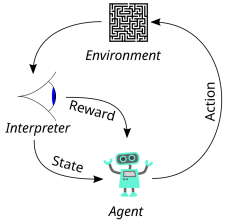
Parte IV: Aprendizado de Máquina.

Prof. Fabiano Araujo Soares, Dr. / FGA 0221 - Inteligência Artificial

Universidade de Brasília

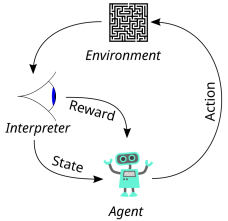
2025

Aprendizado Por Reforço



Imagina que você está ensinando um cachorro a buscar a bolinha. Você não diz "senta aqui" ou "vai ali". Você apenas dá um biscoito quando ele acerta e fica quieto quando ele erra.

Aprendizado Por Reforço



Imagina que você está ensinando um cachorro a buscar a bolinha. Você não diz "senta aqui" ou "vai ali". Você apenas dá um biscoito quando ele acerta e fica quieto quando ele erra.

Como o cachorro aprende sozinho qual é o caminho certo só com essas recompensas/punições? Essa é **EXATAMENTE** a ideia do Aprendizado por Reforço!

Por que Aprendizado por Reforço?

- Usado quando é possível descrever o que é uma ação "boa" ou uma ação "ruim", mas não se tem exemplos o suficiente para explorar as combinações possíveis (e.g.: ensinar um agente a jogar xadrez – aproximadamente 1E8 jogos de mestres e aproximadamente 1E40 posições possíveis);
- No aprendizado por reforço o agente interage com o ambiente e periodicamente recebe reforços positivos ou negativos, por exemplo, no xadrez a recompensa é 1 para ganhar, 0 para perder e 1/2 para um empate.
- Imagine jogar um novo jogo cujas regras você não conhece; depois de cem ou mais movimentos, o árbitro lhe diz "Você venceu" ou "Você perdeu". Isso é o aprendizado por reforço em poucas palavras.

Ponto de vista do Engenheiro

- Fornecer uma recompensa ao agente geralmente é muito mais fácil do que fornecer exemplos rotulados de como se comportar;
- Recompensas são geralmente concisas e fáceis de especificar;
- As recompensas de vitória/perda para xadrez por jogo (e não por jogada) são chamadas de **recompensas esparsas**.
- Em jogos como tênis, podemos fornecer recompensas adicionais para cada ponto ganho;
- Nas corridas de carros, poderíamos recompensar o agente por ganhar posições e progredir na pista na direção certa;
- Essas recompensas intermediárias tornam o aprendizado mais fácil.

Categorias

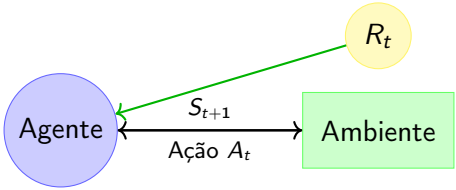
- **Aprendizado por reforço baseado em modelo**: Nessa abordagem, o agente usa um modelo de transição do ambiente para ajudar a interpretar os sinais de recompensa e tomar decisões sobre como agir.
- **Aprendizado por reforço sem modelo**: Nessas abordagens, o agente não conhece nem aprende um modelo de transição para o ambiente. Em vez disso, ele aprende uma representação mais direta de como se comportar. Existem dois tipos:
  - **Aprendizagem de utilidade de ação**: O algoritmo mais popular é o Q-learning,
  - **Pesquisa de política**: o agente aprende uma política $\pi(s)$ que mapeia diretamente os estados para ações.

Aprendizado por Reforço - Intuição

Analogia do Cachorro:

- Ensina sem dizer o caminho exato
- Dá biscoito quando acerta
- Cachorro **experimenta, erra, aprende sozinho**

É exatamente assim que funciona o RL (*Reinforcement Learning*)!



O que é Aprendizado por Reforço?

Definição formal (MDP - Markov Decision Process):

- **Estado** S_t: onde estou agora
- **Ação** A_t: o que faço
- **Recompensa** R_t: ganho/perco pontos
- **Política** π: minha estratégia
- **Valor** V: quanto vou ganhar no futuro

Objetivo: $\max \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_t\right]$ onde $\gamma \in [0, 1)$ é o **fator de desconto** (futuro importa menos)

Definição: O que é MDP? (Processo de Decisão de Markov)

MDP = "Markov Decision Process"

É o "mundo matemático"onde o RL acontece!

