Sistemas de aprendizagem: Reinforcement learning, Genetic Algorithms, Artificial Neural Networks

João Nunes (82300), Mateus Silva (81952)

Universidade do Minho

Resumo O presente artigo pretende apresentar e abordar alguns processos computacionais de otimização e de aprendizagem e que possuem uma capacidade inerente de decisão e previsão mediante um determinado contexto. Dentro dos vários processos aplicados para a aprendizagem será apresentado a metodologia de aprendizagem por reforço, os algoritmos genéticos e as redes neuronais artificiais. Para cada tema será apresentada uma descrição geral no que concerne ao seu funcionamento bem como algumas ferramentas que podem ser usadas para os construir. No final será apresentada algumas áreas que utilizam os tais métodos supracitados.

Keywords: Inteligência Artificial · Aprendizagem por Reforço · Algoritmos genéticos · Redes neuronais artificiais · Sistemas inteligentes · *Machine Learning*.

1 Introdução

Apesar de já ser discutida desde meados dos anos 50, no século XX, quando John McCarthy criou a expressão "Inteligência Artificial", que o próprio definiu como sendo a "ciência e engenharia de tornar máquinas inteligentes", esta continua a ser vista pela sociedade como ficção científica. Contudo, surge como um paradigma de programação do futuro, que está a ganhar um novo impulso e já faz parte do dia-a-dia humano e de inúmeros setores económicos da sociedade.

O interesse por sistemas de aprendizagem (*Machine Learning*) tem vindo a aumentar, e para isso contribuiu a crescente variedade e volume de dados, o processamento de dados mais poderoso e mais barato e o fácil armazenamento de dados.

Para se falar de Inteligência Artificial é obrigatório falar de processos de aprendizagem, pois sem estes não há forma de prover os sistemas de Inteligência Artificial com "inteligência" e capacidade de decisão. De forma simples, sistemas de aprendizagem consistem na aplicação de algoritmos no processamento de dados. Assim, em vez de se codificar à força e na totalidade todas as instruções necessárias para se atingir um determinado objetivo, é dado ao sistema a capacidade de aprendizagem para que este consiga, de forma autónoma, realizar uma determinada tarefa.

Existem três tipos de aprendizagem, a supervisionada, a não supervisionada e a aprendizagem por reforço. Dentro da área de *Machine Learning*, 70% da aprendizagem é supervisionada, de 10% a 20% é não supervisionada e o restante é aprendizagem por reforço. Na aprendizagem supervisionada existe a necessidade de haver um "professor" a fornecer casos de teste para o modelo se basear, isto é, é fornecida informação acerca das saídas desejadas para cada entrada, e após treino, estes algoritmos são capazes de executar tarefas tendo por base o que aprendeu. Quando se refere a aprendizagem não supervisionada, o modelo agrupa dados de acordo com as suas semelhanças sem necessidade e monitorização. Já a aprendizagem por reforço não necessita de um "professor", mas utiliza entradas do ambiente como controlo.

Este relatório aborda três algoritmos de aprendizagem. Em primeiro, o processo de aprendizagem por reforço no qual um agente aprende se as suas ações tiveram consequências positivas ou negativas para atingir um objetivo através de um critério de recompensas. Seguidamente, serão abordados os algoritmos genéticos, onde estes se baseiam na abordagem proposta por *Darwin* sobre a evolução dos

seres vivos. Por fim, irá se apresentar as redes neuronais que também se baseiam no funcionamento e na interligação dos neurónios cerebrais.

2 Reinforcement Learning

2.1 Descrição característica

A aprendizagem por reforço é uma área de *Machine Learning*, que pertence ao ramo de Inteligência Artificial, que é caracterizada pela forma como um agente deve interagir com o ambiente envolvente de modo a maximizar as recompensas recebidas, com o objetivo de melhorar a sua performance. Estes agentes são capazes de melhorar as suas ações através da sua experiência.

Os algoritmos de aprendizagem por reforço introduzem um critério de recompensa de ações que levam o agente a moldar a sua aprendizagem de forma a maximizar a recompensa recebida pelo desempenho das suas ações. Essa recompensa é conhecida como sinal de reforço e devolve ao agente um *feedback* que lhe permite determinar se a ação realizada foi apropriada para a resolução do problema. Este conceito baseia-se na simples ideia de que quanto melhores as consequências da ação tomada, no contexto de atuação do agente, maior será a recompensa recebida pela mesma. É através deste mecanismo de recompensas que o agente consegue determinar o comportamento ótimo capaz de maximizar o seu desempenho na realização de um determinado problema, quer seja obter o menor custo, menor caminho, maior velocidade, etc.

Posto isto, o grande desafio da aprendizagem por reforço recai sobre como o agente irá escolher as ações que irão maximizar as recompensas recebidas. Exemplificando com um exemplo real, quando tentamos ensinar um novo truque a um cão não conseguimos explicar-lhe o que ele deve fazer, no entanto podemos recompensá-lo quando ele realiza a ação desejada e "castigá-lo" quando não o faz. O cão tem de capacidade de entender quais o tipo de ações que o levaram a obter a recompensa desejada. Analogamente, o agente tem de ter perceção de quais são as ações que o levam a obter mais recompensas, evitando as ações prejudiciais. Para além disto, há também uma outra questão que este tipo de aprendizagem tem de ultrapassar, que é o *trade-off* entre *exploration* e *exploitation*. A *exploration* está ligada à aquisição de novos conceitos, enquanto que o *exploitation* consiste na otimização de conceitos já adquiridos. [1]

Neste método de aprendizagem, em vez de serem definidos um conjunto de métodos a serem processados, o agente tenta todas as opções disponíveis de maneira e encontrar a melhor forma de resolver o problema, deste modo, utilização de aprendizagem por reforço como técnica de aprendizagem dedica-se a situações onde o sistema evolui exclusivamente por um processo de tentativa-erro. Assim, não havendo necessidade de existir uma entidade externa que forneça exemplos de treino ou que guie o agente sobre qual o melhor caminho a seguir para alcançar a solução de um problema, este processo de aprendizagem não exige a presença de um "professor". [1]

Comparando com outros algoritmos de *Machine Learning*, este método pode demorar mais tempo até encontrar a melhor solução. Dependendo do problema a ser resolvido, o sistema poderá necessitar de testar todas as alternativas possíveis, e consequentemente leva mais tempo a aprender. Esta tarefa de aprendizagem complica bastante quando passamos de um sistema com apenas um agente para um sistema multiagente. Neste caso, o agente precisa de aprender a comportar-se na presença de outros agentes que também são capazes de aprender. Se temos múltiplos agentes a aprender e a interagir no mesmo ambiente a complexidade aumenta consideravelmente, pois nestas condições o comportamento de cada agente não só é influenciado pelo ambiente, mas também pelas ações dos outros agentes [3]. Para compensar o aumento da complexidade neste tipo de sistemas temos também benefícios associados, tais como o aumento da velocidade no processo de aprendizagem, a partilha de experiência e firmeza, já que em caso de falha de alguns dos agentes, outros podem assumir as suas tarefas. Importante também referir que, dependendo da modelação do sistema, é possível que o sistema comece a procurar soluções para um problema futuro e não apenas para o problema atual.

