Aula 7

Aproximação da função-valor

Os métodos tratados nas aulas anteriores podem ser vistos como atualizações da estimativa da função-valor, movendo o valor estimado em um estado s para um "alvo" u. Denotamos isso por

$$s \mapsto u,$$

onde s é o estado que está sendo atualizado e u é o valor alvo. Essa perspectiva exige um método de aproximação que suporte aprendizado incremental e possa lidar com alvos não estacionários.

Nos métodos tabulares de aprendizado por reforço, mantemos uma estimativa separada para cada estado s, denotada por $v(s)$. Isso funciona bem quando o número de estados é pequeno, mas se torna inviável em ambientes com grandes ou infinitos espaços de estados.

A solução é substituir a tabela por uma função paramétrica que aproxima a função-valor. Essa função é denotada por $\hat{v}(s, \mathbf{w})$, onde $\mathbf{w}$ é um vetor de parâmetros ajustáveis.

Erro Quadrático Médio do Valor ($\overline{VE}$)

Com função-valor em forma tabular, podíamos convergir exatamente para a função verdadeira e os erros em diferentes estados eram independentes. Mas com aproximação, uma atualização em um estado afeta outros estados e não é possível acertar todos simultaneamente. Devemos então decidir quais estados são mais importantes. Introduzimos uma distribuição sobre estados $\mu(s)$, com $\mu(s) \geq 0$ e $\sum_s \mu(s) = 1$. Definimos o Erro Quadrático Médio do Valor (Mean Squared Value Error, abreviado $\overline{VE}$) como

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

Em tarefas *on-policy* episódicas, $\mu(s)$ é chamada de distribuição *on-policy*. Para definir μ, primeiro definimos

$$\eta(s) = h(s) + \sum_{\bar{s}} \eta(\bar{s}) \sum_a \pi(a \mid \bar{s})\, p(s \mid \bar{s}, a),$$

onde $h(s)$ é a probabilidade de iniciar um episódio em s, e $\eta(s)$ é o tempo médio gasto em s por episódio. Em seguida,

$$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}.$$

Note que $\overline{VE}(\mathbf{w})$ só garante um mínimo local.

Método do Gradiente

Primeiramente vamos revisar o que nós temos. Nosso objetivo é achar uma função $\hat{v}(s, w)$ que prevê qual é o valor de um certo estado dado um conjunto de pesos, os quais estão no vetor w. Dessa forma, podemos utilizar um dos métodos mais comuns de aproximações de funções, o SGD (Stochastic Gradient Descent). Esse método se baseia em pegar a nosso função de erro $[v_\pi(S_t) - \hat{v}(S_t, w)]^2$ e conseguir o gradiente dela em relação ao nosso vetor w. Com isso, conseguimos saber para que direção a nossa função de erro está crescendo, o que nos permite ir na direção oposta, já que queremos que o erro diminua.

Dessa forma, imagine que o nosso agente está interagindo com o ambiente, para cada novo estado que ele vivenciar, ele ajustará o vetor w para diminuir o erro, seguindo a seguinte fórmula:

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

Pela regra da cadeia temos:

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

Adicionamos o hiperparâmetro α, chamado de *learning-rate*, pois queremos poder determinar o quanto cada estado vai influenciar na atualização de w_t para w_{t+1}. Isso é importante, pois queremos criar um agente generalista, então o que ele aprende com um estado individual não deve ter tanto peso, o que deve influenciar realmente o que ele aprende é a soma conjunta de mudanças durante vários estados.

Contudo, olhando com atenção, podemos ver que na equação de atualização do w estamos utilizando $v_\pi(S_t)$, que é exatamente o que queremos prever! Por isso, como não temos esse função, temos que utilizar outro valor no lugar. Voltando para o que vimos nas aulas passadas, podemos pensar em 2 soluções:

A primeira seria utilizar o método de Monte Carlo, e dizer que o valor de $v_\pi(S_t)$ (vamos chamar esse valor de U_t, pois essa é uma aproximação e não necessáriamente o valor real) é a soma das recompensas que tivemos no nosso episódio, de forma que $U_t = G_t$. Isso faz com que a fórmula de atualização de w seja:

$$w_{t+1} = w_t - \alpha[U_t - \hat{v}(S_t, w_t)]\nabla\hat{v}(S_t, w_t) \tag{3}$$

Contudo, vimos que os métodos de Monte Carlo não são geralmente a melhor escolha, pois temos que esperar até o fim de um episódia para fazermos as nossas atualizações, o que pode levar muito tempo. Por isso qual seria a outra maneira de resolvermos isso?

