UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

UNIDADE ACADÊMICA DE SERRA TALHADA

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Comparação da construção de redes neurais nas linguagens R e Python

Por

João Paulo Godê Liberal

Serra Talhada - PE,
Fevereiro de 2021

UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
UNIDADE ACADÊMICA DE SERRA TALHADA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

JOÃO PAULO GODÊ LIBERAL

Comparação da construção de redes neurais nas linguagens R e Python

Relatório Técnico apresentado ao Curso de Bacharelado em Sistemas de Informação da Unidade Acadêmica de Serra Talhada da Universidade Federal Rural de Pernambuco como requisito parcial à obtenção do grau de Bacharel. Orientador: Prof. Dr. Sérgio de Sá Leitão Paiva Júnior.

Orientador: Prof. Dr. Sérgio de Sá Leitão Paiva Júnior

Serra Talhada - PE,
Fevereiro de 2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

J62c Liberal, João Paulo Godê
Comparação da construção de redes neurais nas linguagens R e Python / João Paulo Godê Liberal. - 2021.
30 f. : il.

Orientador: Prof Dr Sergio de Sa Leitao Paiva Junior.
Inclui referências e apêndice(s).

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco, Bacharelado em Sistemas da Informação, Serra Talhada, 2021.

1. Redes Neurais Artificiais. 2. Python. 3. R. 4. Diagnóstico. I. Junior, Prof Dr Sergio de Sa Leitao Paiva, orient. II. Título

CDD 004

UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
UNIDADE ACADÊMICA DE SERRA TALHADA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

JOÃO PAULO GODÊ LIBERAL

Comparação da construção de redes neurais nas linguagens R e Python

Trabalho de Conclusão de Curso julgado adequado para obtenção do título de Bacharel em Sistemas de Informação, defendida e aprovada por unanimidade em 12/03/2021 pela banca examinadora.

Banca Examinadora:

Prof. Dr. Sérgio de Sá Leitão Paiva Júnior
Orientador
Universidade Federal Rural de Pernambuco

Prof. Dr. Paulo Mello da Silva
Universidade Federal de Pernambuco

Prof. Me. Hidelberg Oliveira Albuquerque
Universidade Federal Rural de Pernambuco

Este trabalho é dedicado aos meus pais,

pois, sempre deram tudo de si para que eu chegasse onde estou.

AGRADECIMENTOS

Primeiramente agradeço a Deus pela força, por me abençoar com uma ótima família e amigos.

Agradeço a meus pais por terem me proporcionado as oportunidades necessárias para chegar onde cheguei, além de todo o carinho, amor e atenção que eles me proporcionaram, me motivando sempre a alcançar meus sonhos. Agradeço principalmente por estarem sempre presentes e sendo um exemplo a ser seguido.

Agradeço ao meu filho por me fazer ser exemplo em minhas atribuições e responsabilidades.

Agradeço a meu amigo do cuso Vinícius de Oliveira Andrade que me ajudou muito ao longo de nossa jornada no curso, enfrentando sempre cada desafio de cabeças erguidas.

Agradeço aos meus professores, por sua paciência em sanar minhas dúvidas e, me ajudar a ver novos horizontes.

Por fim, agradeço meu orientador por ter sido e ainda ser, um exemplo, tanto na área acadêmica quanto na vida. Me guiando e orientando durante o curso, pois, sem ele eu não teria concluído este trabalho.

“Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.“
(Bíblia Sagrada, Romanos 12, 2)

RESUMO

Este relatório tem como objetivo fazer um comparativo entre Redes Neurais Artificiais (RNAs) escritas nas linguagens R e Python nos sistemas operacionais Windows 10 e Linux-Ubuntu 20.04. As principais métricas observadas foram: tempo de execução em ambas as linguagens nos diferentes sistemas, considerando o tempo para o treinamento e teste da rede; também será avaliada o nível de acurácia nos diferentes sistemas operacionais. A ferramenta utilizada para o desenvolvimento foi o pycharm, juntamente com a utilização das bibliotecas keras, tensorflow e pandas. Para a linguagem Python foi configurado um ambiente no miniconda2 e na linguagem R a execução foi por linha de comando. A base de dados utilizada pelas Redes Neurais foi retirada do Machine Learning Repository (UCI) e trata do diagnóstico de tumor cancerigeno.

Palavras-chave: Redes Neurais Artificiais. Python. R. Diagnóstico.

ABSTRACT

This report aims to make a comparison between Artificial Neural Networks (RNAs) written in R and Python languages on Windows 10 and Linux-Ubuntu 20.04 operating systems.The main metrics observed were: execution time in both languages at the different operating systems, considering the time for training and testing the network; the level of accuracy in the different operating systems will also be assessed.The tool used for the development was pycharm, together with the use of libraries keras, tensorflow and pandas. For the Python language, an environment was set up in miniconda 2 and for the R language the execution was by command line.The database used by the Neural Networks was taken from the Machine Learning Repository (UCI) and deals with the diagnosis of cancerous tumor.

