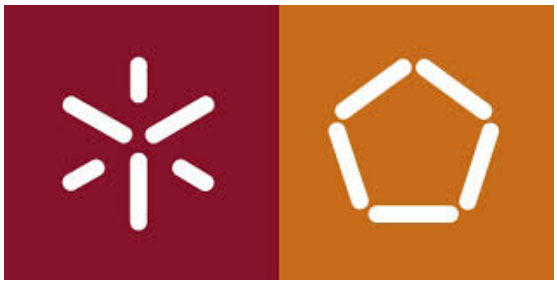

Universidade do Minho - Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Computação Natural

Trabalho prático
Redes Neuronais Artificiais

Carlos José Gomes Campos a74745
José Pedro Ferreira Oliveira a78806
Vitor José Ribeiro Castro a77870

1 de Abril de 2019

Resumo

Este relatório pretende descrever os passos necessários ao desenvolvimento do projeto proposto pela equipa docente, a saber, a deteção de tecidos cancerígenos. O processo compreende o tratamento de dados, até à implementação e otimização final do modelo de previsão, passando por todas as fases de teste e validação, utilizando a biblioteca TensorFlow. Este projeto é feito no âmbito da unidade curricular de Computação Natural, do perfil de Sistemas Inteligentes, do Mestrado Integrado em Engenharia Informática.

Conteúdo

1 Introdução

1.1 Contextualização

Recentemente, com a melhoria das técnicas de imagem e equipamentos disponíveis nos diversos hospitais do país, todo o processo hospitalar pode utilizar de técnicas inovadoras. O *machine learning*, no caso da deteção de massas em mastografia, pode facilitar a filtragem de casos de tumor, e acelerar o processo de tratamento dos mesmos. Cada vez mais se sabe que um diagnóstico atempado é a chave para uma cura bem sucedida, pelo que o estudo que se vai realizar poderia ter, realmente, um impacto positivo na vida de milhares de utentes pelo mundo fora.

1.2 Caso de Estudo

O projeto tem por base a conceção de um modelo de previsão baseado em Redes Neuronais Artificiais, para classificar se uma massa detetada num exame de mastografia é benigna ou maligna, tendo por base um conjunto de dados clínicos de casos reais que apresentam um conjunto de métricas relevantes.

1.3 Estrutura do Relatório

O relatório está dividido em três partes, correspondentes às etapas de criação do modelo.

Na secção 2 são explicadas as formas utilizadas para analisar e visualizar os dados do *dataset* fornecido, bem como o processamento que foi necessário realizar, tendo por base essa análise. Na secção 3 será descrito o processo de construção do modelo, e serão mencionadas quais as APIs do *TensorFlow* que foram utilizadas, que técnicas foram implementadas para treinar e validar o modelo, bem como as principais características, e respetiva justificação, do primeiro modelo implementado, acabando como a análise dos resultados obtidos por este primeiro modelo. Na secção 4 será explicada toda a implementação dos algoritmos genéticos usados para otimizar os hiperparâmetros do modelo, começando pela seleção dos cromossomas, seguindo para a descrição detalhada do algoritmo. Na secção 4.3 serão analisados os resultados obtidos com a otimização e será feita uma comparação com evolução verificada, relativamente ao primeiro modelo.

O relatório termina com as conclusões na secção 5 onde é feita uma reflexão acerca do trabalho realizado bem como do trabalho futuro.

2 Pré-processamento de dados

O processo de criação do modelo de previsão inicia-se com o carregamento e visualização dos dados contidos no *dataset* fornecido. Com recurso à bibioteca *pandas*, os dados foram carregados para um objeto *dataframe*, e os valores em falta foram substituídos pelo caractere '?'. Com o intuito de avaliar a qualidade dos dados a utilizar, foi utilizada a função *describe* que gera estatísticas descritivas que fazem um sumário das tendências e dispersão das variáveis do dataset.

	BI-RADS	Age	Shape	Margin	Density	Severity
count	959.000000	956.000000	930.000000	913.000000	885.000000	961.000000
mean	4.348279	55.487448	2.721505	2.796276	2.910734	0.463059
std	1.783031	14.480131	1.242792	1.566546	0.380444	0.498893
min	0.000000	18.000000	1.000000	1.000000	1.000000	0.000000
25%	4.000000	45.000000	2.000000	1.000000	3.000000	0.000000
50%	4.000000	57.000000	3.000000	3.000000	3.000000	0.000000
75%	5.000000	66.000000	4.000000	4.000000	3.000000	1.000000
max	55.000000	96.000000	4.000000	5.000000	4.000000	1.000000

Figura 1: Sumário com valores estatísticos

Relativamente aos valores em falta, foi criado um *heatmap* para avaliar a sua distribuição, o que permitiu saber que a feature *density* é a que apresenta um maior número de valores em falta e que estes se encontram mais concentrados nas primeiras instâncias do dataset, no entanto, é apenas uma pequena parte dos dados e acaba por ser pouco relevante. As restantes *features* apresentam uma distribuição aleatória, não se verificando correlações diretas entre as mesmas.

