# TP21- Heart Diseases

**André Magalhães**[1], **Miguel Fernandes**[2], **Raul Barbosa**[3]
Departamento Engenharia Informática, Universidade de Coimbra
[1]andrefilipe@student.dei.uc.pt [2]mfernandes@student.dei.uc.pt [3]barbosa@student.dei.uc.pt

## Abstract

No âmbito da cadeira de Computação Evolucionária, uma unidade curricular do Mestrado de Engenharia Informática, foi realizado um trabalho que consistia em encontrar um classificador binário utilizando algoritmos evolucionários. Este trabalho é aplicado a um *dataset* que contém informações sobre problemas cardíacos.

## 1 Introdução

*Neural Networks* e *Genetic Algorithms* são duas técnicas usadas essencialmente para a resolução e/ou optimização de problemas complexos. Ambas têm as suas vantagens e desvantagens. Os dois foram evoluindo e usados ao longo do tempo separadamente, no entanto, neste documento pretendemos demonstrar a resolução de um problema combinando ambas as técnicas. No contexto do problema proposto no âmbito da unidade curricular de Computação Evolucionária - desenvolver um classificador capaz de prever problemas cardíacos - este documento visa descrever teoricamente a implementação realizada para o problema já referido. Para a resolução do problema iremos usar redes neuronais que por sua vez irão usar um algoritmo genético para a atualização dos seus pesos e bias, com a finalidade de melhorar a sua previsão. Apesar do objetivo principal focar-se na inserção de algoritmos genéticos no treino de redes neuronais, também é pretendido realizar uma comparação de ambos os métodos, ou seja, comparar os resultados da rede neuronal com e sem algoritmo genético. O presente documento iniciar-se-à com a descrição do *dataset* fornecido e da sua preparação seguindo de uma breve introdução às redes neuronais. No capítulo 4 será realizada uma descrição sobre algoritmos genéticos e do algoritmo usado para a atualização dos pesos e bias da rede neuronal. Por fim, serão demonstrados os testes realizados e, como consequência, uma conclusão dos mesmos.

## 2 Descrição do Dataset

O *dataset* [1] usado foi *Processed.Cleveland.data*, que contém informação sobre problemas cardíacos de 303 pacientes. Os indicadores utilizados para prever este problema são:

- **Age**:Idade do individuo;
- **Sex**:Género do individuo;
- **Cp**:Tipo de dor no peito;
- **Trestbps**:Pressão arterial em repouso;
- **Chol**:Colesterol em mg /dl;
- **Fbs**:Açúcar no sangue em jejum (maior que 120 mg) (1=verdadeiro;0=Falso);
- **Restecg**:Resultados eletrocardiográfico em repouso;
- **Thalach**:Frequência cardíaca máxima alcançada;
- **Exang**:Angina induzida por exercício;
- **Oldpeak**:Depressão do segmento ST induzido pelo exercício em relação ao repouso;
- **Slope**:A inclinação do segmento ST do pico de exercício;
- **Ca**:Número de vasos coloridos por fluoroscopia;
- **Thal**:3-Normal; 6=Defeito fixo; 7=Defeito reversível
- **Num**:Diagnostico de doença cardíaca (Variável a prever)

## 3 Processamento de dados

Após o carregamento dos dados, antes de aplicar qualquer técnica de escalabilidade e seleção de features, foi necessário algum efetuar algum tratamento dos mesmos:

- Todas as linhas com valor *Null* forem eliminadas, ficando com informação de 297 indivíduos;
- Conversão do tipo das *features* 'ca' 'thal' de *string* para *float*;
- Todos os valores da *feature* 'num' que fossem diferentes de 0 foram convertidos para 1, de modo a indicar se tinha ou não problemas cardíacos;
- Divisão do *dataset* em *features* de treino e *target*;

Uma vez tratados os dados, foi feito o processo de *Feature Selection*. Este tinha como objetivo selecionar as características mais importantes para obter melhores resultados na previsão do *target*. A técnica utilizada para obter a importância de cada *feature* foi a *Feature Importance*, que através do *ExtraTreesClassifier*, obtém a importância de cada característica para com a variável a ser estudada, num(*target*). Decidimos selecionar as primeiras 9. Além de uma seleção de *features*

foi necessário realizar escalabilidade dos dados. Este processo é importante realizar antes do treino, uma vez que uma *feature* com uma grande magnitude pode afetar a previsão. Simplificando com um exemplo fora do contexto do trabalho prático, 100 centímetros e 1 metro (medidas de distância) para nós, seres humanos, são iguais, no entanto, para os algoritmos de treino é interpretado de forma diferente acabando por dar um maior peso, como mencionado, às *features* com uma maior magnitude. O método escolhido para a escalabilidade foi o Min-max Scaler, que consiste em transformar os dados de modo a que estes se encontrem num intervalo especifico, por *default* entre zero e um.

