UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

DANIEL GARDIN GRATTI

Controle ótimo e Aprendizado por reforço

Campinas
02/12/2022

DANIEL GARDIN GRATTI

Controle ótimo e Aprendizado por reforço

Monografia apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado, sob a orientação do(a) Prof. Paulo José da Silva e Silva.

Resumo

Este trabalho tem como objetivo estudar as ideias principais de Aprendizado por Reforço sob a vista de um problema de controle ótimo e apresentado como uma proposta à solução da maldição da dimensionalidade, que surge em problemas de programação dinâmica. Ao longo do texto fazemos uma revisão de programação dinâmica para motivar a necessidade das técnicas de aprendizado por reforço e, por fim, apresentamos alguns resultados conhecidos.

Abstract

The main goal of this text is to study the main ideias behind Reinforcement Learning from the optimal control point of view. It can be viewed as a possible solution for the curse of dimensionality that arises from dynamic programming problems. We introduce a background for dynamic programming to motivate such methods and, finally, we present some well known techniques.

Conteúdo

1 Introdução

Aprendizagem por reforço surge num contexto de controle ótimo, durante a década de 50, em que o objetivo é otimizar um processo em que decisões, chamadas de controles, são escolhidas em estágios, gerando novos resultados e pagando um custo para realizá-la. O resultado da aplicação de várias combinações de controles é um problema não trivial de se prever e muitas vezes não é possível obter uma forma analítica para determinar um controle que direciona para o custo mínimo. Para resolver esta categoria de problemas, durante a década de 50, Bellman [1952] introduziu técnicas de Programação Dinâmica que resolvem tais problemas exatamente, valorando cada possível resultado conforme os melhores controles encontrados até então.

Uma grande classe de problemas podem ser resolvidos por programação dinâmica de forma eficiente. No entanto, técnicas clássicas de programação dinâmica podem não ser suficientes conforme o número de possíveis configurações e controles cresce. Esta barreira foi batizada por Bellman de maldição da dimensionalidade. Para resolver tal problema, técnicas de aproximação da solução ótima foram desenvolvidas sob o nome de programação dinâmica aproximada, em que o objetivo torna-se encontrar uma distribuição de probabilidade que mapeia as condições atuais do problema - chamadas de estado do problema - para possíveis controles a serem aplicados com uma certa probabilidade do controle selecionado ser ótimo. Este conjunto de técnicas, então apelidado de Aprendizado por reforço, foi chamando atenção devido a seu sucesso, primeiramente com o artigo de Tesauro [1995], em que foram utilizadas em conjunto com outras técnicas de aprendizado de máquina para resolver o jogo de gamão. Mais recentemente, os trabalhos de Scherrer et al. [2015] e Silver et al. [2018] mostraram que a aprendizagem por reforço tem seu sucesso atestado ao vencer de humanos habilidosos nessas tarefas.

Apesar de aprendizado por reforço ter se tornado uma subárea do aprendizado de máquina, seu funcionamento é intrinsicamente distinto das demais. Ao contrário das técnicas de aprendizado supervisionado e não-supervisionado, em que o objetivo é inferir uma função de classificação ou agrupamentos a partir de dados já disponíveis para treino, no aprendizado por reforço não há dados coletados *a priori*. O objetivo é realizar uma tarefa a partir de controles. Já os dados a serem correlacionados são obtidos durante

o treinamento. Formalmente, um problema de aprendizado por reforço possui 3 conceitos principais: O ambiente, o agente e a política. O ambiente é um sistema dinâmico que revela ao agente informações sobre sua condição atual. Chamamos tal conjunto de informações de estado do sistema. Um agente é o controlador do sistema, ele pode modificar seu estado atual a partir de possíveis controles aplicados ao sistema e a si mesmo. Após cada controle, o ambiente retorna ao agente o novo estado em que ele se encontra e uma recompensa ou custo da transição. O objetivo geral do problema é encontrar uma política que guia o agente até um estado final com o máximo de recompensa ou mínimo de custo possível. As técnicas de aprendizado por reforço utilizam desta estrutura Ambiente-Agente para explorar possibilidades e tirar proveito do que já foi aprendido.

Este trabalho tem como objetivo introduzir técnicas clássicas de aprendizagem por reforço, sob a perspectiva de controle ótimo, motivar a necessidade de técnicas de aproximação de solução devido a problemas que sofrem da maldição da dimensionalidade, introduzí-las, fazendo uso de outras técnicas de aprendizado supervisionado.

