Aula 6

Unificando Planejamento e Aprendizagem

Este capítulo tem como objetivo desenvolver uma visão unificada dos métodos de aprendizagem por reforço. Até agora, vimos dois grandes grupos de métodos:

- **Métodos Baseados em Modelo (Model-Based):** Requerem um modelo do ambiente para funcionar, como a Programação Dinâmica. A sua componente principal é o **planejamento**.
- **Métodos Sem Modelo (Model-Free):** Podem ser usados sem um modelo, aprendendo diretamente da experiência, como os métodos Monte Carlo e de Diferença Temporal (TD). A sua componente principal é a **aprendizagem**.

Apesar de parecerem diferentes, o coração de ambos os tipos de métodos é o mesmo: o cálculo de **funções de valor**. Este capítulo explora como estes dois métodos podem ser integrados.

Modelos e Planejamento

O que é um Modelo?

No contexto da RL, um **modelo** do ambiente é qualquer coisa que um agente pode usar para prever como o ambiente irá responder às suas ações. Dado um estado e uma ação, um modelo produz uma previsão do próximo estado e da próxima recompensa.

Existem dois tipos principais de modelos:

Modelos de Distribuição (Distribution Models)

Um modelo de distribuição descreve **todas as possibilidades** de próximos estados e recompensas, juntamente com as suas probabilidades de ocorrência.

- **Exemplo (Dados):** Ao modelar a soma de doze dados, um modelo de distribuição produziria uma lista de todas as somas possíveis (de 12 a 72) e a probabilidade exata de cada uma delas ocorrer.
- **Quem usa?** A Programação Dinâmica assume este tipo de modelo, pois precisa da função de dinâmica completa $p(s', r|s, a)$.

Modelos de Amostra (Sample Models)

Um modelo de amostra produz apenas **um dos resultados possíveis**, sorteado de acordo com as probabilidades subjacentes.

- **Exemplo (Dados):** Em vez de listar todas as probabilidades, um modelo de amostra simplesmente simularia o lançamento dos dados e devolveria um único resultado, como "a soma foi 42".
- **Vantagem:** Em muitas aplicações, é muito mais fácil obter um modelo de amostra do que um modelo de distribuição. É mais fácil programar uma simulação do que calcular analiticamente todas as probabilidades.

Tanto os modelos de distribuição como os de amostra podem ser usados para gerar **experiência simulada**.

O que é Planejamento?

O livro define **planejamento** como qualquer processo computacional que recebe um modelo como entrada e produz ou melhora uma política. Em termos simples:

$$\texttt{Modelo} \rightarrow \texttt{Processo de Planejamento} \rightarrow \texttt{Política}$$

A Estrutura Comum de Planejamento e Aprendizagem

A visão unificadora do capítulo é que todos os métodos de planejamento (e de aprendizagem) partilham uma estrutura comum. As duas ideias básicas são:

1. Todos os métodos envolvem o cálculo de **funções de valor** como um passo intermédio para melhorar a política.
2. Eles calculam estas funções de valor através de **atualizações (backups)** aplicadas à experiência.

Esta estrutura comum pode ser diagramada da seguinte forma:

$$\texttt{Modelo} \rightarrow \texttt{Experiência Simulada} \rightarrow \texttt{Backups} \rightarrow \texttt{Valores} \rightarrow \texttt{Política}$$

A grande revelação é que esta estrutura é a mesma para a aprendizagem, com uma única diferença fundamental:

- O **planejamento** usa experiência **simulada** (gerada por um modelo).
- A **aprendizagem** usa experiência **real** (gerada pelo ambiente).

Esta estrutura comum significa que as ideias e os algoritmos podem ser transferidos entre o planejamento e a aprendizagem. Por exemplo, podemos pegar num algoritmo de aprendizagem como o Q-learning, que requer apenas experiência, e aplicá-lo à experiência simulada por um modelo para criar um algoritmo de planejamento.

Dyna: Planejamento, Ação e Aprendizado Integrados

Ideia central

Dyna é uma arquitetura que integra, no mesmo agente, *aprendizado direto por reforço* (a partir de experiência real), *aprendizado de modelo* (a partir dessa experiência) e *planejamento* (a partir de experiência *simulada* gerada pelo modelo). No coração de planejamento e aprendizado estão as mesmas operações de *backup* de funções-valor; a diferença é a fonte da experiência: simulada (modelo) versus real (ambiente). Essa visão unificada permite reutilizar os mesmos algoritmos de RL tanto em dados reais quanto simulados.