É aí que entra a segunda opção, os métodos de TD learning, que usam valores guardados de $\hat{v}$ como aproximações para o valor real, de forma que $U_t = R+\gamma\hat{v}(S_{t+1}, w)$ para TD(0). Entretanto, isso

nos traz um problema, pois, se lembrarmos bem, ao utilizarmos a regra da cadeia para conseguirmos a equação 2 da equação 1, nós tinhamos assumido que v_π era independente de w, por isso derivamos daquela maneira, mas, ao utilizar um valor guardado de $\hat{v}$ como U_t nos invalidamos isso. É por isso que os métodos de TD learning que usam SGD são chamados de métodos *semi-gradient*, pois não usam todo o gradiente.

Uma pergunta que pode surgir é: porque simplesmente não calculamos o gradiente também em relação ao valor guardado de $\hat{v}$ que estamos usando para conseguir U_t, assim poderiamos ter um método de gradiente completo para o TD learning. Infelizmente, como estamos utlizando um valor passado de $\hat{v}$ para o estado $t+1$ não podemos calcular o gradiente, pois estamos apenas pegando o valor que foi guardado e não recalculando a função de novo, pois seria necessário os valores de w naquele instante, o que nós não temos, já que atualizamos w continuamente, e seria muito custoso guardar todos os valores de w.

Mesmo com esse adendo, os métodos *semi-gradient* ainda são muito bons para conseguirmos aproximar a nossa função valor, pois compartilham as vantegens do TD learning, que permite que os valores dos estados sejam atualizados de maneira contínua, sem ter que esperar o fim de um episódio, o que geralmente lava a um treinamento mais rápido.

Redes Neurais em RL

Conforme dito no início dessa apostila, nosso objetivo aqui é discutir maneiras para aproximar nossa função valor, utilizando algoritmos variados para esse propósito, no contexto de Reinforcement Learning. Atualmente, uma das estruturas mais poderosas para aproximar funções são as Redes Neurais Artificiais, e, portanto, é interessante analisar seu potencial no nosso caso.

Neste curso, não explicaremos do zero o que são as Redes Neurais (se você não souber, veja esta aula aqui. Para nossos propósitos em Aprendizado por Reforço, basta saber que esses sistemas complexos são capazes de, com base em quantidades grandes de dados, tentar achar padrões intrínsecos a eles e incorporá-los, podendo gerar Outputs que se assemelhem ao dataset (é uma definição limitante, mas a que importa para nós nesse momento).

É possível utilizar Redes Neurais para estimar nossas Funções-Valor $\hat{v}_\pi(s, w)$: basta que nós treinemos ela com s ou $x(s)$ para diversas passagens por vários estados. Uma das principais vantagens advinda disso é a capacidade da Rede em capturar padrões complexos no dataset, o que é impossível para métodos lineares de aproximação. Além disso, Redes Neurais Profundas conseguem extrair features para o seu próprio aprendizado, eliminando a necessidade do feature-engineering. Essa abordagem também é possível para o caso de Funções Ação-Valor $\hat{q}_\pi(s, a, w)$.

É necessário citar algumas desvantagens desse uso. Redes Neurais geralmente requerem muitos dados para chegar em uma aproximação adequada, o que implica que precisaremos ter vários episódios até que os resultados sejam promissores, o que pode ser custoso em alguns casos. Além disso, a própria natureza do Aprendizado por Reforço é um problema: nossos episódios $(S_t, A_t, R_{t+1}, S_{t+1})$ contém dados interrelacionados (isso prejudica o SGD, utilizado para treinar a Rede), e o nossa Loss Function muda constantemente devido ao caráter online do RL, dificultando a convergência.

Memory-based Function Approximation

Ideia. Em vez de ajustar parâmetros de uma forma funcional global e depois descartar os exemplos, métodos *baseados em memória* armazenam exemplos à medida que chegam e, quando é necessário estimar o valor de um estado de consulta, recuperam exemplos relevantes da memória para computar a estimativa (*lazy learning*). São métodos *não-paramétricos*: a forma da aproximação é determinada pelos exemplos e pela regra de combinação usada, com precisão tendendo a melhorar conforme mais exemplos são acumulados. **Aprendizagem local.** Focamos métodos que aproximam o valor *localmente* na vizinhança do estado de consulta, selecionando exemplos cujos estados são os mais relevantes (tipicamente, por distância); após responder a consulta, a aproximação local é descartada.

