Aula 5

Aprendizado por Diferença Temporal (Temporal-Difference - TD)

A Ideia Central do Aprendizado por Reforço

Se tivéssemos que identificar uma única ideia como sendo "a mais central"e inovadora em Aprendizado por Reforço, essa ideia seria, sem dúvida, o Aprendizado por Diferença Temporal (TD).

O aprendizado TD representa uma combinação poderosa de ideias dos métodos de Monte Carlo (MC) e da Programação Dinâmica (DP). A relação entre TD, DP e MC é um tema recorrente e fundamental na teoria de Aprendizado por Reforço.

TD: Uma Combinação de Monte Carlo e Programação Dinâmica

Os métodos TD herdam as características mais importantes de seus predecessores:

- **Como os métodos de Monte Carlo:** Os métodos TD são livres de modelo *(model-free)*. Eles aprendem diretamente da experiência bruta, sem a necessidade de conhecer a dinâmica do ambiente.
- **Como a Programação Dinâmica:** Os métodos TD atualizam suas estimativas com base em outras estimativas já aprendidas, sem precisar esperar pelo resultado final de um episódio. Esse processo de atualizar uma estimativa a partir de outra é chamado de *bootstrapping.*

Ao longo do estudo, veremos que essas ideias se misturam de várias formas. Em capítulos futuros, serão apresentados os algoritmos *n-step*, que servem como uma ponte entre TD e Monte Carlo, e o algoritmo $TD(\lambda)$, que os unifica de forma elegante.

Foco no Problema de Predição

Como fizemos com os métodos anteriores, vamos começar focando no problema de predição (ou avaliação de política), que consiste em estimar a função de valor v_π para uma dada política π

Para o problema de controle (encontrar a política ótima), tanto DP, quanto TD e MC utilizam alguma variação da Iteração de Política Generalizada (GPI). A principal diferença entre esses métodos reside na forma como cada um deles aborda o problema de predição dentro do ciclo do GPI.

Vantagens dos Métodos de Predição TD

Os métodos de Diferença Temporal (TD) atualizam suas estimativas com base em outras estimativas. Eles aprendem "um palpite a partir de outro palpite"— um processo que chamamos de *bootstrapping*. Mas isso é uma boa ideia? Quais vantagens os métodos TD oferecem em comparação com os métodos de Monte Carlo (MC) e de Programação Dinâmica (DP)?

TD vs. Programação Dinâmica: Independência de Modelo

A vantagem mais direta dos métodos TD sobre a Programação Dinâmica é clara: eles **não exigem um modelo do ambiente**. Assim como os métodos de Monte Carlo, os métodos TD aprendem diretamente da experiência, sem a necessidade de conhecer as distribuições de probabilidade de transição e de recompensa.

TD vs. Monte Carlo: Aprendizado Online e Incremental

A vantagem mais óbvia dos métodos TD sobre os métodos de Monte Carlo é que eles são, por natureza, implementados de forma **online e totalmente incremental**.

Com os métodos de Monte Carlo, é preciso esperar até o **final de um episódio** para calcular o retorno e, só então, realizar a atualização do valor.

Com os métodos TD, basta esperar **apenas um passo de tempo** $(t+1)$ para obter uma recompensa (R_{t+1}) e o valor do próximo estado $V(S_{(t+1)})$, permitindo uma atualização imediata.

Essa diferença é crucial em muitas aplicações. Tarefas com **episódios muito longos** podem tornar o aprendizado com MC excessivamente lento. Além disso, existem **tarefas contínuas**, que não possuem episódios, onde os métodos MC simplesmente não podem ser aplicados da forma como os conhecemos.

Garantia de Convergência: O Bootstrapping é Confiável?

Aprender um palpite a partir de outro pode parecer instável, mas podemos garantir que o processo converge para a resposta correta? Felizmente, a resposta é **sim**.

Foi provado matematicamente que, para uma política fixa π, o algoritmo **TD(0)** (a forma mais simples de TD) converge para o verdadeiro valor da função v_π. A convergência é garantida desde que o parâmetro de passo (taxa de aprendizado) seja ajustado de maneira apropriada.

Eficiência: Qual método aprende mais rápido?

Se tanto os métodos TD quanto os de Monte Carlo convergem para as predições corretas, a próxima pergunta natural é: "Qual deles chega lá primeiro?". Em outras palavras, qual método faz um uso mais eficiente de uma quantidade limitada de dados?