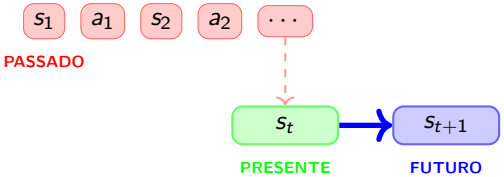
5 Componentes principais:

1. $\mathcal{S}$: Espaço de **Estados** - onde você está
2. $\mathcal{A}$: Espaço de **Ações** - o que você pode fazer
3. $P(s'|s,a)$: Probabilidade de **Transição**
4. $R(s,a)$: **Recompensa** - quanto você ganha/perde
5. $\gamma \in [0,1)$: **Fator de Desconto**

A Propriedade Markov: Memória Curta

Markov = "Memória curta"

O futuro depende APENAS do PRESENTE, não do passado!



Interpretação:

[X] Não importa: Como chegou em s_t (histórico)

[OK] Importa: Onde você está AGORA (s_t)

[!] Resultado: s_{t+1} depende APENAS de s_t

Equação de Markov: A Matemática

A equação que define tudo:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_1, a_1, s_2, a_2, \ldots, s_t, a_t)$$

O que isso significa?

LADO ESQUERDO
"Você precisa saber:"

- Onde estou (s_t)
- O que faço (a_t)

LADO DIREITO
"Você NÃO precisa saber:"

- Onde passei
- O que fiz antes
- Toda historia!

Conclusao Principal

Olhe apenas para o PRESENTE! O passado é IRRELEVANTE.

Onde é usado?

Jogos:

- AlphaGo (DeepMind) ✓ Go
- AlphaStar (StarCraft II) ✓
- Dota 2, Atari, Xadrez

Robótica:

- Braços robóticos
- Drones autônomos
- Carros autônomos

Negócios:

- Otimização de preços
- Gerenciamento de estoque
- Marketing dinâmico

Finanças:

- Trading algorítmico
- Otimização de Portfolio

Vantagens do RL

- **Não precisa de rótulos**: aprende sozinho experimentando
- **Otimização sequencial**: pensa no futuro, não só no agora
- **Ambiente dinâmico**: adapta-se a mudanças
- ***Super-human performance***: vence humanos em jogos complexos
- **Generaliza bem**: aprende estratégias, não regras fixas

Exemplo: AlphaGo venceu o campeão mundial de Go!

Desvantagens do RL

- ***Sample inefficient***: milhões de tentativas necessárias
- ***Exploration vs Exploitation***: equilibrar risco/benefício
- **Instável**: recompensas esparsas $\rightarrow$ difícil convergir
- **Debug difícil**: "caixa-preta" comportamental
- **Custo computacional**: GPU/TPU por dias

Comparação

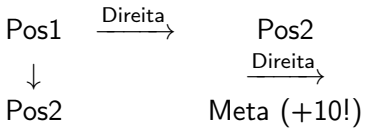
Supervised Learning: 1M exemplos rotulados $\approx$ 1B tentativas RL

Q-Learning: Ideia Central

O que é $Q(s, a)$?

$Q(s, a)$ = "Quanto ganho se fizer ação a no estado s?"

Exemplo: Robô em uma Sala



Valores Q:

- $Q(\text{Pos1, Direita}) = +2$
- $Q(\text{Pos2, Direita}) = +10$
- $Q(\text{Meta, Parar}) = +100$

Tabela Q: Dicionário de Recompensas

Tabela Q = Todos os valores organizados

Exemplo com 3 posições:

Estado	Esquerda	Direita	Parar
Pos1	-5	+2	-1
Pos2	+3	+8	0
Pos3	-1	-1	+100

Como usar a tabela?

- Estou em Pos1: Melhor ação? Direita (+2)
- Estou em Pos3: Melhor ação? Parar (+100)

Regras: SEMPRE escolha maior Q!

