Universidade Federal de Viçosa
Centro de Ciências Exatas e Tecnológicas
Departamento de Informática

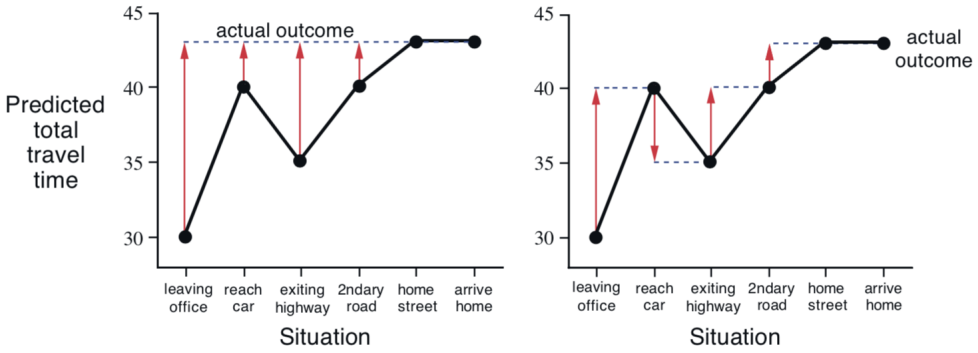
Métodos por Diferença Temporal
Levi H. S. Lelis
levi.lelis@ufv.br

Métodos por Diferença Temporal - Exemplo

Considere o problema de prever o tempo necessário para dirigir do trabalho para casa.

1. Ao sair do trabalho você prevê que irá demorar 30 minutos para chegar em casa.
2. Ao chegar no carro percebe que está chovendo e sua estimativa sobe para 40 minutos.
3. Ao dirigir para via principal, percebe que o trânsito está bom e sua estimativa para o tempo total de viagem cai para 35 minutos.
4. Ao chegar perto de casa você fica preso atrás de um caminhão e sua previsão sobe para 40 minutos.
5. Após 40 minutos de viagem você entra na sua rua e estima que ainda faltam 3 minutos para chegar.
6. Em 43 minutos você termina a sua viagem.

Com um método Monte Carlo precisamos aguardar o final da viagem para aprender as estimativas do tempo de viagem em cada em dos pontos mencionados acima. Isso é mostrado pelo gráfico à esquerda na figura abaixo, onde o valor alvo 43 é obtido e então a estimativa de cada ponto é ajustada.



Na aula de hoje vamos ver uma família de algoritmos que usa um esquema conhecido como **diferença temporal** (TD; sigla em Inglês de *Temporal Difference*), que não espera até o final do episódio para começar a ajustar suas previsões. A ideia é que após cada passo, já conseguimos ajustar as previsões. A figura à direita mostra os ajustes de TD para o problema de previsão do tempo de viagem entre o trabalho e casa.

Previsão TD

Na última aula vimos que o método de previsão Monte Carlo utiliza a seguinte fórmula de atualização:

$$V(S_t) = V(S_t) + \alpha(G_t - V(S_t))\,,$$

onde G_t é a recompensa observada depois do tempo t até o final do episódio e $\alpha = \frac{1}{N(S_t)}$ com $N(S_t)$ sendo o número de vezes $V(S_t)$ foi atualizado. De uma forma geral, α é conhecido como o **tamanho do passo** na direção do alvo G_t; α pode ser variável, como em $\frac{1}{N(S_t)}$, ou fixo em um valor entre 0 e 1.

A equação de atualização TD é dada por:

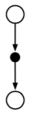
$$V(S_t) = V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))\,.$$

Assim que o valor de R_{t+1} é observado, o valor de $V(S_t)$ já pode ser ajustado. Essa equação de atualização é conhecida como TD(0) e dá origem ao algoritmo abaixo. O Método TD(0) realiza ***bootstrapping***, que

```
1 def TD(0) Previsão On-Policy(π, α ∈ (0, 1], γ):
2   Inicialize V(s) de forma arbitrária, com V(terminal) = 0
3   for each episódio:
4     Inicialize estado inicial S
5     for t in range(1, T):
6       A = π(S)
7       Execute ação A e observe R e S'
8       V(S) = V(S) + α[R + γV(S') − V(S)]
9       S = S'
```

é a ideia de atualizar os valores de V com base em outros valores estimados de V. Os algoritmo de programação dinâmica também fazem *bootstrapping*. Já os algoritmos Monte Carlo não, pois utilizam o valor de recompensa acumulada amostrada de um episódio.

Abaixo, o diagrama de backup do TD(0).



