

Raciocínio Baseado em Casos (RBC)

- É uma técnica dentro da IA que busca resolver novos problemas buscando soluções que já foram utilizadas para resolver problemas anteriores.
- É uma abordagem de solução de problemas em que, em vez de criar uma solução de problemas em uma abordagem de solução totalmente nova, a pessoa (ou um sistema) procura casos antigos parecidos e reutiliza essas soluções, adaptando-as à nova situação.
- Usa uma base de dados explícita de soluções de problemas para lidar com resolução de novos problemas.
- Faz pesquisa em sua memória e verifica se há um caso semelhante com as características do problema, podendo achar um caso ou diversos que serão utilizados na solução do problema.

Exemplos:

- O Médico lembra-se da história da doença de um outro paciente;
- Um profissional jurídico reforça os seus argumentos com jurisprudências semelhantes
- Programadores com orientação a objetos
 - reuso de código

Um Caso pode representar diferentes tipos de conhecimento e assumir distintas formas de representação. Uma forma simples de representar casos é por meio de um conjunto de pares atributo-valor. Esse conjunto é dividido em dois subconjuntos. O primeiro possui os atributos que descrevem o problema e o segundo, os atributos relacionados com sua solução.

Ciclo de RBC (4 Rs)

1. **RECUPERAR:** experiências similares a partir de situações na memória (faz analogias)
2. **REUTILIZAR:** A partir do caso recuperado é feita a reutilização da solução associada ao caso recuperado. Geralmente a solução é transferida ao novo problema diretamente como sua solução;
3. **REVISAR:** uma revisão é feita quando a solução não pode ser aplicada diretamente ao novo problema;
4. **RETER:** Uma vez que a solução revisada é bem-sucedida, o novo problema, juntamente com sua solução (e o método pelo qual a solução foi alcançada), é armazenado na base de dados de casos como um novo caso. Isso enriquece a memória de casos e aprimora a capacidade futura do sistema de resolver problemas.

Desenvolvimento de um sistema RBC - Raciocínio Baseado em Casos

As etapas mais importantes do processo de desenvolvimento de um sistema RBC são:

- Casos
 - Aquisição do conhecimento
- Indexação
- Recuperação de Casos
 - Similaridade
- Adaptação de Casos - Reutilização
- Aprendizagem

Desvantagens

- Dificuldade em obter casos disponíveis e confiáveis
- Não cobrem todo o domínio
- Não existem bons algoritmos de adaptação
- Exige mais espaço para armazenamento

Quando usar RBC?

- Existe um grande volume de dados históricos
- Os especialistas falam sobre seu domínio dando exemplos
- Os problemas não são completamente formalizáveis
 - fraca compreensão do problema, dificuldade de verbalização

Busca em Espaços de Estados

É uma técnica na IA para resolver problemas.

A ideia é modelar um problema como um conjunto de estados possíveis e as ações que podem levar de um estado a outro.

O objetivo é encontrar uma sequência de ações que transforme um estado inicial em um estado objetivo ou em vários estados objetivos.

A busca em espaços de estados é uma técnica fundamental na Inteligência Artificial (IA) para resolver problemas. A ideia central é modelar um problema como um conjunto de "estados" possíveis e as "ações" que podem levar de um estado a outro. O objetivo é encontrar uma sequência de ações que transforme um "estado inicial" em um "estado objetivo" (ou um dos vários estados objetivos).

Para entender melhor, vamos detalhar os componentes:

- * Estado (State): Um estado é uma representação do ambiente ou do problema em um determinado momento. Ele captura todas as informações relevantes necessárias para descrever a situação atual. Pense nisso como uma "fotografia" do problema. Por exemplo, em um jogo de xadrez, um estado seria a configuração das peças no tabuleiro em um dado momento. No problema do aspirador de pó, um estado poderia ser "aspirador na sala A, sala A suja, sala B limpa".

- * Espaço de Estados (State Space): É o conjunto de todos os estados possíveis que o problema pode assumir. Ele pode ser visualizado como um grafo, onde:

- * Nós (Nodes): Representam os estados individuais.

- * Arestas (Edges): Representam as transições entre os estados, que são causadas pela execução de uma ação.

- * Ação (Action) / Operador: Uma ação é uma operação que, quando aplicada a um estado, transforma-o em outro estado. As ações definem as transições possíveis no espaço de estados. No exemplo do aspirador de pó, as ações poderiam ser "mover para a esquerda", "mover para a direita" ou "aspirar".

- * Estado Inicial (Initial State): É o ponto de partida do problema, o estado em que o agente se encontra no início.

- * Estado Objetivo (Goal State): É o estado (ou conjunto de estados) que o agente deseja alcançar. A busca termina quando um estado objetivo é encontrado.

- * Custo da Ação (Action Cost): Cada ação pode ter um custo associado (por exemplo, tempo, energia, dinheiro). O objetivo da busca pode ser encontrar não apenas uma solução, mas a solução com o menor custo total.

O Processo de Busca

O processo de busca em espaços de estados envolve explorar o grafo de estados para encontrar um caminho do estado inicial ao estado objetivo. Os algoritmos de busca fazem isso expandindo os nós, ou seja, gerando os estados sucessores a partir de um estado atual aplicando todas as ações possíveis.

Exemplos de Algoritmos de Busca:

Existem diversas estratégias de busca, que podem ser classificadas em:

- * Buscas Cegas (Uninformed Search): Não utilizam informações sobre o quão "próximo" um estado está do objetivo. Elas exploram o espaço de estados de forma sistemática.

* Busca em Largura (Breadth-First Search - BFS): Explora todos os nós de um nível antes de passar para o próximo nível. Garante encontrar o caminho mais curto (em termos de número de ações) se os custos das ações forem iguais.

* Busca em Profundidade (Depth-First Search - DFS): Explora o mais profundamente possível em um ramo antes de retroceder. Pode encontrar uma solução rapidamente se ela estiver em um ramo profundo, mas não garante a melhor solução.

* Buscas Informadas (Informed Search) / Heurísticas: Utilizam informações adicionais (heurísticas) para guiar a busca em direção ao objetivo, tornando-a mais eficiente.

* Busca Gulosa (Greedy Best-First Search): Expande o nó que parece mais próximo do objetivo de acordo com uma função heurística.

* **Busca A (A Search):**** É um dos algoritmos de busca mais populares e eficientes. Combina o custo do caminho percorrido até o nó atual com uma estimativa heurística do custo restante até o objetivo, garantindo o caminho mais curto se a heurística for admissível. (ALGORITMO A ESTRELA)

Por que é importante?

A busca em espaços de estados é um conceito central em IA porque muitos problemas podem ser formulados dessa maneira. Ao transformar um problema complexo em um grafo de estados e transições, podemos aplicar algoritmos de busca genéricos para encontrar soluções. Isso é amplamente utilizado em diversas áreas da IA, como:

* Planejamento: Encontrar uma sequência de ações para um robô alcançar uma meta.

* Jogos: Encontrar a melhor jogada em jogos como xadrez, damas, etc.

* Navegação: Encontrar a rota mais curta ou mais rápida entre dois pontos (como no GPS).

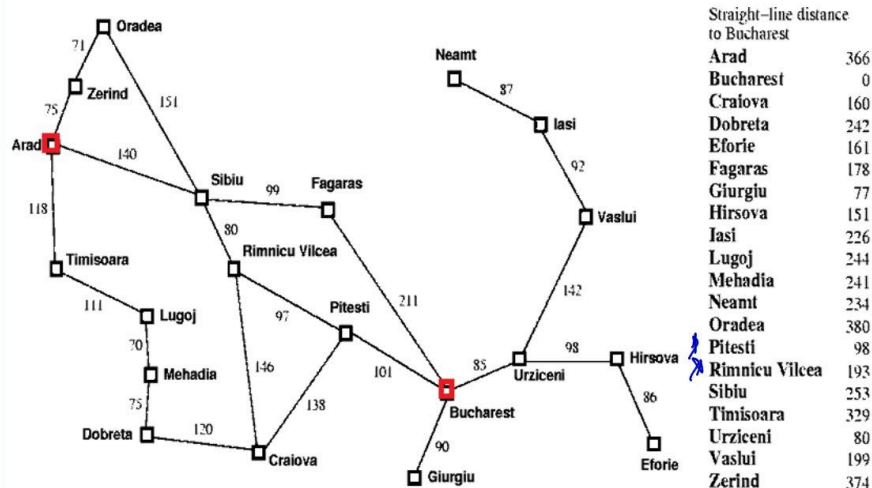
* Resolução de quebra-cabeças: Como o quebra-cabeça de 8 peças ou o cubo mágico.

* Verificação de sistemas: Analisar se um sistema pode atingir um estado indesejável.

Em resumo, a busca em espaços de estados é uma abstração poderosa que permite que agentes inteligentes (sejam eles programas de computador ou robôs) explorem as possibilidades em um ambiente para alcançar seus objetivos de forma sistemática e eficiente.