Atualizando Tabela Q: Regra de Bellman

Fórmula de atualização:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Explicação dos termos:

- r_t: Ganho imediato
- $\gamma \max Q(s_{t+1}, a)$: Melhor futuro
- α: Velocidade de aprendizado

Exemplo prático:

- Q(Pos1,Direita) $= 0$ (inicial)
- Ganhei $r = +2$, fui para Pos2
- max Q(Pos2,*) $= +8$
- $\gamma = 0.9$, $\alpha = 0.1$
- Novo $Q = 0 + 0.1[2 + 0.9 \cdot 8 - 0] = 0.92$

Limitações do Q-Learning

Problema 1: Curse of Dimensionality

"Tabela cresce EXPONENCIAL com estados!"

Problema	Estados	Tabela?
Grid 3x3	9	OK
CartPole	10.000	OK
Xadrez	10^{47}	IMPOSSÍVEL
Atari	Infinitos	IMPOSSÍVEL

Problema 2: Estados Contínuos

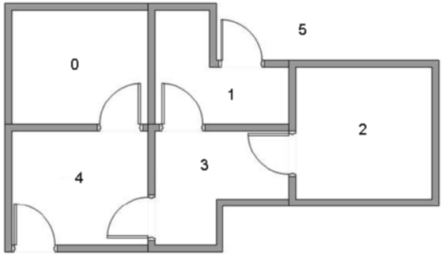
"Mundo real = INFINITOS estados!"

- Ângulo: 0.0, 0.01, 0.011, 0.0111... (infinito)
- Tabela não cabe valores contínuos

- Solução: Deep Q-Network (DQN)

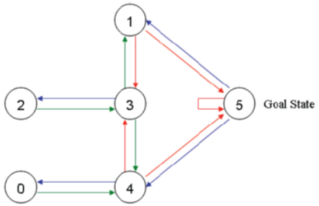
Exemplo: Saída rápida da Casa

- Vamos ver o exemplo discutido em:
  `https://blog.csdn.net/itplus/article/details/9361915`
- Vamos considerar um caso simples onde um agente está em uma casa com 5 áreas (sendo a área 5 a área externa da casa) cada área é ligada a outras por portas:



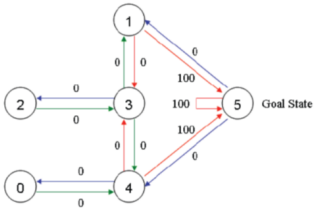
Exemplo: Saída rápida da Casa

- O objetivo do agente é, iniciando em uma das áreas da casa, chegar a região externa (região 5) em um número mínimo de movimentos.
- Podemos representar os estados do problema como um grafo:



Exemplo: Saída rápida da Casa

- Vamos associar a movimentações entre áreas a uma pontuação, a pontuação que leva a área externa será 100 as demais será 0:

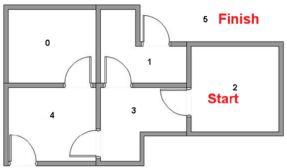


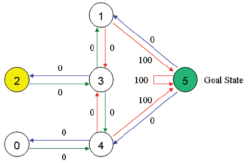
Exemplo: Saída rápida da Casa

- Em Q-learning o objetivo e alcançar o estado com a maior pontuação possível, dessa forma, se o agente chegar a região 5 ele ficará lá para sempre (**absorbing goal**);
- Imagine que o agente não tem ideia da planta da casa e, portanto, não sabe quais sequências de portas leva para fora da casa;
- Nesse problema cada área da casa é um "estado" e o movimento do agente de uma área da casa para outra é uma ação;

Exemplo: Saída rápida da Casa

- Digamos que o agente inicie na área 2 e deseja chegar na área 5:





Exemplo: Saída rápida da Casa

- Vamos construir uma matriz R com os estados e valores de recompensa;
- Para representar transições impossíveis (onde não há portas ligando as áreas), vamos utilizar o valor -1;

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} \multicolumn{6}{c}{\text{Action}} \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

Exemplo: Saída rápida da Casa

- Agora vamos construir uma outra matriz Q que representará o que o agente aprendeu;
- As linhas da matriz Q Representam o estado atual do agente e as colunas representam as possíveis ações que levam ao próximo estado;
- Como o agente inicia sem saber nada, a matriz Q inicial é uma matriz de zeros;
- Para simplificar, vamos considerar que o número de estados é conhecido e igual a 6;
- Se essa simplificação não fosse possível, a matriz Q iniciaria com um elemento e a cada estado novo descoberto o número de linhas e colunas aumentaria;
- A regra de transição do Q-learning é simples:

$$Q(s, a) = R(s, a) + \gamma Max[Q(s + 1, a_1, a_2, a_N)]$$

- Ou seja, o valor de um elemento da matriz Q é igual a soma do valor correspondente na matriz R e o fator de aprendizado γ multiplicado pelo máximo valor de Q para todas as possíveis ações no próximo estado.

