Sistemas de aprendizagem

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal

Aprendizagem e Extração de Conhecimento, Trabalho prático Nº1

17 de Novembro de 2020

Resumo A automatização é uma das áreas informáticas com mais destaque atualmente. Esta tem crescido e evoluído muito rapidamente ao longo dos anos e encontra-se presente no nosso quotidiano. Ela é impulsionada por modelos de inteligência artificial cada vez mais capazes e que utilizam algoritmos de aprendizagem interativos, permitindo fazer uma análise complexa de dados com alta precisão.
Com recurso a estes algoritmos é possível identificar padrões em grandes volumes de dados que, mais tarde, irão permitir prever futuros cenários prováveis, de forma a conseguir uma solução altamente fiável para a resolução dos vários problemas que nos são apresentados.
Existem vários tipos de aprendizagem, todavia apenas iremos abordar três, descrevendo o método de aprendizagem respetivo, bem como as características diferenciadoras, o modo como exibem a capacidade de aprendizagem, as ferramentas usadas nas implementações e qual o papel desempenhado no mercado.

Keywords: Machine Learning · Case-based Reasoning · Genetic Algorithms · Artificial Neural Network

1 Introdução

Este artigo é o resultado da investigação sobre o funcionamento e aplicabilidade dos vários sistemas de aprendizagem, mais precisamente, *Raciocínio baseado em casos*, *Algoritmos genéticos* e *Redes neuronais artificiais*, no âmbito da unidade curricular de Aprendizagem e Extração de Conhecimento.

Esta área têm ganho cada vez mais ”força” devido aumento da utilização de algoritmos *Machine Learning* (processo de ”ensinar” um sistema a fazer previsões com muita exatidão a partir de um conjunto de dados), por exemplo, em sistemas de recomendação na *Netflix*, *YouTube*, *Spotify*, no motor de busca da *Google*, no *feed* de redes sociais como *Facebook*, *Twitter* ou em assistentes pessoais como a *Siri* ou *Alexa*.
Este tipo de algoritmos é *data driven*, onde a aprendizagem é alcançada através da execução de algoritmos de forma a criar modelos de representação de conhecimento, isto é, são encontrados padrões que permitem que sejam produzidas ”conclusões”, sem necessidade de programar a solução e através da análise estatística de dados de *input*.

O universo dos sistemas de aprendizagem é composto por vários algoritmos, cada um com as suas características e especificidades.
Cada um destes algoritmos está acompanhado de uma descrição geral do seu funcionamento e da sua capacidade de aprendizagem, as vantagens e desvantagens e, ainda, as ferramentas de desenvolvimento e soluções no mercado.

2 Case Based Reasoning

2.1 O que é RBC?

O *Raciocínio Baseado em Casos* utiliza experiências passadas para analisar, propor soluções e, por fim, resolver problemas. Esta abordagem seleciona a(s) experiência(s) mais similar(es) bem-sucedida(s), de forma a sugerir uma solução para o novo problema. Sempre que um caso é resolvido, é guardado em memória, permitindo interpretar um problema futuro de uma forma mais informada, algo que, noutra situação, seria necessário começar do zero.
A razão desta abordagem, baseia-se no principio: "*problemas similares têm soluções similares*".

2.2 Como funciona o RBC?

O *Raciocínio baseado em casos* pode ser implementado de diversas formas, sendo estas:

Raciocínio Baseado em Memórias: Nesta perspetiva de raciocínio, as memórias de um caso específico são usadas diretamente para tomar uma decisão.
Essencialmente, a nova situação é observada, sendo feita uma procura de uma ocorrência similar na memória e, imediatamente, temos uma solução com uma confiança associada.
As vantagens deste sistema são a facilidade na geração de exemplos e, visto existir um mecanismo de confiança na resposta [1] que oferece uma noção de qualidade, conseguir obter conhecimento significativo.
No entanto para inferir regras este não será o melhor sistema. [1]
A essência destas abordagens prende-se na organização/acesso dos casos guardados na memória, para realizar uma procura mais eficiente e rápida. [2]

Raciocínio Baseado em Exemplos: O objetivo deste tipo de raciocínio é classificar, ou seja, assumindo que cada caso pertence a uma ou mais classes, deve ser atribuído um conjunto de classes a um dado caso. Assim sendo, um conjunto de classes interligadas resultam numa possível solução.

Raciocínio Baseado em Instâncias: A descrição deste conceito inclui um conjunto de instâncias, possivelmente, um vetor ou até um *set* de tuplos `<key,value>`. Quando há falta de informação relativamente à descrição do caso, usa-se as instâncias em conjunto com a descrição para encontrar os casos similares e classificar o caso com uma maior precisão. [3]

Raciocínio Baseado em Analogia: Os métodos de raciocínio baseados em analogia permitem raciocinar sobre propriedades de um caso, com base em semelhanças entre casos de domínios diferentes, sendo esta a principal diferença entre os métodos apresentados anteriormente. O foco deste método está na adaptação de casos passados ao caso atual.

