CEFET/RJ

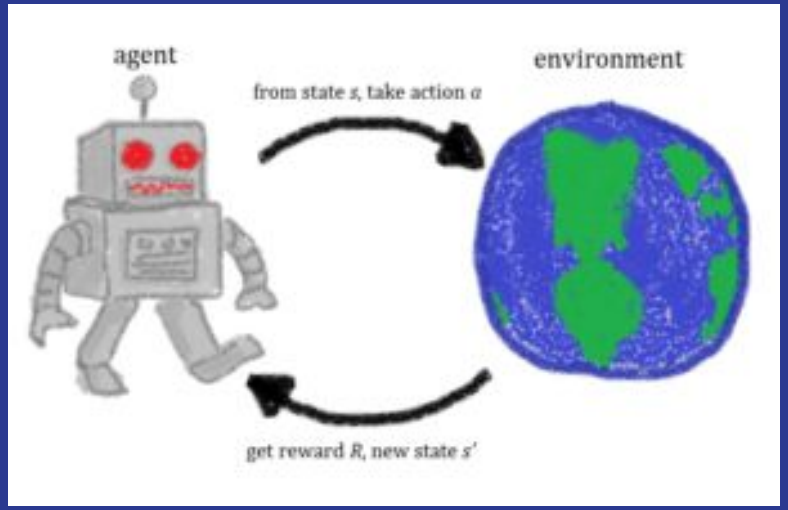
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GCC1734 - **INTELIGÊNCIA ARTIFICIAL**

Eduardo Bezerra (CEFET/RJ)
ebezerra@cefet-rj.br

Essa apresentação é uma tradução e/ou adaptação do material usado no curso CS188 (Introduction to Artificial Intelligence) https://inst.eecs.berkeley.edu/~cs188

APRENDIZADO POR REFORÇO II



Visão geral

- Revisão
- Q-learning com aproximador linear
- Exploração versus aproveitamento

Revisão

Aprendizado por Reforço

- Ainda temos uma PDM:
  - conjunto de estados s $\in$ S
  - conjunto de ações (por estado) A
  - modelo de transições T(s,a,s')
  - função recompensa R(s,a,s')
- Ainda procuramos uma política $\pi(s)$
- Novidade (complicador): agente não conhece nem T nem R, portanto deve experimentar ações para aprender.

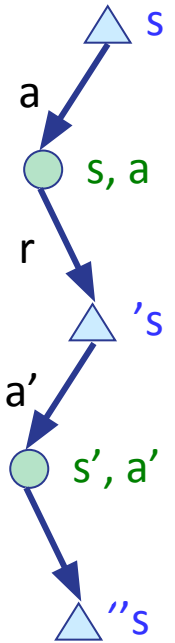
Q-learning

- Treinamento acontece por meio de episódios

$$(s, a, r, s', a', r', s'', a'', r'', s'''' \ldots)$$

- Agente atualiza suas estimativas a cada transição (s, a, r, s') que experimenta.

- No decorrer do treinamento, essas atualizações se assemelham cada vez mais às atualizações de Bellman



Q-learning tabular

- Atualiza iterativamente q-valor para cada q-estado.
- Para isso, o agente computa uma média móvel durante o aprendizado
  - Agente recebe (experimenta) várias transições (i.e., amostras) da forma (s, a, r, s')
  - Cada amostra sugere um valor para um estado-ação (s, a):

    $$Q(s,a) \approx r + \gamma \max_{a'} Q(s', a')$$

  - Mas, é desejável tirar a média sobre os resultados obtidos em (s, a) (Por que?)
  - Para isso, o Q-learning computa uma média móvel:

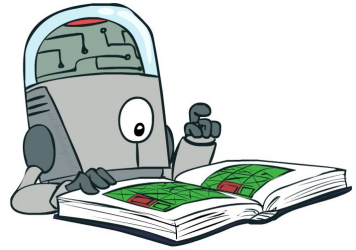
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s', a')\right]$$

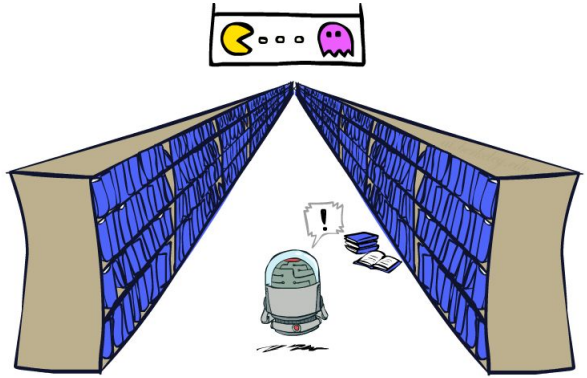
Q-learning tabular

- Vantagens:
  - Tem garantia teórica de encontrar os q-valores adequados.
  - De fácil implementação.
- Desvantagens:
  - **Seu uso é impraticável em problemas com muitos estados e/ou ações.**
  - **Não permite que o agente aprenda a generalizar entre estados.**