Exemplo: Saída rápida da Casa

- O nosso agente vai aprender experimentando caminhos entre o ponto de partida e a chagada;
- Chamaremos de episódio cada vez que o agente iniciar no ponto de partida e chegar a chegada;
- O nosso algoritmos para o Q-learning será o seguinte:

1. Ajuste o parâmetro Gama e a matriz R de recompensas;
2. Inicie a matriz Q com zeros;
3. Para cada episódio:
    1. Selecione o ponto de partida de forma aleatória;
    2. Enquanto a chegada não foi alcançada faça:
        1. Selecione uma das possíveis ações para o estado atual;
        2. Considerando essa ação considere o que aconteceria no novo estado;
        3. Calcule o valor Q máximo para as ações desse novo estado baseado em todas as possíveis ações;
        4. Calcule Q(state, action)
        5. Atualize o estado novo como estado atual;

Exemplo: Saída rápida da Casa

- O parâmetro γ tem valor entre 0 e 1, se γ for próximo de 0 o agente tenderá a considerar apenas recompensas imediatas, se γ for próximo de 1 o agente considerará recompensas futuras;
- Para utilizar a matriz Q o agente deve fazer:
  1. Atualize o estado atual = estado inicial;
  2. Para o estado atual, encontre a ação com o valor mais alto de Q;
  3. Atualize o estado atual = estado novo (para o qual Q é maior);
  4. Repita os passos 2 e 3 até estado atual = chegada.

Exemplo: Saída rápida da Casa

- Para compreender o processo, vamos seguir alguns episódios manualmente. $\gamma = 0,8$ e o ponto de partida é a área 1. Só há duas possíveis ações ir para a área 3 ou para a área 5:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} & & & \text{Action} & & \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

Exemplo: Saída rápida da Casa

- Digamos que o agente decida ir para a área 5;
- Na área 5 o agente pode pode permanecer na área 5, ir para a área 1 ou ir para a área 4. Temos então:

$$Q(1,5) = R(1,5) + 0,8 \times Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0,8 \times 0 = 100;$$

- O estado 5 se torna então o estado atual e a matriz Q é atualizada:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Exemplo: Saída rápida da Casa

- Para o próximo episódio iniciamos no estado 3. Temos três possíveis ações, ir para o estado 1, 2 ou 4. Por escolha aleatória vamos para o estado 1;
- No estado 1, o agente tem duas ações possíveis: ir para o estado 3 ou ir para o estado 5;
- Vamos novamente calcular o valor de Q:

$$Q(3,1) = R(3,1) + 0,8 \times Max[Q(1,3), Q(1,5)] = 0 + 0,8 \times Max(0,100) = 80.$$

- A matriz Q é então atualizada:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Exemplo: Saída rápida da Casa

- Fazendo uma série de interações, o nosso agente finalmente chegará a matriz Q:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

Matriz Q

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

Matriz Q Normalizada

Por que Deep Q-Network (DQN)?

Problema do Q-Learning:
Tabela Q é impossível para problemas GRANDES!

CartPole: 10.000 estados (OK) vs Atari: 256^{21168} (IMPOSSÍVEL!)

Solução: Use Rede Neural!
Em vez de buscar em tabela gigante, use rede que APROXIMA:

$$Q(s, a; \theta) = \text{Rede Neural}$$

Vantagem: Rede comprime 21.168 pixels em poucos parâmetros!

Arquitetura da Rede DQN

Como a Rede Neural Q funciona?