2.3 Ciclo RBC

O **RBC** pode ser descrito como um processo cíclico, como se pode observar na imagem **1**. Para a resolução de um problema são necessários quatro passos:

1. Quando surge um problema novo, este é analisado para que se possam escolher os casos passados que irão ajudar na resolução do problema. (Para isso, é necessário estabelecer regras para que se possa calcular a similaridade entre casos) (**Retrieve**);
2. Através de uma função de similaridade, o sistema propõe uma sequência ordenada de soluções de casos anteriormente resolvidos [4], adaptando a primeira solução, ou seja, a mais similar, ao novo caso (**Reuse**);
3. A solução proposta é avaliada no contexto do problema atual (**Revise**), existindo dois cenários possíveis:
   - **3.1.** No caso de sucesso, avançamos para um processo de aprendizagem (passo 4);
   - **3.2.** No caso de insucesso, é necessária a correção da solução (**Repair**), que passa por enviar o problema novamente para o passo 2, propondo, assim, outra possível solução;
4. Caso o modelo implementado seja hierarquizado, passamos para um processo de aprendizagem que consiste em selecionar a informação que deve ser retida na base de conhecimento, determinar os índices de acordo com regras de indexação (para garantir uma ordem e uma pesquisa mais rápidas) e, por fim, integrar o caso indexado na base de conhecimento (**Retain**). Caso contrário (ou seja, não hierarquizado), apenas é feita a seleção da informação, sendo imediatamente armazenada na base de conhecimento. Deste modo, o caso está apto para solucionar problemas futuros, aumentando a capacidade de resposta do sistema.

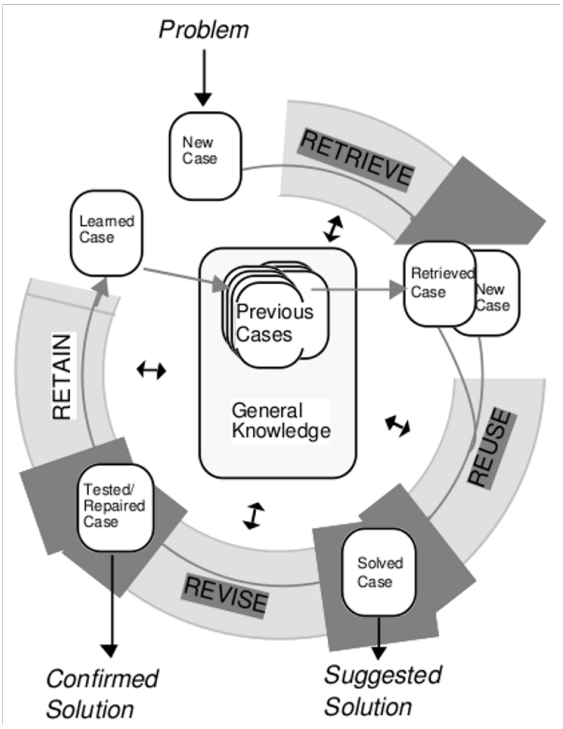

Figura 1. Ciclo de vida de um sistema que implementa *Raciocínio Baseado em Casos*

2.4 Vantagens e Desvantagens

A grande diferença deste algoritmo para outros algoritmos de KBS, *knowledge-based systems*, em inteligência artificial prende-se no facto das experiências serem sempre guardadas como experiências distintas, ao contrário do que acontece normalmente, onde o conhecimento é integrado numa estrutura de regras, redes, entre outros.
Uma vantagem desta abordagem consiste em não ser necessária uma capacidade de extração de conhecimento muito desenvolvida, e, para além disso, oferecer uma enorme flexibilidade na modelação. Também necessita de menos dados de treino que outros algoritmos e o sistema é capaz de aprender/raciocinar mesmo com dados inconclusivos ou imprecisos, sendo que dificilmente cometerá os erros do passado. Sempre que o sistema resolve um problema, este e a sua solução são adicionadas à base de "casos", permitindo que o sistema aprimore o seu raciocínio com o tempo/utilização, tal como um humano quando ganha experiência.
Contudo, muitos argumentam que o **RBC** se baseia em dados não-confiáveis e que a adaptação de soluções de problemas passados a novos problemas pode ser complexa e levar a erros.

2.5 Ferramentas para desenvolvimento

Existem várias aplicações que permitem o desenvolvimento de sistemas com aprendizagem baseada em casos, por exemplo:

- **AIAI CBR *shell***: Realiza a classificação com base na comparação de casos. Os parâmetros do algoritmo podem ser variados como, por exemplo, especificar o número de vizinhos mais próximos, definir manualmente os pesos, entre outros. É possível ser executada através do Java.
- ***Caspian***: Criada na Universidade do País de Gales, *University of Wales*, com o objetivo de analisar falhas e efeitos consequentes na industria, em 1998.
- ***CloodCBR***: Aplicação académica com uma interface *WEB*, criada na Universidade *Robert Gordon*, na Escócia. Esta tem um sistema *CBR* genérico, distribuído e altamente escalonável, baseado numa arquitetura de micro serviços.
- ***Colibri***: Disponibiliza uma interface para desenvolver sistemas **RBC**. Os desenvolvedores desta plataforma oferecem uma biblioteca de Java (*jcolibri3*), que pode facilmente ser incluída num projeto *maven*. É ainda possível realizar o *import* através do ficheiro *.jar* para implementar este tipo de sistemas.
- ***myCBR***: Criada na *University of West London*, tem uma interface que permite configurar tarefas para modelar modelos de conhecimento ou de extração de conhecimento, entre outros. Também possui uma biblioteca em Java que permite maior flexibilidade nas configurações.