Tamanho do espaços de estados

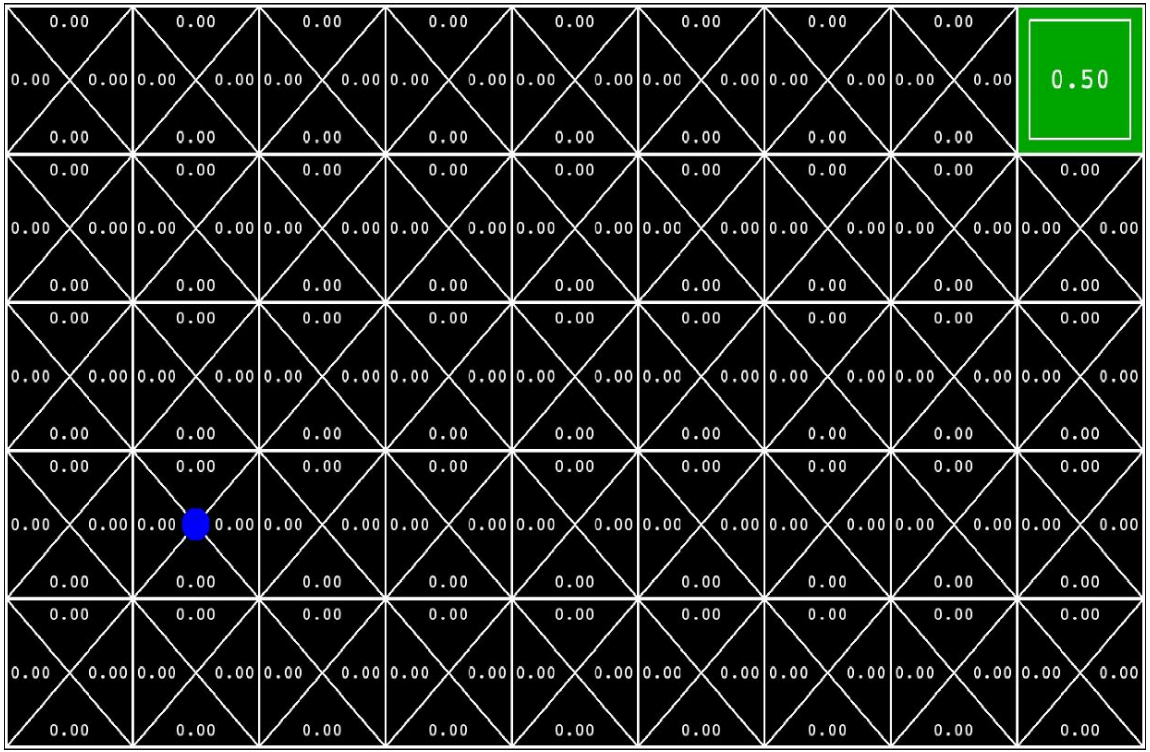
- O Q-Learning tabular mantém uma tabela com todos os q-valores.
- Em situações realísticas, possivelmente não há como o agente aprender os valores de todos os estados!
- Motivos:
  - Muitos estados para visitar durante o treinamento;
  - Muitos estados para manter a tabela em memória.