Keywords: Artificial Neural Networks. Python. R. Diagnosis.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

RNAs	Redes Neurais Artificiais
UCI	Machine Learning Repository
RNN	Recurrent Neural Network
AE	Auto-encode
CNN	Convolutional neural network
DCN	Deep convolutional network
GAN	Generative adversarial network
ELM	Extreme learning machine
API	Application Programming Interface

SUMÁRIO

1 Introdução

1.1 Redes Neurais Artificiais

Uma Rede Neural Artificial (RNA) pode ser definida como um software que é projetado para comportar-se de forma similar à maneira como o cérebro executa uma tarefa específica, a sua implementação se dá através da simulação em um computador ou a utilização de componentes eletrônicos. Para que a execução do software ocorra de forma otimizada, as Redes Neurais Artificiais implementam uma grande quantidade de interligações entre as células computacionais, conhecidas como unidades de processamento ou neurônios (HAYKIN, 2001).

1.2 Elementos de uma Rede Neural Artificial

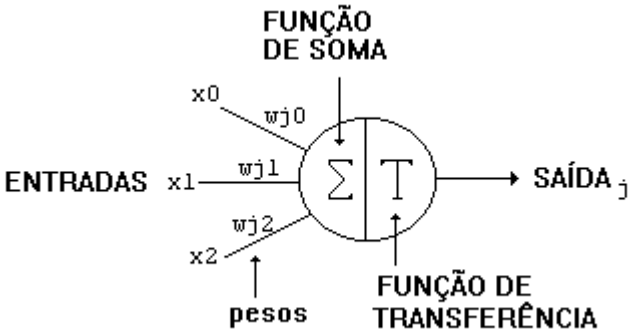
O neurônio é a parte fundamental no processamento das informações de uma RNA (HAYKIN, 2001). Ele é composto por três elementos básicos, são eles:

- As conexões de entrada ou sinapses, onde cada entrada é declarada por um peso sináptico, desta forma o sinal que é recebido na entrada da sinapse que está ligada ao neurônio é ampliado a partir do peso sináptico.
- A junção de soma que vem logo após a entrada e, que realiza a combinação dos sinais de entrada que são declarados a partir dos pesos das sinapses da unidade de processamento.
- A função de ativação, que pode possuir como características um formato sigmoidal e ser não linear, causando uma repleção na ativação de saída do neurônio.

1.3 Configurações de uma RNA

Configurar uma RNA pode se tornar uma tarefa difícil, pois a definição dos parâmetros é fundamental no desempenho da RNA. Na atualidade existem vários modelos de RNA,

Figura 1.1 – Modelo de Neurônio Artificial



Fonte: <https://cerebromente.org.br/n05/tecnologia/rna_i.htm>

apresentando variações na quantidade de neurônios que são utilizadas em cada camada, bem como a conexão entre as eles, o número de camadas ocultas e os tipos de função de ativação. A quantidade de neurônios que ficam na camada oculta em sua maioria são definidas a partir do conhecimento e experiência do usuário, podendo ser definida inicialmente em função do tamanho da camada de input da rede. Alguns erros devem ser evitados em relação à quantidade de neurônios, pois, se a quantidade for bem superior a quantidade realmente necessária, pode-se apresentar um erro de overfitting, causando a memorização dos dados de treinamento, já a situação inversa, uma quantidade de neurônios menor a quantidade ideal, causa o erro de underfitting, tornando a RNA incapaz de realizar a tarefa proposta (SILVA I; SPATTI H; FLAUZINO R, 2010).

1.4 Função de ativação

As operações que são realizadas pelos neurônios possuem como característica a linearidade na maioria dos casos, pois somente é possível auferir relações lineares entre as variáveis de entrada e de saída, no entanto, é possível que sejam modeladas relações não lineares após a saída dos dados, para que isso aconteça os dados de saída da camada devem ser processados pelas funções de ativação (Denny Ceccon, 2020).

Segundo (HAYKIN, 2001) existem diversos tipos de funções de ativação, sendo as mais conhecidas:

- Função limiar: a saída da RNA geralmente está limitada em resultados binários, atribuindo a saída 0 para um resultado negativo e, a saída 1 para um resultado positivo. Este modelo

possui a seguinte fórmula: F(u) = 1 se u>=0 || ou = 0 se u<0

- Função linear por partes: considera-se que o neurônio é representada pelo fator de amplificação no interior da região linear de operação. Este modelo possui a seguinte fórmula: $f(x) = ax$.

- Função sigmoidal: assume valores no intervalo [0, 1], tem como característica a não linearidade e o valor de sua derivada está no ápice quando o valor de x está perto de 0, esta característica é essencial em alguns problemas, como por exemplo os problemas de classificação, no entanto, o fato de não ser linear requer um custo computacional mais elevado, além de manifestar platôs quando x assume um valor extremo, sendo ele muito baixo ou muito alto. Este modelo possui a seguinte fórmula: $f(x) = 1/(1 + e^{-x})$.

