Relatório

Inteligência Artificial

Mestrado Integrado em Engenharia Informática e Computação

Redes Neuronais para a Identificação de Pulsares

Afonso Jorge Ramos	up201506239@fe.up.pt
Bárbara Sofia Silva	up201505628@fe.up.pt
Julieta Pintado Jorge Frade	up201506530@fe.up.pt

8 de Abril 2018

Conteúdo

1 Objetivo

Este trabalho tem como finalidade a aplicação de redes neuronais artificiais na identificação de pulsares.

Os pulsares são estrelas de neutrões altamente magnetizadas que se formam quando uma estrela mais massiva que o Sol colapsa. O seu campo magnético é mais forte que o da Terra e a radiação é irradiada dos seus polos. De facto, é nos polos de um pulsar onde o campo magnético é mais intenso, resultando na produção de radiação em forma de ondas rádio, que por sua vez são emitidas num raio de radiação muito apertado, à semelhança de uma lanterna.

Adicionalmente, devido à sua rotação, cada pulsar produz um padrão de emissão diferente. Também, a existência de interferências rádio e *signal noise* prejudicam a sua deteção, o que faz com que encontrar um pulsar se torne bastante difícil. De modo a facilitar este processo, o uso das ferramentas de *Machine Learning* é fundamental, pois a classificação de pulsares candidatas é automatizada, obtendo assim uma rápida análise dos dados.

Machine Learning consiste na capacidade de uma máquina reconhecer padrões e na habilidade dos computadores aprenderem sem serem explicitamente programados. Além disso, foca-se em um dos dois raciocínios da inteligência artificial, o **indutivo**, visto que extrai regras e padrões de grandes conjuntos de dados. Neste caso, o objetivo é treinar o computador para que este seja capaz de identificar um pulsar.

2 Descrição

2.1 Especificação

2.1.1 Descrição e análise do dataset

O *dataset* que nos foi providenciado provém de uma amostra de pulsares candidatos recolhida durante a *High Time Resolution Universe Survey, South (HTRU2)*. Um Pulsar é uma estrela de neutrões relativamente rara, que produz emissões de rádio detetáveis no nosso planeta, pelo que, são de elevado interesse científico, tanto como sondas do espaço-tempo e meio interestelar, bem como estado da matéria.

Com a rotação dos pulsares, chega-nos um padrão de emissões de rádio *broadband*, padrão esse que se repete conforme a velocidade de rotação dos pulsares, sendo que, a deteção destes padrões de ondas de rádio, nos ajudam a identificar, com a maior certeza possível, estes pulsares. No entanto, cada pulsar produz um padrão de emissão ligeiramente diferente, que varia com a rotação, logo é necessário calcular a média do padrão ao longo das várias rotações. Para além disso, a maioria das deteções de pulsares são falsos positivos causados por interferências da frequência de rádio (RFI) e *signal noise*, o que torna sinais legítimos difíceis de encontrar.

Os dados fornecidos de classificação de dados tratam os candidatos como classificação binária, visto que os custos de classificação com múltiplas classes seriam de um elevado custo, nesta área de trabalho. Neste *dataset*, pulsares legítimos são uma minoria, visto que, como podemos ver, temos apenas 1639 verdadeiros pulsares e 16259 casos de falsos positivos causados por *RFI/noise*, no total de 17898 candidatos.

2.1.2 Pré-processamento dos dados

Cada candidato é descrito por 8 variáveis continuas, bem como uma variável que indica se esse candidato é ou não um verdadeiro pulsar. As primeiro quatro varáveis foram obtidas pelo perfil integrado, isto é, são variáveis que tiveram em consideração o tempo e a frequência para o seu cálculo. Enquanto que as seguintes quatro foram obtidas de uma forma similar, no entanto, através de uma curva DM-SNR, proveniente de uma passagem para as duas dimensões de uma matriz representada pela *Dispersion Measure*, pelo *Signal to Noise Ratio* e pelo *offset* do período, tal como podemos observar no paper original de Robert James Lyon, capítulo 2.2.1, tal como podemos ver na imagem imediatamente abaixo.