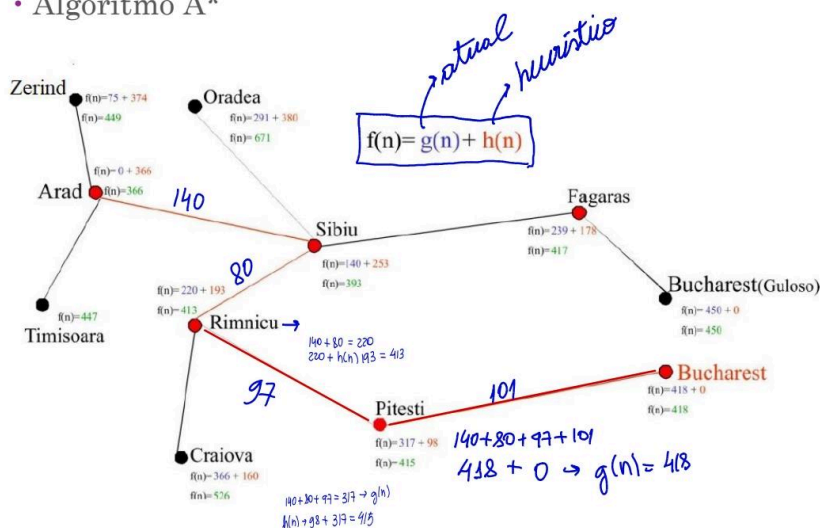
Métodos de Busca

- Busca heurística (ou informada)
- Algoritmo A*



Métodos de Busca

- Busca heurística (ou informada)
- Algoritmo A*



$f(n) = g(n) + h(n)$ -> função de avaliação

O algoritmo **A*** é um algoritmo de busca informado, o que significa que ele utiliza uma função heurística para orientar sua busca em direção à meta. Essa função heurística estima o custo de alcançar a meta a partir de um determinado nó, permitindo que o algoritmo priorize caminhos promissores e evite explorar caminhos desnecessários.

Para entender o algoritmo A*, você precisa estar familiarizado com esses conceitos fundamentais:

- **Nós:** Pontos em seu gráfico (como interseções em um mapa)
- **Bordas:** Conexões entre nós (como estradas que conectam cruzamentos)
- **Custo do caminho:** O custo real de deslocamento de um nó para outro
- **Heurística:** Um custo estimado de qualquer nó até a meta
- **Espaço de pesquisa:** A coleção de todos os caminhos possíveis a serem explorados

No algoritmo A* (A Estrela), a função principal utilizada para guiar a busca e garantir a otimalidade do caminho é a função de avaliação $f(n)$, que é definida como:

$$f(n) = g(n) + h(n)$$

Vamos detalhar cada componente dessa função:

1. $g(n)$ - Custo Real do Caminho:

- Representa o custo acumulado para ir do estado inicial até o nó atual n .
- É o custo real do caminho percorrido até aquele ponto.
- É sempre calculado de forma exata.

2. $h(n)$ - Função Heurística (Custo Estimado):

- Representa uma estimativa do custo para ir do nó atual n até o estado objetivo.
- É uma "adivinhação inteligente" ou um "palpite" de quão longe o nó atual está do objetivo.
- A qualidade da função heurística é crucial para a eficiência do algoritmo A*.
- A heurística deve ser:
 - **Admissível:** Significa que ela nunca superestima o custo real para chegar ao objetivo. Ou seja, $h(n) \leq \text{custo real de } n \text{ ao objetivo}$. Se a heurística for admissível, o A* tem a garantia de encontrar o caminho ótimo (o de menor custo).
 - **Consistente (ou Monotônica):** É uma condição mais forte que a admissibilidade. Uma heurística é consistente se para cada nó n e cada sucessor n' de n , a estimativa do custo de n ao objetivo for menor ou igual ao custo real para ir de n a n' mais a estimativa do custo de n' ao objetivo. Ou seja: $h(n) \leq \text{custo}(n, n') + h(n')$. Uma heurística consistente também é admissível.

Algoritmo Genético

Algoritmos Genéticos (AGs) são uma classe de **algoritmos estocásticos de busca e otimização** inspirados nos princípios da **evolução biológica e genética** de Charles Darwin, como a seleção natural e a reprodução.

Eles são poderosos para encontrar soluções aproximadas para problemas complexos onde as abordagens tradicionais podem ser ineficientes ou impossíveis de aplicar.

A ideia central é simular o processo de "evolução" de uma população de soluções candidatas ao longo de "gerações", buscando a "melhor" solução para um problema.

Como funcionam os Algoritmos Genéticos?

Um AG opera com os seguintes conceitos e etapas principais:

1. População Inicial:

- O algoritmo começa criando uma **população** de soluções candidatas de forma **aleatória**. Cada solução individual é chamada de **indivíduo** ou **cromossomo**.
- Um **cromossomo** é geralmente uma representação codificada do problema, como uma sequência de bits (0s e 1s), números inteiros ou reais, ou até mesmo árvores de expressões. Por exemplo, em um problema de otimização de rota, um cromossomo pode ser uma sequência de cidades a serem visitadas.

2. Função de Aptidão (Fitness Function):

- Para cada indivíduo na população, uma **função de aptidão** (ou *fitness function*) é utilizada para avaliar quão "boa" ou "adequada" é aquela solução para o problema. Essa função retorna um valor numérico que indica a qualidade do indivíduo.
- Quanto maior o valor de aptidão, melhor a solução. Por exemplo, em um problema de busca da rota mais curta, a aptidão pode ser o inverso do comprimento total da rota.

3. Seleção:

- Indivíduos com maior aptidão têm uma probabilidade maior de serem selecionados para se tornarem "pais" da próxima geração. Isso simula o princípio da "sobrevivência do mais apto" da seleção natural.
- Existem várias estratégias de seleção, como:
 - **Seleção por Roleta (Roulette Wheel Selection):** A probabilidade de um indivíduo ser selecionado é proporcional à sua aptidão, como em uma roleta onde fatias maiores correspondem a indivíduos mais aptos.

- **Seleção por Torneio (Tournament Selection):** Um subconjunto de indivíduos é selecionado aleatoriamente da população, e o melhor indivíduo desse subconjunto é escolhido para ser pai.
- **Elitismo:** Garante que os melhores indivíduos de uma geração sejam copiados diretamente para a próxima geração, evitando a perda de boas soluções.

4. Cruzamento (Crossover ou Recombinação):

- Os indivíduos selecionados (pais) são combinados para criar novos indivíduos (filhos), trocando "material genético" entre si. Isso simula a reprodução sexual.
- Por exemplo, se os cromossomos são sequências de bits, o cruzamento pode envolver a escolha de um ponto aleatório e a troca de segmentos de bits entre os pais.
- Existem diferentes tipos de cruzamento: de um ponto, de dois pontos, uniforme, etc.

5. Mutação:

- Pequenas alterações aleatórias são introduzidas nos cromossomos dos novos indivíduos (filhos). Isso simula as mutações genéticas.
- A mutação é crucial para manter a **diversidade genética** na população e evitar que o algoritmo fique preso em "ótimos locais" (soluções que parecem boas, mas não são a melhor globalmente). Ela permite que o algoritmo explore novas partes do espaço de busca.
- Por exemplo, em um cromossomo de bits, a mutação pode significar inverter um bit (de 0 para 1 ou vice-versa) com uma pequena probabilidade.

6. Nova Geração e Iteração:

- Os novos indivíduos gerados por cruzamento e mutação formam a próxima geração.
- O processo então se repete: a aptidão da nova geração é avaliada, os melhores são selecionados, e assim por diante.
- Esse ciclo continua por um número predefinido de gerações, até que uma condição de parada seja atingida (por exemplo, um tempo limite, um número máximo de gerações, ou quando uma solução satisfatória é encontrada).

Por que usar Algoritmos Genéticos?

- **Problemas Complexos:** São eficazes para problemas de otimização com espaços de busca muito grandes e complexos, onde métodos tradicionais (que exigem derivadas ou são muito lentos) falham.
- **Não Requerem Conhecimento Detalhado do Problema:** Eles não precisam de informações sobre a "forma" da função objetivo (se é contínua, diferenciável, etc.), apenas da capacidade de avaliar a aptidão de uma solução.
- **Evitam Ótimos Locais:** A combinação de seleção, cruzamento e mutação ajuda a explorar diferentes partes do espaço de busca, reduzindo a chance de ficar preso em soluções subótimas.

- **Adaptáveis:** Podem ser aplicados a uma ampla gama de problemas, desde que seja possível codificar as soluções e definir uma função de aptidão.

Aplicações Comuns:

- **Otimização:** Encontrar o melhor design para produtos, otimizar rotas de entrega, alocação de recursos.
- **Machine Learning:** Otimização de parâmetros de modelos, seleção de características, treinamento de redes neurais.
- **Engenharia:** Design de circuitos, agendamento de tarefas, controle robótico.
- **Bioinformática:** Análise de sequências genéticas, previsão de estruturas de proteínas.
- **Finanças:** Otimização de portfólios, previsão de mercado.