## 4 Redes Neuronais

Como já referido anteriormente, para realizar uma comparação entre o *accuracy* de uma rede neuronal com e sem algoritmo genético foi necessário a criação de uma rede neuronal, mais especificamente uma rede *Multilayer Perceptron* [2] para a previsão de problemas cardíacos em determinados pacientes. Na realização do treino de redes neuronais, apesar do tratamento dos dados ser um fator importante, a configuração da rede neuronal em si também é um fator a ter em conta para a obtenção de melhores resultados. Dentro dos diversos parâmetros configuráveis na criação da rede neuronal podemos afirmar que os seguintes apresentados são alguns dos mais importantes:

- Pesos associado em cada ligação;
- O bias;
- Função de ativação;
- Algoritmo de optimização dos pesos;
- Número de *hidden layers* e quantidade de neurões;

Dos parâmetros referidos anteriormente, a nossa rede neuronal criada foi configurada da seguinte forma:

- Função de ativação: tanh *(hyperbolic tan function)*;
- Algoritmo de optimização de pesos: sgd *(stochastic gradient descent)*;
- Número de *layers*: Duas camadas com 100 neurões;

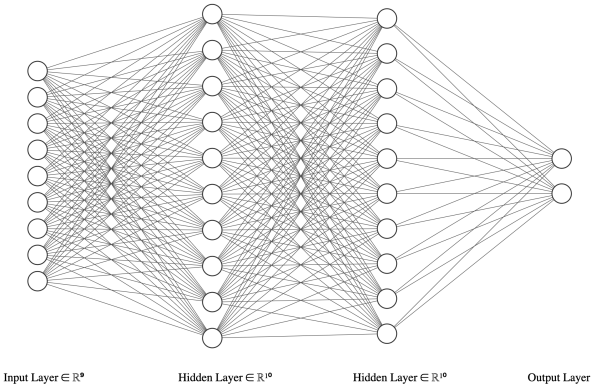


Figure 1: Representação da ANN

Para facilitar a visualização, cada *neuron* das duas *hidden layer* representam 10 *neurons*.

## 5 Algoritmos genéticos

Os algoritmos genéticos baseiam-se em mecanismos de seleção natural e genética [3; 4; 5]. Inicialmente, o processo começa com um conjunto de soluções (cromossomas) denominados "população", onde as soluções destas são utilizadas para formar uma nova população. Estas soluções que são selecionadas para formar novas gerações são escolhidas de acordo com a sua adequação, ou seja, quanto melhores, mais probabilidade de reprodução terão. Para que esta seleção aconteça é necessário que um conjunto de operadores atuem sobre uma dada população para gerar novas que se espera que sejam melhores do que as anteriores. Estes operadores são o *crossover* e *mutation*. O operador *crossover* é responsável pela recombinação de características dos "progenitores" durante a reprodução, com o intuito de que as próximas gerações herdem essas características. Geralmente, a probabilidade de *crossover* deve ser maior que a de *mutation*, uma vez que é o parâmetro genético predominante. O operador *mutation*, considerado um operador secundário, consiste na *troca* de um bit do cromossoma com uma certa probabilidade de mutação, geralmente muito baixa. Aplicado depois do crossover, este operador é responsável pela introdução e manutenção da diversidade genética da população.