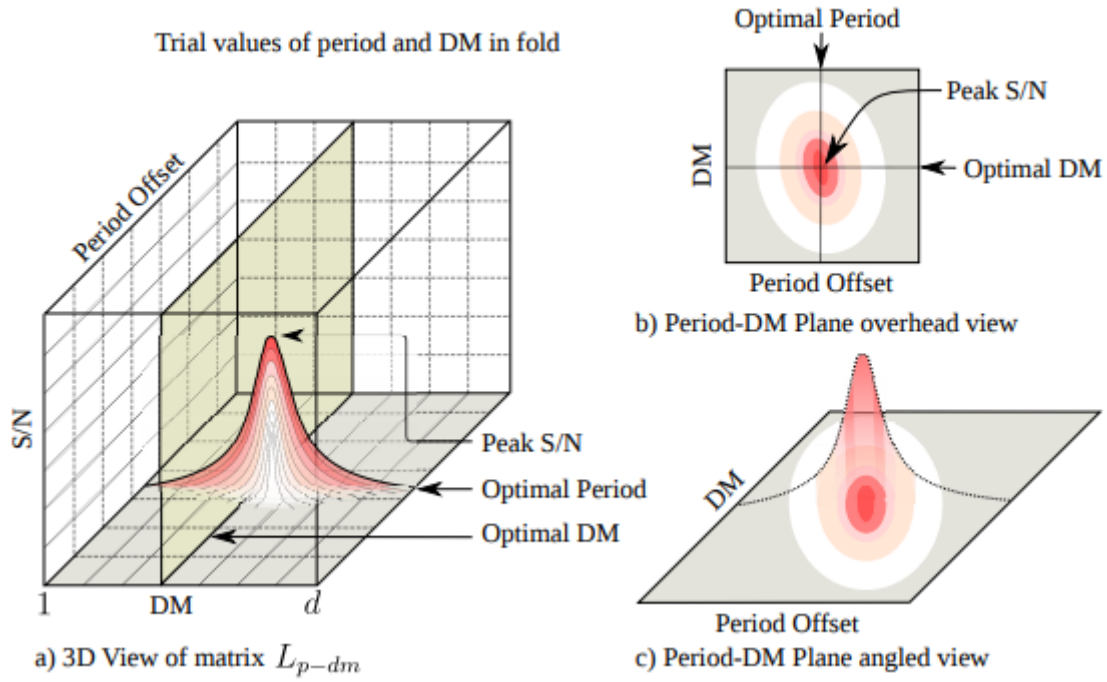

Figura 1: Visualização dos dados recolhidos para cada candidato a pulsar. Em a) podemos observar um pico que representa o maior *Signal to Noise Ratio*. Em b) temos uma visão da matriz de cima, que nos ajuda a distinguir pulsares, visto que estes devem ter regiões circulares de SNR crescente, à medida que os valores do período e do *Dispersion Measure* ficam ótimos. Já em c) temos uma visão lateral de b)

Podemos então observar uma lista ordenada das várias características de cada candidato, as quais estarão a ser tomadas em consideração no processamento dos dados.

1. Mean of the integrated profile.
2. Standard deviation of the integrated profile.
3. Excess kurtosis of the integrated profile.
4. Skewness of the integrated profile.
5. Mean of the DM-SNR curve.
6. Standard deviation of the DM-SNR curve.
7. Excess kurtosis of the DM-SNR curve.
8. Skewness of the DM-SNR curve.

Face a esta notória ausência de dados no dataset providenciado, teremos de proceder a algumas táticas para melhorar o dataset para, igualmente, tornarmos o resultado o mais preciso possível. Desta forma, poderíamos recolher mais dados, algo impossível; mudar a métrica de medição de performance, algo que não se aplica neste caso, pois o objetivo é, de facto, que seja o mais precisa possível; proceder ao *resampling* dos dados, ora com cópias de pulsares verdadeiros, ora como eliminação de falsos positivos, no entanto, isto não seria ideal por si só, visto que o número de dados é bastante elevado e alterações manuais nos mesmos seria muito trabalhosa; ou, podemos tomar o caminho que achámos o ideal, que será gerar candidatos sintéticos utilizando algoritmos como SMOTE (Synthetic Minority Over-sampling Technique), que pode ser aplicado utilizando, por exemplo uma das suas várias implementações incluídas no módulo de Python Imbalanced-learn. De qualquer das maneiras, com a finalização da implementação deste projeto iremos tentar misturar estas várias técnicas, para que possamos ter a maior taxa de precisão possível.

