INF01048 - Inteligência Artificial - Prof. Bruno Castro da Silva

Trabalho #2 – Q-Learning – 2016/2

Neste trabalho você irá implementar o algoritmo Q-Learning para treinar o robô Gladiador—de forma semelhante ao que foi feito no Trabalho 1 da disciplina. Este algoritmo, conforme discutido em aula, aprende uma função Q de preferência para cada ação, quando em um determinado estado: em particular, Q(s,a) é um valor numérico correspondente ao grau de preferencia do agente por executar a ação *a*, quando no estado *s*.

Quando solucionamos, no Trabalho 1, o problema de aprendizado do comportamento do Gladiador via métodos de busca local, tais como Hill Climbing, as funções Q foram descritas por conjuntos de pesos. Agora, esta função será armazenada de forma explícita em uma tabela. Visto que há um número infinito de estados possíveis (p.ex., de possíveis combinações de distâncias do inimigo, presença ou não do inimigo, distância do tiro, etc), precisaremos discretizar tais valores de forma que seja possível criar uma tabela armazenando valores Q para cada possível combinação de estado e ação. Assuma, como exemplo, que estamos discretizando a distância do inimigo em 4 níveis; a distância discretizada corresponderia, portanto, a um número inteiro entre 0 e 3, onde 0 indicaria a distancia mínima possível e 3 indicaria a distância máxima possível. Continuando com este exemplo, assuma que estamos utilizando como *features* de estado o vetor [$d_{inimigo}$, $v_{inimigo}$, d_{bala}, v_{bala}], onde $d_{inimigo}$ é a distância do inimigo (um valor real), $v_{inimigo}$ corresponde a um sensor de presença do inimigo (valor discreto, binário), d_{bala} corresponde à distância da bala (um valor real), e v_{bala} corresponde a um sensor de presença de bala (valor discreto, binário). O primeiro passo seria discretizar tais *features*; caso escolhêssemos discretizá-las em 4 níveis, isso implicaria que um possível estado do agente poderia ser [2, 1, 3, 1]; este estado indicaria que há um inimigo a distância é 2 (em uma escala de 0-3), que ele está visível, que há uma bala a distância 3 (em uma escala de 0-3), e que a bala está visível. Haveria, para este nível de discretização, N=2*4*2*4 possíveis estados. O algoritmo Q-Learning cria e atualiza uma tabela Q onde as linhas correspondem a cada um dos N possíveis estados, e cada coluna corresponder a uma das 4 possíveis ações (Atirar, Esquerda, Direita, AçãoNula).

O agente treinado pelo algoritmo Q-Learning pode, a cada momento, escolher a ação *a* a ser executada através da comparação dos valores Q de cada ação no estado atual, *s*. Isto é, ele poderia verificar qual ação *a* possui o maior valor de Q(s,a). Como os valores Q estão explicitamente armazenados em uma tabela, esta verificação é trivial. No Q-Learning, após executar a ação escolhida, o agente recebe uma recompensa, *r*, e observa qual o estado, *s'*, que resultou da execução da sua ação. De posse das informações de estado atual, ação executada, recompensa recebida, e estado resultante (isto é, da tupla <*s, a, r, s'*>), o agente as utiliza para atualizar o valor de Q(s,a), de acordo com a fórmula do Q-Learning.

Uma vantagem de utilizar Q-Learning (ou outros algoritmos de aprendizado por reforço) é que é possível prover *feedback* muito mais rápido/imediato ao agente a respeito de como ele está se comportando ao longo das batalhas. No caso da utilização de algoritmos de busca local, como os utilizados no Trabalho 1, o agente recebia uma indicação de performance apenas ao final de cada episódio, após ter executado centenas de ações; este agente não recebia, portanto, um *feedback* preciso de quais das ações executadas ao longo do episódio foram boas e quais foram ruins. Agora, com aprendizado por reforço, o agente pode receber uma recompensa *r* imediatamente após executar uma ação, e pode utilizar tal recompensa para melhorar o seu comportamento através da correção dos valores de preferência Q armazenados em sua tabela.

O simulador a ser utilizado neste trabalho pode ser obtido no seguinte repositório: https://bitbucket.org/marcosps96/gladiatorsarena2. Faça um clone deste projeto e crie um repositório no Bitbucket para armazenar o trabalho sendo desenvolvido por seu grupo.

Neste trabalho, utilizaremos um simulador semelhante ao do T1. Você deverá implementar as seguintes funções:

State.compute_features: esta função é semelhante à criada no Trabalho 1. Ela deve acessar as informações básicas de sensores do agente ($d_{inimigo}$, $v_{inimigo}$, d_{bala}, v_{bala}) e calcular as *features* que serão utilizadas por ele para caracterizar o seu estado—isto é, as informações que serão levadas durante a escolha por uma ação. Você poderia calcular/definir, por exemplo, uma *feature* de proximidade do inimigo (feature *prox,* como sugerido no Trabalho 1), ou quaisquer outras *features* que você julgar relevantes para a escolha da ação do agente em cada momento.