Algumas destas ferramentas são antigas, pois, atualmente, são usadas combinações de algoritmos para uma maior fiabilidade nos resultados obtidos.

2.6 Soluções no mercado

Dadas as capacidades de funcionar num mundo onde não existe um acesso perfeito a toda a informação, o *Raciocínio baseado em casos* tem vindo a ser utilizado em diversas áreas e é muitas vezes combinado com outros métodos, para uma maior eficiência.
Por exemplo, no caso da diabetes tipo 1, várias sequências *DNA* associadas às células de tipo T (subtipo de Linfócitos, isto é, células responsáveis pelo sistema imunológico) foram usadas para detetar novos casos. Noutros aspetos da medicina, estes sistemas também são usados como, por exemplo, na deteção de casos oncológicos, apoiar decisões para gestão médica de emergência nas reuniões de grande dimensão, entre outros. [5]
Por outro lado, existem vários sistemas de recomendação que usam este tipo de abordagem. Um deles, recentemente analisado, é um sistema de atribuição do nível de inglês a um aluno, em que este realiza um teste que é separado por secções, sendo atribuída uma pontuação a cada uma. Através da pontuação é atribuído/procurado o caso mais semelhante, prosseguindo o método **RBC**. [6]
Por fim, existem outros campos de aplicação como direito, deteção de fraudes, diagnóstico de falhas em veículos, análise da qualidade do *source code*, entre outros.

Atualmente, existem algumas empresas a trabalhar nesta área, sendo estas algumas das mais relevantes: [7]

- **IBM** - Empresa multinacional americana de tecnologia e consultoria; [8]
- **Microsoft Corporation** - Uma gigante tecnológica na produção de software e outros produtos eletrónicos, com sede em Washington; [9]
- **General Electric** - Conglomerado multinacional de Nova York líder em áreas como energia, aviação, saúde, entre outros. [10]

3 Genetic Algorithms

3.1 O que são *Genetic Algorithms*?

Um algoritmo genético (*genetic algorithm*) é um algoritmo de procura baseado na teoria de *Charles Darwin* da seleção natural. Assim, este algoritmo comporta-se de uma forma semelhante, em que indivíduos (controladores) através de reproduções, mutações e outras operações genéticas, formam uma nova geração, com vista a otimizar um objetivo (normalmente minimizar ou maximizar uma **função objetivo**).

3.2 Seleção natural

De forma a compreender melhor o conceito de *Genetic Algorithms* é necessário entender o que é e como funciona a seleção natural. A seleção natural proposta por *Charles Darwin* baseia-se no facto das características favoráveis de uma geração serem herdadas pelas seguintes e tornarem-se mais comuns, permitindo a sobrevivência da população. Pelo contrário, as características menos favoráveis acabam por se tornar numa minoria nas novas gerações.

Deste modo, os algoritmos genéticos pretendem aplicar este conceito de seleção natural para a aprendizagem e otimização de problemas.

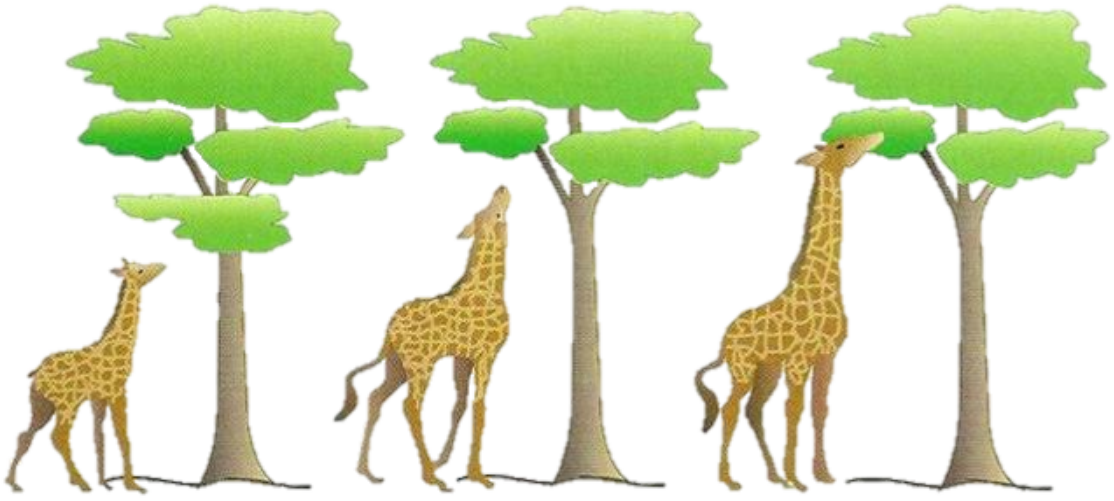

Figura 2. Perspetiva da seleção natural proposta por *Charles Darwin*

3.3 Como funcionam os *Genetic Algorithms*?

Primeiramente, existe uma população inicial (geração $k = 0$). Esta população é constituída por um número de indivíduos/controladores que são formados por uma sequência de *bits*. Assim, cada *bit* do controlador caracteriza um gene, sendo que a sequência total pode ser comparada a um cromossoma que representa o indivíduo. [11]