Entrada: Imagem 84x84 do jogo (21.168 pixels)

Camadas:

- 3 Camadas Convolucionais (extraem características visuais)
- 2 Camadas Densas (raciocínio estratégico)

Saída: Q-value para CADA ação

Ação	Q-value
Esquerda	2.5
Direita	8.3
Pular	-1.2
Parado	0.1

Escolhe: Maior Q-value (Direita = 8.3)

Inovação 1: Experience Replay

Problema: Aprender de forma sequencial (s1, s2, s3...) cria vieses

Solução: Guardar experiencias e aprender aleatoriamente!

3 passos:

1. Guardar cada (s, a, r, s') em BUFFER
2. Acumular 1 milhão de experiencias
3. Treinar com 32 opções aleatórias do buffer

Analogia: Estudar
Errado: Sequencial (exercícios 1, 2, 3...)
Certo: Aleatório (exercícios 50, 10, 200...)

Resultado: Aprendizado muito mais estavel!

Inovação 2: Target Network

Problema: Treinar Q enquanto usa Q causa instabilidade

Solução: Duas redes iguais!

- Online Network: Atualiza RAPIDO (treino)
- Target Network: Atualiza LENTAMENTE (alvo)

A cada 10.000 steps: copia pesos online para target

Analogia: Atirador em alvo movel
Errado: Alvo se move rapidamente
Certo: Alvo se move lentamente

Resultado: Convergência muito mais rapida!

Inovação 3: Epsilon-Greedy

Problema: Explorar (novo) vs Exploitar (melhor)?

Solução: Misture os dois!

Com probabilidade $1 - \epsilon$ (ex: 95%):
Escolhe MELHOR ação: $a = \arg\max_a Q(s, a)$

Com probabilidade ϵ (ex: 5%):
Escolhe ALEATORIA: $a = \text{random}()$

Beneficio:

- 95% explora o melhor
- 5% descobre estratégias novas

ϵ começa em 1.0 e diminui ao longo do treinamento
(mais exploração no inicio, mais explotação no fim)

Algoritmo DQN Completo

Pseudocodigo:

1. Inicializar Online Network θ, Target Network $\theta^- = \theta$
2. Inicializar Buffer vazio
3. Para cada episodio:
   Recebe estado inicial s_0
   Para cada step:
      Com probabilidade ϵ: ação aleatoria
      Se não: $a = \arg\max_a Q(s, a; \theta)$
      Executar, receber (s', r)
      Guardar (s, a, r, s') no Buffer
      Se Buffer tem N experiencias:
         Amostra 32 aleatoias
         Loss: $L = (r + \gamma \max_a Q(s', a; \theta^-) - Q(s, a; \theta))^2$
         Atualiza θ com backpropagation
      Se step % 10000 == 0: $\theta^- \leftarrow \theta$

Comparação: Q-Learning vs DQN

Aspecto	Q-Learning	DQN
Armazenamento	Tabela	Rede Neural
Tamanho Estados	Pequeno	GRANDE
Memoria	exp	Buffer $\approx$ 1M
Estabilidade	Baixa	Alta
Aplicações	Simples	Complexas
Tempo de treino	Minutos	Semanas
Hardware	CPU	GPU

Atari Performance:
Q-Learning: Impossivel (tabela não factível)
DQN: Superou humano em 49/57 jogos!

DQN = Revolução em Deep RL!

Problema com Q-Learning (Depende de Valores)

Q-Learning aprende VALORES, não POLÍTICA

Problema 1: Estados/Ações CONTINUAS

- Robô com braço: ângulos infinitos (0.0, 0.01, 0.001...)
- Carro autônomo: velocidade contínua
- Q-Learning não consegue (precisa de tabela discreta)

Problema 2: Ação ótima pode ser MUITO diferente de ações ruins

- Se todas ações tem Q baixo, qual escolher?
- Difícil encontrar o máximo

Nova Abordagem: Aprender a POLÍTICA diretamente!

Em vez de: Aprender Q(s,a), depois escolher max
Aprenda: $\pi(a|s)$ = Probabilidade de cada ação

Policy Gradient: Ideia Central

O que é $\pi_\theta(a|s)$?

Probabilidade de escolher ação a no estado s

Exemplo: Robô com Braço

Estado: Bola a 45 graus
Política pode ser:

- Mover para 46 graus: 70% de chance
- Mover para 44 graus: 20% de chance
- Não mover: 10% de chance

Objetivo: Aprender que 46 graus é MUITO BOM

Aumentar probabilidade para 95%

Como? Usar Gradiente Ascendente!