- Função tangente hiperbólica: apresenta resultados nos intervalos [-1, 1] possui pontos positivos semelhantes a função sigmoidal, além de que o valor da sua derivada converge a 0 com maior agilidade. Este modelo possui a seguinte fórmula: $tanh(x) = 2/(1 + e^{-2x}) - 1$.

1.5 Tipos de Redes Neurais Artificiais

Na atualidade, existe uma ampla gama de arquiteturas de RNAs que são capazes de desempenhar atividades em específico de forma eficiente, dentre essa seara as mais utilizadas são: as convolucionais, recorrentes, generativas, profundas e autoencoders. A seguir serão listadas as principais RNAs juntamente com suas características e aplicações segundo o (Fjodor van Veen, 2016)

- **Recurrent Neural Network (RNN):** com os dados da camada precedente, a RNA recorrente tem em suas unidades de processamento escondidas o recebimento do resultado de saída da operação realizada no período temporal anterior, por apresentar tal característica, é possível que este modelo de rede possa-lhe dar com problemas que levem em consideração o histórico dos dados em um determinado ciclo de tempo, por exemplo.

- **Auto-encoder (AE):** este modelo de arquitetura possui como principal característica o fato de poder capturar as informações de entrada e as constituir em um espaço dimensional menor que o da entrada, e é por tal motivo que a camada escondida apresenta uma quantidade menor de unidades de processamento em comparação com a entrada da estrutura, a

saída no que lhe concerne possui o mesmo formato da informação de entrada. Este modelo comprime os dados utilizando menos espaço, porem armazena informações de modo que os dados possam ser reconstruídos em seu formato original, essa variação traz como benefícios a compactação dos dados para que posam ser armazenados ou transmitidos pela rede, ou para que os dados compactados sejam enviados para outra RNA que irá executar uma função em específico.

- **Convolutional neural network (CNN) ou Deep convolutional network (DCN):** este modelo de rede possui uma estrutura diferenciada e possui características peculiares, primeiramente pelo fato de possuir diversas camadas que funcionam como filtro de informações, formando padrões ao mesmo tempo, em que organizam os dados em um espaço dimensional menor, após o redimensionamento e padronização, um valor semântico e definido para cada dado e em seguida ocorre a organização por meio de uma classificação FFN. Sua aplicabilidade se torna viável em grandes bases de dados onde é necessário primeiramente fragmentar os dados e aplicar o algoritmo em uma parte para então seguir para a próxima camada apenas com os dados selecionados, desta forma o custo computacional utilizado é menor e a pesquisa se torna mais eficiente.

- **Generative adversarial network (GAN):** Este modelo de rede é composto por duas redes distintas que trabalham em conjunto, a rede generativa é responsável por criar conteúdo, recriando os dados que alimentaram a rede, já a rede contraditória é responsável por observar o conteúdo criado pela outra rede e julga-lo como aceitável ou não, caso não seja, a rede generativa é induzida a ampliar o desempenho e em consequência o custo computacional, até que os dados resultantes possam ser aceitos pela rede contraditória.

- **Extreme learning machine (ELM):** trata-se de um modelo de RNA profunda, onde as conexões das camadas se dão de forma aleatória, possui como característica o treinamento utilizando o ajuste do menor quadrado em apenas uma etapa, sua aplicabilidade é similar as FFNs.

1.6 Biblioteca Tensorflow

Segundo o HD Store (2018) o TensorFlow pode ser definido como uma biblioteca de código aberto compatível com a linguagem Python para a computação numérica, desta forma

tornando o Machine Learning mais rápido e intuitivo, facilitando todo o processo inerente a aquisição de dados, depuração e previsão de resultados futuros e modelos de treinamento.

A biblioteca utiliza o Python para disponibilizar uma API de front-end para confecção dos aplicativos e, utiliza o C ++ de alto desempenho para execução de tais aplicativos, podendo proceder com o treinamento e execução de RNA profundas em diversas situações segundo o HD Store (2018), dentre as quais podemos destacar: reconhecimento de imagens, redes neurais recorrentes, incorporação de palavras, processamento de linguagem natural e simulações baseadas em equações diferenciais parciais.

Segundo o HD Store (2018) a biblioteca ainda possibilita que os desenvolvedores confeccionem gráficos de fluxo de dados, descrição de como os dados se movem no gráfico ou uma série de nós de processamento. Cada nó no gráfico representa uma operação matemática e cada conexão ou borda entre os nós é uma matriz de dados, ou um tensor multidimensional.

1.7 Biblioteca Keras

A biblioteca Keras é direcionada às redes neurais de alto-nível, sua escrita é na linguagem de programação Python e, executada na plataforma de aprendizado de máquina TensorFlow ou Theano. Com a biblioteca Keras é possível a substituição de uma RNA por outra. Seu intuito de desenvolvimento foi facilitar a experimentação rápida, não sendo necessário o domínio de cada um dos backgrounds, de maneira rápida e eficiente (Djames Suhanko, 2016).