2 Programação Dinâmica Exata

2.1 Introdução

A aprendizagem por reforço utiliza de uma base teórica que parte das ideias de programação dinâmica para modelar seus problemas. Um problema de programação dinãmica é baseado de 2 principais hipóteses: As decisões são feitas por estágios por meio de um sistema dinâmico, ou seja, os estados do sistema são instâncias $x_t \in \mathcal{X}$, onde $\mathcal{X}$ é o espaço de estados, de um tempo t em que o próximo estado x_{t+1} é obtido por

$$x_{t+1} = f(x_t, u_t, \omega_t).$$

Onde $u_t \in \mathcal{U}$ é um controle selecionado no tempo t no espaço de ações $\mathcal{U}$, $\omega_t \in \Omega$ é um parâmetro aleatório utilizado em problemas estocásticos e f é uma função de transição $f : \mathcal{X} \times \mathcal{U} \times \Omega \to \mathcal{X}$ que caracteriza o ambiente. No caso de problemas estocásticos, modelamos o sistema regido por f como um processo de decisão de Markov (MDP). Neste modelo, ω_t pode depender do estado atual x_t e da ação escolhida u_t, mas não de

estados anteriores. Portanto, estamos interessados apenas nos problemas que satisfazem $p(x_{t+1}|x_0, u_0, \ldots, x_k, u_k) = p(x_{t+1}|x_t, u_t)$. Essa condição garante que o problema seja consistente e é necessária para o desenvolvimento a seguir.

A outra suposição é de que, a cada decisão feita, uma recompensa dada pela transição de estados pela função $r : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ é adicionada a sua recompensa total de forma aditiva, então a recompensa total obtida durante n estágios é de

$$J(x_0; \mathbf{u}) = \mathbb{E}\left\{\sum_{t=0}^{n} r(x_t, u_t)\right\}, \tag{1}$$

Onde $\mathbf{u} = \{u_1, \ldots, u_n\}$ é uma sequência de controles aplicados que obtêm a sequência de estados $\{x_1, \ldots, x_n\}$ e $\mathbb{E}\{\cdot\}$ indica o valor esperado. Caso o problema não possua um número finito de estágios ou seu horizonte é indefinido e o problema dura até um estado terminal ser alcançado, uma abordagem descontada pode ser utilizada, onde um fator de desconto $\gamma \in (0, 1)$ é aplicado

$$J(x_0; \mathbf{u}) = \lim_{n\to\infty} \mathbb{E}\left\{\sum_{t=0}^{n} \gamma^t r(x_t, u_t)\right\}. \tag{2}$$

Neste caso, se r é limitada, $J(x_0; \mathbf{u})$ converge e está bem definida para todo $x_0 \in \mathcal{X}$. O objetivo é encontrar uma política, uma função $\mu : \mathcal{X} \to \mathcal{U}$ tal que

$$J_\mu(x_0) = J(x_0; \mu) = \max_{\mathbf{u}} J(x_0; \mathbf{u}). \tag{3}$$

Se a igualdade acima vale para uma política μ para todo $x_0 \in \mathcal{X}$, então μ é dita política ótima, denotada por μ^* e $J(x_0)$ também é dita ótima, denotada por $J^*(x_0)$. A programação dinâmica tenta obter esta política por meio do princípio de otimilidade de Bellman

2.2 Princípio de Otimalidade

Proposição 1 (Princípio de Otimilidade de Bellman)**.** *Seja μ^* uma política ótima e $\mathbf{u}^* = \{u_0^*, \ldots, u_n^*\}$ uma sequência de ações ótimas aplicadas a partir de um estado inicial x_0 qualquer, obtendo uma sequência de estados $\{x_1^*, \ldots, x_n^*\}$, então o subproblema onde começamos a partir do estado x_k^* num tempo k qualquer*

$$\max_{u_k,\ldots,u_n} \mathbb{E}\left\{\sum_{t=k}^{n} \gamma^{t-k} r(x_t, u_t)\right\}$$

tem solução igual aos $n-k+1$ *últimos controles ótimos* $\{u_k^*, \ldots, u_n^*\}$*. Em outras palavras, se* μ^* *oferece solução para um caminho,* μ^* *é solução para qualquer subproblema cauda do caminho.*

A demonstração do princípio anterior vem de que se o subproblema começando de x_k^* tivesse uma seleção de controles cuja recompensa esperada fosse melhor que do problema original, então substituindo os controles da solução original resultaria numa recompensa esperada maior. Como a recompensa esperada era máxima, pois μ^* é ótima por hipótese, então uma contradição é evidente e, portanto, o princípio deve valer.

Com o princípio de otimalidade e a próxima proposição é possível obter um algoritmo para gerar uma política ótima exata.