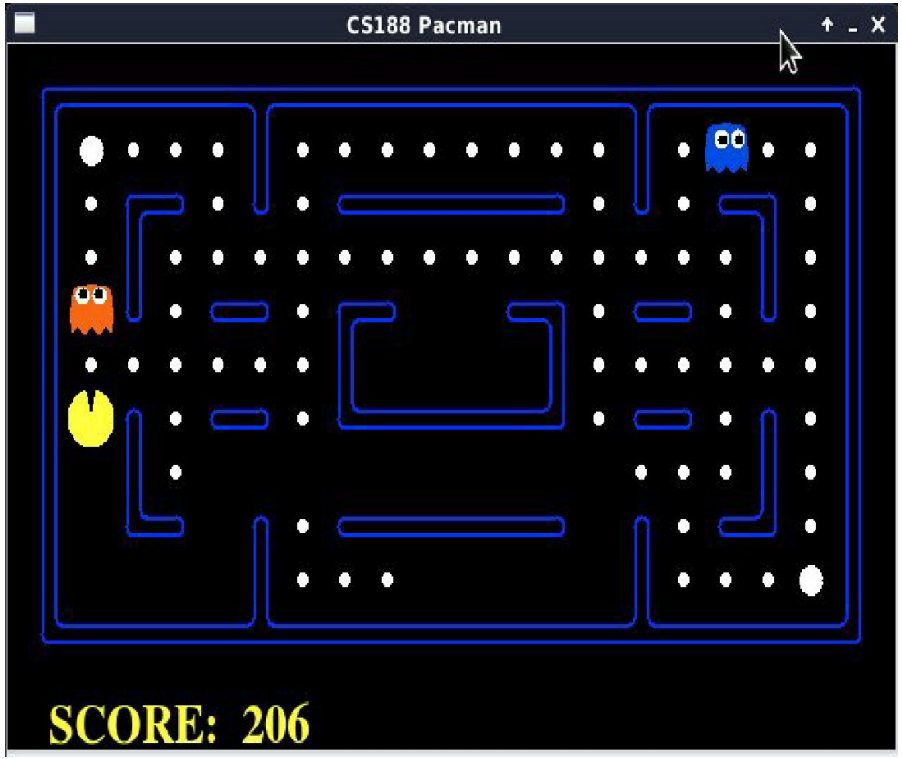


[demo – RL pacman]

Tamanho do espaços de estados



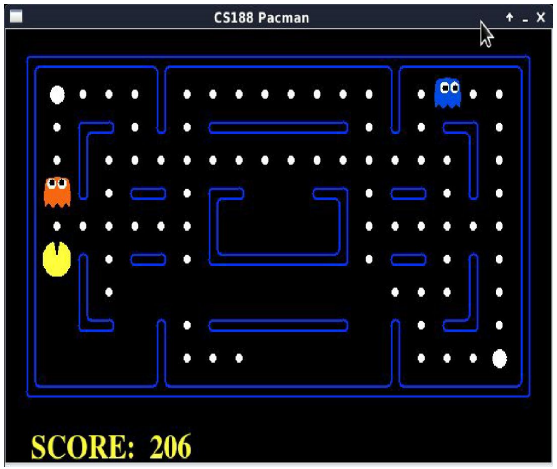
Tamanho do espaços de estados



Quantos estados?

Tamanho do espaços de estados

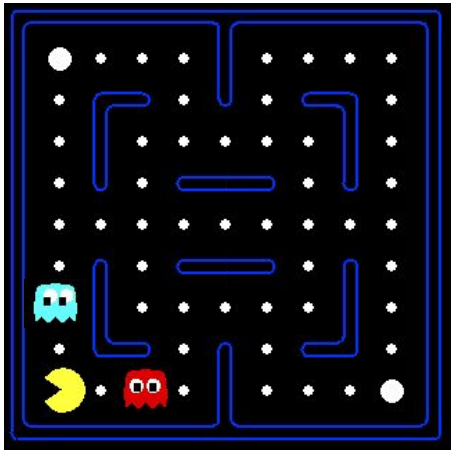
- Localizações para o PacMan: 107.
- Localizações dos fantasmas: 107^2.
- Localizações das comidas: 2^{104}.
  - Nem todas viáveis porque o PacMan não pode pular.
- Pílulas (*power pellets*): 4 possibilidades.
- Se cada fantasma está "assustado": 4 possibilidades (ignorando o *timer*).



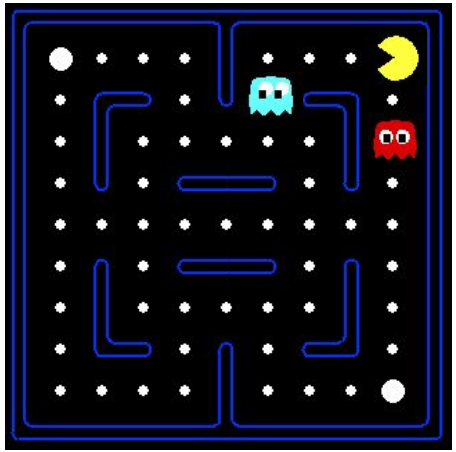
$107^3 * 4^2 = 19.600.688$ estados (sem considerar as comidas!)