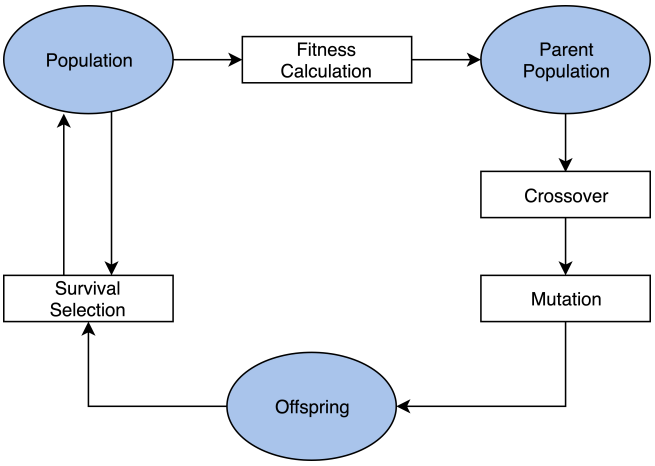


Figure 2: Representação de um algoritmo genético

De seguida descrevemos a implementação do nosso algoritmo genético.

### 5.1 Algoritmo genético implementado

A arquitectura de uma rede neural e o algoritmo de aprendizagem usado para a treinar são importantes decisões a ser tomadas. O algoritmo implementado tenta resolver esse problema usando algoritmos genéticos para evoluir os pesos da rede neuronal [6; 7].

**Crossover**

*Uniform Cross* foi usado para gerar os descendentes onde os *matching* genes são herdados aleatoriamente. A probabilidade de *crossover* ocorrer é controlada pelo o utilizador.

**Mutação**
O tipo de mutação presente no algoritmo é uma perturbação do peso e/ou do bias. A probabilidade de mutação ocorrer é também controlada pelo utilizador. Quando uma mutação ocorre, um número aleatório com uma distribuição gaussiana com media de 0 e *standart deviation* de 1 é adicionado ao peso.

**Função de Fitness**
A função de *FITNESS* utilizada para classificar os melhores elementos da população foi o*mean square error* (MSE),

$$E(g) = \frac{1}{N}\sum_{i=1}^{N}(t-y)^2$$

Onde o t é o *output* da rede neuronal e o y o target. Os melhores elementos por sua vez vão ser os que têm menor erro. O elemento perfeito vai ser um em que o erro seja zero.

## 6 Resultados e conclusões

No presente capítulo vamos apresentar alguns resultados obtidos e algumas conclusões relativamente ao trabalho prático. Como já foi anteriormente referido, um dos principais objetivos é comparar a *accuracy* de uma rede neuronal quando esta usa um algoritmo genético para a otimização dos pesos e bias e uma rede neuronal usando as "suas" próprias funções para a otimização do mesmo. Foram definidas combinações de parâmetros para serem testadas usando o algoritmo implementado onde se foi variando a probabilidade de crossover e mutação, assim como o número de *neurons* e *hidden layers* da rede neuronal. Para cada combinação de parâmetros o algoritmo foi executado 30 vezes de forma a conseguir ter um valor médio com o menor erro possível.

Na figura seguinte podemos visualizar a evolução do *fitness* ao longo das gerações.

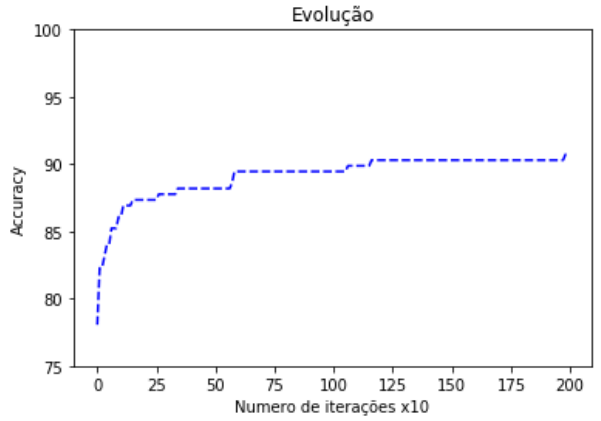


Figure 3: Evolução da *accuracy*

Como podemos concluir, inicialmente o genótipo evolui rapidamente, contudo, começa a estabilizar a partir de um certo ponto mantendo-se quase constante até ao final das gerações.