Proposição 2. *Se* x^* *é um estado terminal, então* $J(x^*) = r(x^*, u) = R(x^*)$ *para todo* $u \in \mathcal{U}$

Este resultado é trivial. Se x^* é um estado terminal então, por definição, o sistema para e não há mais nenhuma recompensa a coletar e portanto a recompensa não depende de nenhum estado senão de si mesmo. Se os estados terminais estão bem definidos, inclusive para a função ótima, os estados que alcançam os estados terminais podem ser calculados de forma exata, tomando o máximo em 1 passo. O teorema a seguir mostra esta relação recursiva dos estados

Teorema 2.1 (Equação de Bellman)**.** *Seja* $x_k \in \mathcal{X}$*. A maior recompensa esperada para este estado é igual a*

$$J^*(x_k) = \max_{u_k} \mathbb{E}\left\{r(x_k, u_k) + \gamma J^*(x_{k+1})\right\}, \tag{4}$$

Onde a esperança contabiliza todos os seguintes estados x_{k+1} *alcançáveis por* x_k *a partir de pelos menos um controle. A equação recursiva acima é chamada de* **Equação de Bellman**.

Demonstração. Realizamos a demonstração do teorema por indução para os problemas de horizonte finito ou indefinido, em que existe pelo menos um estado terminal que é

atingido em um número finito de estágios. Pela preposição 2, se x_n é estado terminal para algum n, então $J^*(x_n)$ pode ser obtido diretamente, independentemente de u. Caso contrário, se x_k não é terminal, então $J^*(x_k)$, por definição, pode ser escrito como

$$J^*(x_k) = \max_{u_k,\dots,u_{N-1}} \mathbb{E}\left\{\sum_{t=k}^{n} \gamma^{t-k} r(x_t, u_t)\right\}.$$

Suponha, por indução, que para todo estado x_{k+1} alcançado por x_k por um único controle, o valor de $J^*(x_{k+1})$ já foi calculado de forma exata. A partir da definição de J^*, podemos fazer

$$\begin{aligned}
J^*(x_k) &= \max_{u_k,\dots,u_{n-1}} \mathbb{E}\left\{\sum_{t=k}^{n} \gamma^{t-k} r(x_t, u_t)\right\} \\
&= \max_{u_k,\dots,u_{n-1}} \mathbb{E}\left\{r(x_k, u_t) + \sum_{t=k+1}^{n} \gamma^{t-k} r(x_t, u_t)\right\} \\
&= \max_{u_k}\left[\mathbb{E}\left\{r(x_k, u_t)\right\} + \max_{u_{k+1},\dots,u_{n-1}} \mathbb{E}\left\{\gamma \sum_{t=k+1}^{n-1} \gamma^{t-(k+1)} r(x_t, u_t)\right\}\right] \\
&= \max_{u_k}\left[\mathbb{E}\{r(x_k, u_k)\} + \mathbb{E}\{\gamma J^*(x_{k+1})\}\right] \\
&= \max_{u_k} \mathbb{E}\{r(x_k, u_k) + \gamma J^*(x_{k+1})\}.
\end{aligned}$$

Por hipótese, como $J^*(x_{k+1})$ já foi calculado de forma exata para todo x_{k+1} alcançável por x_k, $J^*(x_k)$ também pode ser obtido de forma exata pela equação acima. Como todo estado x alcança algum estado terminal x^*, na base da indução por um número finito de controles, $J^*(x)$ pode ser calculado para todo $x \in \mathcal{X}$ a partir da equação recursiva. □

Proposição 3. *Suponha que $J^*(x)$ é ótimo para todo $x \in \mathcal{X}$, então a política obtida por*

$$\mu(x_k) \in \operatorname*{argmax}_{u_k} \mathbb{E}\{r(x_k, u_k) + \gamma J^*(x_{k+1})\} \tag{5}$$

é política ótima

2.3 Fatores Q

Os resultados anteriores mostram um algoritmo para obtenção de uma função de valor J ótima e de uma política ótima, no entanto, J^* não codifica a direção da ação ótima, necessitando de avaliações excessivas dos termos $\mathbb{E}\{r(x_k, u_k) + J^*(x_{k+1})\}$ para sua obtenção. Uma formulação alternativa pode definir este termo como uma parcela, ou fator, do valor ótimo

$$Q^*(x_k, u_k) = \mathbb{E}\{r(x_k, u_k) + \gamma J^*(x_{k+1})\}. \tag{6}$$

Os fatores Q são justamento os termos envolvidos na maximização da equação de Bellman. Métodos que utilizam desses Q-fatores são conhecidos como Q-Learning e têm a vantagem de resgatarem a ação ótima e a função J^*, já que

$$J^*(x_k) = \max_{u_k} Q^*(x_k, u_k). \tag{7}$$

E a poliica ótima pode ser facilmente obtida pela maximização

$$\mu(x_k) \in \operatorname*{argmax}_{u_k} Q^*(x_k, u_k) \tag{8}$$

Além disso, a equação de Bellman pode ser reescrita em termos apenas de Q-fatores

$$Q^*(x_k, u_k) = \mathbb{E}\left\{ r(x_k, x_{k+1}) + \gamma \max_{u_{k+1}} Q^*(x_{k+1}, u_{k+1}) \right\}. \tag{9}$$

Resolver as maximizações para todos os estados e ações retorna os fatores-Q ótimos, assim como os métodos com funções de valor, no entanto, os fatores-Q preservam as informações de ações, não necessitando de uma segunda avaliação para obter a ação que retorna o máximo além de uma simples maximização.