State.discretize_features: esta função deve discretizar os valores das *features* calculadas por State.compute_features. Em particular, assumindo que State.compute_features retorna um vetor com K *features*, a função State.discretize_features deve retornar um vetor de K elementos contendo a discretização de cada uma das *features*. O número de níveis a ser utilizado na discretização de cada *feature* fica a critério do grupo. Um número maior de níveis implicaria mais estados (e portanto aprendizado mais lento, pois a tabela Q a ser aprendida seria maior), mas teria a vantagem de prover ao agente informações mais detalhadas a respeito do seu estado. P.ex., se a distância do inimigo fosse discretizada em apenas 2 níveis, o agente saberia apenas se ele está muito próximo (distância 0) ou extremamente longe (distância 1). A distância poderia ser, por outro lado, discretizada em 1000 níveis, mas provavelmente isso traria poucas vantagens: o agente não precisaria realmente saber se o inimigo está a distância nível 997 ou 998 para decidir se deve atirar ou girar à esquerda.

State.discretization_levels: esta função deve retornar o número de níveis utilizadas para discretizar cada uma das *features* de estado. Assumindo que State.compute_features calcula K *features,* a função State.discretization_levels deve retornar um vetor de K elementos, onde o i-ésimo elemento corresponde ao número de níveis usados pra discretizar a i-ésima *feature*. Você pode utilizar níveis de discretização diferentes para diferentes *features*. *Features* binárias obviamente não precisam ser discretizadas; seu número de níveis é 2.

Controller.save_table_Q: esta função deve salvar os valores armazenados na tabela Q do agente em um arquivo de sua escolha. O propósito desta função é permitir com que os valores Q (os quais determinam o comportamento do agente) sejam gravados em disco e posteriormente re-utilizados, no momento da competição e/ou no modo *evaluate* do simulador. Fica a seu critério o formato em que estes dados serão armazenados no arquivo.

Controller.init_table_Q: esta função deve criar e inicializar a estrutura de dados que armazenará a tabela Q. Caso a função receba como entrada o nome de um arquivo, você deverá ler os valores Q lá armazenados e usá-los para inicializar a tabela. Caso a função não receba um nome de arquivo como entrada, você deverá apenas criar a estrutura de dados da tabela Q e inicializá-la com valores de sua escolha.

Controller.compute_reward: esta função deve calcular qual recompensa você dará ao agente a cada momento de sua vida. Esta escolha é crucial para que o agente consiga aprender o que é bom e o que não é. Por exemplo, você poderia dar uma recompensa negativa sempre que o agente se encontrasse em um estado muito próximo do inimigo; ou poderia dar uma recompensa positiva sempre que ele atirasse e *quase* acertasse o oponente. Recompensas e punições imediatas, logo após a execução de uma ação, servem como *feedback* rápido ao agente a respeito da qualidade do seu comportamento; isso permite com que com ele atualize sua tabela Q (e, portanto, o seu comportamento) *durante* uma batalha—ao contrário do que ocorria com métodos de busca local, onde o valor de preferência Q de cada ação era atualizado apenas ao final de cada episódio/batalha.

Controller.take_action: esta função deve consultar a tabela Q para o estado atual e verificar qual ação possui a maior preferência. Você deve implementar algum sistema de exploração, para que o agente consiga experimentar ações "novas", que não parecem necessariamente as melhores (de acordo com as preferências Q atuais, as quais podem ser incorretas). Um sistema de exploração eficiente é essencial para que o agente experimente diferentes tipos de comportamento em sua busca por um comportamento que o permita vencer as batalhas.

Controller.updateQ: nesta função você deve implementar a regra de atualização do Q-Learning.

Algumas especificações adicionais para sua implementação:

- Fica a seu critério como discretizar as *features* do estado—isto é, em quantos níveis cada *feature* será discretizada. Você deverá fazer testes e descobrir qual tipo de discretização permite com que o agente aprenda comportamentos mais efetivos.

- Fica a seu critério qual taxa de aprendizado α será utilizada. Você deve fazer testes e descobrir qual taxa de aprendizado permite com que o agente aprenda mais rapidamente.

- Fica a seu critério qual taxa de desconto γ será utilizada. Lembre-se que quanto maior γ (mais próximo de 1), maior o horizonte de tempo que o agente considera quando estiver aprendendo. Por exemplo: se $\gamma=0$ e o agente vencer uma batalha, ele irá aprender que apenas a ação executada imediatamente antes do término da batalha foi relevante para a sua vitória. Isso claramente não é verdade: o tiro da vitória pode ter ocorrido 40-50 passos antes do final da partida. O valor de γ, portanto, precisa ser alto o suficiente para permitir com que o agente aprenda esse tipo de padrão—isto é, de que as recompensas recebidas, como aquela dada no momento da vitória—podem ter sido causadas por ações executadas vários passos no passado.