Finalizando, este método de aprendizagem é bastante importante em situações onde não temos dados disponíveis para analisar ou estamos perante um ambiente desconhecido onde o agente necessita de aprender através da sua própria experiência. [1]

2.2 Modo em que se exibe a capacidade de aprendizagem

Exploration versus Exploitation Um dos desafios que surge na aprendizagem por reforço é o *trade-off* entre *exploration* e *exploitation*. O agente deve tirar partido das ações que já conhece (*exploit*) de forma a obter maiores recompensas, mas também deve explorar (*explore*) de forma a conhecer novas ações e poder tomar melhores decisões no futuro.

A escolha de uma destas estratégias tem de ser feita considerando o problema de que uma ação pode ser mais benéfica agora, mas prejudicial a longo prazo (e vice-versa). Desta forma, a escolha tem de depender do ambiente, das ações já tomadas e das ações que ainda faltam tomar. Por exemplo, se estamos perante um problema que está próximo de ser resolvido faz sentido que seja usada *exploitation* para obter maior recompensa, em vez de *exploration* e arriscar uma recompensa baixa ou até mesmo negativa. Mas se estamos numa fase muito precoce da resolução de um problema, onde a maioria das ações ainda não foram testadas, faz sentido usar *exploration* de forma a procurar ações que sejam mais recompensadas.

Apesar de parecer bastante complicado tomar tal decisão, existem diversas formas de resolver este dilema, como por exemplo, usando o algoritmos como o ε *-greedy*, *R-Max*, *E3* e *Expanded Algorithm (REX)*[4].

Componentes Para além dos agentes e do ambiente, há ainda mais quatro sub-elementos nos modelos de Aprendizagem por Reforço[1]:

- Política
- Sinal/Função de recompensa
- Função de valor
- Modelo do ambiente/Estado (este sub-elemento é opcional)

A **política** define o comportamento dos agentes num determinado momento. Isto traduz-se num mapeamento de um estado para uma ação, ou seja, um mapeamento entre os distintos estímulos recebidos do ambiente onde os agentes estão inseridos e as ações a serem executadas como resposta a esses estímulos. Uma política pode ser definida através de uma função simples ou pode envolver uma computação extremamente completa e extensa. Qualquer que seja a política de um agente, este sub-elemento, por si só, é suficiente para caracterizar o comportamento dos agentes.

O **sinal de recompensa** é o valor da recompensa recebido pela realização de uma determinada ação num determinado estado do ambiente. A função que dá este valor de recompensa define o objetivo de um problema de aprendizagem por reforço e, basicamente, mapeia um par estado-ação num único número, a recompensa. O objetivo dos agentes é maximizar o valor recebido como recompensa ao longo do tempo, pelo que, o valor recebido como recompensa por um agente pode mudar a sua política de comportamento. De forma simples, define o que são boas e más ações[5].

A **função de valor** distingue-se da função de recompensa na medida que a função de valor tem em consideração a recompensa a longo prazo, ou seja, é uma previsão das recompensas futuras. Este valor pode ser visto como a soma total de todas as recompensas que um determinado agente espera atingir a partir de um determinado estado. Tem também um maior impacto que o valor do sinal de recompensa, pois é mais vantajoso uma ação ser mais proveitosa a longo prazo do que apenas num determinado instante. Porém, estes valores são mais difíceis de determinar, uma vez que são calculados com base em previsões.

Por último, o é a representação do ambiente por parte do agente e é aprendido com base na experiência. Este modelo pode ser visto como algo que imita o comportamento do ambiente. Este modelo é útil pois permite prever quais serão

os próximos estados após uma determinada ação e um determinado estado e é fundamentalmente usado para elaboração de um plano de ações a realizar no futuro.

São estes os elementos que caracterizam a capacidade de aprendizagem por reforço. O principal objetivo passa por analisar os resultados destes elementos e modificar o comportamento dos agentes de modo a obter uma maior recompensa final.

Aprendizagem A teoria de aprendizagem por reforço é teoricamente restrita a *processos de decisão Markovianos* [7].

Um ambiente satisfaz a *propriedade de Markov* se for possível prever qual será o próximo estado e a próxima recompensa esperada dado o estado e ação atuais [1]. Um *processo de Markov* é uma sequência de estados, com a propriedade de que qualquer previsão de valor de estado futuro dependerá apenas do estado e ação atuais, e não da sequência de estados passados [7]. De forma mais simples, podemos dizer que um sistema satisfaz a *propriedade de Markov* quando a probabilidade de transição de um estado s para um estado s' depende apenas do estado s e da ação adotada em s, isto significa que o estado atual fornece informação suficiente para o sistema de aprendizagem decidir que ação deve ser tomada [3].

Um processo de aprendizagem por reforço que satisfaz a *propriedade de Markov* é chamado de *processo de decisão Markoviano* (*MDP – Markov Decision Process*). Se o espaço de estados e ações for finito, então é chamado de *processo de decisão Markoviano finito*, base para a teoria de aprendizagem por reforço, que assume o ambiente como sendo deste tipo [7].

Formalmente, um *processo de decisão Markoviano* é definido por um conjunto $< S, A, P, R >$ [6], onde temos:

- Um conjunto de estados do sistema, S
- Um conjunto finito de ações, A
- Um modelo de transição, P, que mapeia os pares estado-ação numa distribuição de probabilidades sobre o conjunto de estados
- Uma função de recompensa, R, que especifica a recompensa que o agente recebe por escolher uma determinada ação a pert. A no estado s pert. S.

O estado s e ação a determinam o próximo estado s' de acordo com a probabilidade $P(s'|s, a)$ e a recompensa r(s,a) associada [7].