3 Aprendizagem por Reforço

3.1 Introdução

O algoritmo introduzido pela aplicação da equação de Bellman funciona bem para sistemas pequenos com pouco número de estados possíveis, no entanto, o algoritmo não é escalável. Considere um problema determinístico simples com $n = |\mathcal{X}|$ possíveis estados e $m = |\mathcal{U}|$ possíveis ações para cada estado. A cada estado visitamos no máximo m novos estados, um para cada possível ação. Como este número cresce para cada estado avaliado, avaliamos pelo menos $O(m^N)$ ações no total, no entanto, cada avaliação ocorre pelo menos uma vez para cada estado, já que todos os estados têm de ser atualizados, o número de maximizações feitas pela equação de Bellman é, no máximo, da ordem de $O(nm^N)$ onde N é uma cota superior para o número de estágios até um estado terminal. Para problemas estocásticos, este número cresce ainda mais, já que a avaliação da esperança pode ser computacionalmente custosa.

Desse modo, o método proposto na seção anterior não é eficiente para problemas maiores, este comportamento é o que se chama de maldição da dimensionalidade, em que problemas mais complexos não podem ser resolvidos apenas com programação dinâmica exata, ou simplesmente não é possível calcular a recompensa esperada pois não se conhece a dinâmica do sistema. Essas dificuldades motivam a proposta de novas técnicas sub-ótimas que balanceiem entre tempo de execução e performance adequada.

As duas abordagens gerais em aprendizado de máquina baseado em programação dinâmica aproximada são a aproximação em espaço de valor e a aproximação em espaço de política, a diferença entre as duas abordagens está no objeto matemático que será aproximado. Na aproximação em espaço de valor, nosso objetivo é aproximar J^* a partir de uma função $\tilde{J}$ de modo que $\tilde{J}(x) \approx J^*(x)$ para todo estado $x \in \mathcal{X}$, enquanto na aproximação em espaço de política quem é aproximado é uma política $\tilde{\mu}$ otimizada sobre uma classe de políticas parametrizadas.

3.2 Aproximação em Valor

Proposição 4. *Uma função $\tilde{J}$ é dita uma função de valor sub-ótima para um problema se*

$$\tilde{J}(x) \leq J^*(x) \qquad \forall x \in \mathcal{X}. \tag{10}$$

Esta função de valor introduz uma política sub-ótima $\tilde{\mu}$ definida por

$$\tilde{\mu}(x_k) \in \underset{u_k}{\operatorname{argmax}} \mathbb{E}\{r(x_k, u_k) + \tilde{J}(x_{k+1})\}. \tag{11}$$

De forma equivalente, os Q-fatores podem ser empregados para decompor a função $\tilde{J}$ em ações. A Aproximação de Valor tem como objetivo iterar sobre o espaço de funções de valor obtendo aproximações de J^* cada vez melhores. Isso é feito normalmente parametrizando a função $\tilde{J}$ e encontrando os parâmetros de forma a $\tilde{J}$ se aproximar de J^* em todo domínio. Estes métodos dependem muito da qualidade da aproximação da função sub-ótima, para balancear a dependência apenas da função de valor, algumas técnicas de maximização mais rebuscadas podem ser utilizadas, por exemplo, uma antecipação em múltiplos passos

3.3 Antecipação em múltiplos passos

A ideia da antecipação em múltiplos passos (Multistep Lookahead) é de que, se $\tilde{J}$ não é uma boa aproximação, então considerar a política em 1 passo adiante

$$\mu(x_k) \in \underset{u_k}{\operatorname{argmax}} \mathbb{E}\{r(x_k, x_{k+1}) + \gamma \tilde{J}(x_{k+1})\}.$$

Pode gerar resultados tão ruins quanto a qualidade de $\tilde{J}$, já que a maximização sobre o "futuro"ocorre muito cedo, considerando apenas as recomepensas imediatas $r(x_k, x_{k+1})$. Avaliar mais casos no presente, maximizando as recompensas nestes casos, e deixando a aproximação para valores no futuro é a ideia principal do lookahead. Um Lookahead de ℓ passos é caracterizado com a seguinte equação de Bellman modificada