Incapacidade de generalização

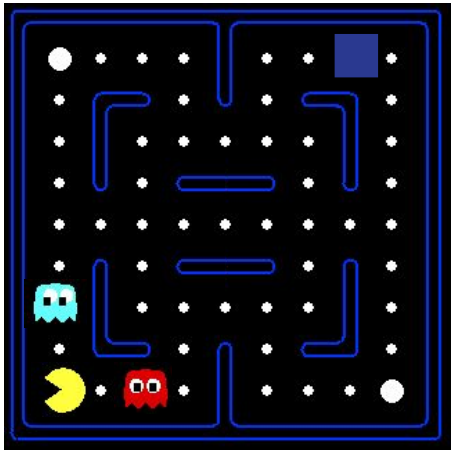
Considere que o agente descobre por experiência que esse estado é ruim:



No Q-learning tabular, isso não diz nada ao agente acerca desse estado:



Ou mesmo desse!



[Demo: Q-learning – pacman – tiny – watch all (L11D5)]
[Demo: Q-learning – pacman – tiny – silent train (L11D6)]
[Demo: Q-learning – pacman – tricky – watch all (L11D7)]

Q-learning com aproximação linear

Q-learning com aproximação linear

- Como alternativa à versão tabular, vamos projetar uma versão do Q-learning na qual o agente:
  - aprende (a partir de experiências) o suficiente visitando um número relativamente pequeno de estados-ações durante o treinamento;
  - generaliza esse aprendizado para situações (estados-ações) novas, mas similares às encontrados durante o treinamento.
- Essas são ideias fundamentais na área da IA denominada Aprendizado de Máquina (*Machine Learning*).

Q-learning com aproximação linear

- No algoritmo Q-Learning tabular, o objetivo é aprender as entradas da q-tabela.
  - i.e., aprender o valor de cada estado-ação.
- No algoritmo Q-learning com aproximação linear, o objetivo é aprender um vetor de coeficientes.
  - Uma vez aprendido, esse vetor pode ser usado para obter uma aproximação linear da Q-função.
- Vetor de coeficientes:

$$[w_1, w_2, \ldots, w_n]$$

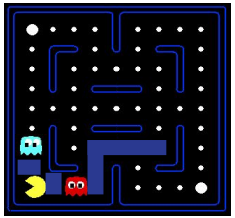
Características (*features*)

- Cada estado-ação usando um conjunto de características.
- Uma característica é uma função que mapeia cada estado-ação para um número real.
  - A ideia é que o vetor de característica capture propriedades importantes daquele estado-ação.
- Vetor de características:

$$[f_1, f_2, \ldots, f_n]$$

Características (*features*) – exemplo

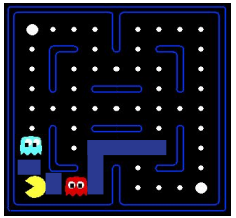
- Considere o seguinte problema no contexto do PacMan:



- Nesse problema, podem ser definidas várias características relevantes, dentre elas:
  - $f_{GST}(s, a)$: inverso da distância ao fantasma mais próximo, após o agente executar a ação a no estação s.
  - $f_{DOT}(s, a)$: inverso da distância à comida mais próxima , após o agente executar a ação a no estação s

Características (*features*) – exemplo

- Considere o seguinte problema no contexto do PacMan:



s

$f_{DOT}(s, \text{NORTH}) = 0.5$

$f_{GST}(s, \text{NORTH}) = 1.0$

Aproximação linear

- Uma vez definidas as características relevantes para um problema, elas são combinadas por meio de uma função linear para computar o valor de um estado-ação (s, a):

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a) \qquad f_i : S \times A \to \Re$$

- É também possível descrever um estado s por meio de características:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \qquad f_i : S \to \Re$$

Aproximação linear – exemplo

- Considere novamente as duas características definidas anteriormente:
  - $f_{GST}(s, a)$: inverso da distância ao fantasma mais próximo, após o agente executar a ação a no estação s.
  - $f_{DOT}(s, a)$: inverso da distância à comida mais próxima , após o agente executar a ação a no estação s.

Aproximação linear – exemplo (cont.)

- Então a aproximação linear da Q-função tem a seguinte forma:

$$Q(s,a) = w_{GST} \times f_{GST}(s,a) + w_{DOT} \times f_{DOT}(s,a)$$

- Suponha também que $[w_{GST}, w_{DOT}] = [+4.0, -1.0]$. Então:

$$Q(s,a) = 4f_{GST}(s,a) - f_{DOT}(s,a)$$

s

$f_{DOT}(s, \text{NORTH}) = 0.5$

$f_{GST}(s, \text{NORTH}) = 1.0$

$Q(s, \text{NORTH}) = +1$

Aprendizado

- Repare que cada característica f_i é multiplicada por um coeficiente (peso) que codifica a importância de f_i:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- No Q-learning com aproximação linear, o objetivo do treinamento é aprender o vetor de coeficientes:

$$[w_1, w_2, \ldots, w_n]$$

$$w_i \in \Re$$

Aprendizado – algoritmo

- Q-learning com aproximação linear:

transition $= (s, a, r, s')$

difference $= \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s, a)$

$$Q(s, a) \leftarrow Q(s, a) + \alpha \, [\text{difference}]$$

Regra de atualização no Q-learning tabular

$$w_i \leftarrow w_i + \alpha \, [\text{difference}] \, f_i(s, a)$$

Regra de atualização no Q-learning com função linear

Aprendizado – algoritmo

- Q-learning com aproximação linear:

$$\text{transition} = (s, a, r, s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s, a)$$

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s, a)$$

- Interpretação intuitiva: expressão acima atualiza os pesos apenas das características ativas em um dado estado/ação.
  - e.g., se algo ruim acontece inesperadamente, “penaliza” as características que estavam ativas: evitar todos os estados com essas características.
- Justificativa formal: mínimos quadrados

Aprendizado – exemplo

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s, a)$$

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s, a)$$

$$Q(s, a) = 4.0 f_{DOT}(s, a) - 1.0 f_{GST}(s, a)$$

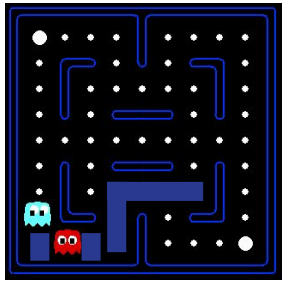
s 

$f_{DOT}(s, \text{NORTH}) = 0.5$

$f_{GST}(s, \text{NORTH}) = 1.0$

$a = \text{NORTH}$

$r = -500$

s' 

$Q(s, \text{NORTH}) = +1$

$Q(s', \cdot) = 0$

$$r + \gamma \max_{a'} Q(s', a') = -500 + 0$$

$\text{difference} = -501$

$$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$$

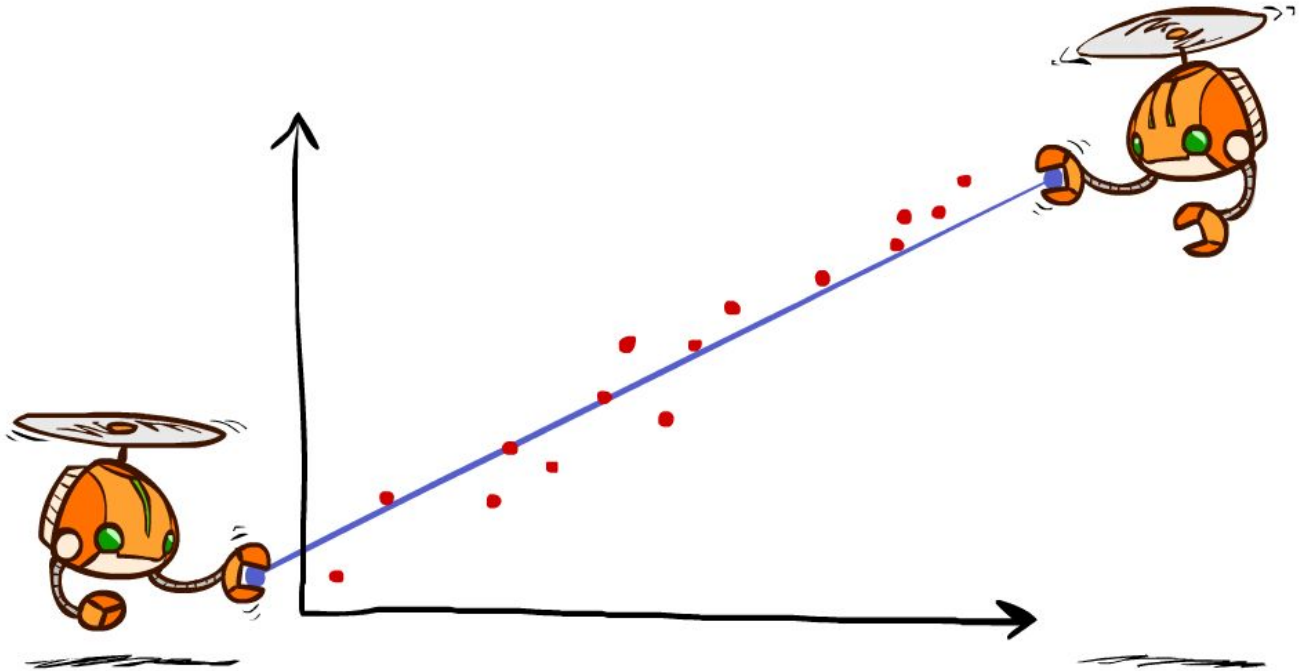
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

