Reinforcement Learning

Caio Petroncini
linkedin/CaioPetroncini

Presença

- Linktree: Presente na bio do nosso instagram
- Presença ficará disponível até 1 hora antes da próxima aula
- É necessário 70% de presença para obter o certificado

Presença

Métodos de Gradiente

O que vimos até agora

- Métodos **tabulares** para resolver MDPs
- DP, MC, TD

O que vimos até agora

- Métodos **tabulares** para resolver MDPs
- DP, MC, TD

Próximo passo:

O que vimos até agora

- Métodos **tabulares** para resolver MDPs
- DP, MC, TD

Próximo passo:

MÉTODOS POR APROXIMAÇÃO

Por que usar aproximação

- Na maioria dos problemas reais, o espaço de possíveis estados e ações é gigantesco
- É impossível armazenar valores associados a cada estado e computacionalmente inviável computar os valores de todos os estados que o nosso agente pode visitar

Erro da aproximação

- Na aproximação ,não é possível acertar todos os estados simultaneamente, então calculamos o Mean Squared Value Error (VE) para decidir os estados mais importantes:

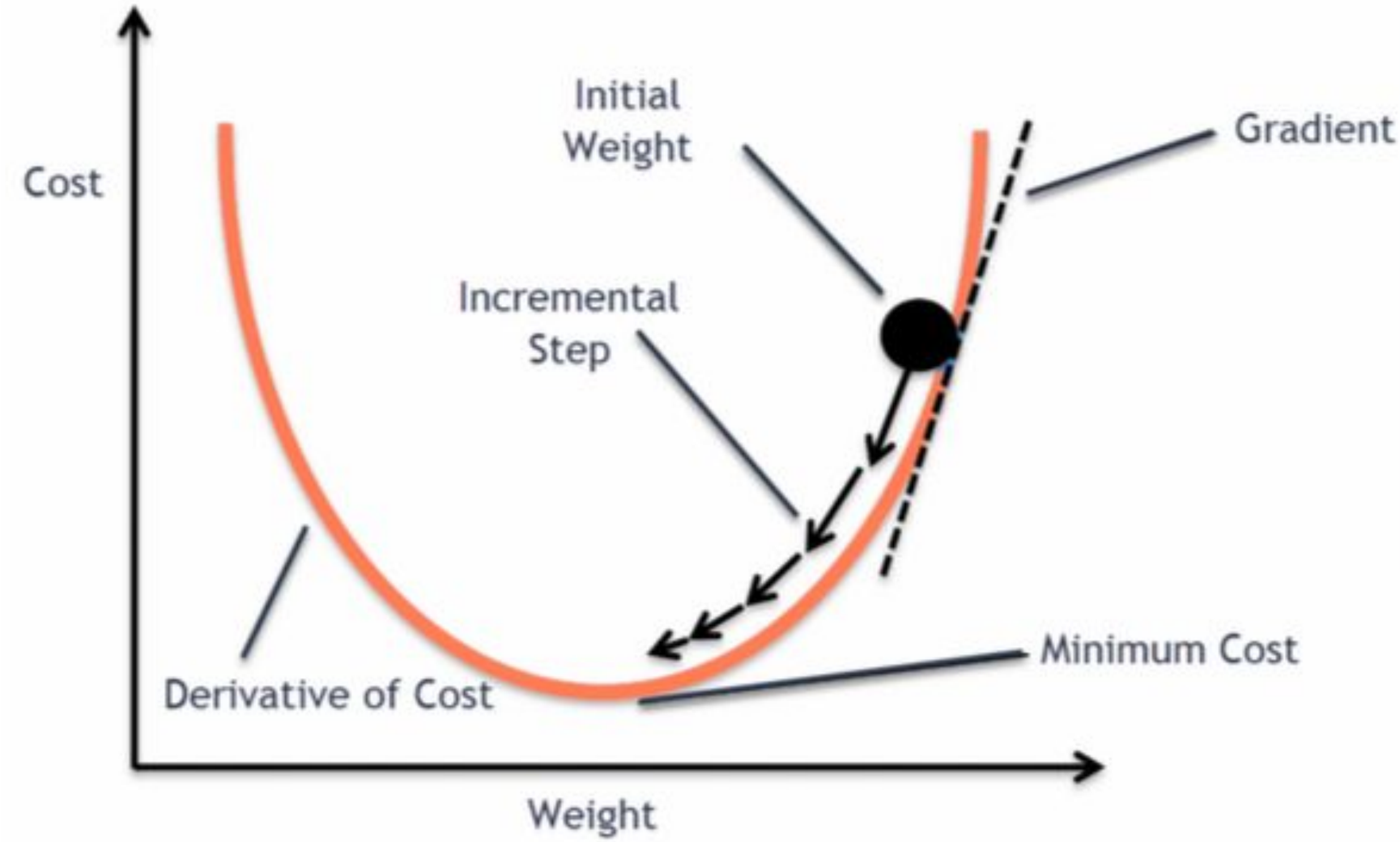
$$\overline{VE}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2.$$