Num problema de aprendizagem por reforço, o domínio deve ser modelado como um MDP, e o agente e o ambiente interagem numa sequência discreta de passos no tempo, t=0, 1, 2, 3... O estado e a ação em dado instante ($s_t \in S$ e $a_t \in A$), determinam a distribuição de probabilidade para o estado s_{t+1} e a recompensa r_t. O objetivo do agente é escolher ações de modo a maximizar a soma descontada das recompensas seguintes

$$R_t = \sum_{k=0}^{t} \gamma^k r_{t+k}$$

onde a taxa de desconto $0 < \gamma \leq 1$ determina o peso temporal relativo das recompensas. Existem fórmulas alternativas para esta função de otimização.

As escolhas das ações dos agentes são feitas a partir da política, $\pi : S \rightarrow A$ [7].

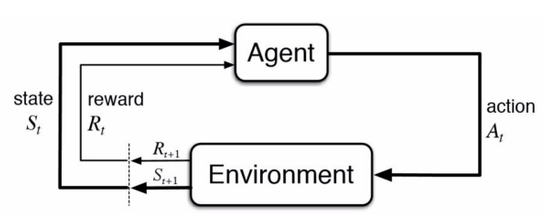

Figura 1. Processo de aprendizagem por reforço usando MDP para descrever a intereção de um agente com o ambiente [9]

Para solucionar o problema de aprendizagem por reforço existem três classes de métodos fundamentais [3]:

- Programação Dinâmica
- Monte Carlo
- Diferença Temporal

De forma bastante resumida e simples, os métodos de Programação Dinâmica são métodos matematicamente bem desenvolvidos, mas requerem um modelo do ambiente completo e preciso como um MDP. Os métodos de Monte Carlo não requerem modelagem do ambiente e são simples em termos conceptuais, baseando-se no cálculo da média de recompensas obtidas em sequências. No entanto, não são viáveis quando a solução do problema é possível apenas de forma incremental. Por fim, os métodos de Diferença Temporal não exigem um modelo exato do sistema e permitem ser incrementais, sendo uma combinação de características dos métodos de Monte Carlo com as ideias da Programação Dinâmica, daí serem mais difíceis de analisar. Q-learning e SARSA são exemplos deste tipo de métodos. Uma explicação mais detalhada destas três classes de métodos e o seu desenvolvimento matemático podem ser encontrados em [3].

O método de aprendizagem por reforço mais popular é o Q-Learning seguido do algoritmo SARSA, ambos baseados em Diferenças Temporais. Mais detalhes sobre algoritmos, vantagens, desvantagens e fórmulas matemáticas podem ser encontradas em [3] e [8].

2.3 Ferramentas de desenvolvimento de Aprendizagem por Reforço

Em virtude de ser um dos métodos mais conhecidos e mais importantes dentro do *Machine Learning*, é fácil encontrar bibliotecas que facilitam a utilização deste método em diversas linguagens de programação.

Em *Python* podemos destacar a biblioteca *PyBrain*[17], onde são apresentados métodos de aprendizagem por reforço, com os quais é fácil usar, de forma simples, os algoritmos e ainda inclui um conjunto de ambientes pré-definidos que possibilitam o teste e comparação destes algoritmos, como por exemplo os famosos algoritmos *Q-Learning* e *SARSA*. Para além desta, existe também a biblioteca *Open Source* chamada *TensorFlow*[18], que foi, originalmente, desenvolvida por investigadores e engenheiros do projeto *Google Brain Team*. Esta biblioteca permite a implementação de diversos métodos de *Machine Learning* e respetiva adaptação para o contexto de aprendizagem por reforço. Temos ainda o *RLLab*[19] e *Keras-RL*[20]. A ferramenta *OpenAI Gym*[21], desenvolvida pela empresa *OpenAI*, que é a mais famosa neste ramo e permite comparar os algoritmos de aprendizagem por reforço e é compatível com o *TensorFlow*.

Em Java temos, por exemplo, a biblioteca *BURLAP* que foi inicialmente desenvolvida e mantida por James MacGlasha (*Brown University*) e que implementa os processos de decisão de *Markov*, bem como os diversos algoritmos de aprendizagem[22]. Podemos também referenciar a biblioteca *RL4J*[23], que é *Open Source*, e que oferece algoritmos de *Deep Reinforcement Learning*.

Em C++ podemos destacar a *framework RL Toolbox*[24], que é *Open Source* e onde é possível desenvolver aprendizagem por reforço usando todos os algoritmos. Por fim, existe também a plataforma *Project Malmo*[25], desenvolvida pela *Microsoft* com a finalidade e investigar e experimentar algoritmos de Inteligência Artificial (incluindo a aprendizagem por reforço) e foi desenvolvida tendo por base o famoso jogo *Minecraft*. A *DeepMind Lab*[26] é também uma plataforma que consiste num jogo 3D em primeira pessoa também desenvolvido com a finalidade de explorar e testar algoritmos de Inteligência Artificial.

2.4 Soluções existentes no mercado

Sendo este um algoritmo que não necessita de grandes modificações e adaptações para serem usados em diferentes áreas, estes podem ser aplicados em qualquer contexto ou problema que se baseie em tomadas de decisões. Para a sua aplicação

basta saber avaliar o desempenho das ações no ambiente, o que é uma enorme vantagem e permite que este algoritmo seja usado em diversos problemas reais[27].

Há inúmeras empresas a adotar algoritmos de *Machine Learning* para melhorar a sua produção, e consequentemente, o seu lucro. Nesta área os *robots* e a aprendizagem por reforço são usados essencialmente para mover objetos de um lugar para outro. A empresa *Fanuc*, é uma das empresas líderes no que toca a automação industrial e abrange uma vasta gama de setores da indústria, como a medicina, agricultura, engenharia aeroespacial, etc. Grande parte das máquinas desta empresa possuem a capacidade de aprender com a experiência através da atribuição de recompensas pelas suas ações. Após algum tempo as máquinas são capazes de executar as suas tarefas de forma precisa. Na imagem seguinte podemos ver o exemplo de um diagrama de fluxo que representa o processo de aprendizagem por reforço de uma ferramenta. Esta imagem foi encontrada numa patente da empresa *Fanuc*, presente no *Google Patents*, onde tem inúmeros exemplos desta mesma empresa [10].