Ainda segundo o autor, é possível aplicar a biblioteca nos casos a seguir:

- Rodar na CPU ou GPU
- Uma prototipagem rápida e fácil
- Ter suporte a esquemas de conectividade arbitrária (inclusive treinos de N para N)
- Ter suporte a redes convolucionais e recorrentes, inclusive com a combinação de ambas

2 Metodologia

2.1 Materiais Utilizados

Recursos utilizados para construir as redes neurais em Python e R:

- Sistema operacional Windows 10 64 bits
- Sistema operacional Linux versão Ubuntu 64 bits (instalado como subsistema do windows)
- Memória RAM: 4096 Mb
- Memória ROM: 128 Gb
- Processador: Intel Celeron 1.83GHz

No sistema operacional Windows, foi feita a instalação da IDE PyCharm na versão 2020.3.2. Com o ambiente de programação pronto para desenvolvimento e para execução do código, foram utilizadas as seguintes bibliotecas, softwares e versão do Python:

- Pandas 1.1.0 (Biblioteca)
- Keras 2.2.4 (Biblioteca)
- Tensorflow 1.5.0 (Biblioteca)
- Miniconda 2 (Software)
- Python 2.7 (Versão do Python)

2.1.1 Configuração do Ambiente Ubuntu

Para realizar o download e configuração do ambiente Ubuntu, foi necessário que na loja (microsoft store) do Windows, fosse baixada a aplicação Ubuntu na versão 20.04. Após a habilitação nas configurações de ativar ou desativar recursos do windows e ativação da opção

de subsistema do windows para linux, foi realizada a instalação da aplicação Ubuntu 20.04, logo após sua instalação e abertura do terminal linux, foram executados os comandos de update, upgrade dos pacotes do sistema através dos códigos "sudo apt-get update"e "sudo apt-get upgrade", e então pôde-se proceder com a instalação do pip na versão 20.0.2 através do comando "sudo apt install python3-pip".

Os códigos do presente relatório foram baseados em códigos disponíveis na internet, sendo procedidas modificações para adaptar ao referido estudo. Endereços dos códigos utilizados:

Stylistic differences between R and Python in modelling data through neural networks

R vs Python: Image Classification with Keras

2.1.2 Execução do Código Python

Para execução do código Python no Windows foi utilizado o software miniconda 2 , executando a versão 2.7 do Python, através da execução do arquivo Spec.txt foi criado o ambiente py36, onde foram instaladas todas as bibliotecas nas versões citadas anteriormente e o código em python pôde ser executado pelo terminal do miniconda.

Para a execução do código Python no Ubuntu foram instaladas todas as bibliotecas nas versões citadas anteriormente no Python e o código foi executado pelo terminal do Ubuntu.

2.1.3 Execução do Script R

Para execução do scriptR no Windows foi instalada a linguagem R e, em seguida o RStudio, foi realizada a instalação do keras na versão 2.4.3 no R e, em seguida executado o scriptR pela IDE.

Para execução do scriptR no Ubuntu foi instalada a linguagem R e, na linguagem instalada a biblioteca keras, realizada a instalação, o terminal foi reiniciado, o console do R foi aberto e o script foi executado gerando as saídas. Todos os comandos de instalação e execução foram realizados pelo terminal.

2.1.4 Configuração da Rede Neural

A rede neural foi montada com três camadas: a camada de input com 30 neurônios, a camada oculta com 16 neurônios e a camada de saída com 1 neurônio. A camada de input foi configurada com a função de ativação "relu"com inicialização randomica dos pesos das sinapses. Entre cada camada foi configurado o comando de droupout, que zera aleatoriamente 20% das sinapses, para evitar overfiting no treinamento da rede. A camada de saída foi configurada com a função de ativação "sigmoid"e valores acima de 0,5 são considerados como positivo para câncer.

- layer_dense(units = 16, activation = 'relu',kernel_initializer = 'random_uniform' ,input_shape = 30)
- layer_dropout(rate = 0.2)
- layer_dense(units = 16, activation = 'relu')
- layer_dropout(rate = 0.2)
- layer_dense(units = 1, activation = 'sigmoid')

Para a realização do experimento foram realizadas 30 repetições com respectivamente 100, 500 e 1000 épocas de treinamento nos sistemas operacionais Linux-Ubuntu e Windows. Foram obtidos o tempo de execução para cada conjunto de épocas de treinamento e a acurácia de cada rede. A acurácia é a relação entre os acertos (verdadeiros positivos e verdadeiros negativos) e a quantidade total de amostras. Os valores obtidos foram plotados no gráfico BoxPlot para uma melhor representação da dispersão dos valores em torno da mediana.

2.1.5 Base de Dados

A base de dados foi retirada da UCI (Machine Learning Repository) um banco de dados de repositórios públicos para teste de algoritmos voltados para aprendizado de máquina. Os dados são obtidos a partir de uma base de dados de imagens digitalizadas referentes a aspiração de massa mamária a partir de uma agulha.