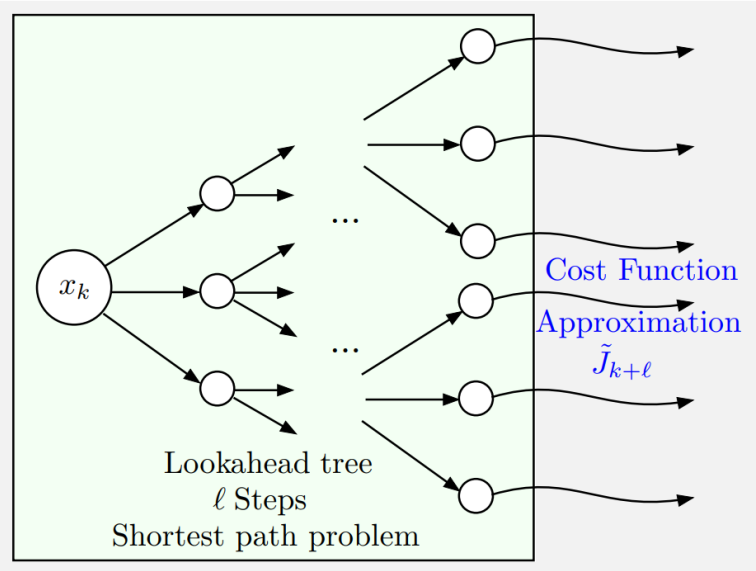

Figura 1: Visualização da Árvore de busca Lookahead. Bertsekas [2019]

$$\tilde{J}(x_k) = \max_{u_k,\ldots,u_{k+\ell-1}} \mathbb{E}\left\{\sum_{t=0}^{\ell-1} \gamma^t r(x_{k+t}, x_{k+t+1}) + \gamma^\ell \tilde{J}(x_{k+\ell})\right\}. \tag{12}$$

Neste caso, um árvore de busca de profundidade ℓ é criada, onde os nós folhas são os estados futuros $x_{k+\ell}$, são aproximados pela função $\tilde{J}$, como a maximização leva em conta ℓ estágios no futuro, o valor de $\tilde{J}(x_k)$ fica menos dependente da própria função $\tilde{J}$ para ser avaliada, se apoiando em maximizações locais mais concretas, o que deve melhorar a performance de uma política

$$\tilde{\mu}(x_k) \in \operatorname*{argmax}_{u_k,\ldots,u_{k+\ell-1}} \mathbb{E}\left\{\sum_{t=0}^{\ell-1} \gamma^t r(x_{k+t}, x_{k+t+1}) + \gamma^\ell \tilde{J}(x_{k+\ell})\right\}, \tag{13}$$

mas sacrificando poder computacional - pois a árvore de busca lookhead é cara computacionalmente - para obter melhores resultados.

3.4 Aproximação Paramétrica

O objetivo da aproximação por valor é escolher uma função que aproxima a função valor ótimo sobre alguma suposição da classe de funções de J^*. Para isso, para-

metrizamos $\tilde{J}(x;\theta)$ em que os pesos θ selecionam uma função de uma classe pré-definida. Corrigimos θ de forma a melhor encaixar na função ótima. Esta é uma tarefa complicada, pois não conhecemos o formato de J^* *a priori*, mas sabemos seu comportamento pela equação de Bellman. A ideia é treinar os parâmetros θ utilizando de estados explorados e tentando generalizar o resultado da transição de um grande número de estados para a função parametrizada. A parametrização pode ocorrer de várias formas, técnicas mais atuais utilizam redes neurais para obter uma classe de funções maior, com o benefício de aprender atributos, melhores que atributos manualmente escolhidos, de forma automática, como discutido nos trabalhos de Mnih et al. [2013] e Krizhevsky et al. [2017].

Proposição 5. *Seja $x \in \mathcal{X}$ um estado do problema e seja $\phi : \mathcal{X} \to \mathbb{R}^n$ uma função, idealmente injetiva, que codifica um estado no espaço $\mathbb{R}^n$*

A construção de ϕ possui os mesmos problemas de codificação de estado e na verdade, o espaço $\phi(\mathcal{X})$ é um espaço de estados, a diferença está em que as componentes de $\phi(x)$ podem, ou não, possui explicabilidade direta de seus valores, uma vez que apenas representam um estado por um vetor numérico. Se ϕ é obtido de forma manual, é natural que suas componenetes representem características físicas dos estados, como pontuação, atributos espaciais ou de posição, entre outros, que são escolhidos como atributos que se acredita serem importantes para uma boa função valor. A partir desta função, é possível realizar a aproximação no espaço de atributos ao invés do espaço de estados. Um exemplo é uma atribuição linear com os atributos construídos manualmente.