- μ(s) é a importância do estado, geralmente sendo a porcentagem do tempo que passamos nele (soma 1)

Explicando o SGD

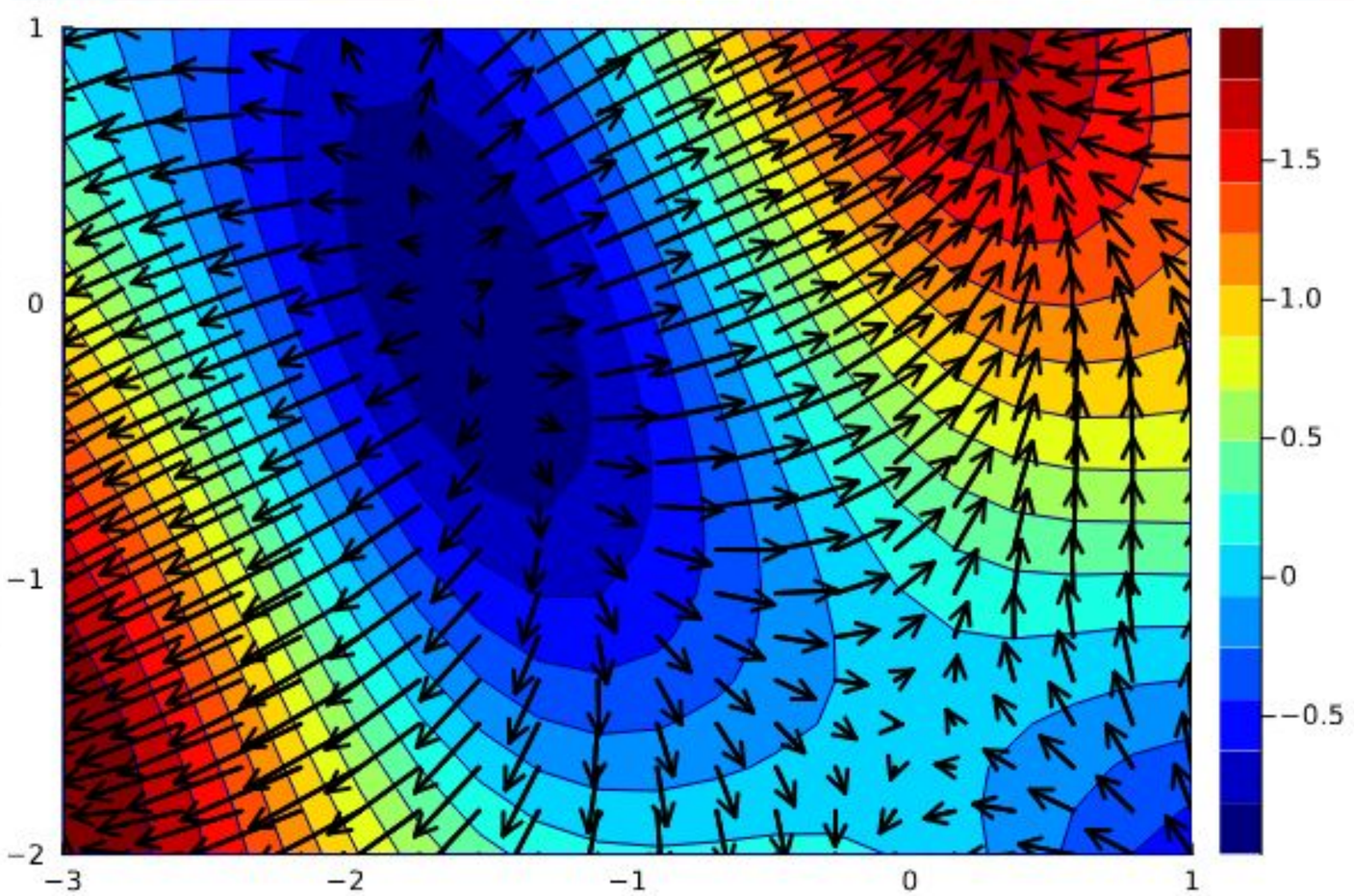
- Stochastic Gradient Descent (SGD) é um método de aproximação de funções
- Utiliza o fato de que o gradiente de uma função mostra para que lado ela está crescendo
- No caso de uma função de erro, vai na direção contrária, pois quer que ela diminua

Explicando o SGD



Explicando o SGD

$$\nabla_{a,b,c} f(a,b,c) \rightarrow \begin{bmatrix} \frac{\partial}{\partial a} f(a,b,c) \\ \frac{\partial}{\partial b} f(a,b,c) \\ \frac{\partial}{\partial c} f(a,b,c) \end{bmatrix}$$



Fórmula para atualizar w

- Como podemos atualizar o w para que a nossa função $\hat{v}(S_t, w_t)$ fique mais próxima da função verdadeira $v_\pi(S_t)$?
- Como o nosso erro, para cada estado, é dado como:

$$[v_\pi(S_t) - \hat{v}(S_t, w_t)]^2$$

- Podemos, utilizando o SGD atualizar o w da seguinte forma:

Fórmula para atualizar w

$$w_{t+1} = w_t - \frac{1}{2}\alpha\nabla[v_\pi(S_t) - \hat{v}(S_t, w_t)]^2$$

- Pela regra da cadeia:

$$w_{t+1} = w_t - \alpha[v_\pi(S_t) - \hat{v}(S_t, w_t)]\nabla\hat{v}(S_t, w_t)$$

Relembrando que o gradiente é $\nabla f(\mathbf{w}) \doteq \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right)^\top.$

Aproximar v_π

- Contudo, nós não temos $v_\pi(S_t)$, pois é essa a função que estamos tentando aproximar
- Por isso temos que usar uma aproximação U_t no lugar

Métodos

- Monte Carlo

$$U_t = G_t$$

- TD Learning

$$U_t = R + \gamma \hat{v}(S_{t+1}, w)$$

Métodos

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
    Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[G_t - \hat{v}(S_t, \mathbf{w}) \big] \nabla \hat{v}(S_t, \mathbf{w})$

Métodos

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose $A \sim \pi(\cdot|S)$
        Take action A, observe R, S'
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
        $S \leftarrow S'$
    until S is terminal

Diferença entre completo e semi gradiente

- Monte Carlo é um método de gradiente completo pois a aproximação não é um valor que depende de w
- TD learning é um método de semi-gradiente, pois utiliza um valor guardado da função para usar como aproximação

Linear Methods

Métodos Lineares: definição

- Aproximamos o valor por função linear nos pesos:

$$\hat{v}(s, w) = w^T x(s) = \sum_{i=1}^{d} w_i x_i(s)$$

- x(s) é o vetor de features do estado s; cada xi é uma função do estado
- As features atuam como funções base; escolher x(s) ⇔escolher a base da aproximação
- Por serem simples, métodos lineares são fáceis de analisar e eficientes

Intuição: representação e generalização

- Estados diferentes podem compartilhar features → a estimativa generaliza para estados “parecidos”.
- A escolha de x(s) determina o que generaliza e o que diferencia entre estados.
- Dimensão d controla capacidade × custo: mais features ↓viés e ↑variância/complexidade.
- Conclusão: a qualidade da aproximação linear depende criticamente de boas features

Atualização por Gradiente (caso linear)

- No caso linear, o gradiente é simples

$$\nabla_w \hat{v}(s, w) = x(s)$$

- SGD (forma geral de predição)

$$w_{t+1} = w_t + \alpha[U_t - \hat{v}(S_t, w_t)]x(S_t)$$

onde Ut é um alvo

- Interpretação: ajusta w na direção que reduz o erro naquele estado St

Propriedades importantes (por que linear?)

- Para perdas típicas (p.ex., VE), com métodos lineares há um único ótimo global; algoritmos de gradiente convergem com passo adequado/decrescente.
- Vantagens: simplicidade, análise teórica sólida, custo computacional previsível.
- Limitações: expressividade depende da base de features; se a base for pobre, sobra erro residual.

Features

- O vetor de features representa cada estado do nosso problema
- Um mesmo vetor de features pode representar vários estados diferentes
- A forma que obtemos as features determina que espécie de dinâmica dos estados estamos explorando e como nossa função valor **generaliza**

Polinomial

- Os estados de um problema podem ser definidos por um conjunto de números como:
  - Coordenadas x, y, z
  - Velocidade e ângulo
  - Capital acumulado
  - Número de cartas na mão

  Usando apenas o método linear nessas características, só é possível capturar **relações lineares** entre as variáveis de um estado

Polinomial

- É possível construir nossas features a partir de polinômios das nossas variáveis de estado, o que permite descrever relações mais complexas entre elas

$$x_i(s) = \Pi_{j=1}^{k} s_j^{c_{i,j}},$$

Exemplo: um problema em que cada estado é uma posição (x,y)

Podemos definir nosso vetor de características como:

$$c(x, y) = [x, y, xy, x^2y, xy^2]$$

Polinomial

$c(x, y) = [x, y, xy, x^2y, xy^2]$

- Assim, o nosso modelo linear pode explorar relações quadráticas entre x e y.
- Podemos definir um vetor de características polinomial arbitrariamente grande, explorando polinômios de graus enormes,
- Na prática, utilizamos conhecimento prévio do problema para escolher os polinômios

