Universidade do Minho

Mestrado Integrado em Engenharia Informática

Computação Natural

Reinforcement Learning aplicado a agentes

Luís Gomes (A78701)

Joel Rodrigues (A79068)

Elisa Valente (A79093)

2 de Fevereiro de 2021

Conteúdo

Figuras

1 Introdução

A Aprendizagem por Reforço (*Reinforcement Learning* - RL) é uma área de *Machine Learning* que pretende encontrar uma solução para um determinado problema, utilizando para isso um mecanismo de tentativa-erro. Como tal, possui uma representação de cada estado possível no ambiente e das ações que podem ser tomadas em cada estado. A partir das ações tomadas e de acordo com as recompensas recebidas, o agente aprimora a ação a escolher em situações futuras semelhantes.

Pretende-se assim apresentar as decisões tomadas na realização do segundo trabalho prático, aplicando *Reinforcement Learning* a agentes computacionais.

Neste documento é apresentada a descrição do problema proposto, na secção 2. Na secção 3 são apresentadas as decisões relativas à preparação do modelo de *Reinforcement Learning* e às funções implementadas. Com o modelo criado, é efetuada a sua análise e validação com a mudança dos diferentes parâmetros (secção 4). De forma a melhorar o desempenho do modelo foi utilizada a otimização de parâmetros com recurso a *Particle Swarm Optimization* (PSO). Este processo é demonstrado na secção 5. Por fim, na secção 6 são mostrados os resultados obtidos. Para além disto foi efetuada uma tarefa extra com o intuito de estudar o modelo de RL para mapas de tamanho médio que pode ser observada na secção 7.

2 Descrição do problema

O objetivo do trabalho proposto passa por criar um modelo *Reinforcement Learning* capaz de aprender a jogar o jogo Pacman, através de códigos padrão do jogo disponibilizados. Para além disto, pretende-se realizar a análise e validação do desempenho de modelo e uma posterior otimização de parâmetros.

Deste modo, numa primeira fase pretende-se treinar o modelo, para o agente aprender os *QValues*, nomeadamente as ações que deve tomar perante qualquer alteração do ambiente. Uma vez que o treino se encontre completo, passa-se para o modo de teste onde é realizada uma análise e validação dos resultados obtidos.

Para além disto, tenciona-se otimizar os parâmetros do modelo *Reinforcement Learning* recorrendo ao método *Particle Swarm Optimization* que tem como objetivo melhorar uma solução tendo por base uma função objetivo.

Como tarefa complementar, objetiva-se, ao fim do desenvolvimento do modelo, apresentar alguns problemas da estratégia usada com o aumento do tamanho dos mapas e futuras soluções para melhorar o desempenho do mesmo.

3 Preparação do modelo de *Reinforcement Learning*

Para a construção do modelo de aprendizagem foi disponibilizado um ficheiro onde existiam funções com código em falta e que foi necessário completar. O preenchimento destas funções teve em vista o correto funcionamento de um algoritmo de RL. Desta forma, as funções alteradas foram as seguintes:

- `computeValueFromQValues` - Este método tem como objetivo a obtenção do maior *QValue* pertencente a um determinado estado. Este cálculo é posteriormente efetuado aquando da atualização dos *QValues*.
- `computeActionFromQValues` - O presente método foi implementado para determinar a melhor ação a ser executada pelo agente, perante um determinado ambiente (estado).
- `getQValue` - Método que retorna o *QValue* para um dado par (estado,ação).
- `getAction` - Método responsável por calcular a ação a ser executada num determinado estado, tendo em conta o fator de exploração, ou seja, com uma probabilidade *epsilon* de ser escolhida uma ação aleatória ao invés da melhor política.
- `update` - O método *update* tem como finalidade a atualização dos *QValues* ao longo das iterações do modelo de RL.

3.1 SARSA vs Q-Learning

A atualização dos *QValues* no modelo de RL, pode ser efetuada de diferentes formas, sendo que as mais usadas são com os algoritmos SARSA e Q-Learning. No primeiro as ações são tomadas tendo em conta as consequências das mesmas num estado futuro. Já no segundo apenas se tem em conta o estado atual, sendo que o agente apenas segue a política ótima em cada estado.