- Fica a seu critério qual função de recompensa utilizar; isto é, quais recompensas você dará ao agente a cada momento. Quanto mais informativas forem as recompensas dadas a cada momento, mais rapidamente o agente descobrirá como se comportar em cada possível situação. Uma recompensa 0 em todos os estados, por exemplo, exceto durante a vitória (recompensa de +100) ou derrota (recompensa -100) não seria muito informativa, pois o agente apenas receberia *feedback* a respeito de seu comportamento ao *final* da batalha. O agente não teria, portanto, condições de, ainda durante a batalha, melhorar o seu comportamento. Caso você desse recompensas *durante* a batalha (p.ex., uma recompensa negativa caso o inimigo estivesse se aproximando demais; ou uma recompensa positiva caso o agente conseguisse mirar no inimigo; ou uma recompensa positiva por conseguir desviar de um tiro que se aproximava; etc), o comportamento do agente pode ser significativamente mais sofisticado, e o aprendizado, mais rápido. Fica a seu critério definir quais recompensas serão dadas em cada situação/estado.

- Fica a seu critério, também, qual método de exploração será utilizado; isto é, de que forma o seu agente irá escolher ações a cada momento. Conforme discutido em aula, o agente de aprendizado por reforço não deve sempre executar a ação com maior valor Q, pois isso poderia impedi-lo de experimentar ações ainda pouco testadas. Um método simples de exploração é conhecido como **Exploração ε-greedy:** ele escolhe a ação com maior valor Q em ε% das vezes, e em *(1-ε)%* das vezes executa uma ação aleatória. Você poderia também considerar um método de exploração que escolhesse ações de forma probabilística, proporcionalmente à qualidade Q de cada ação. Um destes métodos é a **Exploração de Boltzmann**, discutida em aula. O livro-texto principal da disciplina discute outras opções, tal como a exploração baseada em "curiosidade", de acordo com a qual o agente deve preferir executar ações que o levam a estados desconhecidos ou pouco visitados.

Saídas que devem ser geradas pelo seu programa:

O simulador utilizado nesse trabalho automaticamente imprime a performance do agente ao final de cada episódio. *Performance*, aqui, diz respeito à mesma métrica utilizada no Trabalho 1; isto é, valores entre 1 e +1, onde -1 corresponde ao caso extremo onde o agente é derrotado imediatamente, 0 em que de empate, e +1 ao caso extremo onde o agente vence a batalha imediatamente. Valores intermediários de performance, entre -1 e 0 e entre 0 e +1, correspondem, respectivamente, a derrotas e vitórias com performances intermediárias.

Você deverá gerar diversos gráficos onde o eixo x mostra *número do episódio de treinamento*, e o eixo y mostra a *performance do agente naquele episódio*. Gere gráficos deste tipo utilizando:

1. diferentes valores de taxa de aprendizado α (ao menos 3 valores, escolhidos no intervalo entre 0 e 1);
2. diferentes valores da taxa de desconto γ (ao menos 2 valores, escolhidos no intervalo entre 0 e 1);
3. diferentes estratégias de exploração. Gere ao menos 1 gráfico para a estratégia gulosa (isto é, a que sempre executa a ação com maior Q), e ao menos 2 gráficos avaliando uma estratégia de exploração escolhida pelo grupo, e teste-a com diferentes parâmetros. Por exemplo, você poderia gerar gráficos da performance da exploração ε-greedy para dois (ou mais) valores de ε; ou gráficos da performance da Exploração de Boltzmann para dois (ou mais) valores de temperatura T.
4. diferentes funções de recompensa (gere gráficos para ao menos 2 funções de recompensa escolhidas/ testadas pelo grupo).

Seu trabalho irá conter, portanto, no mínimo 10 gráficos, de acordo com os itens descritos acima.

Alguns pontos a serem discutidos no relatório:

Ao apresentar os gráficos mencionados acima, discuta:

1. qual o impacto dos níveis de discretização de *features* no comportamento aprendido pelo agente, e na velocidade do aprendizado? Como você determinou os melhores níveis de discretização?
2. qual o impacto da taxa de aprendizado α no aprendizado? Como você determinou o melhor valor a ser utilizado?
3. qual o impacto da taxa de desconto γ no aprendizado? Como você determinou o melhor valor a ser utilizado?
4. qual o impacto de diferentes estratégias de exploração no comportamento do seu agente e na velocidade de aprendizado?
5. como você escolheu os parâmetros do método de exploração escolhido (por exemplo, o valor de ε e/ou de T)? Você manteve esses valores constantes durante o aprendizado? Tentou diminuir o valor do parâmetro ao longo dos episódios? Ou aumentar? Qual foi o efeito disso na velocidade/qualidade do aprendizado?

Política de plágio: a mesma política de plágio utilizada no Trabalho #1 se aplica a este projeto.