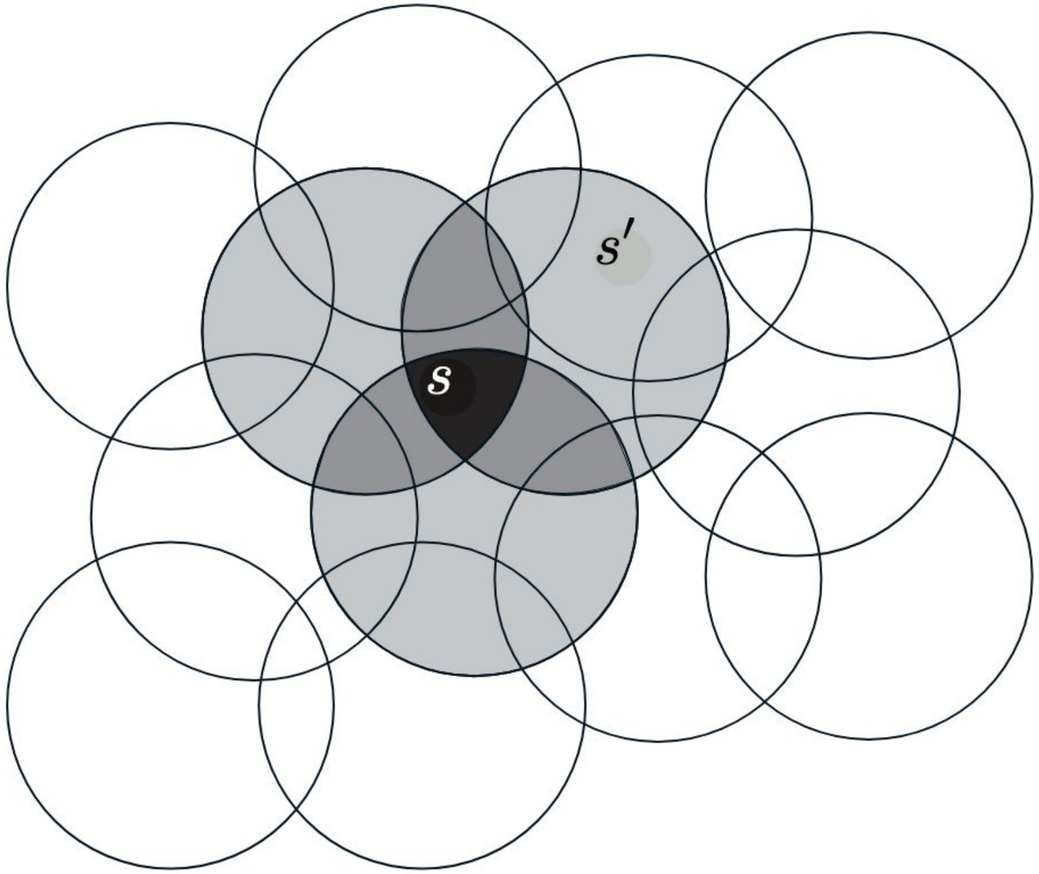
Coarse coding

- No coarse coding, dividimos o nosso espaço de estados em k regiões com sobreposições. Cada estado pode se encontrar dentro de várias dessa regiões
- Nosso vetor de features vai ter k features, cada uma representando uma das regiões criadas e possuindo um valor de 0 ou 1, indicando se o estado se encontra dentro da região

Coarse coding

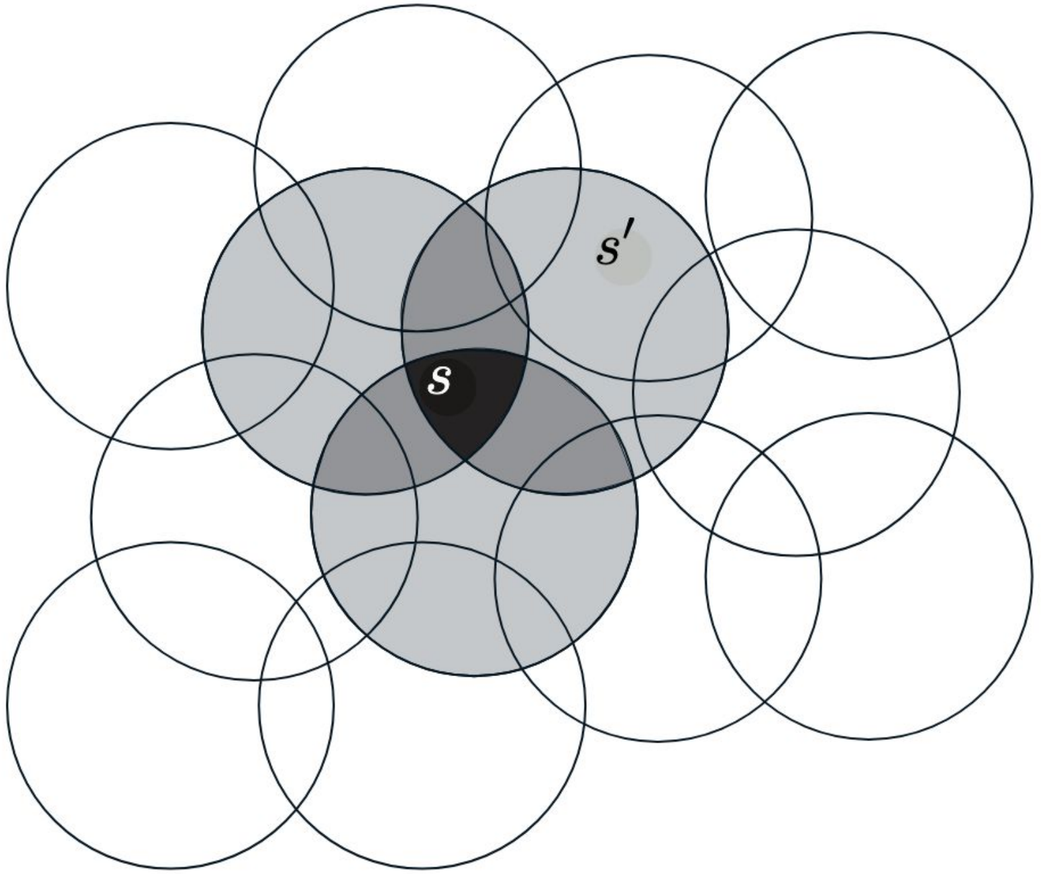
c(s) = [0, 0, 0,1, 0, 1 , … , 0, 1, 0]
k regiões -> k features