TD(0)

O valor de $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ é conhecido como o **erro de TD**. Isto é, a diferença do valor alvo $R_{t+1} + \gamma V(S_{t+1})$ e o valor atual $V(S_t)$.

Previsão TD Off-Policy

A previsão do TD off-policy é obtida fazendo a seleção de ação A com base na política comportamental b e o erro de TD é ajustado pelo fator de $\frac{\pi(S,A)}{b(S,A)}$ de amostragem por importância (veja o algorithm `TD(0) Previsão Off-Policy`). O uso da importância de amostragem para métodos MC não funciona muito bem pois o fator é sob todo o episódio, o que resulta em números muito pequenos (multiplicação de vários valores menores que um). A importância por amostragem funciona melhor com métodos TD. No caso da previsão, apenas um valor de probabilidade é considerado, visto que a atualização é feita com apenas um passo.

Vantagens do TD

- Assim como métodos Monte Carlo, métodos TD aprendem através de experiência.

```
def TD(0) Previsão Off-Policy(π, b, α ∈ (0,1], γ):
  Inicialize V(s) de forma arbitrária, com V(terminal) = 0
  for each episódio:
    Inicialize estado inicial S
    for t in range(1,T):
      A = b(S)
      Execute ação A e observe R e S'
      V(S) = V(S) + α[ π(S,A)/b(S,A) (R + γV(S')) − V(S) ]
      S = S'
```

- Não é necessário esperar o final do episódio para aprender. Pode ser que os episódios sejam muito longos, ou talvez o problema não possa ser dividido em episódios.
- TD(0) converge para v_π se o passo é pequeno o suficiente ou se ele cai ao longo do tempo.
- Qual família de métodos é melhor? TD ou Monte Carlo?

Otimalidade de TD(0)

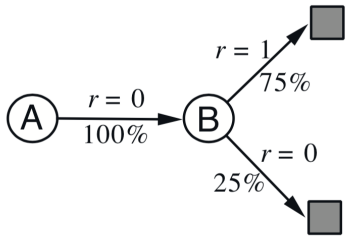
Suponha que a quantidade de dados seja limitada, e.g., 10 episódios. Assim, os valores de erro de TD são todos calculados para os 10 episódios e o valor de V é atualizado após processar os dados dos 10 episódios. Esse processo é conhecido como ***atualização em batch***.

Com atualização em *batch* TD(0) e o algoritmo de previsão MC convergem para um valor, desde que α seja pequeno o suficiente. No entanto, TD(0) converge para um valor e MC converge para outro. Para entender a diferença dentre os dois métodos é importante entender a convergência dos dois métodos no esquema de atualização em *batch*.

Considere os 8 episódios abaixo com estados A e B.

A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

- Qual é o valor de $V(B)$? Se em 6 de 8 episódios obtemos 1 a partir de B, então $V(B) = \frac{3}{4}$.
- Qual o valor de $V(A)$? Duas respostas podem ser consideradas.
  1. $V(A) = 0$, já que todos os episódios em que A aparece termina com recompensa de 0. O método de previsão MC retorna essa reposta, que possui o erro quadrático igual a zero de acordo com o conjunto de treinamento.
  2. $V(A) = \frac{3}{4}$, já que A é sempre seguido por B, que retorna $\frac{3}{4}$. Essa é a solução encontrada por TD(0), que pode ser vista em duas etapas: primeira a modelagem do problema como um processo de Markov, e depois o cálculo da estimativa dado o modelo.



O modelo acima é conhecido como um **processo de recompensa de Markov**, que ignora as ações. Esse tipo de processo é bastante utilizado para analisar problemas de previsão.

De uma forma geral, *batch* MC converge o menor erro quadrático enquanto *batch* TD(0) converge para um valor que bate com a máxima verosimilhança do modelo.

Essa diferença entre MC e TD(0) pode explicar a convergência mais rápida do segundo em casos práticos (ver livro texto para exemplos). Essa diferença no modo *batch* sugere também uma explicação para o modo não-*batch*, onde TD(0) caminha para uma solução "melhor" que aquela para qual MC se aproxima.

Sarsa: Controle TD On-Policy

Assim como nos métodos de controle Monte Carlo, para que o algoritmo de controle seja independente do modelo, utilizaremos TD(0) para estimar valores $q_\pi(s, a)$ para uma política π. Assim como nos métodos anteriores, nós temos duas opções: on-policy e off-policy.