$$\tilde{J}(x,\theta) = \tilde{J}(\phi(x),\theta) = \phi(x)^T\theta.$$

O treinamento então é feito por meio de uma minimização de quadrados mínimos de amostras $\{(x^{(s)}), \beta^{(s)})\}_{s=1}^{q}$, onde β^s pode ser obtido de diversas formas - com dados especializados de soluções ótimas já conhecidas, ou obtidas adaptativamente, como ocorre nas técnicas de Temporal Difference, por Andrew [1998] e SARSA, por Rummery and Niranjan [1994]. Eles consideram, respectivamente, $\beta^{(s)} = r^{(s)} + \gamma\hat{J}(x'^{(s)})$ e $\beta^{(s)} = r^{(s)} + \gamma\hat{Q}(x'^{(s)}, u'^{(s)})$, em que $r^{(s)}$ e $x'^{(s)}$ são, respectivamente, a recompensa e o estado obtidos no estado $x^{(s)}$

$$\min \sum_{s=1}^{q} \frac{1}{2}(\beta^{(s)} - \tilde{J}(x^{(s)}, \theta))^2. \tag{14}$$

A minimização é feita por gradiente descendente estocástico, que garante maior convergência em áreas distantes do ótimo, obtendo um resultado próximo do ótimo (Sutton and Barto [2018]).

3.5 Q-Learning

A ideia de parametrizar $\tilde{J}$ é natural para aproximação em espaço de valor, no entanto, é possível utilizar dos Q-fatores, definidos na seção 2.3, como as funções a serem paremetrizadas, por exemplo, por uma arquitetura linear

$$\tilde{Q}(x_k, u_k; \theta) = \phi(x_k, u_k)^T \theta. \tag{15}$$

Neste caso, os atributos obtidos por ϕ dependem também da ação a ser tomada naquele estado. O Q-Learning utiliza da minimização dos parâmetros θ, mas por meio de uma regra para estimar os valores de β^s em cada amostra. Como sabemos que os Q-fatores têm de satisfazer a equação (6), então é natural minimizar

$$\left(\mathbb{E}\{r(x_k, x_{k+1}) + \gamma \max_{u_{k+1}} \tilde{Q}(x_{k+1}, u_{k+1}; \theta)\} - \tilde{Q}(x_k, u_k; \theta)\right)^2. \tag{16}$$

O cálculo da esperança pode ser feito por meio de árvores de busca, como árvores de Monte Carlo (Coulom [2006]) ou simples amostragem, neste último caso, realizamos a minimização tomando experiências $\{(x^{(s)}, u^{(s)}, r^{(s,)}x'^{(s)})\}_{s=1}^{q}$ obtidas durante uma ou mais rodadas e minimizamos

$$\min_{\theta} \sum_{s=1}^{q} \left(r^s + \gamma \max_{u} \tilde{Q}(x'^s, u; \theta) - \tilde{Q}(x^s, u^s; \theta)\right)^2. \tag{17}$$

Esse método converge para a solução exata, conforme o artigo Watkins and Dayan [1992] mostra. Atualmente, com a popularização de redes neurais profundas, elas passaram a ser uma alternativa para parametrizar tais funções. Isso levou a um aumento interessante

na performance, já que as arquiteruras de redes neurais profundas se caracterizam por conseguir realizar uma extração de atributos de forma automática. Métodos que utilizam destas rede são chamados de Deep Q-Learning, e estão presentes em muitos casos de sucesso de reinforcement learning, como os artigos Mnih et al. [2015] e Silver et al. [2016].

3.6 Aproximação em Política

Os métodos anteriores focavam em obter uma política sub-ótima a partir de uma aproximação da função de valor, no entanto, outras técnicas foram propostas que deixavam de calcular o valor dos estados e apenas aprimorar a política que já tinhamos, um passo de cada iteração é pulado desta forma.

Uma forma de aproximação em política é fazer algo semelhante ao processo de parametrização da função de valor, mas desta vez parametrizar uma política. Supomos que a política ótima pertence a uma classe parametrizada de funções de política $\tilde{\mu}(x;\theta)$ e minimizar um custo relacionado a uma má escolha de ações guiada por esta política. Um modo de realizar isso é minimizando para um conjunto de amostras $\{(x^{(s)}, u^{(s)})\}_{s=1}^{q}$,

$$\min_{\theta} \sum_{s=1}^{q} \left(u^s - \tilde{\mu}(x^s;\theta)\right)^2,$$

em que u^s é um controle considerado melhor, obtido a partir de alguma heurística boa ou a partir de uma política baseada em aproximação de valor. Uma vantagem desses métodos sobre os métodos de aproximação em valor é de que o custo gasto para melhorar uma má aproximação de $\tilde{J}$ pode ser evitado devido a avalição de $\tilde{\mu}$ não depender de maximizações e avaliações de esperanças, assim, realizar buscas em árvore guiada pela política aproximada é muito mais barata, por exemplo, por Rollout.

3.7 Rollout

O rollout é uma técnica de aprimoramento de política que baseaia-se em realizar uma avaliação de um estado ou de uma política utilizando uma árvore de busca, por exemplo por antecipação. Ao fim da profundidade da árvore, estados em ℓ estágios adiante devem ser avaliados por meio de uma aproximação $\tilde{J}(x_{k+\ell})$, no entanto, tal função pode não estar disponível, por exemplo, ao realizarmos aproximação em política. Nessas

situações, a avaliação de $\tilde{J}(x_{k+\ell})$ deve se basear em outra abordagem. No Rollout a avaliação é feita a partir de $J_\mu(x_{k+\ell})$, a recompensa obtida a partir do estado $x_{k+\ell}$ aplicando apenas as ações da política μ.