Em resumo, os Algoritmos Genéticos são uma meta-heurística poderosa que imita o poder da evolução natural para encontrar soluções eficientes para problemas difíceis, explorando o espaço de busca de forma inteligente e adaptativa.

Explicação a partir do slide:

Algoritmos Genéticos: A Natureza como Inspiração para Resolver Problemas

Os Algoritmos Genéticos (AGs) são uma forma muito inteligente e robusta de encontrar soluções para problemas complexos. Pense neles como um "treinamento" para um computador, onde ele aprende a resolver algo da mesma forma que a natureza evoluiu as espécies ao longo de milhões de anos.

A Grande Sacada:

Imagine que você tem um problema que é muito difícil de resolver diretamente. Pode ser, por exemplo, encontrar a melhor rota para um entregador passar por várias cidades, sem repetir nenhuma, e que seja a mais curta possível. Se você tiver muitas cidades, o número de rotas possíveis é GIGANTESCO! Tentar testar cada uma delas levaria um tempo absurdo.

É aí que entram os Algoritmos Genéticos. Eles não tentam testar tudo. Em vez disso, eles usam a ideia de **evolução**.

Os Pilares da Evolução nos AGs:

Para entender os AGs, precisamos lembrar de algumas coisas da biologia:

1. **A Teoria da Evolução:** Na natureza, os indivíduos de uma espécie competem por recursos (como comida e água). Os mais "fracos" ou menos adaptados têm menos

chances de sobreviver e se reproduzir. Aqueles que sobrevivem e se reproduzem passam seus genes para a próxima geração.

2. **Combinação de Genes (Recombinação/Cruzamento):** Quando dois indivíduos se reproduzem, seus genes se combinam para criar um novo indivíduo. Essa combinação pode gerar um filho muito bem adaptado ao ambiente, porque ele "herda" características positivas de ambos os pais. Nos AGs, isso é chamado de **cruzamento** ou **recombinação**.
3. **Mutações:** Às vezes, ocorrem pequenos "erros" no código genético durante a replicação do DNA. Essas mutações são aleatórias e podem ser boas, ruins ou neutras. Nos AGs, a **mutação** é essencial para introduzir novas variações e evitar que o algoritmo fique preso em soluções que não são as melhores globalmente.

Como os Algoritmos Genéticos "Evoluem" Soluções?

Os AGs foram "inventados" por John Holland em 1975, inspirados nas ideias de Darwin e Mendel. Eles funcionam assim:

1. **Começando com uma População Inicial:**
 - Primeiro, o AG cria um grupo de "soluções" aleatórias para o seu problema. Cada solução é um "indivíduo" ou "cromossomo".
 - Um "cromossomo" é a forma como o computador representa uma possível solução. Pode ser uma sequência de 0s e 1s (bits), números, ou qualquer outra estrutura de dados. Por exemplo, no problema do caixeiro viajante, um cromossomo pode ser a ordem em que as cidades são visitadas. Cada "pedaço" dessa representação é um "gene".
2. **Avaliando a Qualidade (Função de Aptidão/Fitness):**
 - Para cada "indivíduo" (solução) criado, o AG usa uma "função de aptidão" (ou *fitness function*) para ver o quão boa ela é. Esta função é como o "meio ambiente" que avalia a adaptação do indivíduo.
 - Quanto maior o valor da aptidão, melhor é a solução. Por exemplo, no problema do caixeiro viajante, a função de aptidão pode calcular a distância total da rota: rotas mais curtas teriam uma aptidão maior (ou inversamente proporcional ao comprimento da rota).
3. **Selecionando os "Pais" (Seleção):**
 - Os indivíduos com a melhor aptidão têm mais chances de serem escolhidos para se reproduzir e gerar a próxima geração. Isso simula a "seleção natural".
 - Um método comum de seleção é a **Roleta**. Imagine uma roleta de cassino onde cada "fatia" é proporcional à aptidão de um indivíduo. Indivíduos mais aptos têm fatias maiores e, portanto, mais chances de serem sorteados para serem pais. É importante notar que mesmo indivíduos com aptidão baixa podem ter uma pequena chance de serem selecionados, pois podem carregar características genéticas valiosas.
4. **Criando Novos Indivíduos (Cruzamento/Recombinação):**

- Os "pais" selecionados trocam partes de seus "cromossomos" para criar "filhos" (novas soluções). Isso se chama **cruzamento** ou **recombinação**.
- Um tipo simples é a **recombinação de um ponto**: escolhe-se um ponto aleatório no cromossomo, e os pais trocam as partes à direita desse ponto para formar os filhos.
- Existem outros tipos, como a **recombinação de dois pontos** (onde se escolhem dois pontos de corte) ou a **recombinação uniforme** (onde, para cada gene, decide-se aleatoriamente de qual pai ele virá).

5. Introduzindo Novas Variações (Mutação):

- Após o cruzamento, uma pequena porcentagem dos "genes" dos filhos pode sofrer **mutação**. Isso significa que um bit pode mudar de 0 para 1, ou vice-versa, de forma aleatória.
- A mutação é vital para garantir que a população não perca a diversidade genética e que o algoritmo possa explorar novas áreas do espaço de busca, evitando ficar "preso" em soluções locais que não são as melhores. A probabilidade de mutação é um parâmetro importante que pode afetar o resultado do algoritmo.

6. Formando uma Nova Geração e Recomeçando:

- Os novos "filhos" (gerados por cruzamento e mutação) formam a próxima geração.
- O processo então se repete: a aptidão da nova geração é avaliada, os melhores são selecionados, e assim por diante.
- Esse ciclo continua até que um **critério de parada** seja atingido (por exemplo, um número máximo de gerações, ou quando uma solução suficientemente boa é encontrada).

Otimização e Aplicações:

Os Algoritmos Genéticos são excelentes para problemas de **otimização**, que buscam a melhor solução para algo. Eles são particularmente úteis quando o "espaço de busca" (todas as possíveis soluções) é muito grande, e uma "função objetivo" (a função de aptidão) pode avaliar a qualidade de cada solução.

Eles são usados em diversas áreas:

- **Busca e Otimização**: Encontrar o máximo ou mínimo de funções, resolver o Problema do Caixeiro Viajante (como o exemplo da rota).
- **Otimização Combinatória**: Escalonamento de tarefas, distribuição de carga horária.
- **Design**: Projetar estruturas e circuitos.
- **Aprendizagem Automática (Machine Learning)**: Ajuda a treinar modelos e otimizar parâmetros.

Em resumo, os Algoritmos Genéticos são uma ferramenta poderosa que imita a sabedoria da natureza para resolver problemas complexos, evoluindo soluções ao longo do tempo através da seleção, recombinação e mutação

Pergunta: você disse que os algoritmos genéticos não tentam testar todos os caminhos. não entendi. como seria isso na prática então?

Resposta:

Você tocou em um ponto crucial que diferencia os Algoritmos Genéticos (AGs) de métodos de busca exaustiva (que testam todos os caminhos)!

Quando digo que os AGs "não tentam testar todos os caminhos", significa que eles não percorrem sistematicamente cada uma das combinações possíveis para encontrar a solução ótima. Em vez disso, eles usam um processo que simula a evolução natural para explorar o espaço de busca de forma mais inteligente e eficiente.

Vamos entender como isso funciona na prática, usando o exemplo do **Problema do Caixeiro Viajante (PCV)**, que é mencionado nos seus slides.

O Problema do Caixeiro Viajante (PCV) na Prática

O Problema: O caixeiro viajante precisa encontrar a rota mais curta para visitar um conjunto de cidades, visitando cada cidade apenas uma vez e retornando à cidade de origem.

Por que "testar todos os caminhos" é impossível/ineficiente?

- **Número de Cidades Pequeno (ex: 4 cidades):**
 - Cidade A, B, C, D
 - As rotas possíveis seriam: A-B-C-D-A, A-B-D-C-A, A-C-B-D-A, A-C-D-B-A, A-D-B-C-A, A-D-C-B-A.
 - Para 4 cidades, há $(4-1)! = 3! = 6$ rotas possíveis (considerando um ponto de partida fixo). Já é um número pequeno e gerenciável.
- **Número de Cidades Médio (ex: 10 cidades):**
 - As rotas possíveis seriam $(10-1)! = 9! = 362.880$ rotas. Isso já é um número considerável, mas um computador pode calcular em um tempo razoável.
- **Número de Cidades Grande (ex: 50 cidades):**
 - O número de rotas é $(50-1)! = 49!$, um número astronômico (da ordem de 6×10^{62}).
 - Testar cada uma dessas rotas levaria mais tempo do que a idade do universo, mesmo com os computadores mais rápidos do mundo. Isso é o que chamamos de **busca exaustiva** ou **força bruta**.