Arquitetura Dyna

No ciclo de interação, o agente: (i) **age** no ambiente, obtendo $(S_t, A_t, R_{t+1}, S_{t+1})$; (ii) faz **aprendizado direto** atualizando Q com essa transição real; (iii) **atualiza o modelo** com a predição para o par (S_t, A_t); e (iv) realiza **planejamento** aplicando o mesmo update de RL em transições *simuladas* geradas pelo modelo. Esse entrelaçamento permite gastar uma fração do tempo por passo interagindo e aprendendo o modelo, e dedicar o restante a n atualizações de planejamento por passo (pequenos passos que podem ser interrompidos e redirecionados com baixo desperdício).

Algoritmo Dyna-Q (tabular)

O Dyna-Q usa Q-learning de uma etapa tanto na experiência real quanto na simulada. Em cada passo de tempo:

1. $S \leftarrow$ estado corrente (não terminal)
2. $A \leftarrow \varepsilon$-gananciosa em $Q(S, \cdot)$
3. Executar A; observar R, S'
4. **RL direto:** $Q(S,A) \leftarrow Q(S,A) + \alpha\,[\,R + \gamma \max_a Q(S',a) - Q(S,A)\,]$
5. **Aprender o modelo:** registrar no modelo Model$(S,A) \leftarrow (R, S')$
6. **Planejamento:** repetir n vezes:
   (a) Amostrar $S \leftarrow$ estado previamente observado e $A \leftarrow$ ação já tomada em S
   (b) $(\tilde{R}, \tilde{S}') \leftarrow$ Model(S,A)
   (c) $Q(S,A) \leftarrow Q(S,A) + \alpha\,[\,\tilde{R} + \gamma \max_a Q(\tilde{S}',a) - Q(S,A)\,]$

Exemplo: Dyna Maze e o papel de n

No problema *Dyna Maze*, há 47 estados, quatro ações determinísticas (movimentos cardinais, bloqueados por paredes/bordas), recompensa $+1$ ao entrar no objetivo e 0 caso contrário, com fator de desconto $\gamma = 0{,}95$. Com valores iniciais nulos, $\alpha = 0{,}1$ e $\varepsilon = 0{,}1$, variar o número de passos de planejamento n por passo real acelera dramaticamente a aprendizagem: o agente sem planejamento ($n = 0$) leva ~ 25 episódios para chegar próximo ao ótimo; com $n = 5$ precisa de ~ 5 episódios; com $n = 50$, apenas ~ 3. A intuição: enquanto o agente ainda está explorando perto do início, o planejamento já "preenche" uma política extensa usando o modelo, o que encurta rapidamente o caminho até a meta quando o episódio seguinte começa.

Observações práticas

- **Fonte das amostras de planejamento ("search control"):** uma estratégia simples é amostrar, ao planejar, apenas pares estado–ação *já experimentados*, consultando o modelo aprendido para produzir a transição simulada.
- **Modelo tabular simples:** no caso determinístico, o modelo guarda a última transição observada para cada (s, a).
- **Passos de planejamento curtos e incrementais:** muitos pequenos backups por passo (em vez de grandes varreduras) facilitam interromper/redirecionar o planejamento e intercalar bem com agir e aprender o modelo.

Gancho para a próxima parte. Até aqui assumimos um modelo consistente com o ambiente. A seguir, discutiremos o que acontece quando o *modelo está errado* e como extensões como o *Dyna-Q+* introduzem bônus de exploração no planejamento para detectar mudanças/atalhos de forma eficiente.

Quando o Modelo Está Errado

Em métodos do tipo Dyna, o agente *aprende* com experiência real e *planeja* com experiência simulada gerada por um modelo. Se o **modelo estiver incorreto** (por estocasticidade, generalização imperfeita ou mudanças no ambiente), o planejamento pode induzir uma política subótima.

Dois padrões de erro

- **Otimismo irrealista:** o modelo prevê recompensas/transições melhores do que as possíveis; a política planejada tenta explorá-las e, ao falhar, *revela e corrige* rapidamente o erro no modelo.

- **Cegueira a melhorias:** o ambiente melhora, mas a política atual não visita as transições que evidenciariam a melhoria; o erro pode persistir por muito tempo.

Estudos de caso

Estes exemplos foram retirados do capítulo 8 do livro Reinforcement Learning An Introduction (Sutton e Barto).