A base de dados de câncer de seio, usada neste trabalho, é um banco de dados com 569 instâncias e 32 atributos, sendo um id do registro, um atributo de diagnóstico (M=Maligno e B=Benigno), os outros 30 atributos estão relacionados ao tumor (tamanho, aspecto, etc.)

Para se obter um melhor índice de precisão preditiva, a base de dados utilizou um plano de separação no espaço 3-D de pior área, pior suavidade e Textura Média, com a utilização da validação cruzada em 10 vezes, a porcentagem de acerto ficou em 97,5%. Foi possível executar esse plano a partir da Árvore de métodos multisuperfície (MSM-T), método de classificação que usa programação linear para confecção da árvore de decisão.

Base de dados e informações disponíveis no site: Breast Cancer WisconsinDiagnostic

2.1.6 Separação Entre Dados de Treinamento e Teste

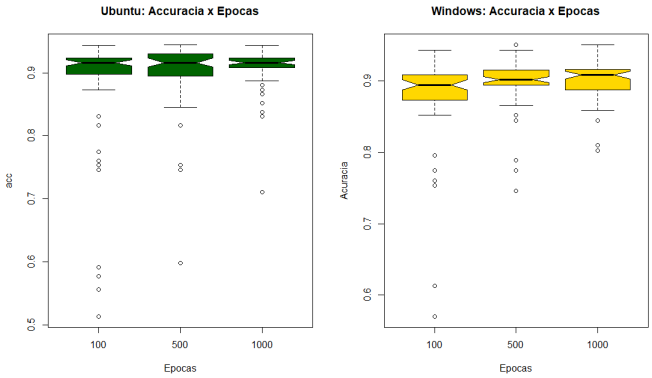
A divisão da base de dados procedeu da seguinte forma, 25% da base serviu para testes e os demais 75% para treinamento, os dados foram escolhidos de forma aleatória a cada execução. Os dados pertencentes a parcela dos 25% foram armazenados com sua classificação e, os já classificados que correspondem a parcela de 75% foram executados pela rede neural, sendo analisados um a um, onde, as sinapses também foram treinadas. Após o treinamento, foi feita a comparação, onde, o modelo gerado atuou na parcela de 25% de teste que gerou um novo resultado. Para chegar ao nível de acurácia, foi feita a comparação entre os resultados obtidos pela rede neural e os que foram classificados manualmente.

3 Resultados e discussões

3.1 Acurácia x Épocas nos sistemas operacionais

No comparativo entre o nível de acurácia x épocas nos sistemas operacionais Linux-Ubuntu e Windows, observa-se que os valores obtidos são similares, no entanto, o sistema Ubuntu se sobressai em comparação com o Windows, é possível notar que a acurácia se aproxima mais de 1 e que os valores obtidos estão mais próximos da mediana com 100 e 1000 épocas, ou seja, com uma taxa de dispersão menor, em contrapartida, o sistema ubuntu possui uma quantidade maior de outliers, com 100 e 1000 épocas. Com 500 épocas, o Windows apresenta uma menor dispersão dos valores obtidos.

Figura 3.1 – Acurácia X Épocas.



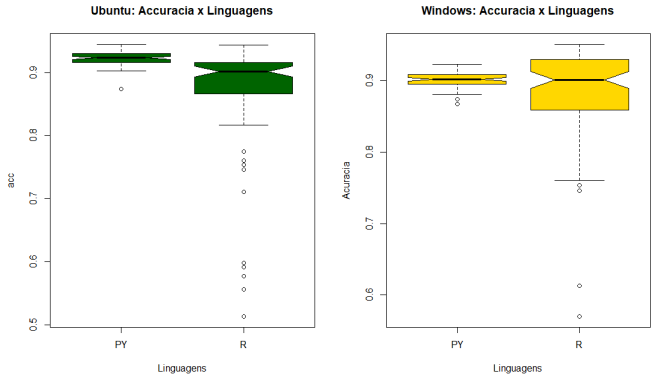
Fonte: Elaborado pelo autor (2021)

3.2 Acurácia x Épocas nas linguagens

No comparativo entre a acurácia e as linguagens R e Python nos sistemas operacionais Linux-Ubuntu e Windows, é possível notar que a linguagem Python teve uma acurácia maior no sistema Linux-Ubuntu e, em ambos os gráficos, a taxa de dispersão foi baixa. Já na linguagem R de programação a taxa de dispersão foi bem mais acentuada em comparação com o Python, as medianas foram bem próximas em ambos os gráficos, com destaque para o sistema operacional Linux-Ubuntu que apresentou uma taxa menor de dispersão, no entanto, com um número maior

de outliers.

Figura 3.2 – Acurácia X Linguagens.