Como o Algoritmo Genético age na Prática (sem testar tudo)

Em vez de gerar e avaliar todas as $49!$ rotas para 50 cidades, um AG faz o seguinte:

1. Geração da População Inicial (Amostragem Aleatória):

- Ele não gera *todas* as rotas possíveis. Ele gera uma **população** de, digamos, 100, 1000 ou 10000 rotas (indivíduos/cromossomos) **aleatoriamente**.

- Por exemplo, para 8 cidades, uma rota aleatória pode ser (3 5 7 2 1 6 4 8). Ele criaria várias dessas sequências.
- **Perceba:** 1000 rotas é um número infinitamente pequeno comparado a 49!. O AG não está testando tudo, está pegando apenas uma pequena "amostra" do espaço de busca.

2. Avaliação da Aptidão (Medindo a Qualidade):

- Para cada uma dessas 1000 rotas aleatórias, o AG calcula a distância total (função de avaliação). Rotas mais curtas são consideradas "mais aptas".
- Por exemplo, a rota (1 3 5 4 2) tem uma avaliação de 230.

3. Seleção (Escolhendo os "Melhores" para Reproduzir):

- As 1000 rotas são classificadas pela sua aptidão.
- O AG não descarta as piores. Ele seleciona um subconjunto de rotas (indivíduos) para serem "pais" da próxima geração. As rotas mais curtas têm uma probabilidade maior de serem selecionadas, mas rotas mais longas (menos aptas) também podem ter uma pequena chance de serem escolhidas.
- Isso é feito usando métodos como a "Roleta", onde fatias maiores são dadas aos indivíduos mais aptos.

4. Cruzamento (Combinando Características de Boas Soluções):

- Os "pais" selecionados trocam partes de suas rotas para criar "filhos" (novas rotas).
- Por exemplo, se um pai tem (A-B-C-D-E) e outro (X-Y-C-D-Z), o cruzamento pode gerar (A-B-C-D-Z) e (X-Y-C-D-E). A ideia é que, ao combinar partes de rotas que já são boas, é possível gerar rotas ainda melhores.
- No exemplo do slide, Pais (3 5 7 2 1 6 4 8) e (2 5 7 6 8 4 3 1) podem gerar um Filho (3 5 7 2 4 6 1 8) através de recombinação uniforme.

5. Mutação (Introduzindo Variações Aleatórias):

- Com uma pequena probabilidade, alguns "genes" (partes das rotas) dos filhos são alterados aleatoriamente.
- No PCV, isso pode significar trocar a ordem de duas cidades em uma rota. Por exemplo, (3 5 7 2 1 6 4 8) pode se tornar (3 1 7 2 5 6 4 8).
- Isso é crucial porque, se o AG só combinasse o que já existe, ele poderia ficar preso em um "ótimo local" (uma boa solução, mas não a *melhor* global). A mutação ajuda a "saltar" para novas regiões do espaço de busca.

6. Formação da Nova Geração e Repetição (Evolução):

- Os novos filhos formam a próxima geração. O processo se repete por centenas ou milhares de gerações.
- A cada geração, a população tende a melhorar, com as rotas mais curtas se tornando mais comuns e as rotas mais longas sendo gradualmente eliminadas.
- O algoritmo para quando atinge um critério de parada, como um número máximo de gerações ou quando a qualidade da solução para de melhorar significativamente.

Conclusão:

O AG não "testa todos os caminhos". Ele usa um processo iterativo e probabilístico de **seleção, cruzamento e mutação** para "explorar" o espaço de busca. Ele se concentra em **evoluir** um conjunto de soluções (a população) ao longo do tempo, direcionando a busca para áreas do espaço de soluções que têm maior probabilidade de conter soluções ótimas ou aproximadamente ótimas. Ele é uma "técnica prática e robusta de busca e otimização" que não exige conhecimento de algoritmos rápidos de solução para problemas complexos.

Então, por exemplo, no exemplo do caixeiro viajante, ele tem um número de cidades para visitar, certo? Disso, de um total de rotas possíveis para ele percorrer, uma única vez, ele cria uma população com uma certa quantidade dessas rotas. Aí, essas rotas vão evoluindo e se mutando e ele vai selecionando sempre as melhores?

Exatamente! Sua compreensão está perfeita. Vamos detalhar isso um pouco mais para solidificar o conceito:

1. O Problema (Caixeiro Viajante): Um Espaço de Busca Gigante

- Sim, o caixeiro viajante tem um número de cidades para visitar uma única vez.
- Para cada problema, existe um "espaço de busca" onde vivem todas as potenciais soluções.
- O total de rotas possíveis (todas as combinações únicas) é o "espaço de busca" completo. Como discutimos, para muitas cidades, esse número é tão grande que **encontrar a melhor solução pode levar tempo demais** se você tentar testar todas as possibilidades.

○

2. Criação da População Inicial: Amostra Aleatória

- Em vez de calcular e testar *todas* as rotas possíveis, o Algoritmo Genético (AG) começa criando uma **população** de soluções, que são escolhidas **aleatoriamente**.
-
- Essa "população" é um conjunto de um certo número de rotas (indivíduos/cromossomos) geradas ao acaso, muito menor do que o total de rotas possíveis. Por exemplo, se há bilhões de rotas, ele pode começar com apenas algumas centenas ou milhares delas.

3. Aptidão e Avaliação: Medindo a Qualidade da Rota

- Para cada rota (indivíduo) nessa população inicial, o AG calcula sua **aptidão** (ou *fitness*).
- No caso do caixeiro viajante, a função de aptidão avalia a qualidade da rota, que geralmente é o inverso da distância total percorrida. Rotas mais curtas têm uma aptidão maior.

4. O Ciclo Evolutivo: Seleção, Cruzamento e Mutação

- **Seleção:** As rotas mais "aptas" (as mais curtas) têm uma probabilidade maior de serem selecionadas para "se reproduzir" e passar seus "genes" (partes da rota) para a próxima geração. Os menos aptos também podem gerar descendentes, mas com menor probabilidade. Isso simula o mecanismo de seleção natural.
-
- **Cruzamento (Recombinação):** As rotas selecionadas (os "pais") combinam partes de si mesmas para criar novas rotas (os "filhos")¹⁰¹⁰¹⁰¹⁰. Essa combinação pode gerar um novo indivíduo que é muito melhor adaptado ao problema, ao combinar características positivas de cada um dos seus pais¹¹.
-
- **Mutação:** Pequenas alterações aleatórias são introduzidas nas novas rotas (nos "filhos"). Isso é crucial para manter a diversidade na população e evitar que o algoritmo fique "preso" em soluções que são boas, mas não a melhor globalmente (ótimos locais).
-
- **Elitismo:** Uma técnica adicional chamada elitismo pode ser usada para garantir que os melhores indivíduos de uma geração sejam copiados diretamente para a próxima, prevenindo que a solução regrede.
-

5. Evolução das Rotas e Seleção das Melhores:

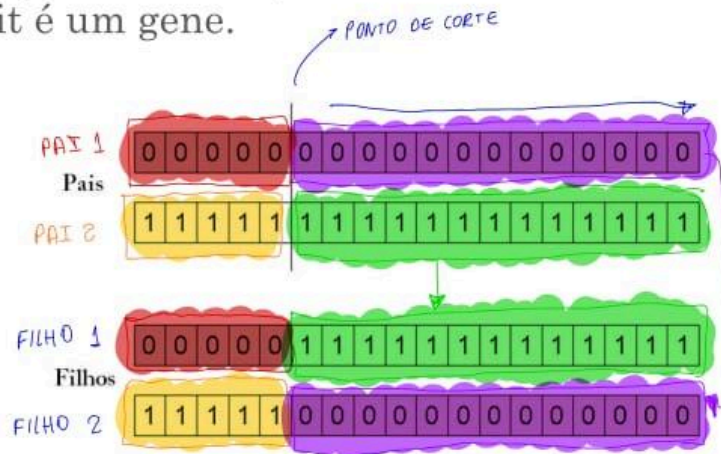
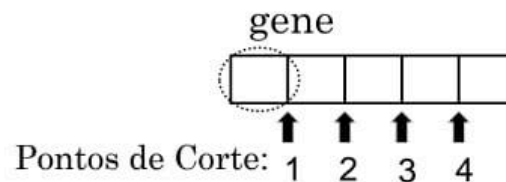
- Esse processo de seleção, cruzamento e mutação é repetido por muitas "gerações".
- A cada iteração (ou geração), o AG analisa a vizinhança da população atual, buscando diversificar a pesquisa no espaço das soluções¹⁶.
- A ideia é que, ao longo das gerações, as rotas (indivíduos) vão "evoluindo" e se tornando progressivamente melhores, ou seja, mais curtas e eficientes.
- O algoritmo **sempre tenta selecionar as melhores** (as mais aptas) para continuar o processo, mas a mutação e a probabilidade na seleção garantem que ele continue explorando novas variações, e não apenas refine o que já tem.

Em resumo, o AG não enumera *todas* as rotas. Ele inteligentemente cria uma amostra, melhora-a iterativamente através de um processo inspirado na biologia, e converge para uma solução que é **aproximadamente ótima** em um tempo razoável, mesmo para problemas com espaços de busca gigantes.