Ex. 8.2: *Blocking Maze*. Um caminho curto até a meta torna-se bloqueado após certo tempo; surge um caminho mais longo. Dyna-Q e Dyna-Q+ inicialmente exploram o caminho curto; após a mudança, há um período sem recompensa até descobrirem a nova passagem, corrigindo o modelo ao *tentar* o caminho bloqueado.

Ex. 8.3: *Shortcut Maze*. Abre-se um atalho sem invalidar o caminho antigo. O Dyna-Q *pode nunca descobrir o atalho*: o modelo vigente não o contém e o planejamento reforça o trajeto conhecido, reduzindo a chance de executar as ações que revelariam o atalho, mesmo com política ε-gananciosa.

Exploração no contexto de planejamento

O conflito exploration–exploitation reaparece com outra cara:

> *Explorar* = tentar ações que **melhorem o modelo**; *Explotar* = agir de forma **ótima dado o modelo** atual.

Queremos detectar mudanças (e atalhos) sem degradar demais o desempenho.

Dyna-Q+: curiosidade via bônus temporal

O Dyna-Q+ mantém, para cada par (s, a), quantos passos se passaram desde sua última tentativa real. A intuição: quanto maior o tempo, maior a chance de que a dinâmica de (s, a) tenha mudado. Nas *atualizações de planejamento*, aplica-se um **bônus de exploração**:

$$\tilde{r} \;=\; r \;+\; \kappa\sqrt{\tau},$$

em que r é a recompensa modelada, τ é o número de passos desde a última tentativa real de (s, a), e $\kappa > 0$ é pequeno. Efeitos desejados:

- Incentiva o planejador a simular e o agente a *testar* transições pouco tentadas, expondo melhorias (atalhos) e mudanças.
- Promove retestes periódicos, inclusive de sequências longas até regiões raramente visitadas.

Práticas úteis no Dyna-Q+.

- Permitir no planejamento ações *nunca tentadas* a partir de um estado.
- Modelo inicial para essas ações: transição para o mesmo estado com recompensa 0 (prior neutro).

Escolhas e cautelas

- **Ganho κ:** alto demais $\Rightarrow$ exploração excessiva; baixo demais $\Rightarrow$ cegueira a mudanças. Ajuste observando a frequência de retestes e o impacto no desempenho.
- **Ambientes estocásticos:** o bônus afeta valores; considere múltiplas iterações de planejamento por interação e políticas comportamentais com pequena ε.

Considerações

- Planejamento só é tão bom quanto o modelo; modelos errados podem congelar a exploração.
- O bônus temporal do Dyna-Q+ ($\kappa\sqrt{\tau}$) nas *atualizações simuladas* encoraja a descoberta de atalhos e a detecção de mudanças.
- Em ambientes dinâmicos, vale aceitar um pouco mais de exploração para permanecer *reativo*.

Atualização Esperada vs. por Amostra

A operação de **atualização de valor** é um eixo central em todos os algoritmos de aprendizagem e planejamento. Esta operação ajusta a estimativa do valor de um estado, ou par estado-ação, com base em informações futuras. Existem duas filosofias principais para realizar esta atualização.

Atualização Esperada

A atualização esperada, ou **expected update**, considera todos os futuros possíveis e calcula uma média ponderada para determinar o novo valor.

- **Como funciona:** Para um dado estado ‘s‘ e ação ‘a‘, o algoritmo olha para todos os possíveis próximos estados ‘s’‘ e recompensas ‘r‘. Ele pondera o valor de cada futuro pela sua probabilidade de ocorrência, ‘p(s’, r | s, a)‘.
- **Requisito:** Exige um **modelo de distribuição** completo do ambiente.

- **Fórmula (para q*):**

$$Q(s,a) \leftarrow \sum_{s',r} \hat{p}(s',r|s,a)[r + \gamma \max_{a'} Q(s',a')]$$

- **Vantagens:** É matematicamente preciso e não sofre de erro de amostragem.
- **Desvantagens:** É computacionalmente muito caro e lento, especialmente se o número de possíveis próximos estados ("fator de ramificação") for grande.

Analogia: É como fazer um censo completo para prever uma eleição. Você entrevista todos os eleitores para ter um resultado exato, mas demora muito tempo.

Atualização por Amostra

A atualização por amostra, ou **sample update**, é muito mais simples: ela usa apenas **um único resultado** do que aconteceu a seguir para ajustar o valor.

