TP3 - Redes Neurais

Luiz Eduardo Lima Coelho

2016020991

1- Introdução

Neste trabalho utilizamos redes neurais artificiais para resolver um problema de classificação supervisionada. O problema apresentado propunha a criação de um modelo de classificação de objetos astronômicos.

A base de dados apresentada continha 18 campos, sendo que 1 deles era a classe de cada objeto, e saber prevê-la é todo o objetivo do modelo. Os outros 17 campos correspondem a *features* numéricas que descrevem um dado objeto. Ao fim da modelagem, gostaríamos portanto, que vistos estes 17 atributos de um novo objeto desconhecido, saber prever se ele á uma estrela, uma galáxia ou um quasar.

Modelos de redes neurais funcionam com aprendizado, isso é, saber resolver um problema por meio de exemplos. Porém um problema de utilizar aprendizado é que o modelo treinado pode se especializar muito no conjunto de dados visto, e então, ser inapto a reconhecer exemplos novos, ou seja, incapaz de se generalizar para dados reais.

Esse e outros problemas serão levados em conta na criação do modelo de classificação proposto neste trabalho.

2- Modelagem e Implementação

2.1 Tratamento dos dados

O primeiro passo do método foi fazer uma análise e preparação dos dados. Os *labels* vieram na base de dados como o nome da classe, como “GALAXY”, por exemplo. Porém para passar os *labels* como estrada da rede, precisamos reescrevê-los de alguma forma numérica. Para tanto, usamos a estratégia de *one-hot-encode*. Nela cada label é um vetor com 3 posições (pois temos 3 classes). E cada classe diferente recebe uma posição nesse, e para cada amostra temos um vetor que terá 1 na posição de sua classe, e 0 nas demais posições. Por exemplo, uma amostra do tipo “STAR” será representada como [0, 0, 1]. Para implementar

esse codificação, foi usada a biblioteca *sklearn,* e sua função *LabelBinarizer()*, que faz exatamente o descrito acima.

O passo seguinte foi analisar os dados das *features*, ou da entrada x da rede. Em uma análise rápida, verificamos que dois atributos são exatamente iguais para todas as amostras da base de dados, independente da classe. Isso foi constatado observando os atributos com desvio padrão zero dentre todas as amostras. Esse tipo de dado não traz nenhuma informação para o problema, e só leva a rede a ter mais parâmetros e dificultar sua convergência. Portanto, as duas *features* que foram identificados como constantes no *dataset* foram removidas dos dados, e mantemos apenas 15 *features* consideradas válidas. Notamos também que a ordem de grandeza das *features* é bastante variada, e como pré-processamento fizemos normalização dos dados para cada *feature*. A normalização utilizada foi apenas de subtrair a média e dividir pelo desvio padrão. Vários estudos indicam que essa prática melhora o desempenho de uma rede neural.

2.2 Separação dos dados

Para evitar *overfitting* do modelo, não devemos usar todo o conjunto de dados para treinamento. Aqui escolhemos fazer uma divisão entre dois conjuntos, um de treino com 70% das amostras e um de teste com os 30% restantes para definir a arquitetura e os parâmetros da rede. Decidimos não utilizar um conjunto de validação, pois ao final, depois de escolhidos todos os hiperparâmetros do modelo, faremos um teste de generalização utilizando *k-fold cross validation*.

2.3 Tipo de rede neural utilizada

Escolhemos utilizar o *Multi Layer Perceptron*, por ser uma abordagem muito utilizada, e comum para resolver uma gama de problemas, em especial a classificação. O *framework* utilizado no projeto foi o Keras. Nele, pode-se construir um MLP usando camadas “Dense”. Elas representam um conjunto de neurônios totalmente conectados com uma função de ativação. Nesse *framework* é muito fácil indicar os hiperparâmetros desejados, e para realizar o treinamento, basta chamar o método *fit()*, com os dados de treino. Analogamente, para testar a acurácia em dados novos podemos chamar o método *evaluate()* passando os dados de treino.