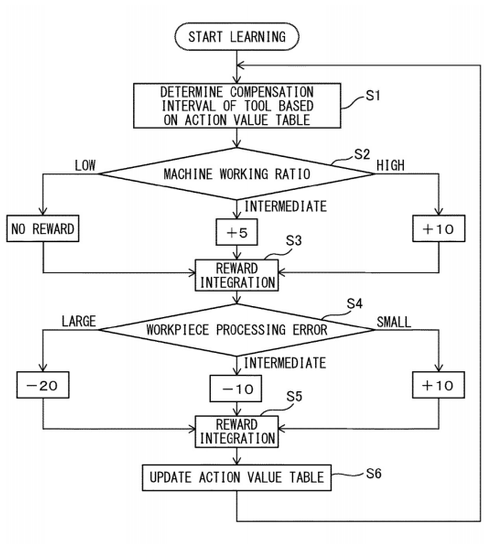

Figura 2. Diagrama exemplificativo do processo de aprendizagem por reforço de uma ferramenta da empresa *Fanuc*.

No setor financeiro, a aprendizagem por reforço apresenta excelentes soluções para encontrar estratégias de negociação comercial e prever as melhores decisões a longo prazo. A *Pit.ai*[28] é uma ferramenta que serve para avaliar estratégias de negociação e baseia-se em algoritmos de *Machine Learning*.

No que diz respeito à gestão de inventários, este algoritmo é essencialmente usado para reduzir o tempo de armazenamento e localização de produtos.

Este algoritmo é também usado na gestão de encomendas, sendo útil em casos em que apenas um veículo tem de fazer inúmeras entregas da forma mais vantajosa (caminho mais curto).

Também no mundo dos jogos este algoritmo tem bastante importância, especialmente devido ao trabalho desenvolvido pela Google, o *DeepMind*. Os algoritmos de aprendizagem por reforço foram implementados com sucesso em quase metade dos 50 jogos *Atari* onde foi testado, sendo que nesses obteve melhores resultados que os métodos usados anteriormente. Há também o famoso caso do *Alpha GO* que é um sistema criado pela Google para jogar GO[29]. Este sistema foi carregado com cerca de 30 milhoes de jogadas de jogadores profissionais. O mais interessante neste sistema e a capacidade de criar as suas proprias estrategias recorrendo para isso a um processo de tentativa-erro, ou seja, aprendizagem por reforço. Este sistema foi o primeiro a conseguir bater o humano. [2] A Microsoft seguiu o mesmo caminho e usou jogos para testar os seus sistemas de Inteligência Artificial, mas

usando o jogo *PacMan* acabando por bater o recorde mundial deste famoso jogo dos anos 80[30].

Na publicidade e marketing podem ser aplicados métodos de aprendizagem por reforço para personalizar e melhorar os sistemas de recomendação presentes nos *websites* tanto na forma de produtos sugeridos como de anúncios, mostrando somente a publicidade que possa interessar ao utilizador. Uma outra solução baseada neste algoritmo está relacionada com aplicações dedicadas a fazer resumos de textos. Para além disso, é também usado em aplicações de diálogo que aprendem com os utilizadores. medicina, há também aplicações que usam aprendizagem por reforço para encontrarem políticas de tratamento ótimas para os pacientes. Nesta área é possível também controlar doses de medicamentos e uso de material médico.

Por último, uma das aplicações muito faladas nos dias de hoje, a condução automática. Empresas como a *Tesla*, *Uber* e *Google* têm investido nos veículos com condução automática, usando aprendizagem por reforço, assim como outros algoritmos.

3 Genetic Algorithm

3.1 Descrição caraterística

O algoritmo genético carateriza-se como sendo um algoritmo de otimização baseado nas mecânicas inerentes à seleção natural, possuindo características *Darwinianas*[13].

Este algoritmo, não pode ser considerado um algoritmo matematicamente guiado uma vez que este baseia-se na aleatoriedade o que lhe proporciona uma melhor performance comparativamente a outros tipos de algoritmos de pesquisa[13].

Como tal, o principal foco de criação deste algoritmo foi balancear a robustez com a eficiência e a eficácia, tendo portanto uma plenitude de vantagens uma vez que computacionalmente são algoritmos simples, e que tal como foi referido não possuem as restrições matemáticas que muitos outros algoritmos de procura possuem, como por exemplo, no que concerne às restrições e premissas sobre, por exemplo, a existência de derivadas.

Portanto, os GA definem-se como:

1. Os GAs trabalham como uma representação do conjunto de parâmetros, e não com os parâmetros em si.
2. Procuram dentro de uma população de pontos e não apenas com um só ponto.
3. Usam informações de recompensa (função objetivo), não utilizando derivadas ou outro tipo de conhecimento auxiliar.
4. Usam também regras de transição probabilísticas ao invés de usar regras determinísticas.

Portanto, verifica-se que ao contrário dos outros modelos de otimização que utilizam alguma regra de transição de modo a transitar de ponto para ponto, os GA trabalham com uma plenitude de pontos ao mesmo tempo num processo de procura paralelo e estruturado, contudo aleatório.

3.2 Modo em que se exibe a capacidade de aprendizagem

Tal como foi dito anteriormente, o algoritmo GA foi inspirado pelas leis naturais de seleção e de genética, onde no processo biológico indivíduos mais *fortes* irão ser com mais probabilidade os vencedores num ambiente competitivo[12]. O *GA* aproveita esta caraterística e tenta estabelecer a ponte entre a natureza e o ramo da otimização.

Como tal, computacionalmente o indivíduo poderá ser representado por um conjunto de parâmetros, este conjunto de parâmetros poderá ser representado por uma string de valores, tipicamente representado por bits, denominando-se como um cromossoma[12].

Sobre estes cromossomas, são efetuadas um conjunto de operações básicas, que também são encontradas na natureza, de **reprodução**, ***crossover*** e por fim de **mutação**. Um cromossoma mais ajustado que os outros, sendo este grau de ajuste definido por uma função objetivo, terá uma maior probabilidade de gerar descendentes com ainda maior qualidade que os seus predecessores, até ser obtida depois de uma extensa procura a solução ideal para o problema de otimização em mão.

De modo a iniciar o processo de otimização, é necessário gerar aleatoriamente um conjunto de cromossomas iniciais que servem como base para a procura da solução ideal, subsequentemente serão gerados novos cromossomas com base na geração anterior, misturando os genes (bits individuais do cromossoma) e recombinando esses mesmos genes de modo a gerar a nova geração de cromossomas.

Os cromossomas são avaliados, tal como foi mencionado, segundo uma função objetivo que indica o quão ajustados estão os cromossomas da geração atual para o problema de otimização apresentado, esta função objetivo poderá ser definida como uma função linear, exponencial, entre outras [12].