- **Como funciona:** Em vez de calcular uma média sobre todos os futuros, o algoritmo pega uma única amostra de '(R, S')'—seja da experiência real ou de um modelo—e usa-a como alvo.
- **Requisito:** Precisa apenas de **experiência real** ou de um **modelo de amostra** (que apenas gera um resultado de cada vez).
- **Fórmula (Exemplo Q-Learning):**

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_{a'} Q(S',a') - Q(S,A)]$$

- **Vantagens:** É computacionalmente muito barata e rápida.
- **Desvantagens:** A atualização tem variância (ruído), pois uma única amostra pode não ser representativa da média.

Analogia: É como fazer uma pesquisa de opinião rápida. Você entrevista uma pequena amostra de eleitores. É rápido e dá uma boa ideia do resultado, mas tem uma margem de erro.

Conclusão: Qual é Melhor?

A questão chave é a eficiência computacional. No tempo que se leva para fazer uma única atualização esperada, podem ser feitas centenas ou milhares de atualizações por amostra. Para problemas grandes e complexos, as **atualizações por amostra são quase sempre superiores**, pois permitem que a aprendizagem se espalhe muito mais rapidamente pelo espaço de estados.

Revisão dos Principais Algoritmos

Esta seção resume os principais modelos e algoritmos vistos até a aula 6.

Fundamentos

Bandits de Múltiplos Braços

- **Problema:** Um único estado com múltiplas ações. O objetivo é encontrar a melhor ação.
- **Conceito-Chave:** O dilema entre **exploração** (tentar novas ações) e **explotação** (usar a melhor ação conhecida).
- **Algoritmo Principal:** ϵ**-greedy**, que age de forma gananciosa na maior parte do tempo, mas com uma pequena probabilidade ϵ escolhe uma ação aleatória para explorar.

Processos de Decisão de Markov (MDPs)

- **Modelo Formal:** Descreve o problema de RL com estados, ações, recompensas e probabilidades de transição ‘p(s’, r | s, a)‘.
- **Propriedade de Markov:** O futuro depende apenas do estado presente, não do passado.
- **Ferramentas Teóricas:** As **Equações de Bellman**, que estabelecem a consistência entre o valor de um estado e o valor dos seus sucessores.

As Três Grandes Famílias de Algoritmos

Programação Dinâmica (DP)

- **Tipo:** Baseado em Modelo. Requer conhecimento total do MDP.
- **Mecanismo:** Usa **atualizações esperadas** e varreduras completas do espaço de estados para calcular os valores ótimos.
- **Algoritmos:**
  - **Iteração de Políticas (Policy Iteration):** Alterna entre avaliar completamente uma política (calcular v_π) e melhorá-la.
  - **Iteração de Valores (Value Iteration):** Uma versão mais direta que combina as duas etapas numa única atualização, aplicando a equação de otimalidade de Bellman.

Métodos Monte Carlo (MC)

- **Tipo:** Sem Modelo. Aprende diretamente da experiência.
- **Mecanismo:** Espera até ao final de um episódio e usa o **retorno completo** (G_t) como alvo para a atualização. Estima valores através da média dos retornos.
- **Característica: Não faz bootstrapping** (não usa estimativas para atualizar outras estimativas).

Aprendizagem por Diferença Temporal (TD)

- **Tipo:** Sem Modelo e com Bootstrapping. A combinação das ideias de DP e MC.
- **Mecanismo:** Aprende a cada passo, sem esperar pelo fim do episódio. O alvo da atualização é a recompensa do próximo passo mais o valor estimado do próximo estado ($R_{t+1} + \gamma V(S_{t+1})$).
- **Algoritmos de Controlo Principais:**
  - **Sarsa (On-Policy):** Aprende o valor da política que está a executar. A atualização usa a ação que foi **realmente** escolhida no próximo passo (A_{t+1}). É considerado mais "cauteloso".
  - **Q-Learning (Off-Policy):** Aprende o valor da política ótima, independentemente da política que está a ser seguida. A atualização usa o valor da **melhor ação possível** no próximo passo ($\max_a Q(S_{t+1}, a)$). É considerado mais "agressivo".

Unificando Planeamento e Aprendizagem

Arquitetura Dyna

- **Dyna-Q:** Uma arquitetura que integra aprendizagem e planeamento.
  1. Age no ambiente e recebe uma experiência real.
  2. Usa essa experiência para uma atualização de Q-learning (**aprendizagem direta**).
  3. Usa a mesma experiência para melhorar um modelo interno do mundo (**aprendizagem de modelo**).
  4. Realiza ‘n‘ atualizações de Q-learning adicionais usando o modelo para gerar experiências simuladas (**planeamento por amostra**).
- **Dyna-Q+:** Uma extensão para quando o modelo pode estar errado (o ambiente muda). Inclui um **bônus de exploração** ($r + \kappa\sqrt{\tau}$) para as ações não testadas há muito tempo durante o planeamento, incentivando a "curiosidade".