$$Q(s, a) = 3.0 f_{DOT}(s, a) - 3.0 f_{GST}(s, a)$$

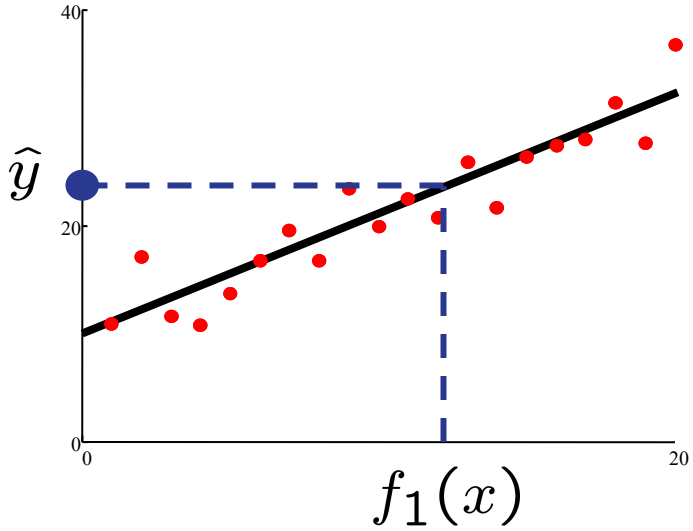
Q-learning com aprox. linear – análise

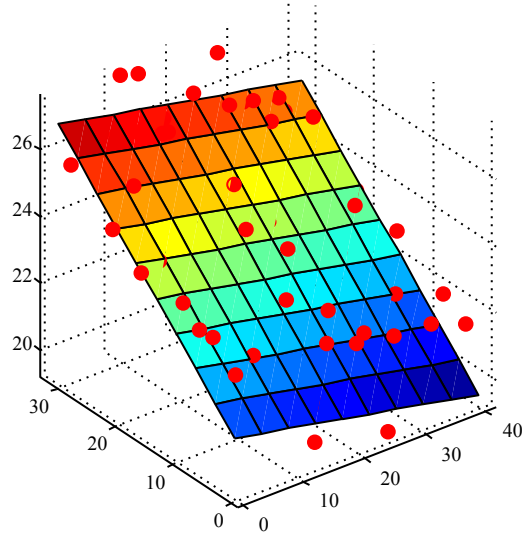
- Vantagens:
  - A experiência do agente é resumida em um conjunto pequeno de números (i.e., o vetor de pesos $[w_1, w_2, \ldots, w_n]$).
  - Pode lidar com espaços de estados contínuos!
- Desvantagens:
  - Pode haver estados que compartilham características, mas efetivamente têm valores bastante diferentes!
  - Requer o projeto de características (normalmente feito manualmente).
  - A Q-função do problema pode não ser adequadamente aproximada por uma função linear.

Q-learning e Mínimos Quadrados (Least Squares)



Aproximação linear: regressão





Predição (em 1D):