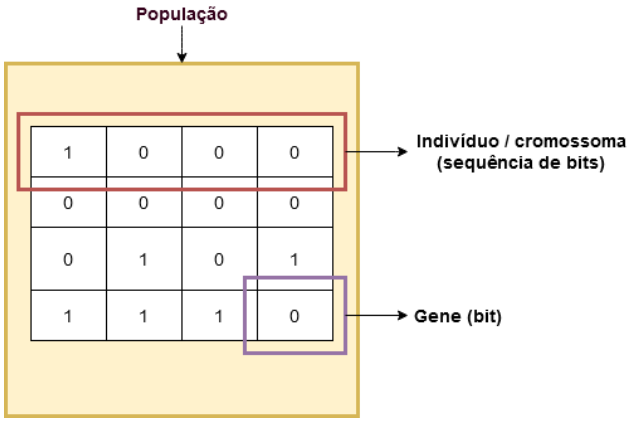

Figura 3. População, gene e cromossoma de um *Genetic Algorithm*

Com a primeira geração definida[1], é preciso implementar uma função de custo (ou *fitness function*) que determina a performance do controlador[2], sendo que a sua probabilidade de seleção é baseada neste custo.

Podemos então, ordenar os controladores consoante a sua performance e avançar para a próxima geração com a aplicação de operações genéticas, como as seguintes mais comuns:

- **Elitism**: Quando um controlador apresenta uma boa performance, é recorrente transferi-lo para a próxima geração sem que sofra alterações na sua sequência de *bits*.
  Esta operação permite ao algoritmo garantir que os melhores indivíduos continuem nas próximas gerações com vista a continuar a optimizá-la, se possível;

- **Replication**: Um dos controladores é replicado para a nova geração sem que ocorra nenhuma alteração. A escolha do controlador é feita de forma aleatória, pelo que pode ser replicado um controlador com baixa performance.
  Esta operação serve para manter a aleatoriedade da transição entre gerações;

- **Crossover**: Esta operação segue uma similaridade com a reprodução de indivíduos, tendo apenas a diferença de se produzirem dois controladores "filhos", ou seja, os controladores resultantes possuem características dos dois controladores que o geraram.
  Assim, são escolhidos dois controladores de alta performance e seleciona-se, aleatoriamente, partes da sequência de *bits* de cada um, isto é, são selecionadas partes de *bits* do controlador X e colocadas em Y, e vice-versa. De seguida, ambos passam para a nova geração.
  Esta operação genética serve para explorar a sequência de *bits* já existente, tentando otimizá-la;

[1] Normalmente as sequências são geradas através de aleatoriadade de *bits*.
[2] Num problema de minimização tenciona-se minimizar a *fitness function*, p.e., a distância percorrida num trajeto.

– **Mutation**: Nesta operação são selecionados *bits* de forma aleatória para a sua alteração.
  Esta operação tem um peso importante para a exploração de novas sequências de *bits* e pode provocar um aumento ou descida drástica da performance do indivíduo. [12]

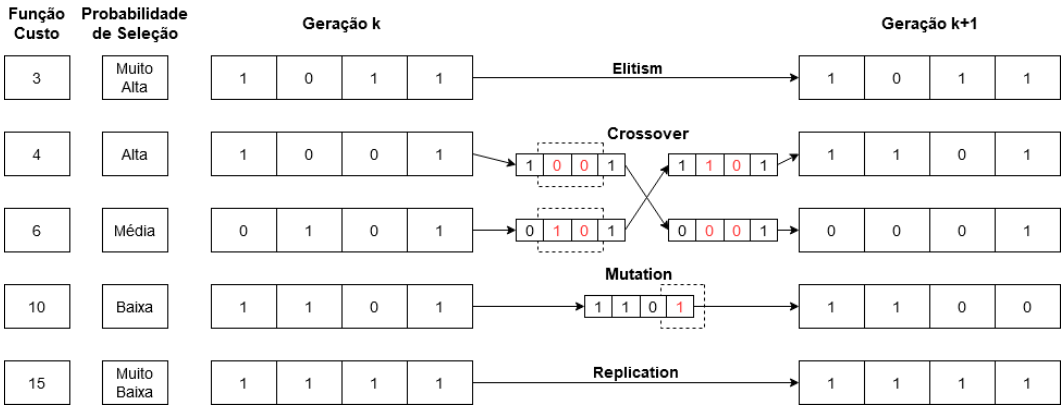

Figura 4. Exemplo de operações genéticas num modelo de minimização

É importante realçar que se podem aplicar todas estas operações numa geração, sendo que podemos definir que, p.e., 30% dos controladores sofrem *Elitism*, 20% *Replication* e 50% *Mutation*, de forma a obter melhores resultados nas próximas gerações.

Desta forma, o algoritmo aprende a cada iteração quais os controladores com melhor performance no problema, pelo que é possível a sua optimização até ser obtido um valor aceitável (previamente definido) ou atingir o limite de recursos disponíveis.