No algoritmo on-policy usamos TD(0) para estimar os valores de uma política ϵ-gulosa, que é melhorada de forma gulosa, como no processo de iteração de política. A regra de atualização é o TD(0) para valores q.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Essa atualização é feita após cada transição $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, onde os valores de A são escolhidos de acordo com a política π. Os valores dão origem ao nome do algoritmo: Sarsa.

Sarsa

Q-Learning: Controle TD Off-Policy

O algoritmo Q-Learning usa uma política exploratória de comportamento b enquanto aproxima os valores Q de uma outra política π.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))\,,$$

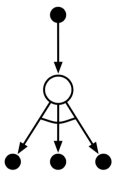
```
def Sarsa(α ∈ (0,1], γ):
  Inicialize Q(S,A) de forma arbitrária, com Q(terminal,·) = 0
  for each episódio:
    Inicialize estado inicial S
    Escolha A em S de acordo com ε-gulosa em Q
    while S não terminal:
      Execute ação A em S e observe R e S'
      Escolha A' em S' de acordo com ε-gulosa em Q
      Q(S,A) = Q(S,A) + α[R + γQ(S',A') − Q(S,A)]
      S = S'
      A = A'
```

Onde A_t é escolhido por b e A' é escolhido por π. Reparem que não é necessário utilizar importância por amostragem pois b e π se diferem apenas na escolha de A' (b poderia escolher algo diferente) e a função Q entrega diferente o valor de A'.

O algoritmo Q-Learning usa uma política ϵ-gulosa como a política comportamental b e uma política que é gulosa sob os valores de Q como política alvo π, resultando na seguinte equação de atualização.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$$

O diagrama de backup de Q-Learning é exibido na figura abaixo.

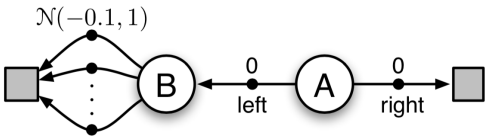


Q-learning

```
def Q-Learning(α ∈ (0,1], γ):
  Inicialize Q(S,A) de forma arbitrária, com Q(terminal,·) = 0
  for each episódio:
    Inicialize estado inicial S
    while S não terminal:
      Escolha A em S de acordo com ε-gulosa em Q
      Execute ação A em S e observe R e S'
      Q(S,A) = Q(S,A) + α[R + γ max_a Q(S',a) − Q(S,A)]
      S = S'
```

Polarização por Maximização e Double Learning

O algoritmo Q-Learning sofre de um efeito conhecido como **polarização por maximização**, que faz com que o algoritmo superestime ações. Isso acontece porque o algoritmo utiliza aproximações dos valores Q. Considere, por exemplo, o problema da figura abaixo.



Nesse problema, um grande número de ações estão disponíveis em B, e elas retornam um valor de recompensa que é amostrado de uma distribuição normal com média -0.1 e variância 1 (o valor esperado de todas as ações é negativo). O estado inicial é o estado A.

Como o número de ações em B é muito grande o máximo das estimativas desses valores será positiva (polarização positiva). Por isso o Q-Learning irá escolher a ação *left* a partir do estado A muita vezes, até conseguir reduzir os valores inicialmente positivos para negativos.

O problema ocorre porque Q-Learning utiliza as mesmas amostras para selecionar as ações e aproximar Q. A solução é utilizar duas aproximações independentes de Q: uma para selecionar ações (Q_1) e outra como aproximação de Q (Q_2). Para um estado S:

$$Q_1(S, \arg\max_a Q_2(S, a)) \text{ ou}$$
$$Q_2(S, \arg\max_a Q_1(S, a))\,.$$

Com o uso de Q_1 e Q_2 a polarização positiva ocorre com probabilidade muito baixa, já que ela requer que uma ação tenha um valor errôneo positivo tanto em Q_1 quanto em Q_2.

```
def Double Q-Learning(α ∈ (0, 1], γ):
  Inicialize Q1(S, A) e Q2(S, A) de forma arbitrária, com Q(terminal, ·) = 0
  for each episódio:
    Inicialize estado inicial S
    for t in range(1, T):
      Escolha A em S de acordo com ε-gulosa em Q1 + Q2
      Execute ação A em S e observe R e S'
      Com probabilidade 0.5:
        Q1(S, A) = Q1(S, A) + α[R + γQ2(S', arg max_a Q1(S', a)) − Q1(S, A)]
      else:
        Q2(S, A) = Q2(S, A) + α[R + γQ1(S', arg max_a Q2(S', a)) − Q2(S, A)]
      S = S'
```

Double Q-Learning aumenta o requerimento de memória por um fator de 2, mas mantém a complexidade de computação inalterada.