Amostragem de Trajetória

Em DP, vimos ser possível atualizar nossas funções ação-valor $Q(a, s)$ percorrendo todo os estados possíveis de ação-valor no ambiente, técnica chamada de *varredura completa*. Agora que estamos explorando algoritmos com modelo $\hat{M}$ do ambiente (que se atualiza conforme o número de experiências), é possível executar uma nova forma de atualização dessas funções: trajetórias simuladas.

Para começar, há um problema evidente com as atualizações feitas com o modelo da DP: elas percorrem todo o espaço disponível de $Q(s, a)$, o que pode ser extremamente custoso ou até impossível em casos em que esse espaço é grande. A solução é utilizar seu Modelo como um simulador, e simular algumas trajetórias com ele seguindo a policy atual.

A ideia por trás da amostragem de trajetória é basicamente essa: amostre a_t de π, tome a ação, receba s_{t+1}, r_{t+1} de $\hat{M}$, e repita até atingir ou um episódio terminal ou um limiar não contemplado pelo seu modelo. Trata-se de uma varredura muito mais eficiente, pois a maior parte dos estados na maior parte dos sistemas é completamente inútil, sendo mais produtivo focar nos estados mais frequentes de acordo com sua política.

Planejamento em Tempo de Decisão (Decision-Time Planning)

O planejamento pode ser utilizado de, pelo menos, duas maneiras distintas no Aprendizado por Reforço. Embora essas abordagens possam ser combinadas, é mais fácil entendê-las separadamente.

Planejamento em Segundo Plano (Background Planning)

Esta é a abordagem que consideramos até agora, tipificada por métodos como a Programação Dinâmica e o Dyna. A ideia central é usar a experiência simulada de um modelo para **melhorar gradualmente** uma política ou função de valor ao longo do tempo. O planejamento aqui **não é focado no estado atual**; é um processo contínuo que melhora os valores para muitos estados do ambiente de forma distribuída. Quando o agente chega a um estado S_t, a seleção da ação A_t é simplesmente uma consulta rápida aos valores já pré-computados em uma tabela ou função. O trabalho pesado de planejamento já foi feito "nos bastidores".

Planejamento em Tempo de Decisão (Decision-Time Planning)

A outra forma de usar o planejamento é iniciá-lo e concluí-lo **após o agente encontrar um novo estado** S_t. O processo é uma computação focada, cujo único objetivo é selecionar uma única ação, A_t. No passo seguinte, ao encontrar S_{t+1}, o planejamento recomeça do zero para produzir A_{t+1}, e assim por diante. Diferente do planejamento em segundo plano, aqui o processo é **totalmente focado no estado atual**. Ele envolve simular e avaliar trajetórias futuras a partir de S_t para determinar qual ação imediata é a melhor. Os resultados desse planejamento (os valores e a política temporária criados para S_t) são tipicamente **descartados** após a ação ser escolhida. Isso é aceitável em ambientes com muitos estados, onde a probabilidade de retornar ao mesmo estado em breve é baixa.

Comparação e Casos de Uso

A escolha entre os dois tipos de planejamento depende diretamente dos requisitos da aplicação:

- **Planejamento em Tempo de Decisão** é mais útil em aplicações onde respostas rápidas **não são necessárias**. O exemplo clássico são os programas de xadrez, que podem ter segundos ou minutos para computar dezenas de movimentos à frente antes de tomar uma decisão.
- **Planejamento em Segundo Plano** é a melhor escolha quando a prioridade é a **seleção de ações de baixa latência (rápida)**. A política é pré-computada e pode ser aplicada instantaneamente a qualquer estado encontrado, sem a necessidade de deliberação.

Busca Heurística (BH)

BH é o nome dado ao conjunto de métodos de planejamento em tempo real pelo espaço de estados possível. Explicando mais a fundo esse conceito, imagine que nós estamos em um estado inicial S_t, o que a BH diz é que, dado esse estado inicial, devemos construir uma árvore com os possíveis estados futuros, visando descobrir qual o melhor caminho a se seguir.