3.4 Vantagens e desvantagens

As principais vantagens deste algoritmo são:

– O seu conceito ser de fácil compreensão;
– Utilizar probabilidades na transição de gerações, pelo que, não é um modelo determinístico;
– Permitir a sua análise com facilidade;
– Ter um bom funcionamento com variáveis contínuas.

Contudo, este algoritmo também apresenta algumas desvantagens:

– O desenvolvimento da função objetivo pode, por vezes, ser difícil;
– Ser computacionalmente pesado.

3.5 Ferramentas de desenvolvimento

Um *Genetic Algorithm* pode ser aplicado de diferentes formas, visto que, é possível a sua implementação com diferentes tecnologias presentes no mercado atual.

Ferramentas como o *Matlab*, *Excel* e várias linguagens de programação (p.e. *Python*, *Java*,...) permitem o uso destes algoritmos de forma a otimizar os parâmetros do problema em causa.

No caso do *Python*, a *package geneticalgorithm* permite o desenvolvimento e implementação deste tipo de modelos de *Machine Learning* com facilidade. [13]
Por outro lado, no *Matlab* o uso de *genetic algorithms* pode ser efetuado através da função `ga` que se encontra disponível no *Global Optimization Toolbox*. [14]
Caso o utilizador prefira linguagens mais tradicionais como o *Java*, o uso da *library Jenetics* permite acesso a todas funções necessárias para o desenvolvimento destes algoritmos. [15]
Por último, até programas como o *Excel* permitem a aplicação destes algoritmos através da *package SolveXL*, escrita em *C++*, que possibilita vários algoritmos de otimização evolucionários. [16]

3.6 Aplicações deste algoritmo no mercado

Os algoritmos genéticos podem ser utilizados para otimizar outros modelos de aprendizagem, pois, como são algoritmos de procura que seguem uma heurística, permitem um melhor desempenho dos modelos previamente desenvolvidos. Um exemplo disso é a junção de redes neuronais com os algoritmos genéticos de forma a aprender melhor e de forma mais eficiente. [17]

Nesta aplicação, muito utilizada por empresas atualmente, os algoritmos genéticos ajudam a definir os *hyperparameters* (valores necessários para a rede neuronal operar) que não conseguem ser aprendidos pela rede neuronal. Assim, o uso de *Genetic Algorithms* vem mudar essa perspetiva, sendo agora possível aprender esses valores.

Para além da otimização de outros modelos, os algoritmos genéticos são muito usados na robótica para o controlo de movimento automático. [18]

A nível de *startups* temos o exemplo da *ReverieLabs*, *VERIKAI* e *Dynamic AI56* que utilizam estes algoritmos para diversas áreas de mercado como, por exemplo, seguros, farmacêutica e telecomunicações. [19]

4 Artificial Neural Networks

4.1 O que é ANN?

Uma **ANN**, *Artificial Neural Network*, ou em português, **RNA**, Rede Neuronal Artificial, é um sistema computacional inspirado nas redes neuronais biológicas que constituem o cérebro de um animal.

Uma **ANN** é baseada numa coleção de nodos interligados denominados de neurónios artificiais que desempenham um papel similar aos neurónios biológicos. Cada ligação, tal como as sinapses num cérebro biológico, pode transmitir um sinal para outros neurónios.
Portanto, um neurónio artificial que recebe um sinal, processa-o e pode enviar outro sinal aos neurónios ligados a si. O sinal é um número real e o *output* de cada neurónio é processado por uma função não linear sobre a soma dos seus *inputs*. Os neurónios e as ”sinapses” normalmente têm um peso que se ajusta durante o processo de aprendizagem, isto é, o peso permite aumentar ou decrescer a força e importância da ligação. Os neurónios podem ter um limite inferior que permite que o sinal seja enviado, caso o limite seja excedido.
Tipicamente, os neurónios estão agregados em camadas, sendo que, diferentes camadas podem ter distintas transformações nos seus *inputs*. Os sinais navegam desde a primeira camada, ou seja, a camada de *input*, até à última camada, a camada de *output*.

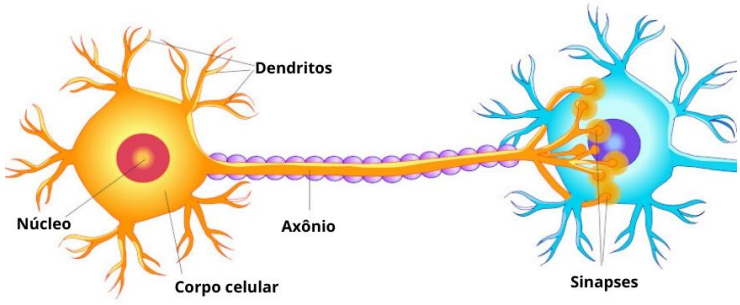

Figura 5. Neurónio Biológico

4.2 Componentes

Neurónios: Os neurónios artificiais são derivados concetualmente dos neurónios biológicos. Cada neurónio tem um ou mais *inputs* e produz um único *output* que pode ser enviado para outros neurónios.
Os *inputs* podem ser provenientes tanto de dados externos, como de imagens ou documentos, ou do *output* de outros neurónios.
Os *outputs* dos neurónios da camada final retornam o resultado da tarefa pretendida, como por exemplo, reconhecer um objeto numa imagem.
Ligações e Pesos: A rede contém ligações, cada uma conecta o *input* de um neurónio com o *output* de outro. Cada ligação tem também um peso associado que representa o grau de importância da ligação. Um neurónio pode ter várias ligações de *input* e *output*.
Função de Propagação: A função de propagação é usada para transportar valores através dos neurónios de uma camada da rede neurológica. Usualmente, os valores de *input* são acrescentados e passados a uma função de ativação que gera um *output*. [20]