2.4 Neurônios da rede

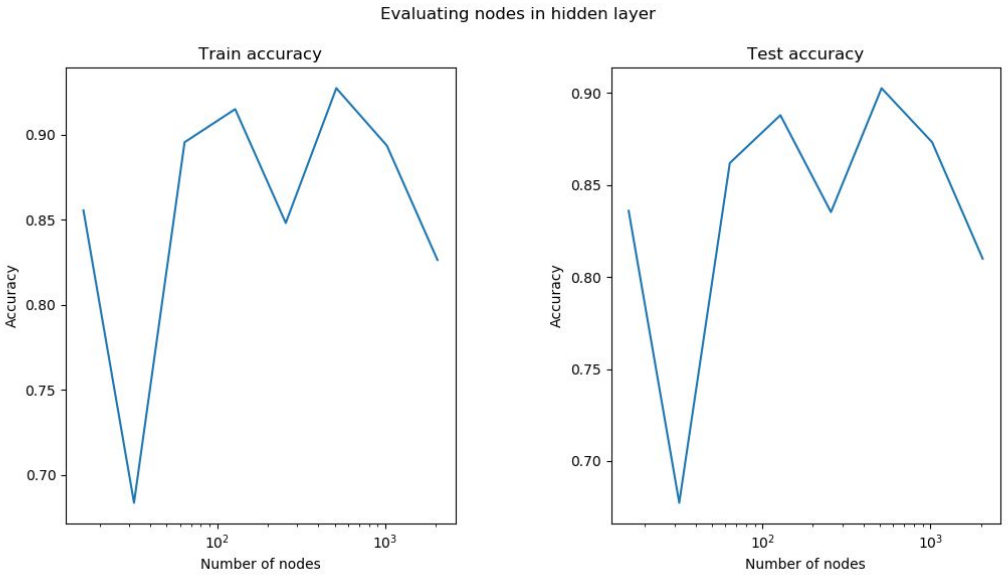
Uma rede MLP precisa ter um tamanho de entrada constante. Como nesse caso estamos utilizando apenas 15 das 17 *features*, obrigatoriamente a camada de entrada deve ter 15 neurônios. Analogamente, como temos 3 classes a serem preditas, devemos ter 3 neurônios de saída. E nesse caso, o valor para cada neurônio representa a confiança ou o *score* de uma determinada classe. Os

neurônios das demais camadas, bem como os demais hiperparâmetros da arquitetura serão discutidos a seguir.

3- Análise Experimental

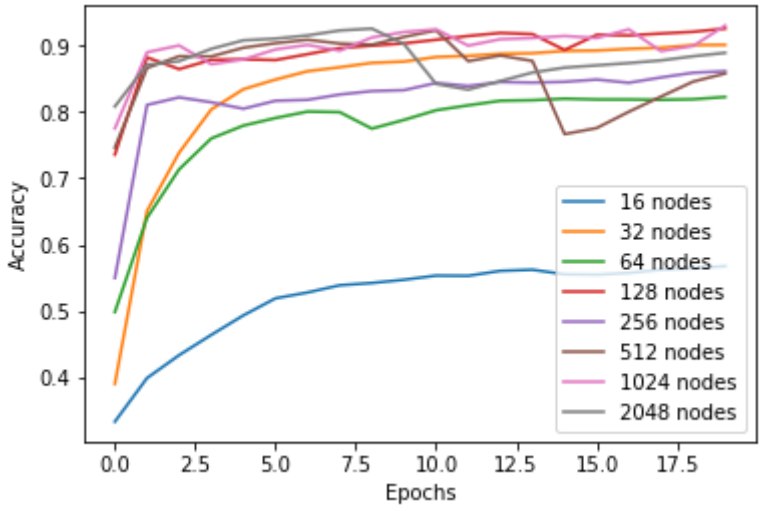
3.1 Número de neurônios na camada escondida

Inicialmente, vamos avaliar como fica a acurácia dos conjuntos de treino e de teste variando o número de nós da camada escondida da rede. Os números testados foram potências de 2, variando de 16 até 2048.



Aqui vemos resultados bem próximos entre a acurácia de treino e de teste, o que é um bom sinal de que o modelo está generalizando. Obtivemos a melhor acurácia de teste para 512 neurônios na camada escondida.