Efetuando um paralelismo com o problema em questão pode ser considerado um exem-

plo concreto: no Q-Learning, caso o agente tenha uma recompensa (comida) na vizinhança, desloca-se sem se preocupar com o ambiente para além do que lhe é circundante (figura 1); já com o SARSA, apesar de existir uma recompensa num estado vizinho, não se deslocará, pois apresenta mais perigo (fantasma) no futuro (figura 2).

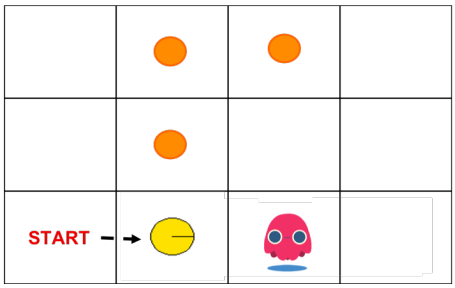

Figura 1: Q-Learnig.

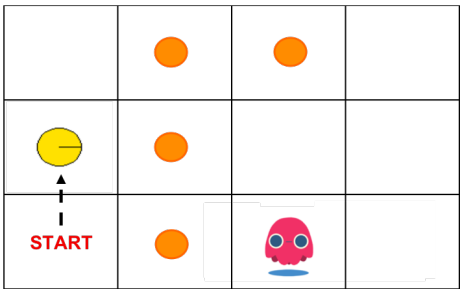

Figura 2: Sarsa.

4 Análise e validação do desempenho do modelo *RL*

Depois de implementado o modelo de RL, numa primeira instância, decidiu-se avaliar o desempenho do modelo sem qualquer processo de otimização. Para isso foram feitos alguns testes, alterando manualmente diversos parâmetros do modelo.

Desde logo, fixou-se o mapa `smallGrid` e foram realizadas as seguintes alterações:

- Teste com diferentes valores para os parâmetros *alpha*, *gamma* e *epsilon*;
- Variação do número de iterações de treino [1000,10000];
- Tipos de fantasmas presentes no mapa (trajetória aleatória ou direcional);
- Teste das diferentes fórmulas, Sarsa e Q-Learning.

Após esta primeira avaliação concluiu-se que a melhor fórmula a utilizar seria o Q-Learning e que 2000 iterações para o mapa `smallGrid` eram suficientes para que o agente aprendesse todas as ações.

Realizando o mesmo tipo de avaliação para outros mapas, os melhores resultados obtidos foram os seguintes:

- **SmallGrid:** 10 vitórias em 10
- **MinimaxClassic:** 4 vitórias em 10
- **TrappedClassic:** 5 vitórias em 10
- **TestClassic:** 0 vitórias em 10

No entanto, estes números de vitórias eram muito instáveis.

5 Otimização de parâmetros do modelo - *Particle Swarm Optimization*

Como os resultados obtidos nos testes referidos na secção 4 foram insatisfatórios para alguns mapas, decidiu-se recorrer ao *Particle Swarm Optimization* para a otimização de parâmetros do modelo. Este método tenta iterativamente melhorar uma solução candidata tendo por base uma função objetivo. A resolução do problema passa pela existência de uma população de soluções candidatas (particulas) que se movem em torno do espaço de procura. O movimento de cada partícula é influenciado não só pela sua melhor posição local mas também pelas posições mais conhecidas da população no espaço de procura, que são atualizadas à medida que melhores posições são encontradas por outras partículas. Assim espera-se que a população convirja para a melhor solução.

Para este problema definiu-se o tamanho da população como sendo apenas de 10 partículas e o número de iterações igual a 10. Escolheram-se estes valores porque eram computacionalmente suportáveis pelas máquinas utilizadas. No entanto, com uma maior população e um maior número de iterações os resultados obtidos seriam possivelmente melhores.