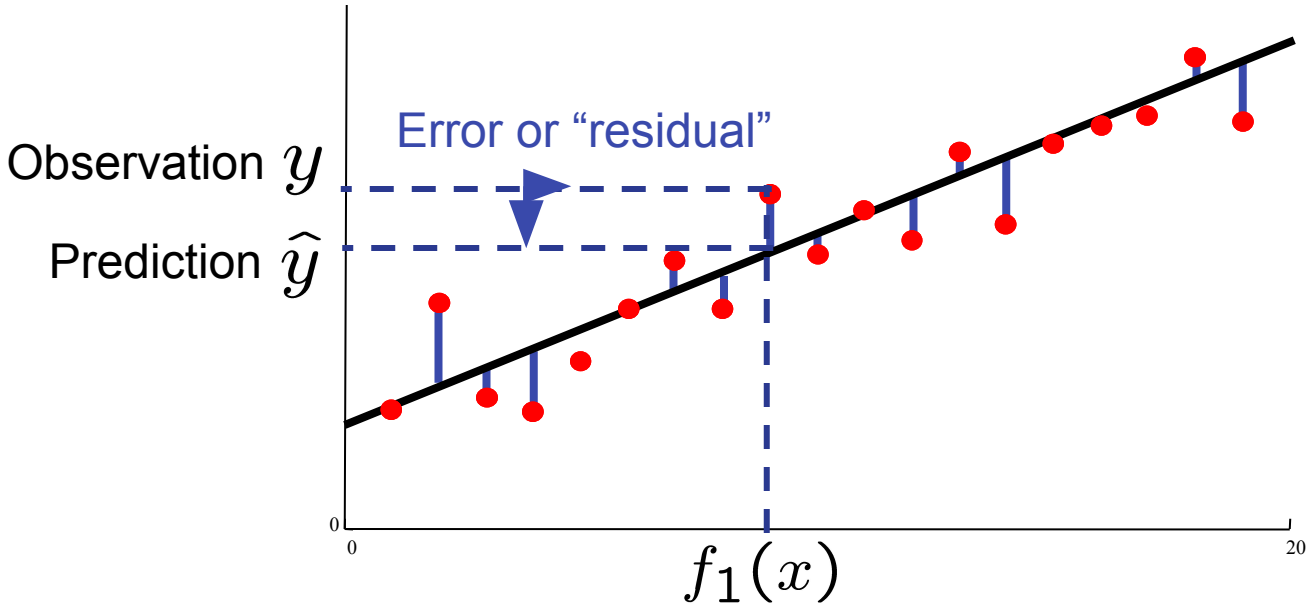
$$\widehat{y} = w_0 + w_1 f_1(x)$$

Predição (em 2D):

$$\widehat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x)$$

Otimização: Mínimos Quadrados

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i) \right)^2$$



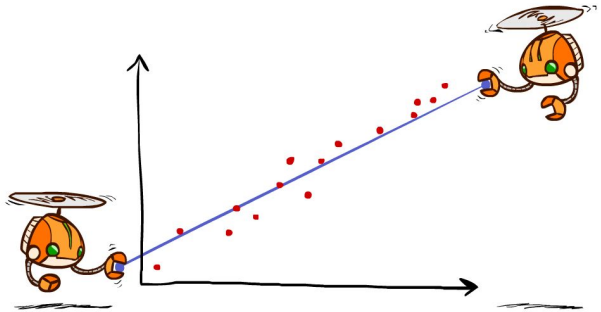
Minimização do Erro

Imagine que tivéssemos apenas um ponto x, com características $f_k(x)$, alvo y, e vetor de pesos $w = [w_1, w_2, \cdots, w_m,]$:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\ \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$



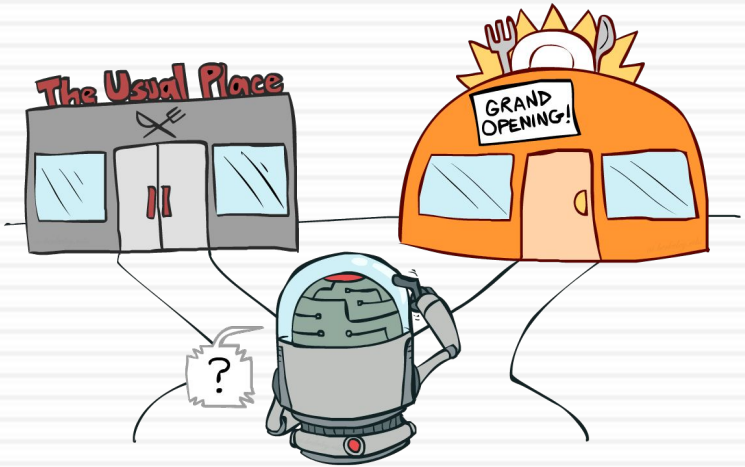
Q-learning com função linear:

$$w_m \leftarrow w_m + \alpha\left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

"alvo" "predição"

Exploração *versus* aproveitamento

exploration versus exploitation



De que forma explorar?

- Durante o treinamento no Q-learning, o agente está diante de um dilema: ele deseja tomar a ação ótima, mas ele ainda não sabe qual ela é, pois está no processo de aprendê-la!
- Esse dilema envolve uma escolha fundamental entre:
  1. **Exploração** (*exploration*): tomar uma ação cujo valor ainda é desconhecido até o momento, mas que tanto pode levar a uma região boa do espaço de estados, quando pode ser desastrosa para o agente.
  2. **Aproveitamento** (*exploitation*): tomar as ações que, no momento, parecem ser boas.

De que forma explorar?

- Há vários esquemas para forçar a exploração durante o aprendizado.
- Dentre eles:
  - epsilon-greedy
  - funções de exploração

Epsilon-greedy

- Solução mais simples: forçar ações aleatórias (ε-greedy)
  - A cada passo de tempo:
    - Com probabilidade (pequena) ε, agir aleatoriamente
    - Com probabilidade (grande) 1-ε, agir de acordo com a política corrente
  - Problemas com ações aleatórias?
    - Agente eventualmente explora o espaço de estados, mas continua explorando mesmo após o aprendizado.
    - Uma solução: diminuir ε ao longo do treinamento

Funções de exploração

- Ideia: explorar áreas cuja qualidade (valor) ainda nã
  eventualmente parar a exploração.