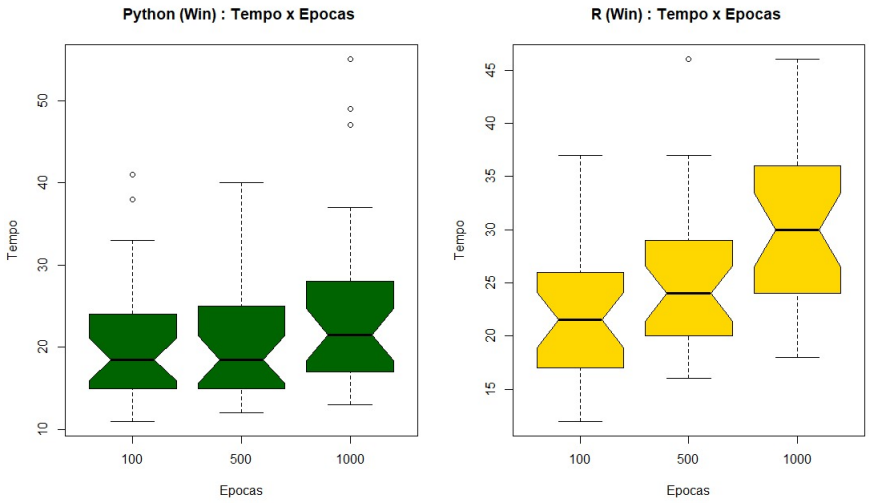


Fonte: Elaborado pelo autor (2021)

3.3 Tempo de execução x Épocas - Windows

No comparativo entre o tempo de execução x quantidade de épocas no sistema operacional Windows, é possível observar que na linguagem Python o tempo de execução teve uma elevação moderada de acordo com o aumento do número de épocas, elevação esta que se deu de forma mais acentuada na linguagem R. Em ambas as linguagens a dispersão se manteve com certa constância, independentemente do número de épocas, com destaque para a linguagem Python que apresenta uma menor dispersão.

Figura 3.3 – Tempo Execução X Épocas (Windows)

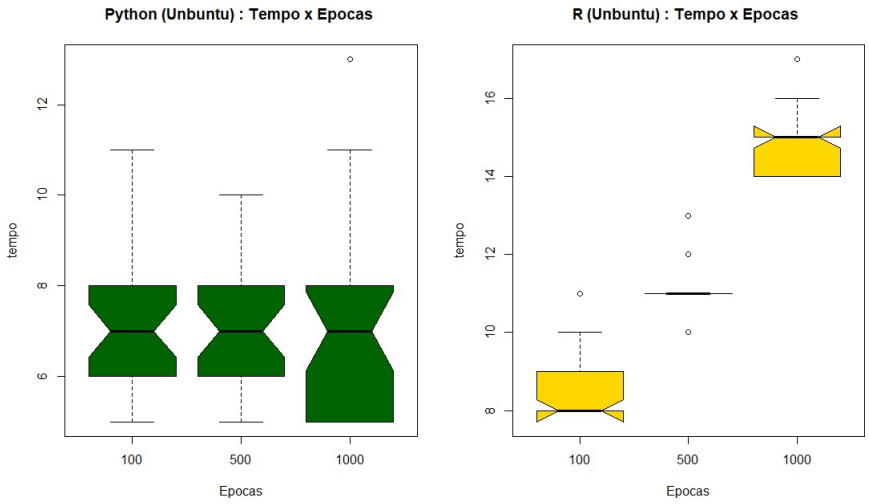


Fonte: Elaborado pelo autor (2021)

3.4 Tempo de execução x Épocas - Ubuntu

No comparativo entre o tempo de execução x quantidade de épocas no sistema operacional Linux-Ubuntu, é possível observar que na linguagem Python o tempo de execução se manteve estático com 100 e 500 épocas, já com 1000 épocas o terceiro quartil se manteve estático, no entanto, o primeiro quartil apresentou uma diminuição, apesar do aumento do número de épocas, isto fez com que a dispersão se tornasse maior entre os dados resultantes. No sistema operacional Windows o tempo de execução foi superior em relação ao Linux em todos os cenários, com destaque para o cenário de 500 épocas, onde o todos os valores permaneceram próximos à mediana, com exceção dos outliers, em 100 épocas o terceiro quartil destoou mais que o primeiro em relação à mediana, já com 1000 épocas, o cenário se inverteu e o primeiro quartil destoou mais que o terceiro em relação à mediana.

Figura 3.4 – Tempo Execução X Épocas (Ubuntu)



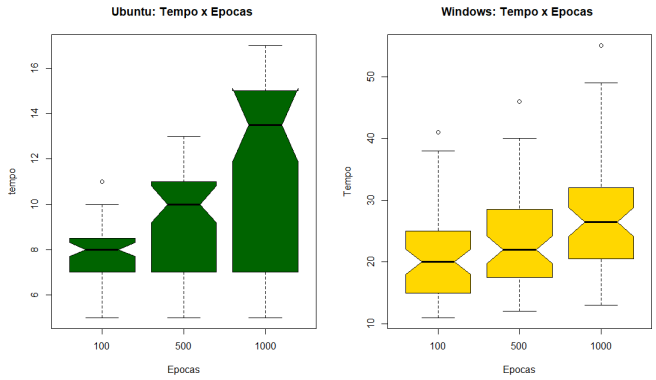
Fonte: Elaborado pelo autor (2021)

3.5 Tempo de execução x Épocas Linguagens

No comparativo da média de ambas as linguagens, considerando o tempo de execução x quantidade de épocas nos sistemas operacionais Linux-Ubuntu e Windows, é possível observar que o Ubuntu teve um desempenho mais satisfatório em relação à unidade tempo e dispersão em relação à mediana, ainda é possível notar que no sistema Linux ouve uma maior variabilidade em relação ao aumento proporcional da dispersão com aumento do número de épocas, métrica

esta que se manteve com maior constância no sistema Windows.