θ = parametros da politica (rede neural)
Atualizar θ para AUMENTAR recompensa futura

Regra de Atualizacao: Policy Gradient

Fórmula (REINFORCE):

$$\nabla J(\theta) = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t\right]$$

Descomposição:

Parte 1: $\nabla_\theta \log \pi_\theta(a_t|s_t)$

- Gradiente do log da probabilidade
- "Para qual direção aumentar essa ação?"

Parte 2: $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$

- Retorno total descontado a partir de t
- "Quão boa foi essa trajetoria?"

Multiplicação:
Aumentar probabilidade se G_t foi ALTA (boa trajetoria)
Diminuir probabilidade se G_t foi BAIXA (trajetoria ruim)

Interpretação: "Faça mais do que funcionou!"

Vantagens e Desvantagens do Policy Gradient

Vantagens:

- Estados CONTÍNUOS: OK! (qualquer valor de entrada)
- Ações CONTÍNUAS: OK! (saida é distribuição continua)
- Política direta: Aprende estratégia, não valores
- Convergência garantida (em muitos casos)

Desvantagens:

- ALTA VARIÂNCIA: Pequenas mudanças causam grandes variações
- Converge LENTAMENTE: Precisa de muitas amostras
- Exploração vs Explotação manual: Precisa definir

Vantagens e Desvantagens do Policy Gradient

Comparação com Q-Learning:

Q-Learning: Bom com estados discretos, ruim com contínuos
Policy Gradient: Bom com estados contínuos, menos eficiente que Q

Quando usar?

- Robô com ações contínuas: Policy Gradient
- Jogos de Atari (discreto): Q-Learning (DQN)

Exemplo Numérico: Policy Gradient

Cenário: Robô Movimentando Braço

Estado: Bola a 45 graus
Política inicial (uniforme):

- Mover 46 graus: 33%
- Mover 44 graus: 33%
- Não mover: 33%

Episodio 1: Escolheu 46 graus

- Recompensa: +100 (pegou a bola!)
- $G_t = +100$
- Atualizar: aumenta probabilidade de 46 graus

Episodio 2: Escolheu 44 graus (por acaso)

- Recompensa: -10 (errou!)
- $G_t = -10$
- Atualizar: diminui probabilidade de 44 graus

Depois de 1000 episodios:

- Mover 46 graus: 90%
- Mover 44 graus: 5%
- Nao mover: 5%

Resultado: Robo aprendeu a melhor ação!

Variantes: REINFORCE vs Actor-Critic

REINFORCE (básico):

- Usa apenas G_t (retorno real)
- Simples de implementar
- Variância MUITO alta

Actor-Critic (melhorado):

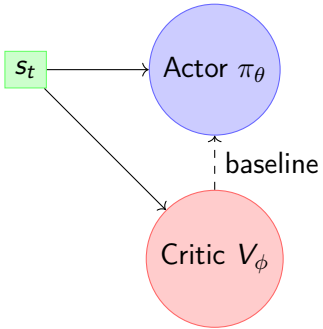
- Usa G_t e $V(s_t)$ (política e valor)
- Reduz variância significativamente
- Precisa aprender $V(s_t)$ (crítico)
- Muito mais estável

PPO, A3C, TRPO (estado da arte):

- Variantes de Actor-Critic
- Adiciona restrições de divergência KL
- Usado em aplicações reais (robôs, jogos)

Actor-Critic (PPO, A2C, A3C)

Híbrido poderoso: Policy (π) + Value (V)



Advantage:

$$A_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Update:

$$\nabla J = \nabla \log \pi(a_t|s_t) \cdot A_t$$

PPO (OpenAI): Estado da arte hoje!

Referências

- Russell, S., Norvig, P., "Artificial Intelligence: A Modern Approach", 4th ed., Person, 2022.
- Muller, A. C., Guido, S., "Introduction to Machine Learning with Python A Guide for Data Scientists", O'Reilly, 2017.
- Watt, J., "A Painless Q-learning Tutorial", 2020.

Obrigado!

E-mail: fabianosoares@unb.br

LinkedIn: https://www.linkedin.com/in/fabiano-soares-06b6a821a/

Site do curso: https://www.fabianosoares.eng.br/fga0221-inteligência-artificial