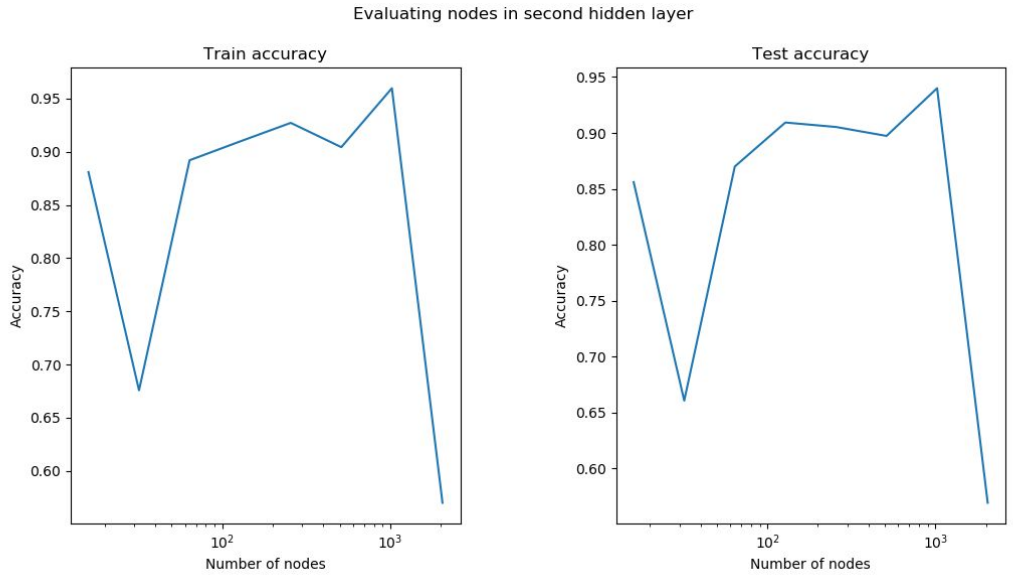
Vemos pelo gráfico seguinte que a convergência do modelo não dependeu muito do número de nós, e que aproximadamente na décima época todos já haviam convergido



3.2 Número de camadas escondidas

Aqui para o teste, fixamos o número de neurônios da primeira camada escondida em 512 e testamos novamente valores de números de neurônio para o segunda camada escondida.

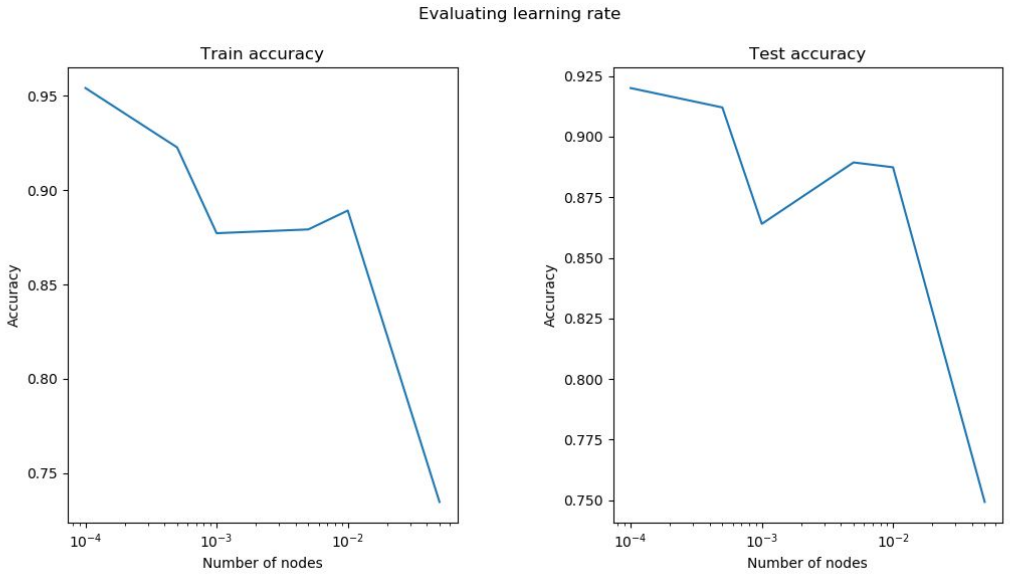
Para duas camadas escondidas:



Vemos que a melhor acurácia do conjunto de treino sobe de 0,902 para 0,940. O valor é pequeno, mas como a rede ainda é simples, o custo computacional não piora muito. Porém inserir ainda mais camadas escondidas pode aumentar demais o número de parâmetros do modelo, o que leva a um treinamento mais difícil e maiores chances de haver *overfitting*.