2.1.3 Modelo de aprendizagem a aplicar

Multilayer Perception (MLP) é uma rede neuronal artificial do tipo *feedforward*, isto é, as conexões entre os neurónios não formam ciclos o que faz com que a informação se mova num único sentido. Esta rede consiste em pelo menos três camadas de nós, sendo então uma rede neuronal multicamada. Além disso, exceto os nós de entrada, cada nó é um neurónio que utiliza a função de ativação não linear. O MLP aplica uma técnica de aprendizagem supervisionada indutiva denominada *backpropagation*.

A aprendizagem supervisionada é uma tarefa de *machine learning* que deduz uma função a partir de um conjunto de dados de treino categorizados. Este tipo de aprendizagem indutiva demonstra ser o mais apropriado para o trabalho, visto que analisa os dados de entrada, os candidatos, e obtém uma função que permite identificar um pulsar, através de aproximações.

Como mencionado anteriormente, ***backpropagation*** *(backward propagation of errors)* é um algoritmo utilizado por esta aprendizagem, onde a rede opera em duas fases e implementa um método de otimização chamado *gradient descent*. Primeiramente, um padrão é apresentado à camada de entrada, e propaga-se, camada a camada, até à última camada da rede. De seguida, o resultado obtido é comparado ao desejado. Se estiver errado, um valor de erro é calculado para cada um dos neurónios da camada de saída. Este valor é propagado por retrocesso, até à camada de entrada, e os pesos das arestas da rede neuronal são atualizados de forma a minimizar a função de perda.

Assim, após serem recebidos dados de entrada suficientes, a rede converge para um estado onde os erros são cada vez menores, ou seja, aprendeu.

2.1.4 Arquitetura das Redes Neuronais

Na realização deste trabalho vai ser usada uma **rede *feedforward* de múltiplas camadas**. Isto implica que a rede tenha vários nós organizados nas seguintes camadas:

1. Uma camada chamada **Input Layer**, que recebe os dados do exterior e os introduz à rede. Não é necessária nenhuma computação nesta camada, pois os nós só têm a responsabilidade de transferir informação para as camadas seguintes.
2. Uma ou mais camadas chamadas **Hidden Layers**, que não têm qualquer ligação ao exterior da rede. Elas realizam a computação da informação proveniente da Input Layer e passam-na para a Ouput Layer.
3. Uma camada chamada **Output Layer**, que é responsável pela computação dos dados provenientes das Hidden Layers e transferência destes para o exterior da rede.

Numa rede *feedforward* a informação move-se unicamente num sentido (da Input Layer para as Hidden Layers e de seguida para a Output Layer), ou seja, cada nó só pode estar ligado a nós da camada seguinte e estas ligações entre nós não fazem ciclos.

2.1.5 Configuração Prevista da Rede

É difícil antes de fazer qualquer teste prever qual a melhor configuração da rede. O número de camadas, número de nós e funções de transferência podem influenciar em muito o resultado dado pela rede. No futuro, com a ajuda de um conjunto de dados, esta rede será validada para verificar se as escolhas tomadas para a sua configuração foram de facto as melhores. De seguida falamos da configuração que achámos mais correta dado o problema e conceitos unicamente teóricos.

2.1.6 Input layer

O número de nós nesta camada é geralmente o número de atributos do *dataset* usado, no caso deste problema 8. Decidiu-se a estes nós adicionar um nó *bias* para ajudar a rede a ajustar-se aos dados recebidos, a camada fica então com **9 neurónios**.

Um **nó *bias*** é um neurónio extra para cada camada da rede exceto a output layer e que guarda o valor de 1. Este nó não está ligado a nenhuma camada anterior à que se encontra e, portanto, não representa uma verdadeira atividade. Sem este nó, num input com todas as variáveis a zero o output seria tudo zero também, esta pode ser uma boa solução para alguns sistemas, mas para a maioria é demasiado restrita.

Na fase de ajustes da rede, muda-se simultaneamente o peso e o valor, portanto uma mudança no peso pode neutralizar a mudança no valor que foi útil para uma instância de dados prévia. A adição de um nó *bias* ajuda no controlo do comportamento de cada camada, pois permite mover a função de transferência horizontalmente ao longo do eixo de input, continuando com a forma/curvatura inalterada. Assim, conseguem-se diferentes resultados e pode-se alterar conforme as necessidades do problema. Uma boa analogia para isto é:

$$y = ax + b$$

Sem a componente b a linha passa sempre pela a origem o que pode não ser o desejado, esta componente ajuda a mover a linha verticalmente de modo a encontrar o local desejado.