Teoricamente: Esta ainda é uma questão em aberto. Ninguém foi capaz de provar matematicamente que um método é universalmente mais rápido que o outro.

Na prática: Em tarefas estocásticas, os **métodos TD geralmente convergem mais rápido** que os métodos de Monte Carlo de passo constante (constant-α MC).

A Otimalidade do TD(0)

A Questão da Eficiência: TD vs. Monte Carlo

Uma observação recorrente na prática de Aprendizado por Reforço é que métodos de Diferença Temporal (TD), como o TD(0), frequentemente convergem para a solução correta mais rápido do que métodos de Monte Carlo (MC) em tarefas com natureza estocástica (aleatória). Para investigar a fundo a razão dessa eficiência superior, analisamos o comportamento de ambos os algoritmos por meio de **Treinamento em Lote (Batch Updating)**.

No treinamento em lote, em vez de atualizar a função valor após cada passo de tempo (online), as atualizações para todo um conjunto ("lote") de episódios de experiência são acumuladas. A função valor é então modificada apenas uma vez, com base na soma de todas essas atualizações. Esse processo é repetido sobre o mesmo lote de dados até que as estimativas da função valor convirjam para uma resposta final e estável. Este método de análise nos permite comparar os pontos de convergência finais de cada algoritmo de forma determinística.

O Paradoxo do Desempenho em Lote

Ao aplicar o treinamento em lote, observamos o seguinte resultado: o método TD(0) converge para uma solução que é consistentemente melhor (ou seja, possui um menor erro quadrático médio em relação à função valor verdadeira, v_π) do que a solução encontrada pelo método Monte Carlo constante-α.

Este resultado apresenta um paradoxo. O método Monte Carlo em lote é considerado "ótimo"em um sentido específico: ele converge para os valores que **minimizam o erro quadrático médio (MSE)** entre as estimativas e os retornos reais observados no conjunto de treinamento. Em outras palavras, a solução de MC é a que melhor se ajusta aos dados que foram vistos. A questão é: como o TD(0) pode ser melhor do que um método que já encontra a solução "ótima"para os dados de treinamento?

A resposta reside no fato de que os dois métodos são "ótimos"de maneiras diferentes, e a otimalidade do TD(0) é mais relevante para a predição de retornos futuros.

Duas Formas de Otimalidade

A Otimalidade do Monte Carlo

A solução encontrada pelo MC em lote é a que melhor se ajusta aos retornos observados no passado. Seu único objetivo é minimizar o erro em relação àquele conjunto específico de experiências.

Embora isso seja ótimo para o conjunto de dados de treinamento, não há garantia de que essa solução seja a melhor para prever o resultado de episódios futuros e inéditos. A solução de MC pode ser vista como uma forma de "memorização"dos resultados passados.

A Otimalidade do TD(0)

O TD(0) em lote, por outro lado, converge para um alvo diferente e mais adaptado: a **Estimativa de Certeza-Equivalência** (*certainty-equivalence estimate*). Esta estimativa é o resultado de um processo de dois passos:

1. **Aprender o Modelo:** Primeiramente, o algoritmo constrói o **modelo de máxima verossimilhança** (*maximum-likelihood model*) do Processo de Decisão Markoviano a partir dos dados do lote. Isso significa que a probabilidade de transição de um estado s para s' é estimada como a fração de vezes que essa transição foi observada, e a recompensa esperada é a média das recompensas observadas naquela transição.
2. **Resolver o Modelo:** Em seguida, o algoritmo calcula a função valor que seria **exatamente correta** se este modelo construído fosse uma representação perfeita do ambiente real.

O TD(0) age com a "certeza"de que o modelo derivado dos dados é o modelo verdadeiro do mundo.

Exemplo Ilustrativo

Suponha que, em um lote de dados, você observe os seguintes oito episódios:

- A, 0, B, 0
- B, 1 (ocorre 6 vezes)
- B, 0 (ocorre 1 vez, além do primeiro episódio)

Para o valor do estado B, V(B), ambos os métodos concordam. O estado B foi visitado 8 vezes, resultando em um retorno de 1 em seis ocasiões e 0 em duas. A média amostral é, portanto, $V(B) = \frac{6}{8} = \frac{3}{4}$.