É IMPORTANTE PARA A EXPLORAÇÃO DO ESPAÇO DE BUSCA. HERDA AS MELHORES CARACTERÍSTICAS DE AMBOS OS PAIS (1 E 2), ACELERANDO O PROCESSO DE ENCONTRAR SOLUÇÕES ÓTIMAS OU QUASE ÓTIMAS PARA O PROBLEMA.

Recombinação - Ponto de Corte

- Cada indivíduo com n genes possui $n-1$ pontos de corte.
- Em um indivíduo com codificação binária, cada bit é um gene.



23

Operadores Genéticos - Mutação

- Depois de compostos os filhos, entra em ação o operador de **mutação**.
- O operador atua com base em uma **probabilidade** extremamente baixa (da ordem de 5%) de **alteração aleatória** do valor de um gene ou mais genes dos filhos.
- O valor da probabilidade que decide se o operador de mutação será ou não aplicado é um dos parâmetros do algoritmo genético que pode alterar o resultado alcançado pelo algoritmo.

24

Okay, vamos explicar o slide sobre "Recombinação - Ponto de Corte" em Algoritmos Genéticos de forma didática.

Recombinação: Como os "Genes" São Misturados nos Algoritmos Genéticos

Este slide explica uma das operações mais importantes em um Algoritmo Genético: a **recombinação** (também conhecida como **cruzamento** ou *crossover*). É aqui que os "genes" dos pais são misturados para criar novos "filhos", simulando o que acontece na reprodução biológica.

O Conceito Principal: Ponto de Corte

Imagine que cada solução (indivíduo ou cromossomo) é como uma sequência de blocos, e cada bloco é um "gene". Para misturar essas sequências, precisamos de um "ponto de corte".

- **Quantos Pontos de Corte?**

- Se um indivíduo tem n "genes" (ou seja, n blocos/bits), ele possui $n-1$ possíveis "pontos de corte".
-
- Pense em uma sequência de 5 blocos: [G1] [G2] [G3] [G4] [G5]. Você pode cortar entre G1 e G2, G2 e G3, G3 e G4, ou G4 e G5. Isso dá 4 pontos de corte para 5 genes.
- No exemplo visual do slide, a sequência tem 5 "genes" (blocos) e 4 possíveis "pontos de corte", marcados com as setas e números 1, 2, 3, 4.
-

- **Codificação Binária:**

- Frequentemente, em Algoritmos Genéticos, os "indivíduos" são codificados como sequências de bits (0s e 1s). Nesses casos, **cada bit é considerado um "gene"**⁵.
-

Como Funciona a Recombinação de um Ponto (Passo a Passo):

O slide demonstra o processo de "recombinação de um ponto":

1. **Seleção dos Pais:** Primeiro, dois "pais" são selecionados da população, geralmente através do processo de seleção que vimos anteriormente (como a roleta). No exemplo, temos dois pais:
 - Pai 1: 000000000000000000 (uma sequência de zeros)
 - Pai 2: 111111111111111111 (uma sequência de uns)
2. **Escolha do Ponto de Corte:** Um ponto de corte é escolhido aleatoriamente em alguma posição entre os "genes" (bits) dos cromossomos. No exemplo, o ponto de corte está na 5ª posição (depois do quinto bit).

3. Troca de Partes para Formar os Filhos:

- Os "pais" trocam as partes de seus "cromossomos" que estão **à direita do ponto de corte**.
-
- A parte à esquerda do ponto de corte do Pai 1 vai para o Filho 1, e a parte à direita do Pai 2 vai para o Filho 1.
-
- A parte à esquerda do ponto de corte do Pai 2 vai para o Filho 2, e a parte à direita do Pai 1 vai para o Filho 2.
-

4. No exemplo visual:

- **Pai 1:** 00000 | 00000000000000
- **Pai 2:** 11111 | 11111111111111
- **Filho 1:** Recebe a parte esquerda do Pai 1 (00000) e a parte direita do Pai 2 (11111111111111). Resultado: 000001111111111111
- **Filho 2:** Recebe a parte esquerda do Pai 2 (11111) e a parte direita do Pai 1 (00000000000000). Resultado: 111110000000000000

Por que a Recombinação é Importante?

A recombinação é fundamental para a **exploração** do espaço de busca. Ao combinar informações de dois pais, ela permite que o algoritmo crie novas soluções que podem herdar as melhores características de ambos, acelerando o processo de encontrar soluções ótimas ou quase ótimas para o problema. Ela é uma das principais formas de **diversificação da procura no espaço das soluções**.

Esse processo, junto com a mutação, garante que o Algoritmo Genético não fique preso em "ótimos locais" e possa explorar um conjunto mais amplo de possibilidades.

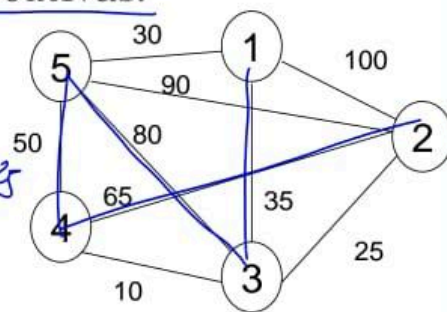
Algoritmos Genéticos – Exemplo

Qual a função de avaliação?

- A função de avaliação consiste em somar todas as distâncias entre cidades consecutivas.

Exemplo:

$$63 + 22 + 44 + 50 + 90 + 35$$



- O cromossomo (1 3 5 4 2) tem avaliação igual a $35 + 80 + 50 + 65 = 230$

34

Algoritmos Genéticos – Exemplo

Recombinação (uniforme):

- Pai1 (3 5 7 2 1 6 4 8)
- Pai2 (2 5 7 6 8 4 3 1)

- 1) Gera-se uma string de bits aleatória do mesmo tamanho que os pais:

1 0 0 1 0 1 0 1

- 2) Copia-se para o filho 1 os elementos do pai 1 referentes àquelas posições onde a string de bits possui um 1:

3 _ _ 2 _ 6 _ 8

- 3) Elementos não copiados do pai1:

5 7 1 4

- 3) Permuta-se essa lista de forma que os elementos apareçam na mesma ordem que no pai 2 e copia-se eles para dentro do Filho1:

3 5 7 2 4 6 1 8

35

Algoritmos Genéticos – Exemplo

• Muta  o:

• Indiv duo (3 5 7 2 1 6 4 8)

- Escolhem-se dois elementos aleat rios dentro do cromossomo e trocam-se as suas posi  es:

(3 5 7 2 1 6 4 8)

Novo indiv duo mutante:

(3 1 7 2 5 6 4 8)

36

AG → RESOLVER PROBLEMAS COMPLEXOS

IA + BIOLOGIA

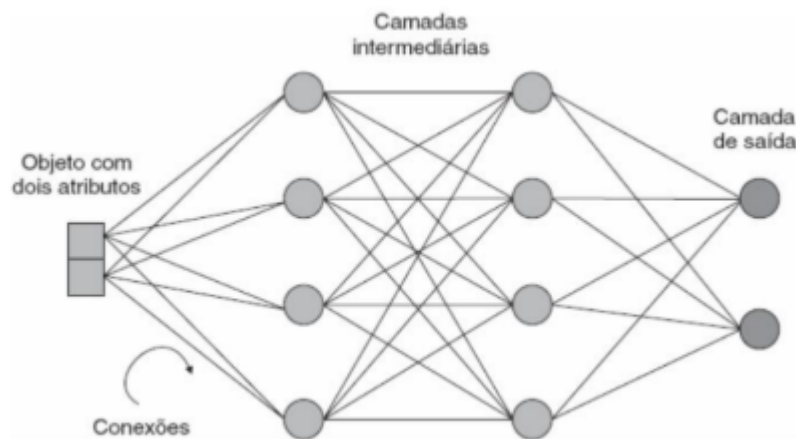
→ TEORIA DA EVOLU  O (DARWIN E MENDEL)

S O UMA T CNICA DE BUSCA E OTIMIZA  O

  UM M TOD0 ESTOC STICO DE BUSCA. → ALEAT RIO PARA EXPLORA  O MAIS AMPLA.

RNA (Redes Neurais Artificiais)

- Em uma RNA, os neurônios podem estar dispostos em uma ou mais camadas.



- As RNAs podem apresentar ou não conexões de retroalimentação, ou feedback.
- O número de camadas, o número de neurônios em cada camada e a presença ou não de conexões de retropropagação definem a topologia de uma RNA

Processo de Aprendizagem

- A propriedade primordial para uma RNA é a sua habilidade de aprender a partir de seu ambiente e de melhorar o seu desempenho através da aprendizagem.
- Definição de aprendizagem no contexto de RNA, segundo Haykin:

Aprendizagem é um processo pela qual os parâmetros livres de uma rede neural são adaptados através de um processo de estimulação pelo ambiente no qual a rede está inserida. O tipo de aprendizagem é determinado pela maneira pela qual a modificação de parâmetros ocorre.