A seguinte tabela mostra os resultados da rede neuronal usando algoritmo genético.

| Crossover | Mutation | Neurons | Accuracy | Mean |
|---|---|---|---|---|
| 0.6 | 0.05 | 100 50 | 89.02 | 86.52 |
| 0.75 | 0.01 | 50 50 20 | 90.72 | 86.01 |
| 0.55 | 0.07 | 50 50 20 | 89.45 | 84.95 |
| 0.8 | 0.1 | 50 50 20 | 89.03 | 84.50 |
| 0.9 | 0.05 | 150 70 | 87.38 | 85.61 |
| 0.9 | 0.05 | 100 50 | 86.70 | 85.69 |
| 0.9 | 0.05 | 100 50 20 | 85.83 | 84.49 |
| 0.8 | 0.05 | 200 80 30 | 85.31 | 83.75 |
| 0.85 | 0.05 | 70 30 | 89.02 | 86.06 |
| 0.85 | 0.05 | 70 50 20 | 88.18 | 84.54 |
| 0.70 | 0.1 | 70 50 20 | 87.76 | 83.99 |
| 0.55 | 0.01 | 50 50 | 91.56 | 87.42 |

Table 1: Tabela de resultados para a evolução

Como podemos visualizar, em relação às células a azul na tabela anterior, quando se aumenta o número de *layers* é possível notar que a *accuracy* diminuí, em média. Podemos constatar também que o melhor resultado (amarelo), em média, aconteceu quando a probabilidade do *crossover* e a *mutation* tinham valores menores. De seguida, foram feitos testes para o *testing set* com os parâmetros que apresentavam melhores resultados na tabela anterior.

| Crossover | Mutation | Neurons | Accuracy |
|---|---|---|---|
| 0.75 | 0.01 | 50 50 20 | 78.33 |
| 0.9 | 0.05 | 100 50 20 | 81.67 |
| 0.55 | 0.01 | 50 50 | 82.33 |

Table 2: Tabela dos melhores resultados para conjunto teste

Conseguimos verificar que a *accuracy*, em relação ao treino, é mais baixa. A *accuracy* da rede *Multilayer Perceptron* para o testing set foi de 86.7%. Verificamos assim que esta é um pouco mais alta que a nossa implementação usando algoritmos genéticos. Conclui-se que o nosso algoritmo não é superior a uma rede neuronal que utiliza *stochastic gradient descent*, sendo o nosso melhor resultado 82.33%. No entanto a aplicação de algoritmos genéticos podem ser utilizados para os mais diversos parâmetros como, por exemplo, para escolha da melhor arquitectura de rede neuronal, seleção de *features* e até mesmo o tipo modelo de *machine learning* a ser usado.

## 7 Conclusão

Com este projeto é possível concluir que redes neuronais genéricas continuam a ter melhores resultados que a nossa variável de estudo. No entanto os resultados não foram desapontantes e mostram promessa.

Por esta razão, para um trabalho futuro, propomos aplicar um algoritmo genético mais complexo que permitisse uma maior manipulação dos parâmetros. Para além da perturbação dos pesos e do bias, permitir o algoritmo decidir o número de *layers* e *neurons* que cada rede pode ter. Durante a fase de experimentação reparou-se que havia casos onde o *accuracy* apresentava valores muitos semelhantes ou mesmo iguais. Isto deve-se ao fato de existirem genótipos com a

mesma solução apesar de terem codificações diferentes. Por outras palavras, duas redes neuronais que tenham *hidden layers* diferentes podem ser funcionalmente equivalentes. Uma maneira de resolver este problema seria aplicar marcadores de historia a cada *network*.

## References


[1] Robert Detrano. Heart Disease Data Set. https://archive.ics.uci.edu/ml/datasets/Heart+Disease.

[2] John B. Hampshire II and Barak A. Pearlmutte. Equivalence proofs for multi-layer perceptron classifiers and the bayesian discriminant function. *Elsevier Inc.*, 1991.

[3] Algoritmos genéticos. https://web.fe.up.pt/~ee98221/paginas/algo.htm.

[4] Melanie Mitchell. An introduction to genetic algorithms. *MIT Press.*, 1998.

[5] Marek Obitko. Algoritmos genéticos. https://www.obitko.com/tutorials/genetic-algorithms/portuguese/ga-basic-description.php, 1998.

[6] David J. Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. *BBN Systems and Technologies Corp.*, pages 762–767, 1989.

[7] William T. Kearney. Using genetic algorithms to evolve artificial neural networks. 2016.