4.3 Treino

Redes neuronais são treinadas processando exemplos, sendo cada um destes constituído por um *input* e um resultado conhecidos. Através da sua associação, são formados pesos probabilísticos que são armazenados diretamente nas estruturas de dados da rede.
Normalmente, o treino da rede rege-se pela diferença entre o *output* processado pela rede e o resultado esperado (através de exemplos). Este desfasamento é conhecido por erro.
A rede reajusta os pesos das suas associações de acordo com a regra de aprendizagem e o valor do erro. Estes ajustes sucessivos irão permitir à rede produzir um *output* cada vez mais similar ao resultado pretendido. Assim que um certo critério é verificado, o processo de treino pode ser terminado. Este processo é denominado por *supervised learning*.
Estes sistemas aprendem a realizar tarefas considerando exemplos, geralmente sem serem programados com regras baseadas em tarefas. A título de exemplo, no reconhecimento de imagens, pode ser pedido à rede para tentar identificar quais fotos contém gatos e, para isso, são fornecidas várias imagens, umas classificadas de ”Gato” e outras de ”Sem Gato”, cujo resultado de treino é usado para identificar gatos noutras imagens. Isto é feito sem qualquer noção do que é um gato (que possuem quatro patas, cauda,...), ao invés disso, a rede gera automaticamente as características através dos exemplos processados.

4.4 Organização

Os neurónios estão, tipicamente, distribuídos por várias camadas, especialmente em *deep learning*.
Os neurónios de uma dada camada só estão ligados a neurónios ou da camada imediatamente precedente ou da seguinte, sendo possível haver diversos padrões de conexões.
A primeira camada, camada que recebe os dados externos, é chamada de camada de entrada, *input layer*. A última camada, camada que produz o resultado final, é a camada de saída, *output layer*. Entre estas duas, podem ou não existir várias camadas escondidas, *hidden layers*, assim como, diversos padrões de conexões.
As redes podem ser *fully connected*, isto é, os neurónios de uma dada camada estão ligados a todos os neurónios da camada seguinte. Outro tipo de organização é o *pooling*, onde um conjunto de neurónios de uma dada camada estão ligados a um único neurónio da camada seguinte.
As redes que têm neurónios cujas conexões formam um grafo acíclico dirigido são chamadas de *feedforward networks*, enquanto que redes que permitem conexões recorrentes entre os neurónios na mesma ou em camadas anteriores são conhecidas por *recurrent networks*.

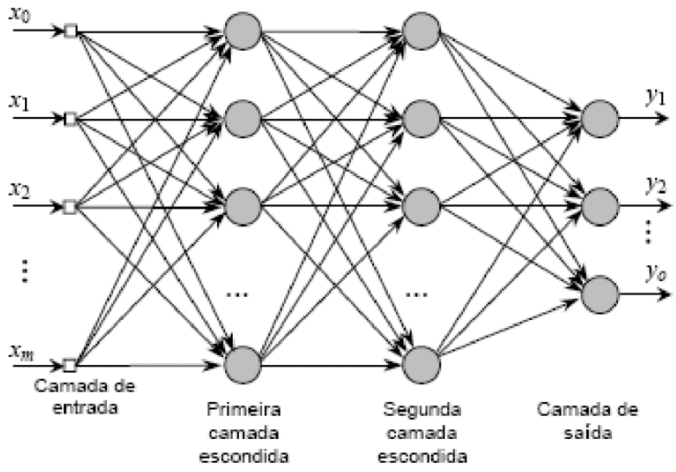

Figura 6. Rede Neuronal

4.5 Paradigmas de treino

Existem três grandes paradigmas de aprendizagem - *supervised learning*, *unsupervised learning* e *reinforcement learning* - tendo, cada uma, diferentes métodos de aprendizagem.

Supervised learning: Usa um par de *inputs* e os *outputs* desejados. O objetivo da aprendizagem é a produção do *output* desejado para cada *input*. Neste caso o custo da função está relacionado com a eliminação das deduções erradas. Este método é usado em tarefas de reconhecimento de padrões, regressões e dados sequenciais, como é o caso do reconhecimento de voz e gestos. No geral, pode-se pensar neste paradigma como aprender com um "professor", em forma de uma função que fornece sempre *feedback* sobre a qualidade das soluções obtidas pela rede.

Unsupervised learning: Procura padrões não detetados anteriormente num conjunto de dados, sem qualquer tipo de etiquetas e com o mínimo de supervisão humana, ao contrário do método anterior.