Posto isto, decidiu-se eliminar estes valores do *dataset*, através da função *dropna*.

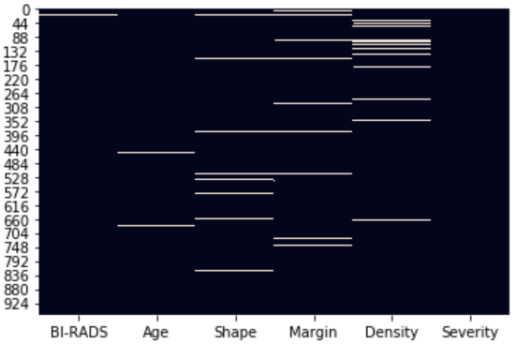

Figura 2: Heatmap para visualização dos dados em falta

Posteriormente, foi necessário separar as *features* e as *labels* e redefinir os índices, dado que foram eliminados alguns valores previamente.

Para a criação de um bom modelo de RNA é necessário normalizar os dados de input, uma vez que todos as *features* devem estar numa gama de valores tal, que garanta uma distribuição de importância equitativa entre as mesmas, prevenindo situações em que valores maiores se tornam dominantes relativamente a inputs menores durante o treino da RNA. Para tal, foi usado o *StandardScaler* para normalizar o conjunto de *features*.

3 Construção da Rede Neuronal Artificial

3.1 Criação do modelo

Para a criação do modelo foi utilizado o classificador *DNNClassifier* disponível na biblioteca TensorFlow.

```
classifier = tf.estimator.DNNClassifier(hidden_units=[10,10,10],
                                        n_classes=2,
                                        feature_columns=feat_cols,
                                        activation_fn=tf.nn.relu,
                                        dropout=0.5,
                                        optimizer=tf.train.AdamOptimizer(
                                           learning_rate=0.01
                                        ))
```

Figura 3: Classificador e respetivos parâmetros

Analisando os parâmetros em 3, numa primeira implementação do modelo criou-se uma RNA com três camadas intermédias, cada uma com dez nodos e definiu-se que o número de classes a prever seriam duas. O argumento *feature_columns* foi gerado anteriormente e representa um array com os nomes das *features* num formato reconhecido pelo TensorFlow. A função de ativação escolhida foi a **ReLu**, dado ser a mais utilizada em modelos de deep learning. Para estratégia de otimização, utilizou-se o otimizador *AdamOptimizer*, visto ser o que atualmente exibe melhores resultados em modelos semelhantes, como se observou por pesquisas do grupo. Ao argumento *learning_rate* foi atribuído o valor de 0.01, pelo facto de se enquadrar no intervalo de valores comummente utilizados. Por fim, utilizou-se um *dropout* com cerca de 0.5, de maneira a adicionar um fator de probabilidade de ativação ou desativação aos neurónios da rede. Assim, forçou-se os dados a encontrarem caminhos alternativos ao longo da rede, evitando o *overfitting* do modelo.

3.2 Treino e Avaliação

Como metodologia de validação do nosso modelo, utilizou-se a técnica de *cross-validation* com dez folds. Através da biblioteca *Kfold* do *sklearn*, os índices das divisões da coleção de dados foram gerados automaticamente e apenas foi necessário iterar pelo objeto do *Kfold* dez vezes e definir os dados de treino e teste, para proceder à validação do modelo.

Para cada subset gerado pelo KFold:

- Definir a função de input para alimentar o classificador com os dados de treino, considerando alguns argumentos como *Shuffle* a *True* de maneira a garantir aleatoriedade na escolha das instâncias para treino. O tamanho do *batch_size* é vinte e este define a quantidade de instâncias que é carregada para memória de cada vez para o treino do modelo.
- Treinar o modelo com esses dados, utilizando a função *train* com o argumento *steps* definido a quinhentos;
- Definir a função de input para fornecer ao classificador os dados de teste;
- Iterar sobre o objeto resultante da previsão e obter os dados da classificação;
- Calcular a *accuracy* obtida neste subset e armazená-la numa variável;

Finalmente, faz-se a média de todas as *accuracies* obtidas em cada subset.

```
Accuracy: 0.791566265060241 STDEV 0.06144709555781371 Coefficient of variation 0.07762722878688795
```

Figura 4: Score obtido pelo primeiro modelo implementado

Adicionalmente, verificou-se o desvio padrão no conjunto de *accuracies* e através do coeficiente de variação, que representa a razão entre o desvio padrão e a média, podemos verificar se existem grandes variações entre as *accuracies* de cada *fold*. Caso exista, pode indicar que o modelo está a ter um bom desempenho para uns *folds* e mau desempenho para outros, indicando problemas de *overfitting*. O valor percentual obtido é de cerca de 7.8% o que significa que os resultados são homogéneos, não existindo grande dispersão nos valores.