5.1 Espaço de procura

O espaço de procura é definido pela taxa de aprendizagem, taxa de exploração e fator de desconto. Todos estes três aspetos podem variar entre 0 e 1 no entanto, decidiu-se reduzir a gama de valores para diminuir o espaço de procura.

- **Taxa de Aprendizagem *(alpha)* = [0.1, 0.7]**

  A Taxa de Aprendizagem determina a mudança de peso em cada etapa de aprendizagem. Grandes valores de *alpha* levam a que um novo conhecimento seja aprendido mais rapidamente, no entanto, o conhecimento existente passa a não ter tanto valor. Como tal, decidiu-se que os valores de *alpha* apenas podem variar entre 0.1 e 0.7.

- **Taxa de Exploração *(epsilon)* = [0.0, 0.15]**

  A Taxa de Exploração indica o quão explorador é o agente. Um *epsilon* grande significa que o agente muitas vezes irá explorar novas ações, não sendo estas as ótimas. Contudo, não é isso é que se pretende para este problema e, deste modo, decidiu-se que o *epsilon* apenas poderá variar entre 0.0 e 0.15 para se dar maior importância ao facto do agente aprender melhor a política ótima.

- **Fator de desconto *(gamma)* = [0.5, 1.0]**

  O Factor de Desconto é uma medida que indica o quão longe no tempo o algoritmo está disposto a observar. Se o *gamma* estiver mais próximo de zero, o agente tenderá a considerar apenas recompensas imediatas. Se o *gamma* estiver mais próximo de um, o agente irá considerar recompensas futuras com maior peso, estando assim disposto a atrasar a recompensa. Como tal, decidiu-se dar maior importância a recompensas futuras reduzindo a gama de valores para o intervalo [0.5, 1.0].

5.2 Função Objetivo

Inicialmente, para a função objetivo, pensou-se que o melhor seria tentar minimizar a percentagem de derrotas ($1.0 - winRate$). No entanto, após alguns testes concluiu-se que essa não era a melhor opção uma vez que apesar de se estar a otimizar para ganhar não se estava a otimizar para ganhar com o melhor resultado. Posto isto, a função objetivo é a seguinte: -$\sum scores$.

6 Resultados Obtidos

De forma a avaliar o desempenho do modelo em diferentes ambientes, testou-se e otimizou-se um modelo de RL para cada mapa, com 2000 episódios de treino. Os resultados que se obtiveram foram os seguintes:

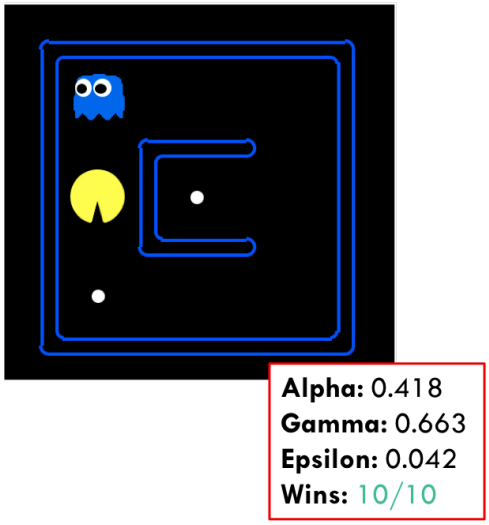

Figura 3: Mapa - SmallGrid.

Figura 4: Mapa - MinimaxClassic.

Figura 5: Mapa - TrappedClassic.

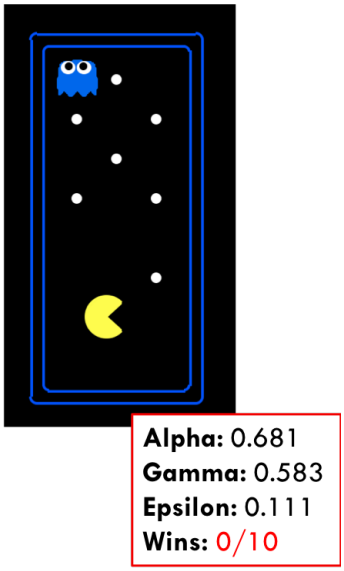

Figura 6: Mapa - TestClassic.