- A definição do processo de aprendizagem implica a seguinte sequência de eventos:
- A rede neural é estimulada por um ambiente
- A rede neural sofre modificações nos seus parâmetros livres como resultado desta simulação
- A rede neural responde de uma maneira nova ao ambiente, devido às modificações ocorridas na estrutura interna

- Um conjunto de regras bem definidas para a solução de um problema de aprendizado é chamado de algoritmo de aprendizado.
- Há vários algoritmos de aprendizado. Cada um dos algoritmos oferece vantagens próprias.
- Basicamente, algoritmos de aprendizado diferem na forma como os pesos sinápticos são ajustados.
- Um outro fator é a maneira como a RNA se relaciona com seu ambiente.

Tipos de Aprendizagem

A propriedade mais importante das redes neurais é a habilidade de aprender de seu ambiente e com isso melhorar seu desempenho. Isso é feito através de um processo iterativo de ajustes aplicado a seus pesos denominado treinamento.

Denomina-se algoritmo de aprendizado a um conjunto de regras bem definidas para a solução de um problema de aprendizado.

Aprendizagem Supervisionada

- a saída desejada é conhecida e informada para que a rede compare com a saída processada
- se houver erro, a rede tenta corrigir este erro até que a mesma forneça uma saída igual a saída desejada

Aprendizagem não supervisionada

- a saída desejada é obtida através de entradas repetitivas até a rede reter o conhecimento
- não existe saída informada para comparação
- deve existir redundâncias nos dados para a rede encontrar padrões ou características dos dados

Vantagens:

- solução naturalmente paralela
- robusta, tolerante a falhas
- permite a integração de informações oriundas de fontes ou tipos diferentes
- sistema adaptativo, capaz de aprender
- mostra certo grau de autonomia no aprendizado
- performance muito rápida no reconhecimento.

Limitações:

- ainda muito difícil explicar seu comportamento, por causa da falta de transparência
- soluções não escalam bem... computacionalmente cara para problemas de maior porte
- ainda muito distante da realidade biológica

Formas mais básicas de aprendizado em Redes Neurais:

- Perceptron: Algoritmo para aprendizagem de redes neurais simples (uma camada) desenvolvido nos anos 50.
- Backpropagation: Algoritmo mais complexo para aprendizagem de redes neurais de múltiplas camadas desenvolvido nos anos 80.

Aprendizagem de Perceptron

Usa-se um conjunto de exemplos de treinamento que dão a saída desejada para uma unidade, dado um conjunto de entradas.

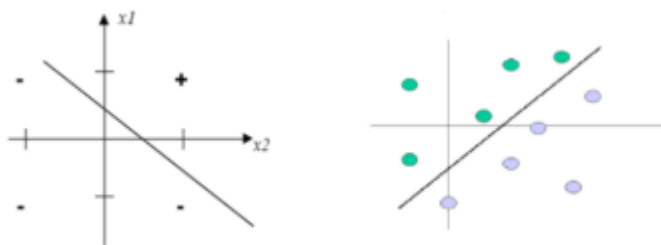
O objetivo é aprender pesos sinápticos de tal forma que a unidade de saída produza a saída correta para cada exemplo. Ou seja, o conhecimento é representado através de pesos.

O algoritmo faz atualizações iterativamente até chegar aos pesos corretos.

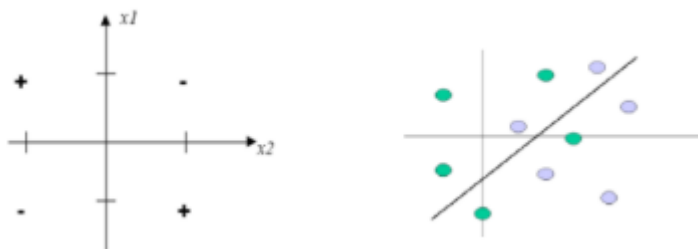
O Perceptron possui uma única camada.

Limitações

Um único Perceptron consegue resolver somente funções linearmente separáveis.



Em funções não linearmente separáveis o perceptron não consegue gerar um hiperplano para separar os dados.



Perceptrons expressam somente superfícies de decisão linear.

Entretanto, é possível combinar vários perceptrons lineares para gerar superfícies de decisão mais complexas.

Dessa forma podemos, por exemplo, gerar uma superfícies de classificação para o operador XOR.

Para resolver problemas não linearmente separáveis utilizando RNAs, a alternativa mais emprega é adicionar uma ou mais camadas intermediárias.

As redes do tipo Perceptron multicamadas (MLP, do inglês multilayer Perceptron) apresentam uma ou mais camadas intermediárias de neurônios e uma camada de saída.

Backpropagation

Um obstáculo que havia para utilizar redes multicamadas era a ausência de um algoritmo para o treinamento dessas redes, o que foi transposto com a proposta de um algoritmo baseado em gradiente descendente denominado backpropagation (Rumelhart et al., 1986). Para que esse algoritmo seja utilizado, a função de ativação precisa ser contínua, diferenciável e, de preferência, não decrescente. A função de ativação do tipo sigmoidal obedece a esses requisitos.

O Backpropagation não é um algoritmo ótimo e não garante sempre a melhor resposta.

O algoritmo de descida do gradiente pode ficar preso em um erro mínimo local.

É possível refazer o treinamento variando os valores iniciais dos pesos.

Backpropagation é o algoritmo de aprendizagem mais comum, porém existem muitos outros.

Explicação feita pelo Gemini: com alterações

As Redes Neurais Artificiais (RNAs) são um paradigma de computação inspirado na arquitetura paralela do cérebro humano. Elas buscam simular a capacidade de aprendizado do cérebro humano na aquisição de conhecimento, especialmente na busca pela construção de máquinas inteligentes.

Aqui está uma explicação das RNAs de forma didática e fácil de entender:

1. A Inspiração: O Cérebro Humano e os Neurônios

O cérebro humano é um modelo natural para a construção de máquinas inteligentes. No cérebro, o comportamento inteligente é uma propriedade emergente de um grande número de unidades simples. A unidade fundamental do sistema nervoso, do qual faz parte o cérebro, é a célula nervosa, o neurônio. Os neurônios se distinguem por apresentar excitabilidade, o que lhes permite responder a estímulos externos e internos. Isso possibilita a transmissão de impulsos nervosos a outros neurônios e a células musculares e glandulares.

Um neurônio possui três componentes principais:

- **Dendritos:** Recebem sinais
- **Corpo Celular (Corpo):** É a parte principal do neurônio.
- **Axônio:** Transmite sinais para outros neurônios.
- Quando um sinal é forte o suficiente (ultrapassando um certo limiar), ele segue pelo axônio; caso contrário, é bloqueado.

2. Neurônios Artificiais: Os Blocos Construtores das RNAs

Os estudos pioneiros na área de RNAs foram realizados por McCulloch e Pitts. Em 1943, eles propuseram um modelo matemático de neurônio artificial, a unidade lógica com limiar (LTU), que podia executar funções lógicas simples.

Um neurônio artificial, como mostrado na página 6 do documento, possui elementos básicos:

- **Entradas:** Podem ser saídas de outros neurônios, entradas externas, um bias ou qualquer combinação desses elementos. Pense nelas como a informação que flui para o neurônio.
- **Pesos:** Cada entrada é multiplicada por um "peso". Esses pesos representam as forças de conexão sináptica. Um valor de peso igual a zero equivale à ausência da conexão associada. Neurônios podem apresentar conexões de entrada negativas ($w_j < 0$) ou positivas ($w_j > 0$).
- **Somador:** O somatório de todas as entradas, multiplicadas por suas respectivas forças de conexão sináptica (os pesos)
- **Função de Ativação:** Define a saída do neurônio por meio da aplicação de uma função de ativação à entrada total. Ela restringe (limita) o intervalo permissível de amplitude do sinal de saída a um valor finito. Tipicamente, o intervalo normalizado da amplitude da saída de um neurônio é escrito como o intervalo unitário fechado $[0, 1]$ ou alternativamente $[-1, 1]$.

Exemplo: Imagine um neurônio tentando decidir se você deve sair de casa.

- **Entradas:** Está ensolarado (sim/não)? Está quente (sim/não)? Está ventando (sim/não)?
- **Pesos:** Você pode dar um peso alto para "ensolarado", um peso médio para "quente" e um peso negativo para "ventando" (porque você não gosta de vento).
- **Somador:** Ele soma (ensolarado * peso_ensolarado) + (quente * peso_quente) + (ventando * peso_ventando).
- **Função de Ativação:** Se a soma estiver acima de um certo "limiar" (por exemplo, 0.5), o neurônio "ativa" e sua saída é "Sair de casa". Caso contrário, a saída é "Ficar em casa".

3. Tipos de Funções de Ativação: Como os Neurônios "Decidem"

Diferentes funções de ativação permitem que os neurônios produzam diferentes tipos de saídas.