Figura 3.5 – Tempo Execução X Épocas.

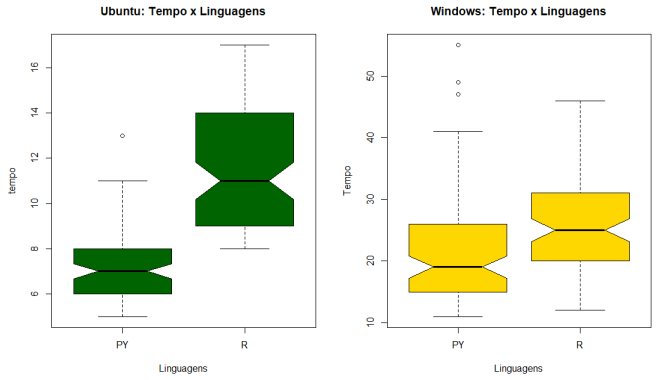


Fonte: Elaborado pelo autor (2021)

3.6 Tempo de execução x Linguagens

No comparativo da média das épocas nas linguagens Python e R nos sistemas operacionais Linux-Ubuntu e Windows, pode-se inferir que no sistema Linux o Python apresentou um tempo de execução mais otimizado em relação à linguagem R, com valores mais proximos da mediana, gerando assim uma menor dispersão. No sistema operacional Windows o tempo de execução em ambas as linguagens foi maior em comparação com o sistema linux, ainda sendo possível observar que a taxa de dispersão foi bem similar em ambas as linguagens e que a diferença proporcional em tempo de execução foi menor no sistema Windows.

Figura 3.6 – Linguagem X Tempo de Execução (Linux e Windows)



Fonte: Elaborado pelo autor (2021)

Para a execução de ambos os softwares, houveram algumas dificuldades. A versão da biblioteca tensorflow 1.15 é incompatível com o Python 3.6, versão mínima que o software

Anaconda 3 tem suporte. O site oficial do Anaconda não disponibiliza as releases anteriores da ferramenta, mas existe um software similar, o Miniconda nas versões 2 e 3, o miniconda 2 utiliza o Python 2.7 e a partir desta ferramenta foi possível proceder com a instalação de todas as bibliotecas e execução do código.

Para a execução do script R, o windows apresentava erros relacionados a biblioteca tensorflow e keras e uma de suas DLLs, com o seguinte código de erro: "caught illegal operation address 0x7f8455cf78c0, cause 'illegal operand'", este erro está relacionado a incompatibilidade entre a versão mais atualizada da biblioteca tensorflow e o processador mais antigo da máquina utilizada para executar o software. Foi necessário então realizar o mesmo procedimento em uma máquina com versão do processador mais atualizada.

4 Conclusão

Após a aplicação do experimento, pôde-se observar que a linguagem Python apresentou um desempenho superior na maioria dos testes realizados, principalmente quando o sistema operacional utilizado era o Linux-Ubuntu, sistema qual a linguagem pôde ofertar uma acurácia mais proxima de 1 nos experimentos, um tempo de execução mais otimizado e uma dispersão menor dos valores. No sistema Windows a linguagem Python se manteve como mais satisfatória, mas com algumas peculiaridades, exemplo: na figura 2, o Windows manteve equilibrada a mediana entre as duas linguagens, enquanto que no sistema Linux a mediana da acurácia apresentava um melhor rendimento na linguagem Python, o Windows também aumentou a dispersão da linguagem R em comparação com o Python mas, ao mesmo tempo, em que a quantidade de outliers da lingagem R diminuiu ainda comparando com a linguagem Python. Uma outra observação bastante pertinente é que em ambos os sistemas operacionais a linguagem Python teve um mínimo de elevação do seu tempo de execução mesmo com um aumento expressivo do número de épocas, cenário este bastante viável para um Rede Neural Artificial que precise passar por uma grande quantidade de épocas para a sua aprendizagem.

REFERÊNCIAS BIBLIOGRÁFICAS

Denny Ceccon. *Funções de ativação: definição, características, e quando usar cada uma*. 2020. Disponível em: <https://iaexpert.academy/2020/05/25/funcoes-de-ativacao-definicao-caracteristicas-e-quando-usar-cada-uma/>. Acesso em: 04 janeiro 2021.

Djames Suhanko. *Rede Neural com keras*. 2016. Disponível em: <https://www.dobitaobyte.com.br/rede-neural-com-keras-mais-anotacoes/#O_que_e_Keras>. Acesso em: 12 janeiro 2021.