De seguida existem vários esquemas que podem ser aplicados de modo a selecionar os cromossomas que serão a base para a geração seguinte, como por exemplo um esquema baseado numa roleta, onde os cromossomas com mais probabilidade de serem escolhidos para o processo de reprodução serão os cromossomas com maior grau de ajuste.

Após terem sido escolhidos os cromossomas, os cromossomas escolhidos sofrem um processo *genético* de *crossover*, que se trata de uma recombinação entre subpartes dos cromossomas, produzindo como resultado cromossomas que contêm informação genética dos dois predecessores. O modelo mais simples de crossover trata-se do *single point crossover* onde é escolhido aleatoriamente um ponto de *crossover* entre os dois cromossomas, sendo depois os genes recombinados formando dois novos cromossomas.

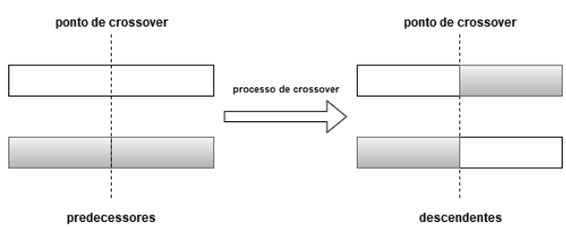

Figura 3. Diagrama exemplificativo do processo de crossover.

Num contexto real, caso os genes sejam compostos bits, no processo de *crossover* irão ser recombinados os zeros em uns e os uns em zeros.

Tal como foi mencionado anteriormente, a mutação tem um papel secundário mas não menos importante que as duas operações básicas que acontecem anteriormente. Na mutação são introduzidas variações aleatórias no cromossoma [11], de modo a manter a variedade das soluções uma vez que não se pretende que as gerações seguintes de cromossomas sejam homogéneas. A operação de mutação ocorre baseada numa probabilidade, tipicamente pequena, onde é alterado um

gene numa posição aleatória do cromossoma, passando por exemplo o bit nessa posição de um para zero.

Após todo este processo ter sido concluído, a nova geração de cromossomas está pronta para substituir a geração antiga de cromossomas, existindo para tal duas estratégias diferentes de substituição [11]:

- **Generational-Replacement** - onde cada população de tamanho n de cromossomas gera n cromossomas novos para substituir a geração antiga.
- **Steady-State Reproduction** - apenas alguns cromossomas são substituídos pelos novos descendentes, tipicamente são escolhidos os cromossomas com pior grau de ajuste para se substituir.

Todo este processo se repete, até ser atingido o critério de paragem, que poderá ser, por exemplo, atingir um *threshold* de ajuste, ou apenas parar após a criação de um certo número de gerações cromossomas.

De uma maneira simplista e abreviada, os GA exibem o seguinte comportamento:

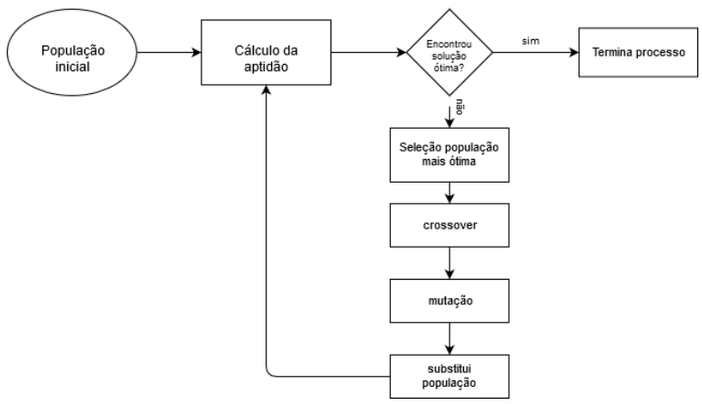

Figura 4. Diagrama exemplificativo do funcionamento geral do GA.

De modo a melhor o desempenho que é um dos principais *stumbling blocks* deste algoritmo, e uma vez que intrinsecamente o GA possui caraterísticas de paralelismo podem ser construídas arquiteturas que aproveitam este facto tais como esquemas *Master/Slave* ou até mesmo esquemas de migração onde se subdivide os genes em diferentes sub-populações sendo cada uma independente no decorrer do algoritmo. No caso do esquema de migração, ocorrem migrações de cromossomas adequados entre as várias sub populações. Um exemplo de uma topologia com este comportamento é o *Ring Migration Topology*.

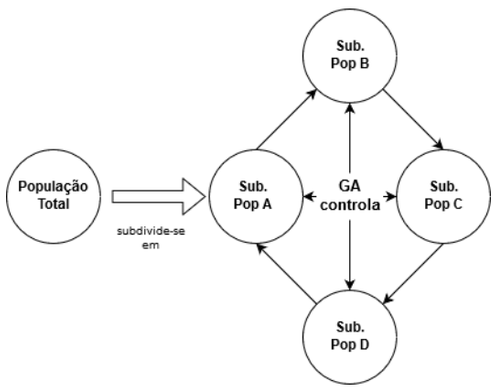

Figura 5. Diagrama exemplificativo da topologia do Ring Migration.

Para além de melhorar o desempenho, esta topologia devido às migrações aleatórias de cromossomas entre os vários sub-conjuntos de cromossomas, permite também manter a diversidade [12] o que irá garantir robustez no que toca ao encontrar a solução ideal para o problema da otimização.

No caso anterior, a migração é direcional, contudo existem também esquemas onde a migração pode ser bidireccional e ao contrário destas duas topologias anteriores, sem restrições no que toca à vizinhança entre as sub populações.

3.3 Ferramentas de desenvolvimento GA

No que toca à prática, o algoritmo pode ser implementado via um plenitude de maneiras. No R, pode ser usado o *package genalg*[31] de modo a montar o modelo, o *matlab* também pode ser utilizado sendo para tal necessário utilizar uma *toolbox* externa tal como o *GEATbx*[32] ou o *Global Optimization Toolbox*[33]. No *excel* também é possível adicionar um *addon* externo tal como o *GeneHunter*[34] de modo a resolver problemas de otimização segundo a metodologia do algoritmo genético no *excel*. Existe também uma libraria externa para o java denominada *Jenetics*[35], para o python poderá ser utilizada a *framework DEAP*[36].