A divergência ocorre no cálculo de V(A):

- **Solução Monte Carlo:** O método MC observa que o estado A foi visitado apenas uma vez, e o retorno final daquele episódio foi 0. Para minimizar o erro nos dados de treino, a solução é, portanto, $\mathbf{V(A) = 0}$.
- **Solução TD(0):** O método TD primeiro constrói um modelo a partir dos dados. Nesse modelo, a probabilidade de transição de A para B é de 100%. Como o valor de B já foi determinado como $\frac{3}{4}$, a solução de certeza-equivalência é $\mathbf{V(A) = \frac{3}{4}}$.

A resposta do TD(0) é intuitivamente superior. Ela captura a estrutura do problema (que A sempre leva a B) e, por isso, é mais provável que generalize melhor para experiências futuras do que a resposta de MC, que se ajustou excessivamente a um único dado observado.

Eficiência e Viabilidade

A análise em lote revela o motivo da eficiência do TD(0).

- **Generalização:** A superioridade do TD(0) vem da capacidade de encontrar um modelo que simule o ambiente, permitindo uma melhor generalização. A solução de MC, ao focar apenas em minimizar o erro nos dados passados, corre o risco de se ajustar excessivamente (overfitting) a particularidades do conjunto de treinamento.
- **Eficiência no Aprendizado Online:** Embora não convirjam completamente para a estimativa de certeza-equivalência a cada passo, suas atualizações estão constantemente se movendo na direção deste "alvo"mais inteligente, o que acelera o processo de aprendizado em comparação com o MC, que se move em direção à média dos retornos passados.
- **Viabilidade em Grande Escala:** Para problemas com um grande número de estados (n), construir explicitamente o modelo de máxima verossimilhança (que pode exigir memória de ordem $O(n^2)$) e resolvê-lo (com complexidade computacional de ordem $O(n^3)$) é inviável. Os métodos TD são notáveis porque fornecem uma maneira de aproximar esta mesma solução de alta qualidade com requisitos de memória e computação muito mais baixos (ordem $O(n)$), tornando-os uma ferramenta extremamente poderosa e prática no campo do Aprendizado por Reforço.

Algoritmos de Predição TD

Vamos estabelecer aqui então alguns algoritmos tipicamente usados no contexto de TD. É importante frisar que, no contexto mais geral de GPI, os algoritmos aqui mencionados serão aplicados na parte de policy-evaluation, e não na de policy improvement, ou seja, utilizaremos eles para estimar $q_*(s, a)$ dada uma política π sendo seguida. Um último adendo quanto à notação: chamaremos de 'target' o fator utilizado nos algoritmos abaixo para atualizar as estimativas das funções ação-valor (veremos que mudá-lo é a diferença essencial entre eles):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[\text{target} - Q(s_t, a_t)\Big] \tag{1}$$

Sarsa

É um algoritmo on-policy, isto é, a política usada para simular o episódio é a mesma utilizada para atualizar nossos parâmetros. No caso, o Sarsa envolve um ciclo básico a ser seguido durante as simulações episódicas:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big] \tag{2}$$

note que usamos na atualização o seguinte conjunto:$(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ formando daí o nome do do algoritmo.

Outro fator importante sobre o Sarsa é frisar como, conforme dito anteriormente, sempre estamos atualizando nossas funções valor-ação com base na política utilizada pelo modelo até o momento. Como nossa política frequentemente tem um teor explorativo (ex: ϵ-greedy), o Sarsa acaba tornando o treinamento mais "seguro", fazendo o agente tomar o caminho óptimo considerando esse teor de alatoriedade. Para exemplificar isso, considere que queremos treinar um robô muito caro a atravessar uma ponte que tem falhas estruturais em alguns pontos: se o caminho óptimo envolver alto risco de cair, o modelo considerará isso (durante exploração de treinamento), e evitará seguir uma política muito arriscada, preservando mais nosso robô. Esse ponto ficará mais claro quando falarmos do algoritmo seguinte.