Fjodor van Veen. *The Neural Network Zoo*. 2016. Disponível em: <https://www.asimovinstitute.org/neural-network-zoo/>. Acesso em: 02 janeiro 2021.

HAYKIN, S. *Redes Neurais, Princípios e Prática*. [S.l.]: BOOKMAN, 2001. v. 2ª ed.

HD Store. *O que é o TensorFlow? A biblioteca de machine learning explicada*. 2018. Disponível em: <https://blog.hdstore.com.br/o-que-e-o-tensorflow/>. Acesso em: 28 fevereiro 2021.

SILVA I, N.; SPATTI H, D.; FLAUZINO R, A. *Redes Neurais Artificiais: para engenharia e ciências aplicadas*. [S.l.]: Artliber, 2010. v. 1ª ed.

APÊNDICE A – CÓDIGO PYTHON

```
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense,Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from datetime import datetime as dt
import gc

previsores = pd.read_csv("entradas-breast.csv",sep=",")
classe = pd.read_csv("saidas-breast.csv",sep=",")
previsores_treinamento,previsores_teste,classe_treinamento,
   classe_teste = train_test_split(previsores,classe,test_size=0.25)

arq=open("out.csv","w")
arq.write("Linguagem;Epocas;Tempo;Acuracia\n")
arq.close()
print("aguarde a finalizacao")
for tt in range(30):
  gc.collect()
  epocas=100
  bts=10
  df_out=[]
  ln=""
  for t in range(3):
    classificador = Sequential()
    classificador.add(Dense(units = 16, activation = 'relu',
              kernel_initializer = 'random_uniform', input_dim = 30))
    classificador.add(Dropout(0.2))
    classificador.add(Dense(units = 16, activation = 'relu',
```

```
30                kernel_initializer = 'random_uniform'))
31      classificador.add(Dropout(0.2))
32      classificador.add(Dense(units = 1, activation = 'sigmoid'))
33      start = dt.now()
34      classificador.compile(optimizer = "adam", loss = '
           binary_crossentropy',metrics = ['binary_accuracy'])
35      hist=classificador.fit(previsores_treinamento, classe_treinamento
           , batch_size = bts,epochs = epocas,verbose=0)
36      cros_v=10
37      resultados=0
38      resultados = classificador.evaluate(previsores_teste,
           classe_teste)
39      out= {'epocas':epocas,'delta_loss': (hist.history['loss'][0]-hist
           .history['loss'][epocas - 1]),
40        'time': (dt.now() - start).seconds,"Acuracia": resultados[1]}
41      df_out.append(out)
42      if (t==0):
43        epocas=500
44        bts=50
45      elif(t==1):
46        epocas=1000
47        bts=100
48    i=1
49    for l in df_out:
50      ln=ln+("PY;%i;%f;%f\n"%(l['epocas'],l['time'],l['Acuracia']))
51    arq=open("out.csv","a")
52    arq.write(ln)
53    arq.close()
```

APÊNDICE B – Código R

```
library(keras)
dt_in <- read.csv("entradas-breast.csv", sep=",")
dt_out <- read.csv("saidas-breast.csv", sep=",")
n <- nrow(dt_in)
x_train=as.matrix(head(dt_in,round(n*0.75)))
y_train <- as.matrix(head(dt_out,round(n*0.75)))
x_test <- as.matrix(tail(dt_in,round(n*0.25)))
y_test <- as.matrix(tail(dt_out,round(n*0.25)))

df_out=data.frame()
for (tt in 1:2)
{
  epocas=100
  bts=10
  print(paste("iteracao ",tt))
  for (t in 1:3)
  {
    print(paste("Epocas ",epocas))
    model <- keras_model_sequential()
    model %>%
      layer_dense(units = 16, activation = 'relu',kernel_initializer
         = 'random_uniform' ,input_shape = 30) %>%
      layer_dropout(rate = 0.2) %>%
      layer_dense(units = 16, activation = 'relu') %>%
      layer_dropout(rate = 0.2) %>%
      layer_dense(units = 1, activation = 'sigmoid')
    start <- Sys.time()
    model %>% compile(
        loss = 'binary_crossentropy',
        optimizer = 'adam',
```

```
30          metrics = c('binary_accuracy')
31        )
32      hist=model %>% fit(x_train, y_train, epochs = epocas, batch_size
          = bts,verbose=0)
33      resultados = model %>% evaluate(x_test, y_test,verbose=0)
34      out <-  data.frame(epocas = epocas,
35                         'delta_loss'=(hist$metrics$loss[1]-
                              hist$metrics$loss[epocas]),
36                         time = as.integer(Sys.time()) - as.integer(
                              start),
37                        acuracia=resultados[2])
38      df_out=rbind(df_out,out)
39      if (t==1)
40      {
41        epocas=500
42        bts=50
43      }else if (t==2)
44      {
45        epocas=1000
46        bts=100
47      }
48
49    }
50
51  }
52  write.csv(df_out,"R_out.csv")
```