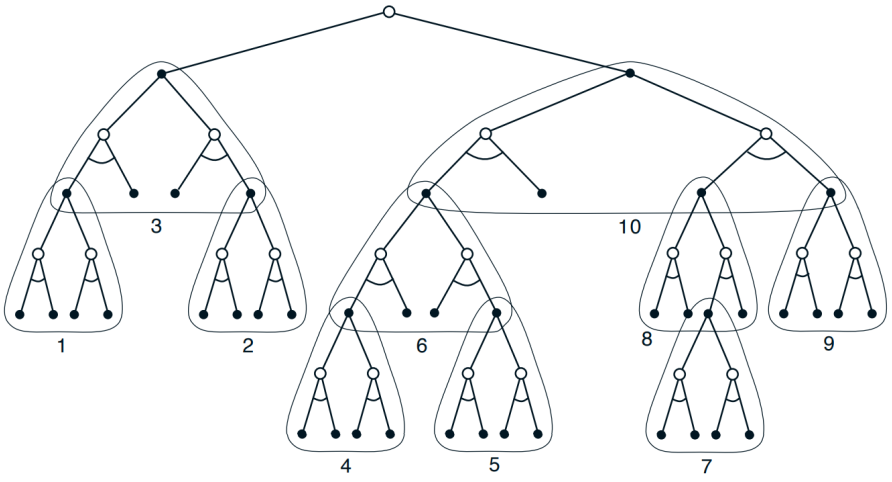

Figura 1: Exemplo de árvore, com S_t sendo o nó raiz, fonte: livro base do curso

Analisando essa ideia, podemos perceber que já vimos isso ao longo do curso diversas vezes, pois, toda vez que agimos de forma ganaciosa para escolhermos a nossa ação, como em uma política *greedy*, estamos analisando os estados futuros que nos darão os melhores resultados, sendo que agora, com a BH, estamos aumentando o nosso espaço de busca para mais do que apenas 1 estado a nossa frente.

Outro ponto importante que devemos destacar é como são feitos as alterações na função valor na BH. Para fazermos isso, podemos simplesmente atualizar os valores dos nossos estados folhas até voltarmos para o nosso estado inicial, fazendo isso por meio, por exemplo, de um DFS (Depth-First Search), que desce na árvore até uma folha, atualiza ela e depois volta atualizando os estados anteriores.

Pensando em um exemplo mais prático, imagine que o nosso agente é um jogador de xadrez, que tem por objetivo, claro, ganhar o jogo. Como um ser humano faria, o nosso agente, antes de fazer qualquer jogada, ele pensa nos próximos estados que o tabuleiro pode estar, analisando qual a jogada que melhora sua chance de vitória. No caso do ser humano, nós imaginamos esses cenários futuros com os nossos cérebros, já o nosso agente fará isso por meio da construção da árvore de possíveis estados futuros, mas as duas abordagens seguem o mesmo princípio.

Além disso, uma das grandes vantagens da BH é que os *updates* feitos na nossa função valor são realizados baseados sempre no estado em que estamos atualmente, dando prioridade aos estados mais imediatos, devido ao gamma. Dessa forma, o nosso agente consegue focar seus *updates* em estados que realmente vão ser importantes para ele, pois são esses os próximos estados mais prováveis, o que é diferente do que algortimos como o Dyna, que seleciona estados para atualizar de maneira aleatória no seu estado de planejamento. Isso ajuda ao agente desenvolver uma política centrada em sempre conseguir achar o melhor caminho a se seguir, olhando vários estados a frente, mas dando prioridade para atualizar os estados mais próximos e prováveis de acontecerem.

Algoritmos de Rollout

Damos esse nome ao conjunto de algoritmos que unem a ideia de BH, fazendo a busca do melhor caminho em uma árvore, com Monte Carlo Control, estimando os valores das ações em um certo estado por meio de uma média dos retornos obtidos. Vamos ver esse tipo de algoritmo em específico para termos uma ideia mais clara de como as ideias da BH são usadas. A ideia central é simples: a partir de um estado atual, como podemos decidir qual é a melhor ação a ser tomada? Em vez de depender de uma função de valor pré-calculada, que pode ser imprecisa, nós simulamos as consequências de cada ação possível.

O processo funciona da seguinte forma:

1. **Ponto de Partida:** Dado um estado atual S_t, o agente avalia todas as ações possíveis que pode tomar a partir deste estado.