2.1.7 Hidden layers

O primeiro problema para determinar a configuração desta parte da rede é determinar o número de camadas que a constituem e para isso é importante ter em conta a performance da rede. As situações em que a performance aumenta com a adição de uma segunda ou terceira camada são raras, pois na maioria dos problemas uma Hidden Layer é suficiente, portanto a nossa rede terá **uma só camada deste tipo**.

O segundo problema é determinar quantos nós a Hidden Layer terá. Para assegurar a habilidade da rede de generalizar, o número de nós precisa de ser o mínimo possível. Se houver um excesso de nós, a rede torna-se num banco de memórias que consegue lembrar-se do conjunto de treino, mas não tem uma boa performance em amostras que não estejam presentes neste conjunto. Para determinar o número de nós ideal foram tidas em conta em três regras:

1. O número de neurónios de uma Hidden Layer deve estar entre o número de nós da Input Layer e o número de nós da Output Layer.
2. O número de nós de uma Hidden Layer deve ser 2/3 da soma do número de nós da Input Layer e da Output Layer.
3. O número de nós de uma Hidden Layer deve ser menor que o dobro dos nós da Input Layer.

Assim, concluímos que a nossa camada deve ter **7 nós**, sendo que um destes nós é um nó *bias*.

O terceiro e último problema é determinar que função é usada para assumir o papel de função de transferência. Esta função é usada para determinar o output de cada neurónio. Foram consideradas dois tipos de funções: Sigmoid e ReLU.

A função Sigmoid tem um domínio entre 0 e 1 e tem como expressão:

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

Enquanto que a função ReLU tem um domínio de 0 a infinito e expressão:

$$R(x) = x^+ = max(0, x)$$

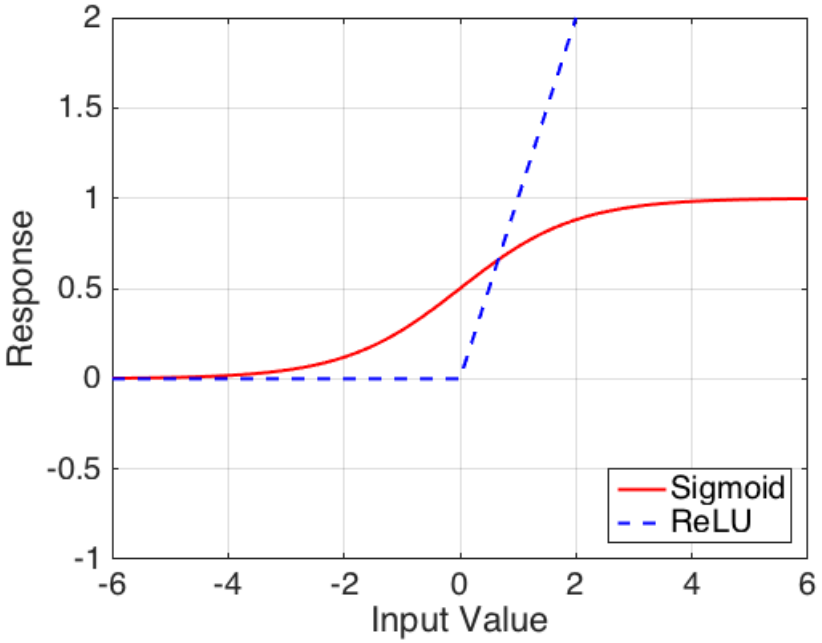

Figura 2: Funções Sigmoid e ReLU

O problema da função Sigmoid é que o máximo da sua derivada é igual a 1/4, ou seja, quando se aumenta o valor de x, o gradiente diminui exponencialmente. Como o gradiente da função ReLU é 1 ou 0, esta não tem este problema.

Foi então decidido usar esta última função **ReLU como função de transferência**.