Exemplos. (i) *Vizinho mais próximo*: retorna o alvo do exemplo cujo estado é o mais próximo do estado de consulta. (ii) *Média ponderada de vizinhos*: recupera vizinhos próximos e retorna a média ponderada de seus alvos, com pesos decrescentes com a distância. (iii) *Regressão localmente ponderada*: ajusta, na vizinhança, uma superfície paramétrica minimizando um erro ponderado por distâncias; a superfície é descartada após o uso.

Por que em RL? Por serem não-paramétricos, não impõem formas predefinidas e podem ganhar acurácia com mais dados. Em RL, onde amostragem por trajetórias é central, métodos locais concentram a aproximação nas regiões realmente visitadas e permitem que novas experiências afetem imediatamente estimativas próximas ao estado atual, sem exigir ajustes incrementais de uma aproximação global.

Memória e dimensionalidade. Armazenar exemplos exige memória proporcional à dimensionalidade do estado por exemplo, e linear no número de exemplos n (nada exponencial em k ou n), embora permaneça a questão de responder consultas suficientemente rápido conforme a base cresce. **Eficiência prática.** Para acelerar busca de vizinhos, empregam-se paralelismo/hardware especial e estruturas como *k-d trees*, que particionam recursivamente o espaço e permitem eliminar grandes regiões durante a busca; em LWR, é necessário também acelerar os cálculos de regressão local e, na prática, adotar esquemas de esquecimento para manter a base sob controle.

Interesse e Ênfase

Motivação: O Problema da Aprendizagem Uniforme

Até agora, os algoritmos que estudamos tratam todas as atualizações de valor com a mesma importância. Se um estado é visitado, o seu valor é atualizado. No entanto, esta abordagem "uniforme"tem uma limitação fundamental, especialmente quando usamos **aproximação de funções**.

- **Recursos Limitados:** Com aproximação de funções, temos muito menos pesos (w) do que estados. Mudar um peso para melhorar a estimativa de um estado irá inevitavelmente afetar (e

talvez piorar) a estimativa de muitos outros estados.

- **Nem todos os estados são igualmente importantes:** Em muitos problemas, estamos mais interessados em ter estimativas precisas para alguns estados do que para outros. Por exemplo, em tarefas com desconto, os estados no início de um episódio têm um impacto muito maior no retorno total.

A questão central é:

Como podemos alocar os nossos recursos limitados de aprendizagem para focar nos estados que mais nos interessam?

A Solução: Introduzindo Interesse e Ênfase

Para direcionar a aprendizagem, o livro introduz dois novos conceitos que nos permitem controlar o processo de atualização de forma mais inteligente.

Interesse (I_t)

O **Interesse** é um número escalar ($I_t \geq 0$) que define o nosso **grau de interesse** em estimar corretamente o valor do estado no tempo t.

- **Função:** Ele define **o que** é importante para o nosso objetivo.
- **Como é usado?** O Interesse pode ser definido de qualquer forma. Por exemplo, podemos definir $I_t = 1$ para o estado inicial de um episódio e $I_t = 0$ para todos os outros. Isto diz ao algoritmo: "O seu único objetivo é acertar o valor do estado inicial, não me importo com o erro nos outros estados.".
- **Impacto:** O Interesse modifica o nosso objetivo de aprendizagem. Em vez de minimizarmos o erro médio em todos os estados, passamos a minimizar o erro médio ponderado pelo nosso Interesse.

Ênfase (M_t)

A **Ênfase** é um número escalar ($M_t \geq 0$) que **multiplica (ou pondera) a atualização** de aprendizagem.

- **Função:** É o **mecanismo** que efetivamente aplica o foco definido pelo Interesse.
- **Como funciona?** Uma Ênfase alta resulta numa grande atualização de pesos; uma Ênfase zero significa que nenhuma atualização ocorre para aquele estado, mesmo que o erro TD seja grande.
- **Relação com o Interesse:** A Ênfase é calculada a partir do Interesse. Um estado só recebe Ênfase se ele, ou os estados que o seguem na trajetória, tiverem Interesse.

As Fórmulas da Aprendizagem Focada

A regra de atualização de n-passos que vimos no Capítulo 9 é modificada para incluir a Ênfase (M_t).

- **Regra Geral de Atualização (Semi-gradiente n-passos com Ênfase):** A atualização para o estado S_t é ponderada pela Ênfase M_t. A atualização é aplicada no tempo $t + n$:

$$w_{t+n} \leftarrow w_{t+n-1} + \alpha \mathbf{M_t}[G_{t:t+n} - \hat{v}(S_t, w_{t+n-1})]\nabla \hat{v}(S_t, w_{t+n-1}) \tag{4}$$

  Fonte: Equação (9.25), página 234.