- **Função Limiar (threshold):** Empregada no modelo de neurônio artificial de McCulloch e Pitts (1943). O valor do limiar define quando o resultado da função limiar será igual a 1 ou 0 (alternativamente, pode-se empregar o valor -1). Quando a

soma das entradas recebidas ultrapassa o limiar estabelecido, o neurônio torna-se ativo (saída +1).

- **Exemplo:** No nosso exemplo de "sair de casa", se a soma calculada é 0.6 e o limiar é 0.5, a saída é 1 ("Sair de casa"). Se a soma é 0.4, a saída é 0 ("Ficar em casa").
- **Função Sigmoidal Logística (bipolar):** Representa uma aproximação contínua e diferenciável da função limiar. Sua saída é um valor no intervalo aberto (0, 1), podendo apresentar diferentes inclinações.
 - **Exemplo:** Em vez de um sim/não estrito, uma função sigmoidal pode gerar uma probabilidade, como 0.8 (altamente provável sair) ou 0.2 (menos provável).
- **Função Tangente Hiperbólica:** É uma variação da função sigmoidal que utiliza o intervalo aberto (-1, +1) para o valor de saída.
- **Função Linear Retificada (ReLU):** Mais recentemente, com a popularização das redes profundas, passou a ser cada vez mais utilizada. Essa função retorna 0 se recebe um valor negativo ou o próprio valor, no caso contrário.

4. Arquiteturas das Redes Neurais Artificiais

Em uma RNA, os neurônios podem estar dispostos em uma ou mais camadas.

- **Redes Feedforward (Alimentação para frente):** A informação flui em uma única direção, da entrada para a saída, sem ciclos.
- **Redes Recorrentes:** Apresentam conexões de retroalimentação, ou feedback, permitindo que a informação flua em ciclos. O número de camadas, o número de neurônios em cada camada e a presença ou não de conexões de retropropagação definem a topologia de uma RNA.

5. Processos de Aprendizagem: Como as RNAs Melhoram

A propriedade primordial para uma RNA é sua habilidade de aprender a partir de seu ambiente e de melhorar o seu desempenho através da aprendizagem[cite:42]. Isso é feito através de um processo iterativo de ajustes aplicado a seus pesos, denominado treinamento. Um conjunto de regras bem definidas para a solução de um problema de aprendizado é chamado de algoritmo de aprendizado. Basicamente, algoritmos de aprendizado diferem na forma como os pesos sinápticos são ajustados.

Duas formas principais de aprendizado são:

- **Aprendizagem Supervisionada:** A saída desejada é conhecida e informada para que a rede compare com a saída processada. Se houver erro, a rede tenta corrigir esse erro até que a mesma forneça uma saída igual à saída desejada.
 - **Exemplo:** Ensinar uma criança a identificar animais. Você mostra uma foto de um gato e diz "gato" (saída desejada). Se ela diz "cachorro", você a corrige. A rede (o cérebro da criança) se ajusta até que ela identifique corretamente o gato.

- **Aprendizagem Não Supervisionada:** A saída desejada é obtida através de entradas repetitivas até a rede reter o conhecimento, não existindo saída informada para comparação. Deve haver redundâncias nos dados para a rede encontrar padrões ou características dos dados.
 - **Exemplo:** Dar a uma criança uma caixa de brinquedos e pedir para ela organizá-los. Ela pode organizá-los por cor, tamanho ou tipo, sem ser instruída sobre como categorizar.

6. Perceptrons: Redes Neurais Simples

O Perceptron é um algoritmo para aprendizagem de redes neurais simples (uma camada), desenvolvido nos anos 50. Ele é treinado por um algoritmo supervisionado de correção de erro e usa a função de ativação do tipo limiar. O objetivo é aprender pesos sinápticos de tal forma que a unidade de saída produza a saída correta para cada exemplo. O algoritmo faz atualizações iterativamente até chegar aos pesos corretos

O Perceptron aprende ajustando seus pesos com base no erro ($\text{saída_desejada} - \text{saída_real}$).

Após algumas iterações com todas as entradas, o Perceptron aprenderá os pesos corretos para consistentemente produzir a saída "AND".

7. Limitações dos Perceptrons e Redes Multicamadas

Um único Perceptron consegue resolver somente funções linearmente separáveis. Isso significa que ele só pode traçar uma única linha reta para separar os pontos de dados em diferentes categorias. Em funções não linearmente separáveis, como o operador XOR (onde uma única linha reta não consegue separar os casos "verdadeiro" e "falso"), o Perceptron não consegue gerar um hiperplano para separar os dados.

Para resolver problemas não linearmente separáveis utilizando RNAs, a alternativa mais empregada é adicionar uma ou mais camadas intermediárias. As redes do tipo Perceptron multicamadas (MLP, do inglês multilayer Perceptron) apresentam uma ou mais camadas intermediárias de neurônios e uma camada de saída. Ao combinar vários perceptrons lineares, é possível gerar superfícies de decisão mais complexas.

8. Backpropagation: Treinando Redes Multicamadas

Um obstáculo para utilizar redes multicamadas era a ausência de um algoritmo para o treinamento dessas redes, o que foi transposto com a proposta de um algoritmo baseado em gradiente descendente denominado back-propagation (Rumelhart et al., 1986)⁹⁴. Para que esse algoritmo seja utilizado, a função de ativação precisa ser contínua, diferenciável e, de preferência, não decrescente. A função de ativação do tipo sigmoidal obedece a esses requisitos.

O Backpropagation busca determinar um vetor de pesos que minimiza o erro, modificando-o repetidamente em pequenos passos. A cada passo, o vetor de pesos é alterado na direção que produz a maior queda ao longo da superfície de erro.

Embora amplamente utilizado, o Backpropagation não é um algoritmo ótimo e não garante sempre a melhor resposta. O algoritmo de descida do gradiente pode ficar preso em um erro mínimo local. É possível refazer o treinamento variando os valores iniciais dos pesos.

9. Aplicações das Redes Neurais Artificiais

As RNAs são utilizadas em quase todas as tarefas de aprendizado, incluindo:

- Classificação
- Previsão numérica
- Reconhecimento não supervisionado de padrões
- Programas de reconhecimento de voz e escrita
- Automação de dispositivos inteligentes
- Modelos sofisticados de padrões climáticos
- Análise de desempenho em esportes
- Detecção de Spams em e-mails
- Aplicações na Bolsa de Valores e Finanças

Elas também são fundamentais para tecnologias como AlphaGo e Google Fotos, Netflix, Spotify, Robótica e Chatbots.

SISTEMAS MULTIAGENTES

raio de comunicação : alcance

“Um agente é um sistema informático situado em um ambiente que é capaz de realizar ações de forma autônoma / independente para conseguir seus objetivos.”

Imagine uma equipe de especialistas trabalhando juntos para resolver um problema complexo. Cada especialista tem seu próprio conhecimento e habilidades, e eles se comunicam e coordenam para alcançar um objetivo comum. Um **sistema multiagente (SMA)** funciona de maneira semelhante, mas com **agentes de inteligência artificial (IA)** em vez de pessoas.

Em essência, um sistema multiagente é um sistema computacional composto por **múltiplos agentes inteligentes** que:

- **São autônomos:** Cada agente pode tomar decisões e agir por conta própria para atingir seus objetivos (que podem ser individuais ou parte de um objetivo maior).
- **Interagem:** Os agentes se comunicam, cooperam, competem ou negociam entre si para trocar informações, coordenar ações e resolver problemas.
- **Percebem seu ambiente:** Cada agente tem a capacidade de observar e interpretar o ambiente em que está inserido.
- **Têm objetivos:** Cada agente é projetado para alcançar um ou mais objetivos específicos.
- **São capazes de aprender:** Muitos sistemas multiagentes incorporam mecanismos de aprendizado para que os agentes possam melhorar seu desempenho ao longo do tempo com base em suas experiências e interações.

Características dos Sistemas Multiagentes

1. AUTONOMIA
2. REATIVIDADE
3. PROATIVIDADE
4. HABILIDADE SOCIAL

Pense da seguinte forma:

Em vez de ter um único programa de IA tentando fazer tudo, um SMA divide o problema em partes menores e atribui essas partes a diferentes agentes especializados. Esses agentes trabalham juntos, trocando informações e coordenando suas ações para alcançar uma solução que seria difícil ou impossível para um único agente alcançar sozinho.

Alguns exemplos de sistemas multiagentes em ação:

- **Robótica de enxame:** Vários robôs simples trabalhando juntos para limpar um derramamento, explorar um ambiente perigoso ou construir estruturas.
- **Sistemas de tráfego inteligente:** Agentes de IA controlando semáforos e gerenciando o fluxo de veículos para otimizar o tráfego e reduzir congestionamentos.