Cada peso de w está unicamente associada a umas das regiões já que realizamos o produto escalar

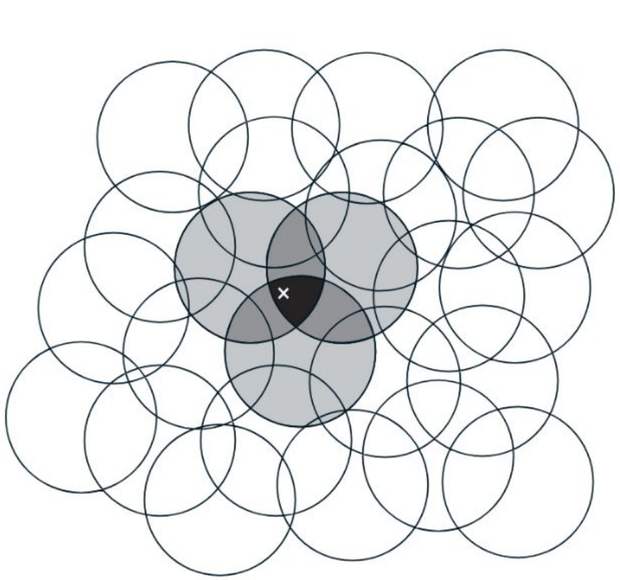


Coarse coding

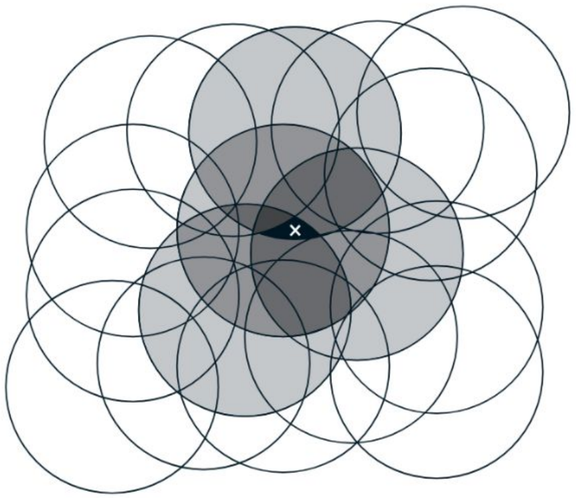
- Estados que estão em regiões em comum possuem features em comum. Ou seja, estão associados
- Todos os estado na mesma interseção são totalmente associados (possuem o mesmo vetor de features)
- Mudanças nos pesos associados a uma região afeta o valor previsto de todos os estados nessa região