De entre os mapas apresentados, o *SmallGrid* (Fig. 3) é aquele que apresenta um conjunto menor de estados possíveis para o agente avaliar, como tal, comparativamente aos outros resultados, o modelo neste mapa, apresenta resultados elevados e estáveis.

Nos mapas *MinimaxClassic* (Fig. 4) e *TrappedClassic* (Fig. 5) verifica-se que apesar da percentagem elevada de vitórias que se conseguiu em alguns casos de teste, nem sempre os resultados obtidos foram assim tão favoráveis. Com 2000 episódios de treino, para os valores de *alpha*, *gamma* e *epsilon* resultantes do modelo de PSO, os resultados são instáveis.

Após os resultados obtidos nos mapas considerados, verifica-se que a percentagem de vitórias diminui com o aumento do tamanho do mapa. Sendo que esta premissa tem como consequência o aumento exponencial do conjunto de estados possíveis, concluí-se que 2000 episódios de treino não são suficientes para o agente aprender em todos os mapas.

7 Modelo de RL em mapas médios

O modelo de RL criado apresenta bons resultados para mapas de tamanho pequeno, especialmente quando o número de estados possíveis é diminuto em relação ao número de iterações de treino. De forma a perceber a abrangência do modelo foi efetuado um estudo em mapas de tamanho superior, que apresentam assim um número de estados e ações muito superior. Os mapas escolhidos foram assim submetidos aos processos de treino e teste como visto na secção 4 para os mapas pequenos. Os resultados obtidos, no entanto foram muito aquém daquilo que seria razoável. Os mapas utilizados e os respetivos resultados obtidos encontram-se representados na figura 7.

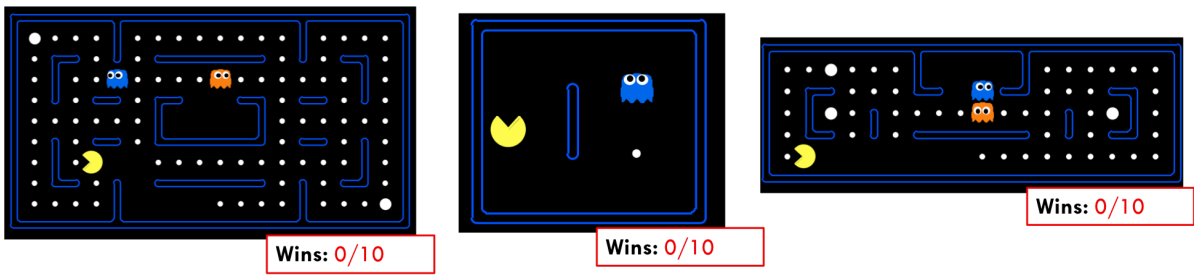

Figura 7: Resultados obtidos em mapas médios.

7.1 Limitações

Uma das primeiras observações a fazer é que o processo de treino destes mapas é muito superior aos mapas pequenos. Os testes foram efetuadas com até 5000 iterações de treino sem nenhuma melhoria nos respetivos resultados. Desta forma, foi revisto o processo de representação e atualização dos diferentes estados numa tentativa de perceber a razão de o agente não se comportar como deveria em situações que pareciam óbvias. Os estados são representados na forma de uma *string*, onde cada ponto do mapa é exibido como um carácter, representativo de um elemento do jogo (Pacman, fantasma, comida, espaço livre, parede e bónus). Este tipo de representação remete para o problema mostrado na figura 8 onde uma mesma situação é representada por dois estados distintos. Esta

situação leva a que o aumento do número de estados seja exponencial com aumento de tamanho no mapa devido ao elevado número de combinações de caracteres existentes.

Assim o maior problema do treino do agente em mapas grandes reside no facto de o mesmo estado não ocorrer mais que uma vez, não ocorrendo a atualização dos *Qvalues* e consequentemente não existindo uma melhoria no comportamento do agente.