2. **Simulação (o "Rollout"):** Para cada ação candidata a, o algoritmo executa uma simulação que se estende por múltiplos passos no futuro. A partir da ação a, o agente segue uma política de rollout, que é definida antes de começar o algorimo, para escolher as ações subsequentes. Essa política é geralmente rápida e simples, mas não necessariamente ótima. A simulação continua, gerando uma trajetória completa até um estado terminal.

3. **Cálculo do Retorno:** Ao final de cada trajetória simulada, calculamos o retorno total (a soma das recompensas) obtido a partir daquela ação inicial. Esse valor funciona como uma estimativa de Monte Carlo do verdadeiro valor daquela ação.

4. **Seleção da Melhor Ação:** Após simular todas as ações possíveis a partir do estado S_t e obter uma estimativa de retorno para cada uma, o algoritmo simplesmente escolhe a ação que levou ao maior retorno médio simulado. Esta é a ação que o agente executará no ambiente real.

Depois de executar a ação escolhida, o agente recebe um novo estado e todo o processo se repete.

Os algoritmos de Rollout oferecem uma série de benefícios significativos, tornando-os uma excelente ferramenta para problemas de decisão sequencial:

- Toda a capacidade computacional é concentrada em tomar a melhor decisão para o estado atual. Em vez de tentar calcular a política ótima para todos os estados possíveis do ambiente (o que pode ser computacionalmente inviável), o esforço é direcionado para onde ele é mais necessário, característica comum aos algoritmos baseados em BH.
- As simulações para cada ação candidata são independentes umas das outras. Isso significa que elas podem ser executadas em paralelo em múltiplos processadores, permitindo que o algoritmo escale para problemas com muitas ações possíveis e encontre boas soluções rapidamente.

Monte Carlo Tree Search (MCTS)

A Busca em Árvore Monte Carlo (MCTS) é um dos mais recentes e bem-sucedidos exemplos de planeamento na hora da decisão. É, na sua essência, um algoritmo de rollout melhorado. A sua principal inovação é que ele **acumula as estimativas de valor** obtidas das simulações numa árvore de busca, usando essa informação para direcionar sucessivamente as simulações futuras para trajetórias mais promissoras. Este algoritmo foi o grande responsável pelos avanços que permitiram aos computadores atingir um nível de Grão-Mestre no complexo jogo de Go.

O que é a MCTS?

A MCTS é um algoritmo de planejamento na hora da decisão, executado sempre que o agente encontra um novo estado para selecionar a próxima ação. O processo é iterativo e simula muitas trajetórias a partir do estado atual.

A ideia central é construir gradualmente uma **árvore de busca** onde a raiz é o estado atual. Esta árvore armazena estimativas de valor para os pares estado-ação que parecem mais promissores. Diferente de um algoritmo de rollout simples que descarta tudo, a MCTS usa a "memória"da árvore para fazer uma exploração muito mais inteligente.

O Processo MCTS em 4 Passos

Cada iteração da MCTS, que é repetida milhares de vezes para tomar uma única decisão, consiste em quatro passos. Usaremos a analogia de um jogador a "pensar"na sua próxima jogada de Go.

1. **Seleção (Selection):** A partir da raiz (a posição atual do tabuleiro), o algoritmo atravessa a árvore já construída. Ele usa uma **política de árvore (tree policy)**, como o UCB, para escolher os nós (jogadas) que têm as melhores estimativas de valor, equilibrando exploração e explotação