4 Otimização dos hiperparâmetros

O primeiro modelo criado apresentou uma boa percentagem de acerto, no entanto, caso isso não se verificasse era necessário fazer uma otimização dos hiperparâmetros, pois o modelo não era viável.

A estratégia para fazer esta otimização passou por utilizar algoritmos genéticos, em que cada cromossoma representa um *setup* de configuração de um modelo de RNA e cada gene representa um hiperparâmetro desse modelo.

Cada cromossoma possui quatro elementos, o primeiro corresponde à variável *learning_rate*, o segundo número de nodos por cada camada intermédia, o terceiro ao número de camadas intermédias e o último é a função de ativação. A função de fitness será representada pela *accuracy* obtida por cada configuração.

4.1 Funções auxiliares

Para a integração de GA foram definidas um conjunto de funções descritas de seguida.

- **create_new_population** Esta função é responsável por gerar a população inicial, numa situação em que não houvesse a menor ideia dos parâmetros iniciais a usar, torna-se necessário generalizar e, portanto, o que esta função faz é cria um conjunto de dez cromossomas com configurações aleatórias, tomando em atenção um intervalo de valores razoável em que cada hiperparâmetro pode estar. Para a *learning_rate* é gerado um número aleatório entre 10^-2 e 10^-1, para o número de nodos é gerado uma potência de dois, para o número de camadas intermédias é obtido um número entre um e vinte e, finalmente, para a função de ativação é gerado um zero (associado à função Softmax), um um (associado à função ReLu) ou um dois (associado à função Leaky ReLu).

- **update_classifier_parameters** Esta função recebe como argumento um determinado cromossoma e faz o mapeamento dos valores simbólicos dos genes para as respetivas configurações na criação de um modelo.

- **select_mating_pool** Esta função recebe como argumento uma população, os valores de fitness associados a cada indivíduo e o número de "pais"que serão selecionados para servirem de ponto de partida para a próxima geração. Assim, iterativamente é feita a verificação dos valores mais altos de fitness e a seleção do cromossoma associado.

- **crossover** Esta função recebe como argumento um conjunto de cromossomas e o número de "filhos"a serem gerados. O processo passa por gerar novos cromossomas com metade das características de outros dois cromossomas da geração anterior.

- **mutation** Função responsável por introduzir e preservar diversidade, garantindo que o algoritmo não estabilize num máximo local. Através disto, é possível prevenir a situação de os cromossomas serem muito parecidos e

levar ao abrandamento ou paragem da evolução. Esta função recebe o conjunto de cromossomas "filhos"e aleatoriamente escolhe um gene para ser modificado, de seguida atribui um valor também obtido de forma aleatória mas dentro de um intervalo de valores admissível.

4.2 Algoritmo

O processo inicia-se com criação de uma população através da função *create_new_population*.

- O algoritmo corre durante vinte gerações, para cada geração:
  - Para cada cromossoma de uma geração:
    - Como metade da população da nova geração são os cromossomas da geração anterior, é possível acelerar o processo verificando se o cromossoma em questão já foi avaliado anteriormente e nesse caso é apenas necessário obter o resultado do modelo criado na geração anterior;
    - Caso isso não se verifique, estamos perante um novo cromossoma e terá de ser avaliado através da função *classify_create_folds* que para um dado cromossoma atualiza o modelo, treina e valida com base na construção do modelo anteriormente mencionada, retornando a precisão do modelo;
- Guarda-se a melhor prestação de cada geração para analisar posteriormente a evolução;
- Seleciona-se os melhores cromossomas para "pais"da nova geração, através de função *select_mating_pool*;
- Aplica-se o processo de *crossover*, através da função *crossover*;
- Aplica-se o processo de *mutação*, através da função *mutation*;
- Gera-se a nova população, agregando os "pais"e os "filhos";

Finalmente, obtém-se o valor máximo de fitness da última geração e a respetiva configuração de genes.