3.4 Soluções existentes no mercado

Tendo discutido o funcionamento geral do algoritmo genético, falta abordar as áreas em que o mesmo foi aplicado e com sucesso, áreas essas que são diversas na sua natureza.

Na área do controlo de sistemas, o algoritmo genético tem sido utilizado devido às suas caraterísticas de otimização uma vez que é crítico que nestas áreas os parâmetros sejam otimizados de modo a aperfeiçoar a performance de todo o sistema. No que toca à área da robótica, o algoritmo genérico também é utilizado no que concerne à navegação do robô, uma vez que a trajetória do robô final pode ser generalizada numa série de movimentos dos seus membros, esta série poderá ser codificada numa string (cromossoma), sendo então possível aplicar o algoritmo GA. Na área de reconhecimento de padrões, por exemplo no que toca ao reconhecimento facial [11], pode ser aplicado um cromossoma binário para codificar as características faciais, sendo então gerado inicialmente um conjunto de faces com base nos cromossomas aleatórios gerados inicialmente. Este conjunto de faces é então classificado por um supervisor do algoritmo, sendo que o GA continua a partir do cromossoma melhor classificado.

4 Artificial Neural Networks

4.1 Descrição caraterística

Uma rede neuronal artificial é um sistema de processamento de dados consistido por um elevado número de unidades computacionais interligadas entre si que atuam sobre os dados instantaneamente e de maneira paralela [15], inspirando-se na estrutura do córtex cerebral humano. Nos dias de hoje, os computadores modernos superam a capacidade intelectual dos humano em relação a operações numéricas, por exemplo, contudo os humanos conseguem resolver problemas complexos apenas com base à percepção, algo que os computadores têm bastante problemas em resolver. Como tal advém-se a criação das redes neuronais de modo a aproximar esta capacidade humana com a capacidade computacional de um computador.

Como tal, as redes neuronais pretendem possuir as seguintes caraterísticas [14] que são visíveis nos seres humanos:

1. Capacidade de aprendizagem;
2. Capacidade de generalização;
3. Elevado nível de paralelismo;
4. Adaptação a novas situações;
5. Tolerância a falhas;

4.2 Modo em que se exibe a capacidade de aprendizagem

Portanto, uma RNA é formada, e à semelhança do cérebro humano, por um conjunto de neurónios que são a unidade computacional da RNA com a capacidade de aprendizagem. Os neurónios estão conectados entre si através dos axónios, sendo que a informação transmitida circula em apenas um sentido. Os axónios ligam-se às sinapses, sendo que esta ligação possui um peso (importância) do sinal a entrar no neurónio (input). A soma dos valores do input é por norma feita na função de transferência, sendo apenas gerado um output para os valores de input no neurónio se a soma dos valores excede um certo *threshold* (patamar).

No processo de construção de uma RNA é primeiro necessário identificar o número de neurónios de entrada e saída da rede, e de seguida escolher a topologia mais indicada para a rede, topologia essas que serão abordadas mais à frente. É também necessário possuir dados de treino, de modo a providenciar à rede dados sobre os quais ela irá usar para se treinar e para iterar sobre. É também elaborada uma distribuição aleatória de pesos pelas sinapses na rede. A partir daqui, são calculados os valores de ativação em cada um dos neurónios consoante o seu input, através do processo descrito anteriormente, sendo originado valores de output que são transferidos para à próxima camada de neurónios através axónios. Na camada de saída, e consoante o valor obtido, os pesos das sinapses são novamente ajustados segundo alguma regra de aprendizagem, que também serão apresentada posteriormente, todo este processo é iterado até se obter uma solução ideal para o problema apresentado.

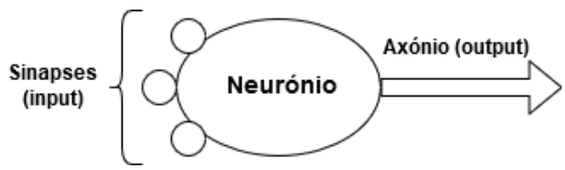

Figura 6. Componentes base de um neurónio.

As RNA podem ser vistos como um grafo pesado, onde os neurónios são os nodos e as arestas com peso podem ser visto como os axónios [14]. Como tal, este padrão de conexão pode ser agrupado em duas categorias distintas:

1. *feed-foward*: não existem ciclos no grafo, e poderá ter uma ou mais camadas de neurónios.
2. recorrente: quando existem ciclos no grafo.

Como tal, uma arquitetura *feed-foward* irá possuir o seguinte aspeto:

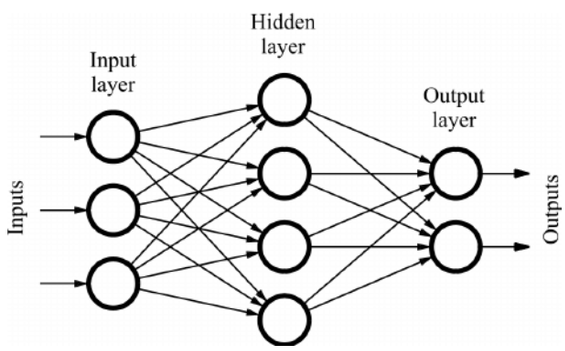

Figura 7. Arquitetura *feed-foward*.

Uma arquitetura recorrente, possuirá o seguinte aspeto:

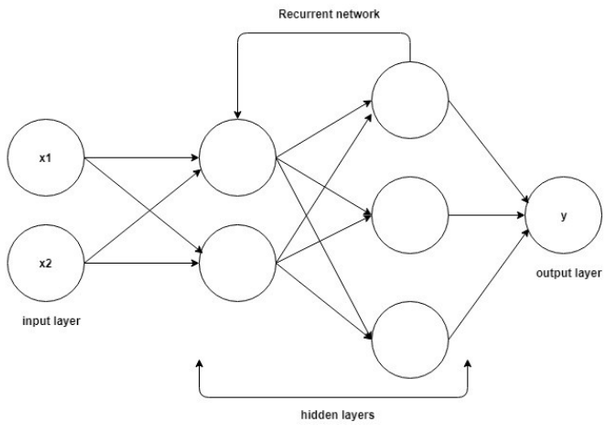

Figura 8. Arquitetura recorrente.

Geralmente, a estrutura recorrente é mais dinâmica no seu funcionamento que a estrutura *feed-foward* sendo esta mais estática [14], os algoritmos de aprendizagem variam agora consoante a arquitetura escolhida.