Reinforcement learning: Pretende escolher as melhores ações num determinado ambiente, de maneira a maximizar a recompensa. Não precisa de etiquetas nos pares *input*/*output* e também não necessita de ações subótimas para ser corrigido. Em vez disso, foca-se num equilíbrio entre *exploration* e *exploitation*.

4.6 Vantagens e Desvantagens

Algumas das vantagens que este algoritmo apresenta são: [21]

- A informação fica guardada em toda a rede, ao invés de numa base de dados;
- Habilidade de trabalhar com conhecimento incompleto;
- Tolerância a falhas e corrupção de células;
- Poder ter a memória distribuída;
- Habilidade de aprender com eventos passados e aplicar esse conhecimento em eventos similares;
- Capacidade de processar dados em paralelo, podendo realizar várias tarefas ao mesmo tempo.

Também se podem encontrar desvantagens deste algoritmo, como por exemplo: [21]

- Dependência de *Hardware* (necessitam de processadores com poder de processamento paralelo de acordo com a estrutura da rede);
- Comportamento complexo ou inexplicável da rede, pois quando é dada uma solução, não é fornecida nenhuma informação de como, nem porquê;
- Difícil determinação de como deve estar estruturada a rede;
- Duração de treino da rede é desconhecido e dependente de um valor de erro previamente definido.

4.7 Ferramentas de desenvolvimento

As **ANN**s podem ser utilizadas de diversas formas, sejam estas mais simples ou mais complexas. Assim, existem várias ferramentas de apoio ao desenvolvimento destas redes.

Estas ferramentas podem ser encontradas, por exemplo, no *MATLAB*, como é o caso das ferramentas *Adaptive Neural Networks* e *Deep Learning Toolbox*.
Na linguagem *Python* existem várias ferramentas conhecidas, como *PyTorch*, *NeuroLab*, *ffnet* (*feed-forward neural network*), entre outras. Na linguagem `C` é possível importar a biblioteca *FANN* (*Fast Artificial Neural Network*) e, ainda, na linguagem `C++` a biblioteca *OpenNN*. [22]

Estas são algumas enumerações de ferramentas em linguagens, contudo é possível encontrar muitas mais ferramentas destas e de outras linguagens.

4.8 Aplicações deste algoritmo no mercado

Este algoritmo está espalhado por todo mundo, com grandes empresas a dar o seu contributo para o seu desenvolvimento e utilização.

Os maiores vendedores no mercado global incluem *Starmind* (Suíça), *Google*, *IBM*, *Oracle*, *Microsoft*, *Intel*, *Qualcomm*, *Alyuda*, *LLC*, *NeuralWare*, *Neurala* (todas dos EUA), entre outras. Estes vendedores adotaram estratégias de crescimento diferentes, assim como lançamentos de vários novos produtos, parcerias, colaborações e aquisições, de modo a expandir a sua presença no mercado global de redes neuronais.
Algumas áreas onde este algoritmo é utilizado são Serviços de Gestão, Serviços Profissionais (Consultoria, Suporte e Manutenção, Implementação e Integração), Aplicações (Reconhecimento de Imagem, Reconhecimento de Sinais, Mineração de Dados, Sistemas de Recomendações e Descoberta de Medicamentos). [22]

4.9 Exemplos de aplicações deste algoritmo no mercado

Alguns exemplos de acontecimentos deste tipo são: [22]

- Em Novembro de 2015, *Google* introduziu *TensorFlow*, um motor AI *open-source* que implementa tecnologia *deep-learning* para as suas operações computacionais.
- Em Junho de 2017, *Google* lança *MultiModel*, uma arquitetura de rede neuronal que utiliza o sucesso da visão, linguagem e redes auditivas para resolver um número de problemas que abrangem vários domínios, como reconhecimento de imagens, reconhecimento por voz e tradução.
- Em Maio de 2018, a *Microsoft* lançou uma nova funcionalidade para o *Microsoft Translator*, que permite aos utilizadores criar personalizações na tradução de texto e voz.
- Em Agosto de 2019, a *IBM* anunciou o primeiro lançamento do *Python SDK* e *Node.js SDK* para interação com as descobertas da `API` do serviço da *IBM* denominado *Cloud Security Advisor*, de modo a acelerar a integração da plataforma dos clientes com o serviço.

5 Conclusão

Durante a elaboração deste artigo, concluímos que, apesar de todos os sistemas de aprendizagem serem diferentes, têm um propósito comum, o de solucionar problemas de forma inteligente, rápida e fiável.
Todos os algoritmos aqui apresentados mostraram-se úteis quando implementados para problemas adequados, portanto, a escolha do algoritmo é muito importante nos sistemas de aprendizagem, visto que é uma decisão que irá condicionar todo o processo.

Outro aspeto de ressalvar é o ”desuso” atual de certos algoritmos, como é o caso do *Raciocínio baseado em casos*, pois, atualmente, existem várias alternativas, sendo uma delas a combinação de vários algoritmos disponíveis ou até a combinação de vários que permitem uma solução ótima. Para além disso, existem poucas ferramentas disponíveis no mercado.