3.3 Variação da taxa de aprendizagem

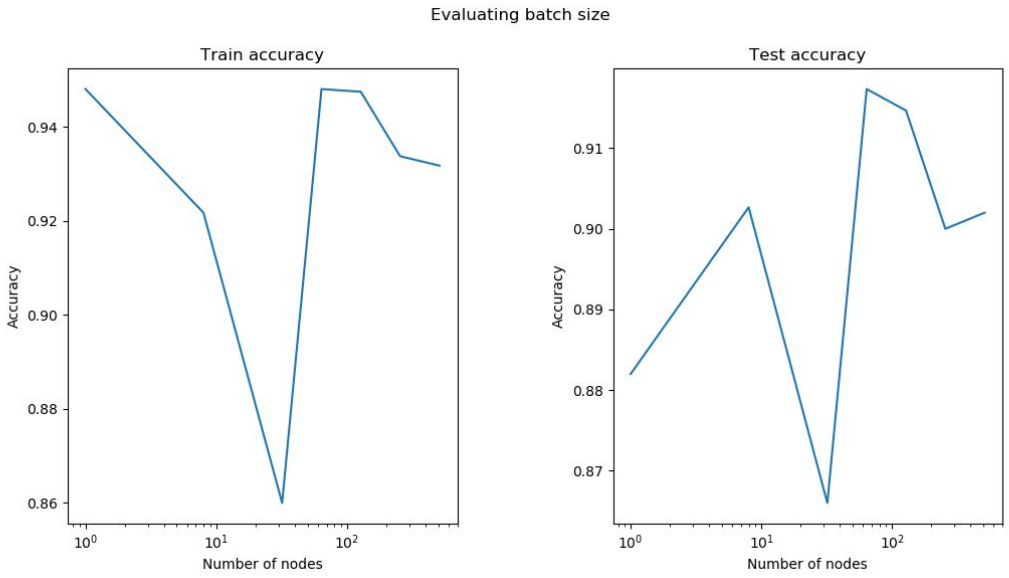
Aqui mantemos fixos os hiperparâmetros anterior, e variamos a taxa de aprendizado



Vemos que a taxa mais baixa trouxe os melhores resultados. Para taxas muito altas, a rede não converge muito bem e os resultados ficam bem baixos.
Foram testados alguns valores de taxa de aprendizagem com decaimento ao longo do tempo (parâmetro *decay*) do Keras, mas nenhum obteve acurácia melhor que a menor das reportadas acima (0,92 de acurácia no teste).

3.4 Analisando impacto do tamanho do *batch*

Não parece haver uma relação tão direta entre a acurácia e o tamanho do *batch*, como visto abaixo



Entretanto, usando um *batch* maior, a velocidade de treinamento do algoritmo sobe consideravelmente, especialmente usando uma GPU. Nesse caso, é preferível adotar *batches* maiores.

3.5 *Overfitting*

Em todos os experimentos reportados aqui as curvas de teste acompanhavam de perto as curvas de treino, ficando um pouco abaixo. Esse é um comportamento normal de treino e indica que há pouco overfitting dos dados. Porém, uma análise melhor pode ser feita na próxima seção, com resultados usando *k-fold*

3.6 Resultados finais usando *oversampling*

Por fim, replicamos a classe menos representada, que é o Quasar, em 6 vezes para termos um *dataset* mais balanceado. O modelo foi re-treinado com os hiperparâmetros selecionados nas etapas anteriores. Para verificar que a arquitetura se generaliza, foi feito um *k-fold cross validation*, com 3 partições. O processo é feito três vezes, em cada vez uma partição é separada para teste, e as outras duas para treino.

O resultado final reportado é a acurácia média entre os 3 *folds:*

Acurácia de teste final = 0.902 +- 0.006

Arquitetura final da rede:

- 2 camadas escondidas, ambas com ativação Relu
- 512 neurônios na primeira

- 1024 neurônios na segunda
- Taxa de aprendizado de 0.0001
- Tamanho do batch de 512
- Atualização de pesos usando Adam Optimizer
- 20 épocas de treinamento
- Total de parâmetros treináveis: 536579

4- Conclusão

Neste trabalho, construímos um classificar do começo ao fim, a partir de uma base de dados utilizando o *framework* Keras e a biblioteca SKLearn. Tivemos que lidar com a preparação dos dados, que é essencial para o bom funcionamento de um algoritmo de aprendizado. Com isso identificamos duas *features* que eram constantes e não adicionavam nenhuma informação ao problema.

O resultado final obtido foi um pouco menor que o encontrado em alguns testes preliminares, que foram de até 0,94. Porém o resultado final ainda é satisfatório. Talvez a pequena diferença tenha aparecido com o balanceamento do *dataset*. Em *datasets* desbalanceados, a rede tende a favorecer as classes mais presentes. Com o balanceamento, perdemos esse *bias*. Além disso, os resultados entre as 3 partições foram bastante próximos, indicando uma boa generalização da arquitetura da rede.

Dessa forma, esse trabalho foi uma boa oportunidade para praticar decisões e escolhas de parâmetros, em especial para redes neurais. Por mais que a acurácia não seja 100% confiável, vemos que a predição da rede é um indicativo muito forte de que o objeto pertence a determinada classe.

5- Bibliografia

- https://keras.io/
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelBinarizer.html
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html