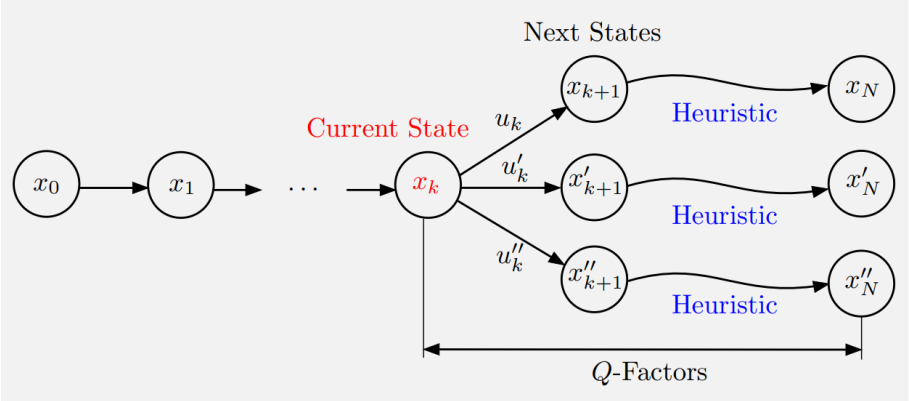

Figura 2: Funcionamento do Rollout em 1 passo. Os estados sucessores são avaliados pela heurística até alcançarem um estado terminal e então os controles são minimizados. Bertsekas [2019]

O algoritmo de rollout inicia a partir de uma política base $\hat{\mu}$ também chamada de heurística base - é uma política sub-ótima conhecida que garante uma solução sub-ótima para o problema. Esta heurística pode utilizar de alguma característica especial do problema, alguma estratégia gulosa ou, ainda, uma política aproximada $\tilde{\mu}$ a ser aprimorada. Um exemplo de heurística conhecida é a heurística do vizinho mais próximo para o problema do caixeiro viajante. Nesta heurística, a cidade vizinha mais próxima ainda não visitada é a cidade escolhida para o caixeiro visitar. Normalmente uma heurística pode vir de um avaliador de estados $H(x)$ que possui um papel semelhante ao da função de valor. Já $\hat{\mu}$ é obtida por

$$\hat{\mu}(x_k) \in \underset{u_k}{\operatorname{argmax}}\, H(x_{k+1}).$$

Aplicando puramente esta heurística chegamos numa solução sub-ótima para o problema. No entanto, é possível aprimorar, a partir dessa heurística, uma política que performa melhor que a própria heurística base. Considere uma política base (ou heurística) $\hat{\mu}$ para um problema de decisão de Markov e suponha que estamos num estado genérico $x_k \in \mathcal{X}$. A política $\tilde{\mu}$ oferece uma escolha sub-ótima para o estado, no entanto, é possível gerar uma nova política que atuará, pelo menos, igual ou melhor que $\tilde{\mu}$ aplicando as ideias de

antecipação.

Para um estado x_k, crie uma árvore de busca de profundidade ℓ, tomando todos os possíveis estados alcançáveis x_{k+1} por pelo menos um controle u_k, o valor de u_k pode ser obtido por meio dos fatores Q, utilizando a equação

$$Q^*(x_k, u_k) = \mathbb{E}\left\{\sum_{t=0}^{\ell-1} \gamma^t r(x_{k+t}, u_{k+t}) + \gamma^\ell J^*(x_{k+\ell})\right\}$$

Como J^* não é conhecido, mas uma heurística para $x_{k+\ell}$ sim, podemos aproximar Q^* por

$$\bar{Q}(x_k, u_k) = \mathbb{E}\left\{\sum_{t=0}^{\ell-1} \gamma^t r(x_{k+t}, u_{k+t}) + \gamma^\ell H(x_{k+\ell})\right\}$$

A política $\bar{\mu}$ gerada pela maximização destas aproximações deve atuar melhor ou igual a $\hat{\mu}$.

$$\bar{\mu} \in \operatorname*{argmax}_{u_k} \bar{Q}(x_k, u_k)$$

Na prática, a avaliação da esperança é muito custosa para obter um ganho computacional na maximização dos termos aproximados pela heurística. Muitas vezes é muito mais vantajoso, se disponível, simular várias possibilidades de estados em alguma profundidade ℓ e a partir desta profundidade apenas aplicar a heurística $\hat{\mu}$ até que um estado terminal seja atingido e então contabilizar a recompensa esperada. Esta é a ideia por trás de um dos mais famosos algoritmos do tipo em aprendizado por reforço, como Monte Carlo Tree Search (MCTS), presente em algoritmos conhecidos, como AlphaZero e AlphaGo.