Desta forma, a aprendizagem baseada em casos mostra-se bastante adequada para situações onde a rapidez é requisito, pois a resolução para os vários problemas é baseada em soluções passadas e não é necessário começar o processo do zero. Também assegura que os erros passados não são propagados, pois tem ”memória” das experiências passadas e permite obter soluções para todo o tipo de domínio, visto que consegue trabalhar com dados inconclusivos.
No entanto, é um algoritmo que aceita resultados sem espírito crítico e pode tornar-se tendencioso para certo tipo de soluções.

Já os algoritmos genéticos são vantajosos quando o espaço de procura é grande, assim como a existência de variáveis a considerar. Este algoritmo é bastante preciso e eficiente, devido à sua componente de aleatoriedade na procura de soluções, aos métodos de desenvolvimento e à evolução de soluções encontradas.
Todavia, apesar de ser um algoritmo com uma capacidade de aprendizagem enorme, assim como de evolução do conhecimento retido, este apenas consegue focar-se na otimização de valores. É também computacionalmente dispendioso e não existe a garantia de encontrar uma solução.
Uma vantagem deste algoritmo é a autonomia que apresenta, *unsupervised learning*, pois, assumindo que a função objetivo está bem definida, o crescimento da população é feito sem intervenção externa, e, dessa forma, chega à solução.

Por fim, os algoritmos de redes neuronais são vantajosos quando se quer trabalhar com eventos similares ou que se repetem, pois estes têm a capacidade de tomar decisões tendo em conta eventos passados.
Este algoritmo tem uma grande capacidade de processamento paralelo permitindo trabalhar em vários aspetos ao mesmo tempo.
A informação obtida fica guardada em toda a rede em vez de numa base de dados, todavia tem a capacidade de trabalhar com conhecimento incompleto e corrupção de células sem gerar resultados errados.

Referências

1. L. KOLODNER, Janet (1988). '**Proceedings of a Workshop on Case-Based Reasoning**'.
2. NOVAIS, Paulo, NEVES, José. '**Raciocínio baseado em casos**'.
3. W. AHA, David, KIBLE, Dennis, K. ALBERT, Marc (1991). '**Instance-Based Learning**'.
4. LAZA, R., CORCHADO, J. M. (2003). '**Constructing Deliberative Agents with Case-Based Reasoning Technology**'.
5. ELISABET, Damayanti, SENSUSE, Dana Indra, AL HAKIM, Shidiq (2019). '**Implementation of Case-Method Cycle for Case-Based Reasoning in Human Medical Health: A Systematic Review**', *3rd International Conference on Informatics and Computational Sciences (ICICoS)*.
6. SUTRISNO, Mirza, BUDIYANTO, Utomo (2019). '**Intelligent System for Recommending Study Level in English Language Course Using CBR Method**', *6th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI 2019)*.
7. **Top Organizations with Patents on Technologies**. Disponível em: `https://ieeexplore.ieee.org/document/8977047`; `https://ieeexplore.ieee.org/document/8982438`
8. OSUSZEK, Lukasz. '**A Case Based Reasoning as an Element of Case Processing in Adaptive Case Management Systems.**'
9. BREESE, J. S., HECKERMAN, D. (1996). '**Decision-Theoretic Case-Based Reasoning**'. *Society for Artificial Intelligence in Statistics* 26, 838-842
10. CHEETHAM B., CUDDIHY P., GOEBEL K. (2001). '**Applications of Soft CBR at General Electric**'.

11. MALLAWAARACHCHI, Vijini (2017). **Introduction to Genetic Algorithms**. Disponível em: `https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3`
12. BRUNTON, Steve (2018, jun 10). **Machine Learning Control: Genetic Algorithms**. Disponível em: `https://www.youtube.com/watch?reload=9&v=CZE86BPDqCI`
13. **Genetic Algorithms em Python**. Disponível em: `https://pypi.org/project/geneticalgorithm/`
14. **Genetic Algorithms em Matlab**. Disponível em: `https://www.mathworks.com/help/gads/genetic-algorithm.html?s_tid=CRUX_lftnav`
15. **Genetic Algorithms em Java**. Disponível em:`https://jenetics.io/`
16. **Genetic Algorithms em Excel**. Disponível em: `https://www.solvexl.com/`
17. WHITLEY, D., STARKWEATHER, T., BOGART, C. (1990). '**Genetic algorithms and neural networks: Optimizing connections and connectivity**'.
18. WALKER, M., MESSOM, C. (2002). '**Intelligent robotic sensor agents for environment monitoring**'.
19. **5 Top Genetic Algorithm Startups**. Disponível em: `https://www.startus-insights.com/innovators-guide/5-top-genetic-algorithm-startups/`

20. AGATONOVIC-KUSTRIN, S., BERESFORD, R. (2000). '**Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research**'. Disponível em: `https://www.sciencedirect.com/science/article/pii/S0731708599002721`
21. MIJWIL, Maad M. (2018). **Artificial Neural Networks Advantages and Disadvantages**. Disponível em: `https://www.linkedin.com/pulse/artificial-neural-networks-advantages-disadvantages-maad-m-mijwel`
22. **Artificial Neural Network Market by Component**. Disponível em: `https://www.marketsandmarkets.com/Market-Reports/artificial-neural-network-market-21937475.html`