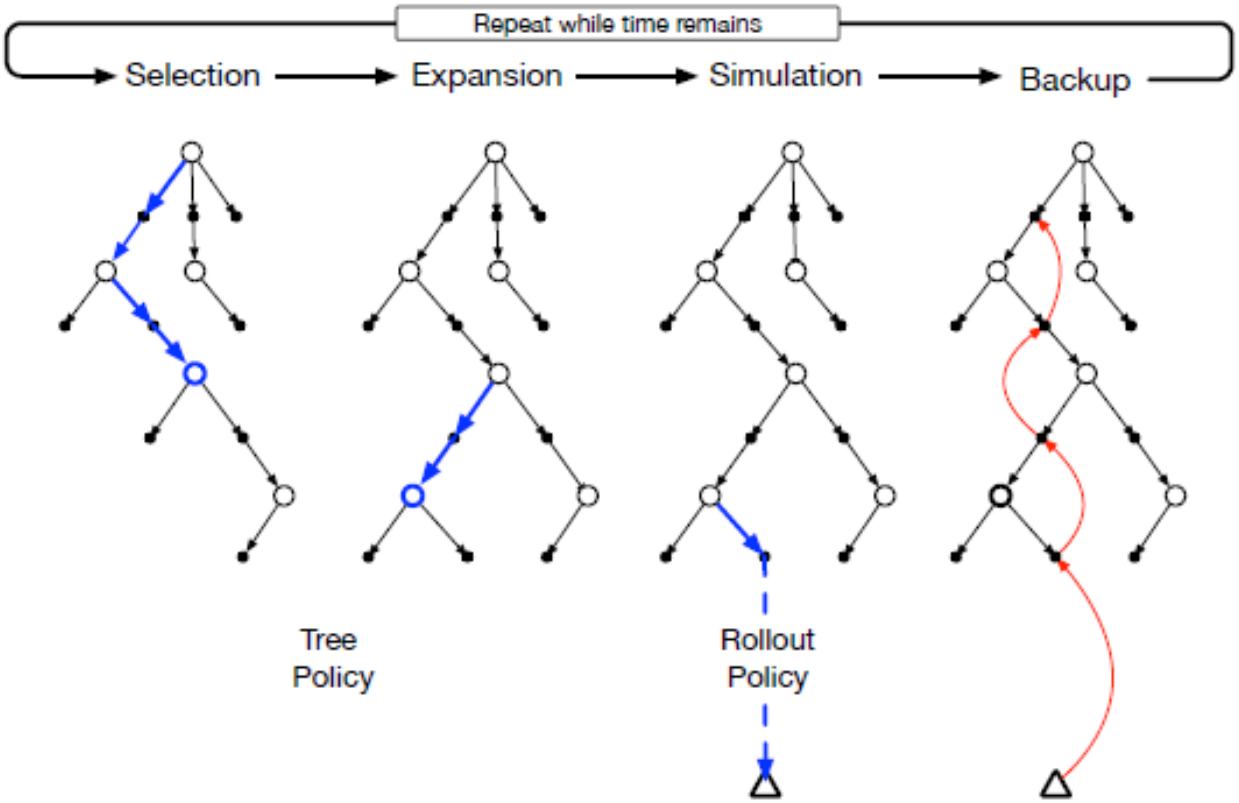

Figura 2: Os quatro passos da MCTS: Seleção, Expansão, Simulação e Backup.

dentro das linhas de jogo já conhecidas. Este processo continua até que um nó folha (uma posição menos explorada) seja alcançado.

2. **Expansão (Expansion):** A partir do nó folha, a árvore é expandida. Uma ou mais novas jogadas (ações) que ainda não foram exploradas a partir daquele estado são adicionadas à árvore como novos nós filhos.

3. **Simulação (Simulation):** A partir de um desses novos nós, o algoritmo para de "pensar"profundamente e entra em modo rápido. Ele executa uma simulação Monte Carlo até ao final do jogo, usando uma **política de rollout** que é computacionalmente leve e rápida. O resultado desta simulação é uma vitória (+1) ou uma derrota (-1).

4. **Backup:** O resultado da simulação (o retorno) é propagado de volta para cima na árvore. Todos os nós e ações que foram visitados durante o passo de **Seleção** têm as suas estatísticas (ex: número de vitórias e de visitas) atualizadas com base no resultado. Por exemplo, se a simulação resultou numa vitória, a "contagem de vitórias"de todos os nós no caminho percorrido aumenta.

Após o tempo disponível para a decisão se esgotar, o agente escolhe a ação a partir do nó raiz que tiver as melhores estatísticas (por exemplo, a mais visitada ou com a maior taxa de vitórias).

Por que a MCTS é Tão Poderosa?

A MCTS combina de forma elegante vários princípios da aprendizagem por reforço:

- **Amostragem e Melhoria Online:** Beneficia-se da estimativa de valor baseada em amostras e da melhoria da política, tal como outros métodos de controlo Monte Carlo.
- **Memória Focada e Eficiente:** Em vez de tentar aprender uma função de valor global para todos os estados, a MCTS cria uma "tabela de consulta"parcial e temporária (a árvore) que é altamente relevante para a decisão atual. Ela aloca memória e computação para os segmentos iniciais das trajetórias que se mostraram mais promissoras.
- **Exploração Guiada:** A MCTS resolve um dos maiores desafios dos algoritmos de rollout. Em vez de simular aleatoriamente, ela usa as estatísticas da árvore para focar as simulações em áreas mais promissoras, tornando a busca exponencialmente mais eficaz.