Um algoritmo de aprendizagem generaliza-se num procedimento onde são aplicadas regras de aprendizagem de modo a variar os pesos das sinapses, que é o que dita a aprendizagem da RNA. Existem portanto, três paradigmas principais de aprendizagem, a aprendizagem supervisionada, sem supervisão e a aprendizagem híbrida.

Na aprendizagem supervisionada é apresentada à rede a solução para o problema ou seja o output para cada input dado à rede, de seguida os pesos das sinapses são ajustados de forma a melhor acomodar esta solução apresentada o que permite obter cada vez respostas mais fiéis ao problema em mão. Na aprendizagem sem supervisão não é necessário para cada padrão de input da o output correto ao invés, neste tipo de aprendizagem é explorada a estrutura por detrás dos dados tentando encontrar correlação e padrões entre os dados apresentados, variando depois o peso das sinapses consoante essas descobertas. Na aprendizagem híbrida é misturado a aprendizagem supervisionada com a aprendizagem sem supervisão, onde parte dos pesos são determinados através da aprendizagem supervisionada e outra parte utiliza o algoritmo da aprendizagem híbrida.

Existem também regras de aprendizagem [14] das quais se podem destacar:

1. *Hebbian*: o valor do peso das sinapses é atualizado se tanto o valor de input como o valor desejado de output é alto, matematicamente a fórmula de aprendizagem pode ser descrita em baixo, onde x_i e y_j são os valores de *ouput* dos neurónios i e j que estão conectados pela sinapse w_{ij} e η é a taxa de aprendizagem;

$$w_{ij}(t+1) = w_{ij}(t) + \eta * y_j(t) * x_i(t)$$

2. Gradiente decrescente: a diferença entre o output real e o output desejado é minimizado através da aplicação do método dos mínimos quadrados;
3. Competitiva: neurónios competem entre si e o que gerar o melhor output para um dado input atualiza o peso das suas sinapses, na fórmula seguinte o i^* representa o neurónio com o maior valor de output (vencedor), sendo que nesse caso o peso da sinápse é atualizado segundo uma taxa de aprendizagem η.

$$w_{ij} = \begin{cases} \eta * (x_j - w_{ij}), & i = i^* \\ 0, & i \neq i* \end{cases}$$

4.3 Ferramentas de desenvolvimento das RNA

A *Theano*[37] é das principais librarias utilizadas no desenvolvimento das RNA na linguagem python, bem como o *Tenserflow*[18] que foi criado pela Google. No caso do R, o *package neuralnet*[38] é utilizado para o desenvolvimento das redes neuronais artificiais. No que toca ao *matlab* existe a *Deep Learning Toolbox*[39] de modo a criar, analisar e treinar as RNA. Para o java existe a *framework* externa *Neuroph*[40] que para além de simplificar o processo de criação de uma rede neural artificial em java devido à sua extensa biblioteca, oferece também uma interface gráfica onde é possível modificar graficamente a rede criada. Para o C existe uma biblioteca minimalista a *Genann*[41].

4.4 Soluções existentes no mercado

Uma vez que os algoritmos RNA consistem em unidades interconectadas com poder de processamento, o que torna este algoritmo bastante adequado para várias áreas.

Na área da conversão [15] *text to speech* a RNA com o input de símbolos ou letras é capaz de converter e as pronunciar, treinando a RNA da mesma maneira que se treina uma pessoa a ler.

A RNA pode também ser usada na área de previsão e na identificação de padrões, por exemplo, no que toca ao êxito académico [16] e para além desta área, pode também ser utilizada para tomar grandes decisões financeiras, por exemplo no que toca a prever o preço dos stocks, devido à sua flexibilidade e capacidade de extracção de conhecimento sobre os dados.

5 Conclusão

Os assuntos abordados ao longo deste documento permitem-nos entender a importância dos sistemas de aprendizagem na elaboração de um sistema inteligente. Apresentados os três sistemas de aprendizagem, é agora possível concluir sobre as suas diferenças. Se a finalidade é comum aos três, isto é, tornar uma máquina inteligente, a forma como levam a máquina a aprender é distinta. Todos possuem vantagens e desvantagens e devem ser usados consoante o objetivo final a que são propostos, pois há situações mais ou menos apropriadas para um tipo de sistema, podendo uns apresentar melhor desempenho que outros.

A aprendizagem por reforço é uma técnica de aprendizagem que testa todas as alternativas possíveis para atingir um objetivo. Em troca disto o agente recebe recompensas conforme a sua performance para a resolução do problema. Se atinge um bom resultado, recebe uma boa recompensa, se não, recebe uma má recompensa. Quando mais o agente for recompensado, mais aprende e, consequentemente melhor será a sua performance.

As principais vantagens associadas ao algoritmo de aprendizagem por reforço passam pelo facto de ser um processo, computacionalmente, pouco complexo, nem ser necessário um grande conhecimento sobre o contexto e o ambiente onde o agente se insere. Para a obtenção de bons resultados basta apenas saber avaliar o desempenho de uma ação.

A nível de desvantagens, podemos referir o facto de possuir um processo de exploração de ações baseado num processo de tentativa-erro para aprender, ou seja, necessidade de um custoso e iterativo processo de treino até que se adquira uma solução dita ótima.

De seguida, aborda-se e explica-se os conceitos dos algoritmos genéticos, que tal como foi referido, incorporam bases *Darwinistas* usando o conceito de que os elementos mais aptos sobrevivem de modo a gerar soluções cada vez melhores para um dado problema. Soluções essas que são obtidas através de um conjunto de operações biológicas como seleção, reprodução e mutação, procurando simular o procedimento evolutivo verificado na natureza.

Tal como foi referido, juntamente com as bases do algoritmo genético é possível aplicar um esquema de paralelismo o que sem dúvida aumenta a eficiência e a rapidez de computação deste algoritmo, que era uma das principais desvantagens apontadas aquando da sua concepção. Para além disto, a forma de iteração do algoritmo é geral, pelo que não necessita de ter conhecimento adicional e específico em relação ao problema em questão. Inicialmente, é apenas necessário definir a função objetivo que calcula o grau de ajuste de um cromossoma, o critério de paragem e o tamanho da população inicial, de seguida o algoritmo continua com o seu processo, iterando quantas vezes são necessárias.

Contudo o algoritmo possui uma desvantagem importante, no que toca à sua natureza, uma vez que o processo de obtenção das soluções no GA é estocástico e não linear não é possível obter tempos de resposta garantidos uma vez que a base do projeto é a aleatoriedade, o que dificulta prever a performance deste algoritmo para um dado problema, algo que num contexto real é importantíssimo.