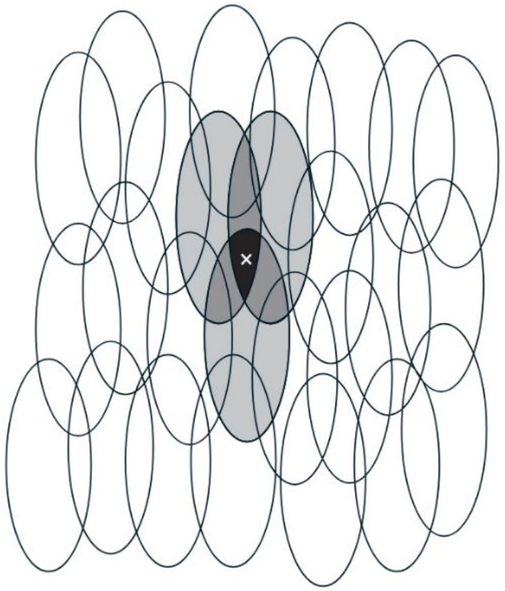
Coarse coding



Narrow generalization



Broad generalization



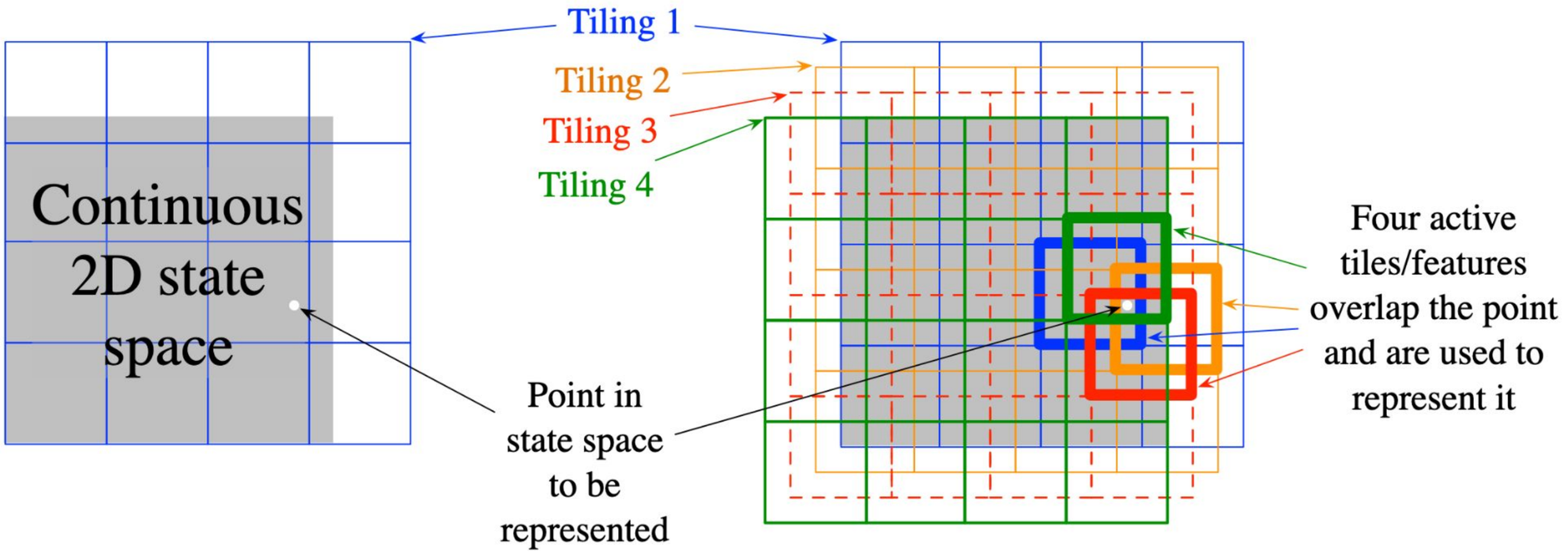
Asymmetric generalization

- Generalização é determinada pelo formato das regiões e o grau de sobreposição
- Acuidade é determinada pelo número de regiões

Tile coding

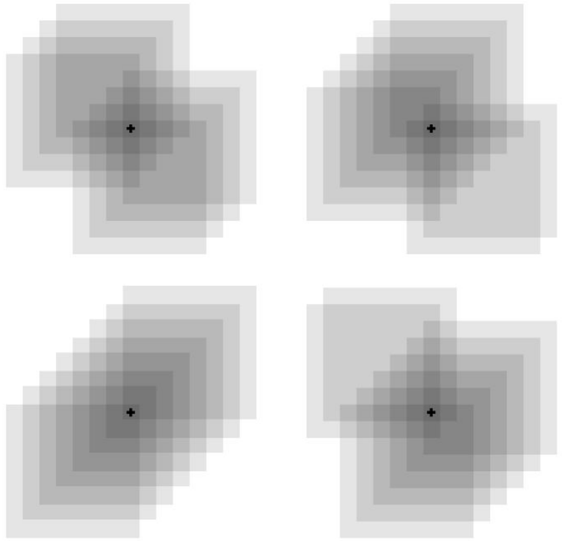
- Tile coding é uma forma de coarse coding em que o espaço em vários tilings (particões), cada tiling tendo vários tiles (células)
- O vetor de features funciona da mesma forma que qualquer outro coarse coding.
- Se temos k tilings n por n, teremos k * n * n features, cada uma representando um tile
- Para cada tilling, é computacionalmente trivial encontrar em qual tile um estado está. O Tile coding é então muito mais rápido e barato de implementar

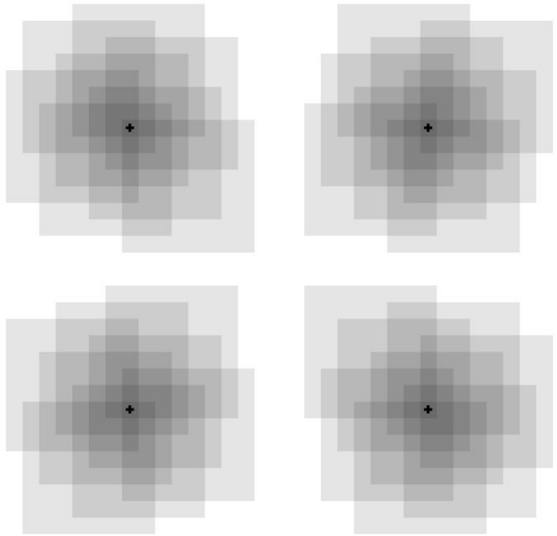
Tile coding



- Mesma dinâmica de generalização com estados que ativam os mesmo tiles
- Estados que ativam **exatamente** os mesmos tiles são considerados idênticos
- Note que somente um tiling não é possível ter sobreposição de tiles