2.1.8 Output layer

Para determinar o número de neurónios nesta camada considerámos duas opções:

1. Usar um único nó de output e a função de transferência deste seria uma função Sigmoid. Dependendo do resultado desta função ser menor ou maior que 0.5 concluiríamos se os dados fornecidos no input correspondiam ou não a um Pulsar.
2. Usar dois nós que representariam as classes Pulsar e Não-Pulsar. O resultado da função de transferência de cada um diria a sua probabilidade de se verificar sendo que a soma dos dois daria 1 (ex.: Pulsar:0.70 e Não-Pulsar:0.30 concluindo assim que os dados correspondiam a um Pulsar). Esta função de transferência seria do tipo Softmax. Esta função achata a unidade de cada output entre 0 a 1 tal como uma função Sigmoid, mas também divide cada output de modo a que a soma deles dê 1. O output desta função diz-nos a probabilidade de cada uma das classes. A expressão da função da Softmax é:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad for \quad j = 1, ..., K$$

Foram tidos estes dois tipos de funções em consideração pois são as duas usadas em casos onde se pretende prever a probabilidade como output. Se como output fosse preciso mais do que uma classe, seria obrigatório usar Softmax, mas como o resultado é binário (é ou não é pulsar) **vai ser usada a função Sigmoid** pois matematicamente é mais simples.

2.2 Trabalho Efetuado

Face a todo o conhecimento que obtivemos ao longo da análise do projeto, passámos ao início da sua implementação:

1. **Investigação de que ferramentas utilizar**, pelo que optámos por escolher usar Keras e Tensorflow.
2. **Configuração das ferramentas**, para que possamos testar a nossa implementação nas várias máquinas, que se demonstrou mais morosa que o inicialmente previsto.
3. **Processamento do dataset**, utilizando a biblioteca Pandas, que reduziu bastante o tempo despendido nesta parte do projeto.

4. **Implementação inicial**, minimamente funcional, para que possamos, calmamente, melhorar cada aspeto da implementação, de forma a tomarmos as melhores decisões no futuro.

De agora em diante, iremos aperfeiçoar o estado atual da implementação, para depois integrarmos a biblioteca *Synthetic Minority Over-sampling Technique* no nosso trabalho e, finalmente, passarmos à melhoria da heurística, para obtermos a maior precisão possível.

2.3 Resultados esperados e forma de avaliação

Depois de a rede estar implementada, é necessário passar por 3 fases para adaptar e verificar o funcionamento desta:

1. **Treinar a rede**, ajustando os pesos das arestas.
2. **Validar a rede**, ajustando a sua configuração (número de camadas, número de neurónios e funções de transferência usadas).
3. **Testar a rede**, analisando a sua performance e estimar a taxa de erro.

Para cada uma destas fases, serão usadas instâncias do dataset fornecido. Como a fase de treino é fundamental para a boa performance da rede, vão ser reservados mais dados para esta. Assim, a fase de treino terá 40% dos dados, a fase de validação 30% dos dados e a fase de teste 30% dos dados.

3 Conclusões

O facto de o tema deste trabalho reunir grandes matérias da inteligência artificial, nomeadamente redes neuronais, com a astronomia torna a sua implementação e compreensão mais interessante.

Em suma, o grupo planeia concluir este trabalho com sucesso, atingindo todos os objetivos e chegando aos resultados esperados.

4 Recursos

Durante a elaboração deste relatório o grupo recorreu às seguintes fontes:

- Ryan S. Lynch, Searching for and Identifying Pulsars
- Robert James Lyon, Why are Pulsars Hard to Find?
- https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your
- https://en.wikipedia.org/wiki/Machine_learning
- https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-
- https://en.wikipedia.org/wiki/Softmax_function
- https://www.quora.com/Artificial-Neural-Networks-Why-do-we-use-softmax-function
- https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidde
- https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks
- https://www.quora.com/What-is-bias-in-artificial-neural-network
- https://stats.stackexchange.com/questions/185911/why-are-bias-nodes-used-in-neu
- https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o_sigm%C3%B3ide
- https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/
- https://datascience.stackexchange.com/questions/15484/sigmoid-vs-relu-function-
- https://www.quora.com/What-is-special-about-rectifier-neural-units-used-in-NN-l
- https://en.wikipedia.org/wiki/Multilayer_perceptron
- https://en.wikipedia.org/wiki/Supervised_learning
- http://conteudo.icmc.usp.br/pessoas/andre/research/neural/MLP.htm