Por fim, é necessário frisar que esse algoritmo tem 2 parâmetros: α e n. O α controla a rapidez da convergência das estimativas de $Q(s, a)$, e n é o número de recompensas que queremos incluir na equação acima (a equação abaixo exibe a atualização para $n = 2$). O interessante é notar como, para $n \to \infty$, Sarsa se torna equivalente ao algoritmo do α-MC, já que apenas atualiza no final de um episódio.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma r_{t+2} + \gamma^2 Q(s_{t+2}, a_{t+2}) - Q(s_t, a_t)\Big] \tag{3}$$

Algorithm 1 Sarsa: on-policy TD

Inicialize $Q(s, a),\ \forall s \in \mathcal{S},\ a \in \mathcal{A}(s)$ arbitrariamente, e $Q(\text{terminal-state}, \cdot) = 0$
repeat ▷ p/cada episódio
  Inicialize S
  Escolha A de S usando a política obtida de Q (isto é, ϵ-greedy)
  repeat ▷ para cada parte do episódio
    Faça A, observe R, S'
    Escolha A' de S' usando política obtida de Q (isto é, 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 é terminal
until convergir

Q-Learning

Esse algoritmo é off-policy, e, portanto, o agente aprenderá as funções ação-valor da política óptima, mesmo não seguindo ela na prática. A atualização será dada por:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big] \tag{4}$$

Note que nosso target agora considera a função ação-valor óptima para atualização, e não a que ele de fato utiliza seguindo sua política atual. Isso na prática significa que sempre atualizaremos o conhecimento de nosso agente sobre o mundo de modo a incentivá-lo a seguir os passos mais

óptimos possíveis para seus objetivos. Isso pode ser perigoso, pois, voltando ao exemplo do robô, ele acabaria seguindo um caminho óptimo tendo uma política exploratória, a qual foi desconsiderada em seu treinamento, fazendo com que ele tome riscos frequentes. Há os mesmos parâmetros α, n que já estavam presentes no Sarsa.

Algorithm 2 Q-learning: Algoritmo de controle TD off-policy

Inicialize $Q(s,a),\ \forall s \in \mathcal{S},\ a \in \mathcal{A}(s)$ arbitrariamente, e $Q(\text{estado-terminal}, \cdot) = 0$
repeat ▷ para cada episódio
    Inicialize S
    repeat ▷ para cada passo do episódio
        Escolha A a partir de S usando uma política derivada de Q (por exemplo, ϵ-gulosa)
        Execute a ação A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha\Big[R + \gamma \max_a Q(S',a) - Q(S,A)\Big]$
        $S \leftarrow S'$
    until S é terminal
until convergência

Expected Sarsa

Este algoritmo pode ser interpretado de duas maneiras: uma versão com menor variância do Sarsa, o que permite aumentar α e acelerar o aprendizado; ou uma versão on-policy de Q-learning, em que um caminho aproximadamente óptimo é aprendido enquanto o agente também se atenta à aleatoriedade intrínseca à sua política π. Nesse sentido, ES é considerado uma espécie de "meio termo"entre os resultados dois algoritmos acima. Matematicamente:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\Big[r_{t+1} + \gamma \sum_a \pi(a \mid s_{t+1})\, Q(s_{t+1},a) - Q(s_t,a_t)\Big] \tag{5}$$

Sendo que o nome "expected"advém do fato de usarmos o valor esperado da função ação-valor como target, buscando reduzir os efeitos da aleatoriedade de π na convergência da estimativa. Empiricamente, esse algoritmo se iguala em desempenho ao q-learning para a maioria dos casos, até superando-o muitas vezes. Uma desvantagem importante é ter que calcular $\sum_a \pi(a \mid s_{t+1})\, Q(s_{t+1},a)$ constantemente, o que requer consideravelmente mais recursos computacionais. O pseudocódigo é idêntico ao do q-learning, sendo somente o target trocado.

Double Learning

Os algoritmos discutidos até aqui utilizam a maximização da *policy* para analisar os resultados e selecionar a próxima ação. No entanto, esse processo pode introduzir um viés de maximização, ou seja, uma superestimação dos valores esperados. No algoritmo *Q-Learning*, por exemplo, a mesma estimativa da função Q é usada tanto para escolher a melhor ação quanto para avaliá-la, o que pode levar à superestimação dos valores reais.

O algoritmo conhecido como *Double Learning* busca mitigar esse problema ao empregar duas estimativas independentes da função Q, denotadas por Q_1 e Q_2. A ideia é utilizar uma das estimativas para selecionar a ação e a outra para avaliá-la. A seguir, é apresentada a versão modificada do *Q-Learning*:

$$Q_1(s,a) \leftarrow Q_1(s,a) + \alpha \left[r + \gamma \cdot Q_2 \left(s', \arg\max_{a'} Q_1(s',a') \right) - Q_1(s,a) \right]$$