Tile coding

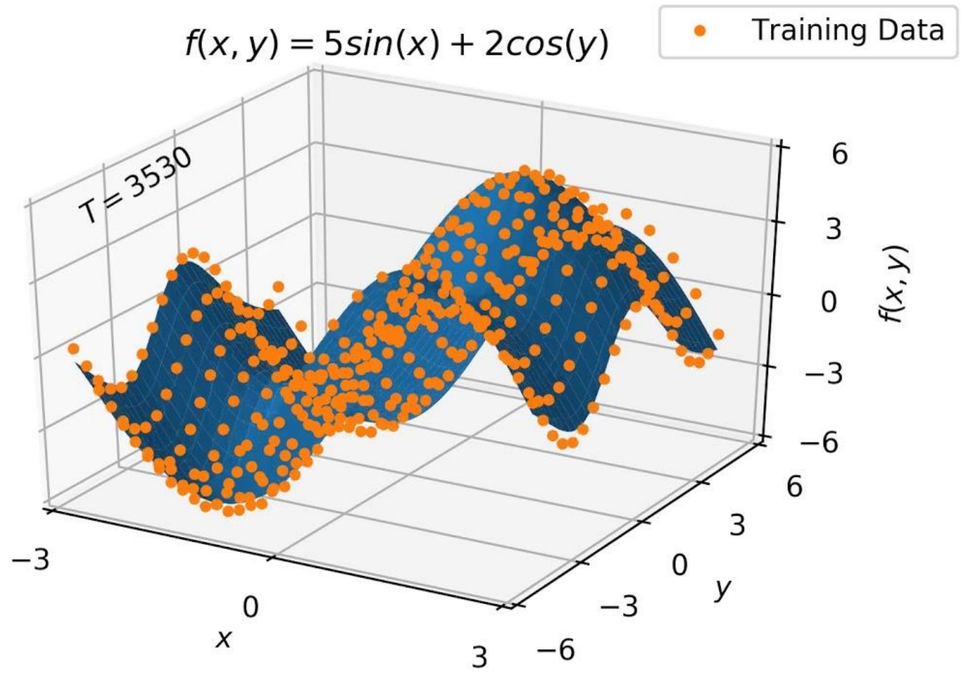




- A forma como fazemos o **offset** dos tilings determina que forma da generalização
- Se os tilings estão desalinhados diagonalmente, cada estado vai estar mais fortemente associado com seus vizinhos diagonais
- Se a disposição for mais assimétrica, terá um viés menor na generalização,

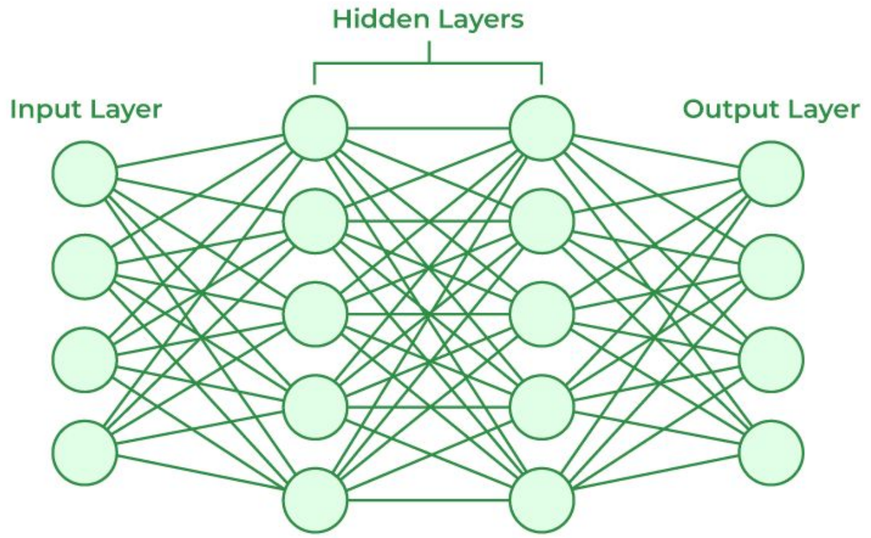
Redes Neurais em RL

- **Propósito:** aproximar Funções Valor / Ação-Valor



Redes Neurais em RL

- **Recapitulando:**
  - RNNs são function-approximators
  - inspiração neurológica



Redes Neurais em RL

- **Vantagens:**
  - capturar padrões complexos / não lineares
  - extração automática de features

Redes Neurais em RL

- **Desvantagens:**
  - DataSet grande
  - natureza do RL
    - aprendizado Online
    - dados interrelacionados

Redes Neurais em RL

- **Exemplo:** AlphaGo (Google)

Memory Based

- Guardam exemplos e só calculam a estimativa quando consultados **lazy learning**.
- Aproximação **não paramétrica**: a forma não é pré-fixada; é determinada pelos exemplos e pela regra de combinação.
- Precisão tende a melhorar conforme mais exemplos são armazenados.