Na próxima secção foram abordadas as redes neuronais artificiais, que tal como o algoritmo genético, também se baseiam num processo biológico que ocorre no córtex cerebral no que toca à interação e comunicação dos neurónios uns com os outros através de estímulos. A aprendizagem na RNA é exibida através de uma aplicação de regras de aprendizagem variando o peso das sinapses.

Como tal, e como foi possível observar, as RNA possuem um esquema que só por si é intrinsecamente paralelo, o que tal faz com que demonstre uma maior performance quando comparado com outros algoritmos de aprendizagem. Para além disto, este algoritmo possui a capacidade de reconhecer padrões nos dados, o que significa que possui uma elevada capacidade de adaptação mesmo quando apresentado com informação incompleta.

Porém, também existem um certos inconvenientes associados a este tipo de algoritmo, por exemplo, no que toca à sensibilidade na definição dos parâmetros, por exemplo, no que concerne à definição do número de camadas intermédias que compõem a arquitetura do sistema, quantos neurónios por camada e também

relativamente à quantidade e tamanho dos dados de teste que são dados como input a este algoritmo. Se não existir uma certa conjugação entre estes parâmetros todos com o problema a resolver, é muito possível que este algoritmo não apresente a performance nem solução desejada.

Referências

1. Richard S. Sutton and Andrew G. Barto, *"Reinforcement Learning: An Introduction"*. The MIT Press, 2nd edition, 2012.
2. Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn TensorFlow*, março 2017, pp. 437-460.
3. Maurício Rangel Guimarães Serra, *Aplicações de Aprendizagem por Reforço em Controle de Tráfego Veicular Urbano*, 2004.
4. Michael Piñol, *Exploration versus Exploitation Dilemma in Reinforcement Learning*, California State University Stanislaus.
5. Fernando Domínguez-Estévez, Antonio A. Sánchez-Ruiz, and Pedro Pablo Gómez-Martín, *Training Pac-Man bots using Reinforcement Learning and Case-based Reasoning*, Dep. Ingeniería del Software e Inteligencia Artificial Universidad Complutense de Madrid (Spain).
6. R. E. Bellman, *Dynamic Programming*, Princeton Univ. Press, Princeton, New Jersey, 1957.
7. T. Monteiro, Sidolmar, H. C. Ribeiro, *Desempenho de Algoritmos de Aprendizagem por Reforço sob condições de ambiguidade sensorial em robótica móvel*, Divisão de Ciência da Computação, Instituto Tecnológico de Aeronáutica, São Paulo
8. Carlos Henrique Costa Ribeiro, *A Tutorial on Reinforcement Learning Techniques*, Divisão de Ciência da Computação, Instituto Tecnológico de Aeronáutica, São Paulo
9. Hochländer aus Wiesbaden, Aaron, *Deep Learning for Reinforcement Learning in Pacman*, Technische Universität Darmstadt, Julho 2014
10. Fumito Yukawa, *Machine Learning Device and Method for Optimizing Frequency of Tool Compensation of Machine Tool, and Machine Tool have the Machine Learning Device*, FANUC CORPORATION, Patent No.:US10,121,107 B2, United States Patent, Date of Patent: Nov.6, 2018
11. K. S. Tang, K. F. Man, S. Kwong and Q. He, "Genetic algorithms and their applications,"in IEEE Signal Processing Magazine, vol. 13, no. 6, pp. 22-37, Nov. 1996.
12. K. F. Man, K. S. Tang and S. Kwong, "Genetic algorithms: concepts and applications [in engineering design],"in IEEE Transactions on Industrial Electronics, vol. 43, no. 5, pp. 519-534, Oct. 1996.
13. GOLDBERG, David, Genetic Algorithms in Search, Optimization, and Machine Learning", Addison Wesley, 1989.
14. A. K. Jain, Jianchang Mao and K. M. Mohiuddin, "Artificial neural networks: a tutorial,"in Computer, vol. 29, no. 3, pp. 31-44, March 1996.
15. R. E. Uhrig, "Introduction to artificial neural networks,"Proceedings of IECON '95 - 21st Annual Conference on IEEE Industrial Electronics, Orlando, FL, USA, 1995, pp. 33-37 vol.1.
16. P. Dharmasaroja and N. Kingkaew, "Application of artificial neural networks for prediction of learning performances,"2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), Changsha, 2016, pp. 745-751.
17. PyBrain, `http://pybrain.org/`
18. TensorFlow, `https://www.tensorflow.org/`
19. RLLab, `https://github.com/rll/rllab`
20. Keras-RL, `https://github.com/keras-rl/keras-rl`
21. OpenAI Gym, `https://gym.openai.com/`
22. BURLAP, `http://burlap.cs.brown.edu/information.html`
23. RL4J, `https://github.com/deeplearning4j/rl4j`
24. RL Toolbox, `https://s-rl-toolbox.readthedocs.io/en/latest/`
25. Project Malmo, `https://www.microsoft.com/en-us/research/project/project-malmo/`
26. DeepMind Lab, `https://deepmind.com/research/publications/deepmind-lab`
27. Oreilly, Practical applications of reinforcement learning in industry, `https://www.oreilly.com/radar/practical-applications-of-reinforcement-learning-in-industry/`
28. Pit.ai `https://pit.ai`
29. Diário de Notícias, Computador derrota campeão de jogo de tabuleiro go,`https://www.dn.pt/sociedade/interior/computador-derrota-campeao-chines-de-jogo-de-tabuleiro-go-8499621.html`
30. The Verge, Sorry humans, Microsoft's AI is the first to reach a perfect Ms. Pac-Man score, `https://www.theverge.com/2017/6/14/15801700/microsoft-ai-perfect-ms-pac-man-score`
31. genalg, https://cran.r-project.org/web/packages/genalg/
32. GEATbx, http://www.geatbx.com/
33. Global Otimization Toolbox, https://www.mathworks.com/products/global-optimization.html
34. GeneHunter, http://www.wardsystems.com/genehunter.asp
35. Jenetics, http://jenetics.io/
36. DEAP, https://deap.readthedocs.io/en/master/

37. THEANO, http://deeplearning.net/software/theano/
38. neuralnet, https://github.com/bips-hb/neuralnet
39. Deep Learning Toolbox, https://www.mathworks.com/products/deep-learning.html
40. Neuroph, http://neuroph.sourceforge.net/
41. genann, https://codeplea.com/genann