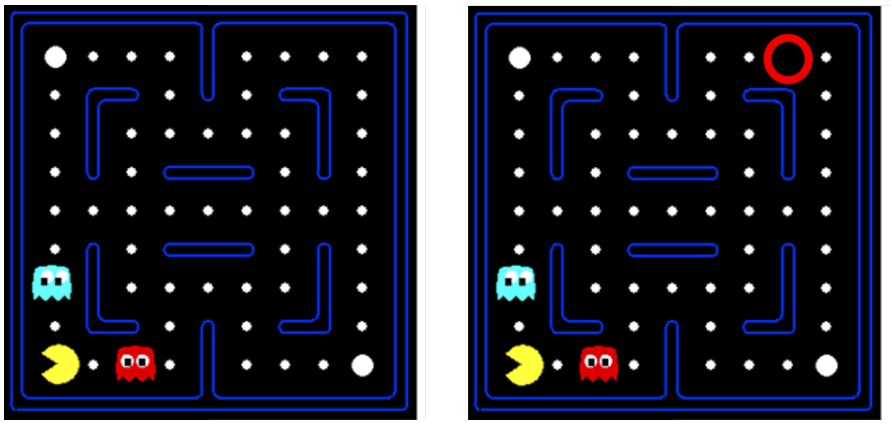

Figura 8: Mesma situação mas diferentes estados.

7.2 Otimizações

Para melhorar os aspetos referidos na secção 7.1 foram pensadas algumas otimizações para o algoritmo desenhado. A primeira, e mais óbvia seria a utilização de um processo de treino prolongado onde houvesse a possibilidade de o agente visitar a maioria dos estados possíveis, atualizando os *Qvalues* em conformidade. Apesar da sua simples implementação, esta operação seria muito custosa e demorada.

Outra forma de melhorar o desempenho do processo de aprendizagem seria representar apenas o mapa na zona circundante do pacman. Este tipo de representação seria mais eficiente na medida em que o mapa seria subdividido e seriam diminuídos os estados possíveis, sendo descartada informação que se encontra "longe"do agente.

No entanto, a melhor solução passaria por utilizar um conjunto de características que representassem de uma forma mais generalizada o mapa sem perder as informações mais

relevantes. Essas características poderiam ser mapeadas manualmente num vetor, como por exemplo:

- distância aos k pontos de comida mais próximos;
- distância aos n fantasmas mais próximos;
- número de pontos de comida restantes;
- número de fantasmas em jogo.

Em alternativa, poderiam ser deduzidas automaticamente outras propriedades com o recurso a técnicas de extração de *features* como o *Approximate Q-learning*. Neste tipo de método os pesos são aprendidos após a extração de propriedades dos estados de jogo originais.

8 Conclusão

O problema apresenta um conjunto de características que fazem com que o *Reinforcement Learning* seja adequado para a sua resolução. As duas principais características são a existência de estados do ambiente e ações possíveis de executar pelo agente em cada estado.

Os resultados obtidos demonstram que este método de aprendizagem consegue resultados ótimos quando o número de estados e ações é relativamente pequeno, no entanto, o seu desempenho degrada-se à medida que a dimensão do problema aumenta. Desta forma, e de acordo com os resultados obtidos, pode inferir-se que a utilização do RL puro não é a mais adequada, devido à sua natureza que impõe um número de iterações de treino exponencialmente crescente de forma a cobrir todos os estados mais que uma vez. A utilização de técnicas de representação ou extração de *features* seria uma solução mais viável neste contexto.

Relativamente ao método de otimização utilizado para decidir os valores das taxas de aprendizagem e de exploração e o fator de desconto (PSO), foram encontrados resultados satisfatórios. De facto, os valores destes parâmetros encontrados pelo algoritmo levam a resultados bastante superiores aos obtidos com a procura manual e, em alguns casos, retirando instabilidade no número de vitórias em execuções consecutivas de todo o processo de treino e teste.