4.3 Resultados finais

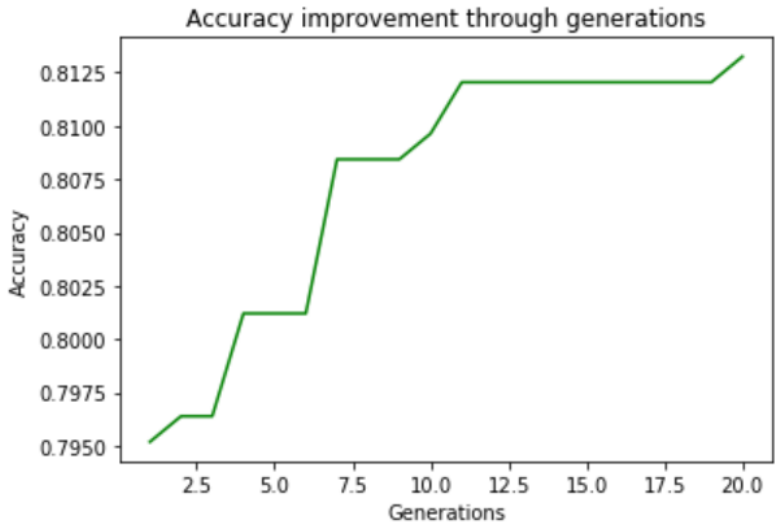

Figura 5: Accuracy improvement

Através da análise do gráfico 5, é possível verificar uma evolução da *accuracy* ao longo das gerações, na ordem de grandeza das décimas. A partir da décima geração a evolução estagnou, verificando-se uma evolução na ordem de grandeza das centésimas nas últimas gerações. A carga computacional verificada nas vinte gerações, podia ter sido reduzida para metade, uma vez que o resultado obtido não foi significativo.

Depois de analisados os resultados obtidos posteriormente à execução do algoritmo genético, comparou-se o modelo sugerido com o modelo original. O modelo original é aquele que tinha sido desenvolvido com o que seria, pela análise da equipa, o melhor modelo possível, de acordo com parâmetros escolhidos através de pesquisa.

O modelo original é o seguinte:

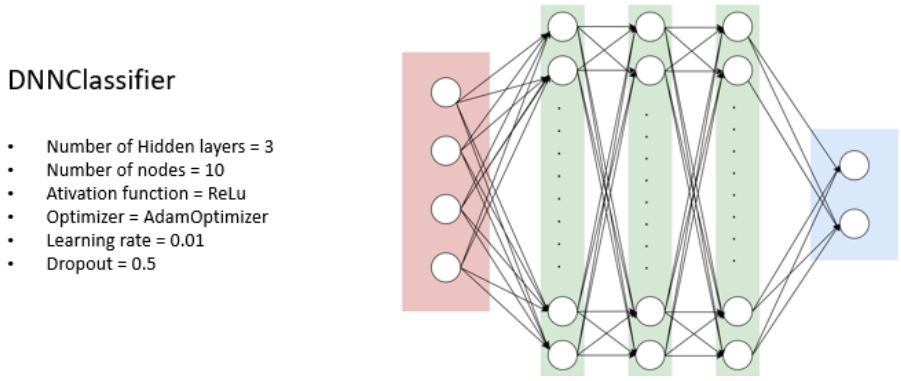

Figura 6: Modelo original

O modelo encontrado depois de corrido o algoritmo genético tem os seguintes hiperparâmetros:

```
The best hyperparameters obtained are [ 0.07056695  8.         16.          0.        ] with an accuracy of 0.8132530120481928
```

Figura 7: Hiperparâmetros obtidos após otimização

Learning rate = 0.07056685
Number of nodes = 8
Number of hidden layers = 16
Activation function = Softmax

Figura 8: Modelo obtido após algoritmo genético

Foram feitos cálculos relacionados com a *std-deviation* e foi possível concluir que a margem de erro em relação à média de *accuracy* era de apenas 4%, para uma *accuracy* final de 81%. O número de nodos e de *hidden layers* é dentro do normal, mas o grupo esperava encontrar na função da ativação a *relu* e não a *softmax*. De facto, uma vez que para o gradiente o poder computacional exigido não é contabilizado, o *softmax* acabou por ser escolhido pelo algoritmo genético.

5 Conclusão

Este trabalho foi particularmente desafiante e interessante. Se, por um lado, tinham sido fornecidas todas as bases para a resolução do problema inicial, através da criação de uma rede neural específica, surgiram vários desafios.

A proposta de uso de algoritmos genéticos foi de difícil compreensão, ao início, sendo que devido aos valores que os genes podiam tomar eram vastos. Mais ainda, o número de genes e o seu significado podia ser diferente dependendo do contacto, pelo que a função que cria o *classifier* ocupou um largo tempo a ser explorada.

Assim, apesar de ter sido extenso o tempo em que se procurou perceber o significado do uso de cada um dos valores diferentes nos genes e, por conseguinte, nos classificador, o grupo conseguiu entender o porquê e as situações em que se deve aplicar cada um desses parâmetros. Foi também possível perceber a carga computacional que se exige quando se enfrentam problemas de tamanhos cada vez maiores. De facto, o problema que o grupo tinha em mãos, era infinitamente pequeno quando comparado com muitos outros problemas do quotidiano. É, por isso, necessário agir com precaução aquando do uso de algoritmos genéticos ou outros métodos de treino e teste consecutivo de redes neurais, pelo tempo que este processo pode tomar.