- Função de exploração
  - Toma uma estimativa de valor **u** e um contador de visitas **n**, e retorna uma utilidade otimista, e.g.

"Em face de alguma incerteza, seja otimista!"

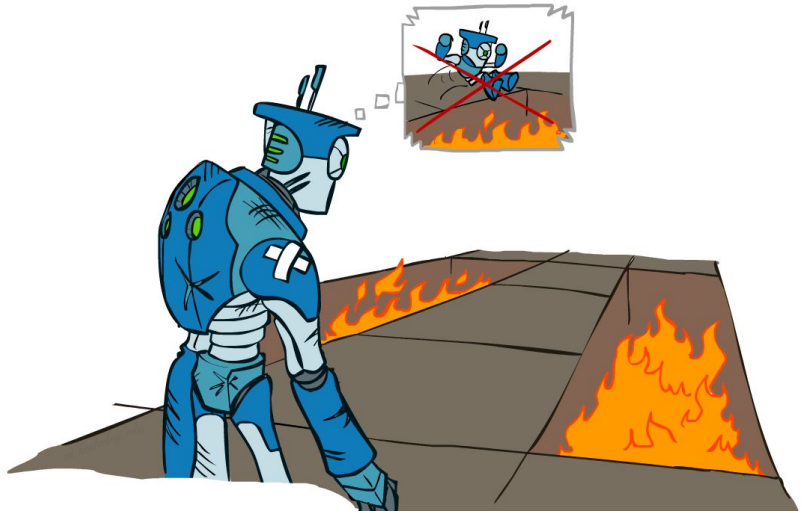
$$f(u, n) = u + k/n$$

Q-Update normal: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

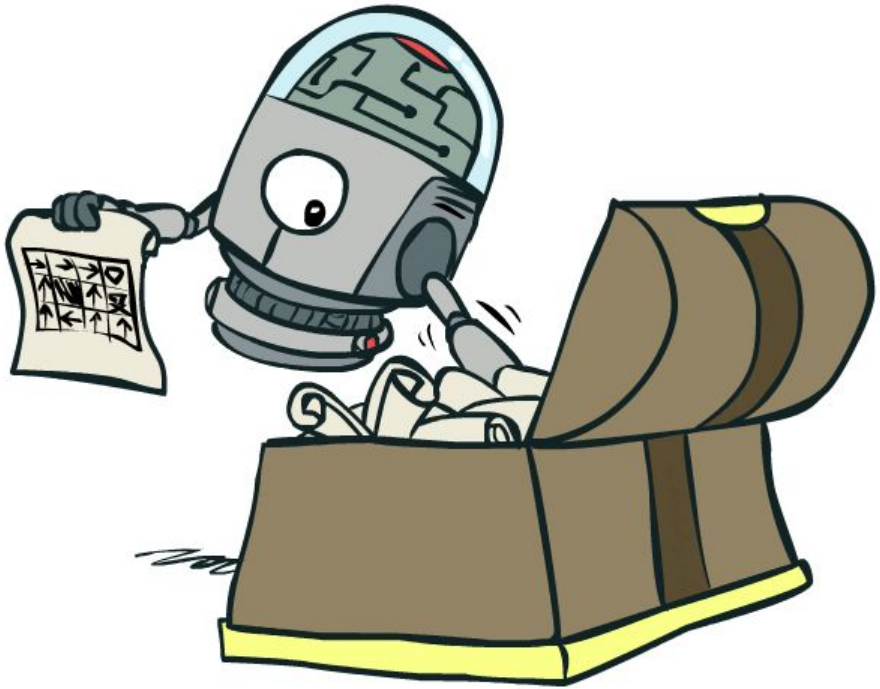
  - Nota: isso propaga o "bônus" para estados que levam a estados desconhecidos!

Arrependimento

- Even if you learn the optimal policy, you still make mistakes along the way!
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
- Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret



Policy Search



Policy Search

- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)
  - We'll see this distinction between modeling and prediction again later in the course

- Solution: learn policies that maximize rewards, not the values that predict them

- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights

Policy Search

- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Nudge each feature weight up and down and see if your policy is better than before
- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical
- Better methods exploit lookahead structure, sample wisely, change multiple parameters…

Policy Search

[Andrew Ng]

[Video: HELICOPTER]