- **Cálculo da Ênfase (M_t):** A Ênfase é calculada de forma recursiva a partir do Interesse (I_t):

$$M_t = I_t + \gamma^n M_{t-n} \quad (\text{com } M_t = 0 \text{ para } t < 0) \tag{5}$$

  Fonte: Equação (9.26), página 234.

Intuição da Fórmula da Ênfase: A ênfase para atualizar um estado visitado no tempo t depende do nosso interesse nesse exato momento (I_t), mais a ênfase de n passos atrás (M_{t-n}), trazida para o presente (descontada por γ^n). Isto cria uma "cadeia de importância"que se propaga para trás no tempo, garantindo que os estados que levam a estados de alto interesse também recebam ênfase.

Analogia: O Aluno Eficiente

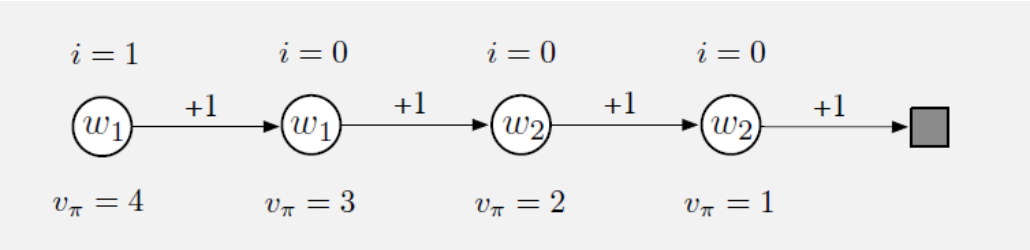
Para entender a interação entre Interesse e Ênfase, vamos usar a analogia de um aluno a estudar para uma prova.

- **Aprendizagem Normal:** O aluno estuda todos os capítulos do livro com a mesma dedicação, tentando minimizar o seu "erro"de conhecimento em todas as páginas.
- **Com Interesse:** O professor avisa: "A matéria mais importante é do Capítulo 5". Esta informação define o **Interesse** do aluno. O seu objetivo agora é apenas dominar estes capítulos.
- **Com Ênfase:** Ao resolver um exercício prático, o aluno erra. Ele precisa de rever a matéria.
  - Ele dá **Ênfase máxima** à revisão do capítulo 5 que o levaram ao erro. A correção do seu conhecimento (a atualização dos pesos) é grande.
  - Ele dá **Ênfase zero** à revisão dos outros capítulos, mesmo que os tenha consultado. Ele não gasta energia a corrigir o seu conhecimento sobre tópicos de baixo interesse.

Conclusão da Analogia: O **Interesse** é o plano de estudo que define o objetivo. A **Ênfase** é o mecanismo que aloca o esforço de correção (aprendizagem) para atingir esse objetivo de forma eficiente.

Exemplo Prático (Exemplo 9.4)

O livro ilustra o poder desta abordagem com um exemplo numérico simples, um processo de recompensa de 4 estados.



- **Cenário:** Os valores verdadeiros dos estados são 4, 3, 2 e 1.
- **Restrição (Aproximação de Função):** Os dois primeiros estados partilham o mesmo peso, w_1, e os dois últimos partilham w_2.
- **Objetivo (Interesse):** Estamos interessados apenas no valor do primeiro estado. Portanto, $I_0 = 1$ e $I_{t>0} = 0$.

Resultados

- **SEM Interesse e Ênfase:** O algoritmo tenta minimizar o erro em todos os estados e converge para $w_1 = 3,5$ (a média de 4 e 3). O valor do nosso estado de interesse fica **errado**.
- **COM Interesse e Ênfase:** O algoritmo foca todo o seu esforço de atualização no primeiro estado e nos estados que o influenciam. Ele converge para $w_1 = 4,0$. O valor do nosso estado de interesse fica **exato**, à custa de uma estimativa menos precisa para os outros estados (dos quais não nos importamos).

Resumo Prático

Conceito	**Pergunta que Responde**	**O que é?**
Interesse (I)	"Onde é que eu me importo em ser preciso?"	Uma forma de **definir o objetivo** da aprendizagem. É um parâmetro que nós fornecemos.
Ênfase (M)	"Quanto esforço de atualização devo aplicar aqui?"	Um **peso que multiplica** a atualização de aprendizagem, calculado com base no Interesse e na dinâmica do problema.

Tabela 1: Resumo prático dos conceitos de Interesse e Ênfase.