Aprendizagem local

- K vizinhos mais próximo: retorna o alvo do exemplo cujo estado é o mais próximo do estado de consulta.
- Média ponderada de vizinhos: média dos alvos dos vizinhos, com pesos decrescentes com a distância.

$$\hat{v}(s,\mathcal{D}) = \sum_{s' \in \mathcal{D}} k(s,s')g(s').$$

- A função kernel pode ser distância simples, RBF com parametros (média e variância) determinada por SGD entre outros. A distância pode ser entre features do estado

Por que em RL e desafios

- Concentram a aproximação onde as trajetórias realmente visitam; muitas regiões talvez nunca sejam alcançadas.
- Experiências novas afetam rapidamente estimativas perto do estado atual.
- Memória: custo linear em ***k*** por exemplo e linear em ***n*** exemplos (evita exponencial em ***k***).
- Gargalo: busca por vizinhos e regressão local; acelerar com paralelismo, hardware dedicado e **k-d trees**.

Interesse e Ênfase

Motivação:

- Como podemos tratar os estados, de maneira eficiente, com recursos limitados?

- **Problema:** Em problemas complexos, os recursos de aproximação são muito mais limitados que o número de estados, ou seja, não podemos aprender o valor exato para cada estado.

- **Questão:** Todos os estados têm a mesma importância?
  - Como podemos focar os recursos de aprendizagem nos estados que mais nos interessam?

Solução: Interesse e Ênfase

Uma **aprendizagem** mais **focada**

- **Interesse:** número que define **o quanto nos importamos** com o valor do estado no tempo *t*
  - I_t = 1 → **importante;**
  - I_t = 0 → **não** importa;
- **Ênfase:** número que **multiplica a atualização** de aprendizagem
  - Mecanismo que aplica o foco. Só recebe ênfase se tiver interesse (I_t maior ⇒ mais ênfase)
  - Ênfase alta → grande atualização

Como é feito?

- **Regra Geral de Atualização:**

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha M_t \left[G_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1}) \right] \nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}), \qquad 0 \leq t < T$$

- **Cálculo da Ênfase:**

$$M_t = I_t + \gamma^n M_{t-n}, \qquad 0 \leq t < T,$$

- **Interesse:**
  - Valor escalar não negativo que nós definimos
  - “Quão importante é ter uma estimativa precisa desse estado?

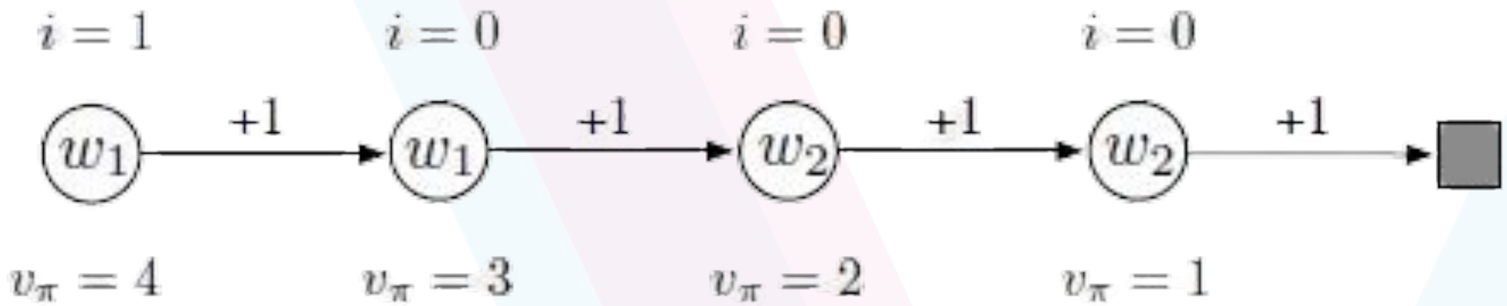
Analogia: Estudar para prova

- **Aprendizagem Normal:** estudar o livro todo da disciplina
- **Com interesse:** *"A matéria que mais vai cair é do Capítulo 5"*
  - $I_t = 1$ → conteúdo do Cap. 5
  - $I_t = 0$ → demais conteúdos

A enfase acumula interese através do tempo, então estados que eu visitei logo depois de visitar um estado interessante recebem enfâse mesmo eles mesmo não sendo interessantes

Exemplo prático:

$\mathbf{w} = (w_1, w_2)^\top$



SEM Interesse e Ênfase:

- Minimizar o erro em todos os estados
- w1 = 3,5 e w2 = 1,5 ← **Errado**

COM Interesse e Ênfase:

- Foca o esforço de atualização no primeiro estado
- w1 = 4 ← **Correto**

Controle

- Como fazer o controle agora?
- Não muda muita coisa, podemos ter uma noção já nessa aula com SARSA de exemplo

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Controle com SARSA

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

data@icmc.usp.br

@data.icmc

/c/DataICMC

/icmc-data

data.icmc.usp.br

obrigado!