- **Mercados online:** Agentes representando compradores e vendedores negociando preços e realizando transações.
- **Gerenciamento de cadeia de suprimentos:** Agentes representando fornecedores, fabricantes e distribuidores coordenando a produção e a entrega de bens.
- **Sistemas de recomendação:** Múltiplos agentes analisando diferentes aspectos das preferências de um usuário para fornecer recomendações mais precisas.
- **Saúde:** Agentes auxiliando no diagnóstico, monitoramento de pacientes e coordenação de cuidados.

Em resumo, sistemas multiagentes são uma abordagem poderosa para construir sistemas de IA complexos e flexíveis, capazes de lidar com problemas distribuídos e dinâmicos através da colaboração e coordenação de múltiplos agentes autônomos.

A fonte "Aula SMA-Introd.pdf" oferece uma introdução abrangente ao campo da Inteligência Artificial, com foco particular nos Sistemas Multiagentes e nos Agentes Inteligentes.

A **revisão** dos principais pontos da fonte destaca:

- **Definição e Natureza dos Agentes:** A fonte inicia definindo um agente como um sistema computacional situado em um ambiente, capaz de realizar ações de forma autônoma para atingir seus objetivos. É enfatizado que os agentes interagem com o ambiente através de sensores e atuadores. O "estado" de um agente é composto por crenças, objetivos, planos e ações, sendo que as crenças englobam seu conhecimento e memórias, os objetivos são estados futuros desejados, e os planos são sequências de ações para alcançar esses objetivos. Os agentes são ativos, executando sem intervenção direta e controlando seu próprio estado e comportamento, podendo até aprender e modificar seus planos.
- **Propriedades Essenciais dos Agentes:** Quatro propriedades são destacadas como fundamentais para os agentes:
 - **Autonomia:** Operam sem intervenção direta e possuem controle sobre suas ações e estados internos.
 - **Reatividade:** Percebem o ambiente e reagem a essas percepções, especialmente em ambientes dinâmicos e variáveis.
 - **Pró-Atividade:** Não se limitam a reagir, mas tomam a iniciativa e exibem comportamento direcionado a objetivos.
 - **Habilidade Social:** Capazes de interagir com outros agentes através de uma linguagem de comunicação, muitas vezes dependendo da cooperação para cumprir objetivos.
- **Conceito de Racionalidade e Medidas de Desempenho:** A racionalidade de um agente é avaliada por uma "medida de desempenho" que quantifica o sucesso. A definição de racionalidade em um dado momento depende da medida de desempenho, do conhecimento prévio do agente sobre o

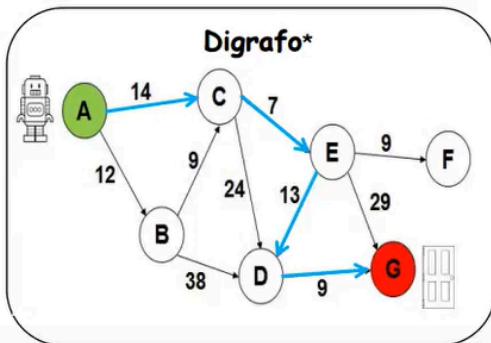
ambiente, das ações que pode executar e da sequência de percepções recebidas. O comportamento de um agente é avaliado por suas consequências (Consequencialismo).

- **Sistemas Multiagentes (SMA):** A fonte explora a transição de agentes individuais para sistemas multiagentes, onde vários agentes interagem entre si, seja colaborando, cooperando, negociando ou competindo para atingir objetivos. Os SMA são uma estratégia de "dividir e conquistar" para problemas complexos, oferecendo divisão de responsabilidades, distribuição de conhecimento, heterogeneidade (agentes especialistas), concorrência e distribuição, o que proporciona flexibilidade, escalabilidade e tolerância a falhas.
- **Aplicações dos Agentes e SMAs:** Diversas aplicações práticas são apresentadas, demonstrando a versatilidade dos agentes inteligentes:
 - **Logística e Cadeias de Suprimento:** Otimização de rotas, estoques e negociações (Ex: Amazon, FedEx).
 - **Veículos Autônomos e Tráfego Inteligente:** Redução de congestionamentos e prevenção de acidentes em simulações urbanas.
 - **Jogos e Simulações:** Simulação de NPCs e estratégias em jogos (Ex: The Sims, StarCraft).
 - **Saúde e Diagnóstico Distribuído:** Monitoramento remoto de pacientes e auxílio em diagnósticos (Ex: Triagem automatizada).
 - **Finanças e Mercados:** Robôs de trading e simulações de mercado.
 - **Ambientes Inteligentes (Smart Homes):** Gerenciamento de energia, segurança e conforto via dispositivos IoT.
 - **Controle de Satélites:** NASA buscando satélites mais autônomos.
- **Propriedades dos Ambientes:** Os ambientes em que os agentes operam são classificados por diversas propriedades:
 - **Determinístico x Não-Determinístico:** Se o próximo estado é completamente determinado (Ex: mundo do aspirador de pó) ou incerto (Ex: motorista de táxi).
 - **Estático x Dinâmico:** Se o ambiente muda enquanto o agente pensa (dinâmico) ou permanece fixo (estático). Ambientes dinâmicos apresentam desafios devido a processos paralelos que modificam o ambiente sem controle do agente.
 - **Discreto x Contínuo:** Refere-se à natureza finita/infinita e claramente definida (discreto, Ex: xadrez) ou contínua (contínuo, Ex: dirigir um táxi) das percepções, ações e estados.
 - **Acessível x Inacessível:** Se o agente pode obter informações completas, precisas e atualizadas sobre o estado do ambiente

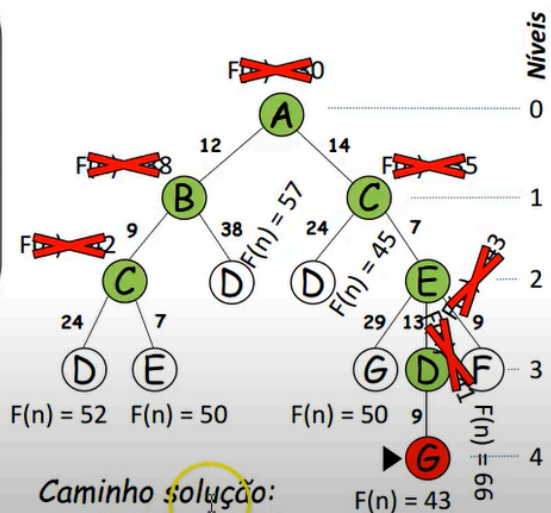
- (acessível) ou não (inacessível). Ambientes mais acessíveis simplificam o desenvolvimento de agentes.
- **Episódico x Sequencial:** Se cada episódio de percepção e ação é independente dos anteriores (episódico, Ex: controle de qualidade na linha de montagem) ou se as ações têm consequências de longo prazo (sequencial, Ex: xadrez, dirigir táxi).
 - **Agente Único x Multiagente:** Se há um único agente ou vários agentes interagindo, podendo ser cooperativos ou competitivos.
 - **Framework PEAS:** Introduzido como um acrônimo para descrever um ambiente de tarefa em termos de Desempenho (Performance), Ambiente (Environment), Atuadores (Actuators) e Sensores (Sensors).
 - **Desempenho:** Medida de sucesso ou eficácia do agente.
 - **Ambiente:** O contexto externo em que o agente opera.
 - **Atuadores:** Mecanismos que o agente usa para agir sobre o ambiente.
 - **Sensores:** Mecanismos que permitem ao agente perceber e obter informações do ambiente. Um exemplo prático de um aspirador de pó automático ilustra como aplicar o PEAS.
 - **Tipos de Agentes:** A fonte categoriza cinco tipos básicos de agentes:
 - **Agentes Reativos Simples:** Baseados em regras condição-ação, selecionam ações apenas com base na percepção atual (Ex: agente aspirador de pó).
 - **Agentes Reativos Baseados em Modelos:** Podem lidar com ambientes parcialmente observáveis, mantendo informação interna para distinguir estados do mundo aparentemente iguais.
 - **Agentes Baseados em Objetivos:** Tentam atingir um estado específico, podendo exigir técnicas de planejamento e busca para sequências de ações (Ex: táxi decidindo virar).
 - **Agentes Baseados na Utilidade:** Definem um grau de satisfação ("utilidade") para os estados, avaliando metas competitivas e guiando a busca. Permitem decisões racionais em caso de objetivos conflitantes ou incertos (Ex: jogos, balancear velocidade e segurança).
 - **Agentes com Aprendizagem:** Permitem que o agente atue em ambientes desconhecidos e se torne mais eficiente, com componentes como elemento de aprendizagem, crítico, elemento de desempenho e gerador de problemas.

Em suma, a fonte é um material introdutório e didático que estabelece as bases conceituais para a compreensão de agentes inteligentes e sistemas multiagentes, desde suas definições e propriedades até suas aplicações e classificações de ambientes e tipos de agentes.

3. Busca A*



Estado	H
A	30
B	26
C	21
D	7
E	22
F	36
G	0



Caminho solução:
A → C → E → D → G
Custo: 43

<https://www.youtube.com/watch?v=FU6JQaRMMDM>