esquerda executava o movimento para a direita para terminar o nível. No caso do nosso agente ele precisa primeiro explorar o ambiente e tirar notas para depois se tornar eficaz a resolvê-lo.

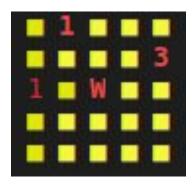


Fig. 2. Nível de exemplo usado nos testes

No nosso caso para treinar o agente com o algoritmo de Q-Learning é preciso executar o ficheiro train q learning e especificar o nível (neste caso o 4) bem como o total de episódios a executar por treino. A pergunta que se levanta é se é garantido que o agente chega a um estado final nesse processo de treinos, a resposta é não, posto isto é importante deixar o agente experienciar o ambiente durante várias iterações e ir alterando os hiperparâmetros (α , γ , ε) de forma a se adequarem a cada nível em específico. No caso deste nível as recompensas futuras não são de destacar pois não existem muitas expansões entre peças no tabuleiro, ou seja, deve-se priorizar mais as recompensas atuais de forma a atingir o objectivo. Em muitos agentes de aprendizagem por reforço o valor de ε (epsilon) tem como valor um número muito próximo de 0 (ex: 0,1) mas como já averiguamos cada nível é um nível e por vezes um valor mais baixo de epsilon implica menos experimentação de caminhos que são à partida piores por parte do agente naquele determinado nível, mesmo que possam ser estes a originar a solução final depois de completamente explorados. Para este exemplo usaremos então valores adequados:

- a = 0.8
- $\varepsilon = 0.2$

Correndo o treino para o nível 4 usando Q-Learning com um total de 5000 episódios obtemos o seguinte resultado:

Results after 5000 train sections: Total timesteps: 24521 Total penalties: 10377 Total solutions found: 2992 Average timesteps per episode: 4.9042 Average penalties per episode: 2.0754

Fig. 3. Após 5000 episódios verificamos que por episódio houve a escolha de 2 ações inválidas por parte do agente, bem como uma média de 5 movimentos por episódio encontrando assim 2992 soluções.

Correndo novamente o mesmo teste com 5000 episódios e como epsilon prioriza as ações já exploradas obtemos o seguinte resultado:

Results after 5000 train sections:
Total timesteps: 23653
Total penalties: 9422
Total solutions found: 3200
Average timesteps per episode: 4.7306
Average penalties per episode: 1.8844

Fig. 4. Após 5000 episódios verificamos que por episódio houve a diminuição de passos por episódio bem como uma redução significativa nas penalizações. É de notar também o aumento de conclusões de nível de 2992 para 3200

Para o SARSA fizemos o teste com diferentes valores nos hiperparâmetros:

- a = 0.8
- $\varepsilon = 0,1$

Obtendo os seguintes resultados depois de 5000 episódios:

Fazendo outro treino de 5000 episódios obtivemos os seguintes resultados:

Results after 5000 train sections: Total timesteps: 36684 Total penalties: 23115 Total solutions found: 2099 Average timesteps per episode: 7.3368 Average penalties per episode: 4.623

Fig. 5. Após 5000 episódios verificamos que teve mais passos comparado ao primeiro teste com Q-Learning, bem como mais penalizações durante o treino mesmo tendo um valor inferior de epsilon o que significa que o Q-Learning teve uma melhor performance no primeiro treino

Results after 5000 train sections: Total timesteps: 44490 Total penalties: 31263 Total solutions found: 2050 Average timesteps per episode: 8.898 Average penalties per episode: 6.2526

Fig. 6. Após outros 5000 episódios verificamos que não houve uma subida de performance por parte deste algoritmo, muito pelo contrário. O que nos leva a querer que para o contexto do nosso jogo, seria melhor usar q-learning

Para depois verificar a integridade destes algoritmos bem como as tabelas de valores por eles gerados temos um script que simplesmente escolhe a melhor ação para um determinado estado a partir do maior valor para a ação dessa tabela. Correndo o mesmo verificamos que encontramos facilmente uma solução:

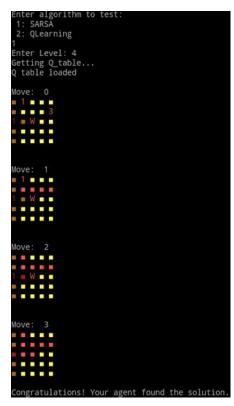


Fig. 7. Um episódio basta para chegar ao objectivo do jogo utilizando o algoritmo SARSA (o mesmo acontece com o Q-Learning

Conclusão

Após a realização deste trabalho concluímos que apesar de tanto o **Q-Learning** como o **SARSA** serem algoritmos de *Reinforcement Learning* existem algumas diferenças entre eles.

Primeiro detectamos que para obter resultados significativos usando *SARSA* teríamos de efetuar cada vez mais testes comparado ao *Q-Learning*. Também detectamos que *SARSA* aplicado ao nosso contexto não é assim tão bom quanto esperávamos.

Finalmente verificamos que diferentes valores de a, γ e ε originam resultados bastante diferentes, sendo que não foi possível treinar o agente com sucesso para todos os níveis.

Dados os parâmetros apresentados, após 10.000 treinos o algoritmo Q-Learning resolveu com sucesso os níveis 1, 2, 3, 4 e 5 enquanto que o algoritmo SARSA resolveu com sucesso os níveis 1, 2, 3 e 6.

Trabalhos relacionados e referências

- ZHED Solver https://www.wilgysef.com/articles/zhed-solver/
- Sarsa https://www.geeksforgeeks.org/sarsa-reinforcement-learning/
- Q-Learning https://www.geeksforgeeks.org/q-learning-in-python/
- OpenAl Gym http://gym.openai.com/docs/
- OpenAl Gym Example -https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-pytho
 n-openai-gym/
- Russel, S., & Norvig, P. (2010) Artificial Intelligence A Modern Approach, New Jersey.