3.8 Actor-Critic

Aproximação em valor e aproximação em política, apesar de possuírem ideias diferentes, podem ser utilizados em conjunto, conforme pode ser visto nas ideias do Rollout de aprimoramento de política. Esta ideia de unir aproximação em valor e aproximação em política é também utilizada em métodos chamados Actor-Critic.

O nome dado a este conjunto de métodos se dá devido aos nomes dados às parametrizações de $\tilde{Q}(x, u; \xi)$ e $\tilde{\mu}(x; \theta)$. O algoritmo que realiza a avaliação de política,

utilizando funções de valor é chamado de crítico (*critic network*) e é responsável pela aprimoração da política, enquanto o algoritmo que realiza as ações a partir de uma parametrização de política é chamado de ator (*actor network*). A ação das duas partes é utilizada para dar origem aos métodos *Actor-Critic*, que estão atualmente no estado da arte de aprendizado por reforço (Chu et al. [2019]). Uma diferença principal quanto aos métodos anteriores é a forma em que a política é vista. Nesta abordagem, uma política é uma função $\pi : \mathcal{X} \times \mathcal{U} \to [0, 1]$ que representa uma distribuição de probabilidade no estado de ações. Uma vantagem desta representação é a obtenção de $\nabla J_{\pi_\theta}(x)$, de acordo com o teorema do gradiente de política (A demonstração completa pode ser encontrada no capítulo 13 de Sutton and Barto [2018])

$$\nabla J_{\pi_\theta}(x) = \sum_{x \in \mathcal{X}} \mu_\theta(x) \sum_{u \in \mathcal{U}} \nabla \pi(x, u) \tilde{Q}(x, u; \xi). \tag{18}$$

A principal ideia do Actor-Critic é realizar o treino das redes a partir de exemplos gerados progressivamente. O método inicia com uma política e função de valor base e começa a gerar amostras de experiência $\{(x^{(s)}, u^{(s)}, r^{(s)}, x'^{(s)})\}_{s=1}^{n}$, calculados pela rede de ação (ator), que são utilizados para atualizar $\tilde{Q}(x, u; \xi)$ de forma semelhante ao Q-Learning. Com o teorema acima e a partir da experiência e dos Q fatores melhorados, os pesos θ na política são atualizados. Essa etapa é chamada de aprimoramento de política, realizada pela rede crítica. Com a política aprimorada, realiza-se uma nova amostragem de estados até convergir para uma política ótima. Outros métodos Actor-Critic foram propostos e têm performance superior à outras estratégias, como o artigo Haarnoja et al. [2018] discute.

4 Conclusão

Muitas das ideias de aprendizado por reforço vem diretamente da programação dinâmica através dos resultados de Bellman para otimalidade de sistemas dinâmicos discretos, generalizado para processos de decisão de Markov. Como visto, obter exatamente a solução ótima para um problema pode não ser fácil - em muitas vezes impossível de se obter de forma eficiente. Deste modo, a necessidade de resolver problemas mais complexos trouxe a necessidade de desenvolver métodos que poderiam resolvê-los de forma

aproximada.

Esses métodos tornaram-se uma área própria, chamada Aprendizado por reforço. Ela possui várias abordagens para resolver problemas complexos. Seu poder se mostra quando unido a outras técnicas, como aprendizado supervisionado e aprendizado profundo, ou até mesmo, com a composição de suas próprias estratégias, como exemplificado pelo Actor-Critic.

Impulsionado por problemas cada vez mais complexos em diversas áreas, como em jogos digitais e máquinas autônomas, e em diferentes formulações, como no caso em que há mais de um agente e em problemas com estados contendo informações parciais do estado todo, a aprendizagem por reforço cresce em número de diferentes estratégias para cobrir a grande dificuldade dos problemas que são propostos. Essas técnicas são teoricamente sólidas, mas também tem sua eficácia comprovada por diversos casos recentes de problemas complexos resolvidos com sucesso.

Referências

Alex M Andrew. Reinforcement learning:: An introduction. *Kybernetes*, 1998.

Richard Bellman. On the theory of dynamic programming. *Proceedings of the national Academy of Sciences*, 38(8):716–719, 1952.

Dimitri Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019.

Tianshu Chu, Jie Wang, Lara Codecà, and Zhaojian Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):1086–1095, 2019.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, and Matthieu Geist. Approximate Modified Policy Iteration and its Application to the Game of Tetris. *Journal of Machine Learning Research*, 16(49):1629–1676, 2015. ISSN 1533-7928.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687. doi: 10.1038/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144, December 2018. doi: 10.1126/science.aar6404.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, March 1995. ISSN 0001-0782. doi: 10.1145/203330.203343.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.