Avaliação de Deep Learning para Predição de Mensagens Processadas pela Plataforma de Integração Guaraná

Matheus H. Rehbein[1], **Rafael Z. Frantz**[1]

[1]Departamento de Ciências Exatas e Engenharias
Universidade Regional do Noroeste do Estado do Rio Grande do Sul
98700-000 – Ijuí – RS – Brasil

matheus.rehbein@sou.unijui.edu.br, rzfrantz@unijui.edu.br

Abstract. *Companies have many applications in their software ecosystems that need to be integrated. Integration frameworks allow integrating with efficiency those applications, however there are parameterization that must be defined manually, like the number of available threads. This paper presents an experimental study that created and evaluated deep learning models for realizing the prediction of the number of messages that will be processed by using a determinate number of threads and the message rate as input in the model, to allow the parametrization of the number of threads automatically and in real time.*

Resumo. *Empresas possuem diversas aplicações em seus ecossistemas de softwares que precisam ser integradas. Plataformas de integração permitem uma eficiência para realizar a integração, entretanto, algumas configurações devem ser definidas de forma manual, como o número de threads disponíveis. Este artigo apresenta um estudo experimental que criou e avaliou modelos de deep learning para realizar a predição do número de mensagens que serão processadas a partir de um determinado número de threads e da taxa de mensagens, para permitir uma definição de forma automática e em tempo de execução do número de threads nas plataformas de integração.*

1. Introdução

Para dar suporte aos seus processos de negócio, geralmente as empresas utilizam diversas aplicações em seu ecossistema de software [Manikas 2016]. Para agilizar as rotinas diárias da empresa, é necessário que as aplicações operem de forma integrada [Hohpe and Woolf 2004], portanto é necessário integrá-las. Atualmente, existem diversas plataformas que visam auxiliar o desenvolvimento e manutenção de integração entre diferentes aplicações, abstraindo as principais atividades envolvidas na integração. Destacam-se Apache Camel [Ibsen and Anstey 2018], Guaraná [Frantz et al. 2016], Mule ESB [Dossot et al. 2014], e Spring Integration [Fisher et al. 2012]. Essas são plataformas desenvolvidas a partir do paradigma de mensageria, que visa realizar a troca de informações entre as aplicações por meio de mensagens [Frantz et al. 2020]. Algumas configurações são necessárias para obter maior desempenho durante a integração, como a quantidade de *threads*.

Quando realizado uma superestimação ou subestimação da quantidade de *threads* a aplicação pode ficar lenta ou causar uma sobrecarga no servidor. A partir dessas questões, a configuração das plataformas de integração se torna fundamental, e usar uma

configuração de forma automática pela própria plataforma de integração é necessário, para isso modelos matemáticos específicos para cada processo de integração se constituem em uma boa alternativa de solução ao processo de configuração.

O campo de *machine learning* é constituído por modelos capazes de obter conhecimento a partir de dados de exemplos, e então realizar predições [Géron 2017]. Sendo um dos principais modelos o modelo de *deep learning*, que é composto por múltiplas camadas de processamento capazes de aprender com diversos níveis de abstração. Para realizar uma boa definição da quantidade de *threads* nas plataformas de integração, o modelo de *deep learning* pode ser utilizados, a partir de predições da quantidade de mensagens que serão processadas. O conhecimento inferido no modelo pode ser obtido por meio de logs e de experimentos realizados anteriormente.

Neste artigo, será utilizado um conjunto de dados (*dataset*) extraído da plataforma Guaraná, que foi desenvolvida pelo grupo de pesquisa, para criar modelos capazes de predizer a quantidade de mensagens que serão processadas. Além da disso, será realizada a comparação entre os modelos de *deep learning*, com diferentes configurações para encontrar qual configuração teve maior assertividade. O restante do artigo está organizado da seguinte forma: na Seção 2 será apresentado um conhecimento prévio sobre os assuntos, seguida pela Seção 3 que apresenta os trabalhos relacionados; a Seção 4 apresentará a metodologia utilizada; o objeto de estudo será apresentado na Seção 5, e os experimentos na Seção 6; por fim, será realizada as conclusões do artigo na Seção 7.

2. Background

Nesta Seção serão apresentados os conceitos chaves para entendimento do trabalho. Será descrito o desenvolvimento da técnica de *deep learning*, seguido pela evolução histórica. Em seguida, é apresentado o Guaraná e suas principais definições.

2.1. Deep Learning

Rede Neural é uma técnica de *machine learning* composta por uma série de elementos (chamados de neurônios) caracterizados como elementos simples e possui como principal característica a forte conexão entre os neurônios; essa ideia vem da forma como o cérebro humano é estruturado. O neurônio é constituído por: sinapses, entradas, pesos, função de ativação e saída. Sendo que as sinapses são as ligações lógicas entre os neurônios. Haverão diversas sinapses em cada neurônio; as entradas consistem nos valores que serão inferidos no neurônio; o peso é um valor gerado pelo modelo, que multiplicado pela entrada será utilizado para auxiliar na excitação/inibição do neurônio. O peso é iniciado de forma randômica e posteriormente será ajustado no treinamento. O peso não é atribuído diretamente no neurônio, mas sim em cada sinapse; uma função de ativação é utilizada para realizar o processamento interno no neurônio, para representar se o neurônio será excitado ou inibido; por fim a saída é o resultado da função de ativação [Géron 2017].

O *perceptron* é um exemplo de uma arquitetura de rede neural, na qual suas entradas e saídas são números e cada entrada é associada com um peso [Géron 2017]. Uma rede *perceptron* é composta por apenas uma camada onde todos os neurônios estão internamente conectados. Muitos problemas foram encontrados na rede *perceptron*, e foram solucionados com um conjunto de redes *perceptron*, chamada de *multi-layer perceptron*. Uma rede *multi-layer perceptron* consiste em diversas redes *perceptron* interligadas, ou

seja, com diversas camadas. Sendo necessário uma camada de entrada, uma ou mais camadas invisíveis, e uma camada de saída, como representado na Figura 1. É considerado um modelo de *deep learning* quando uma rede possui duas ou mais camadas invisíveis.

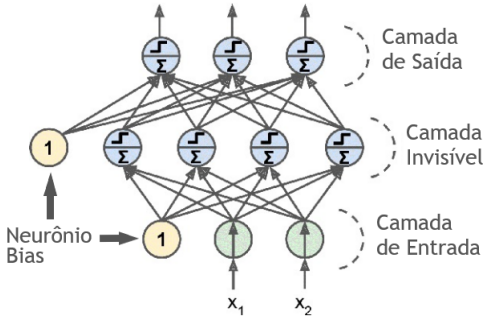


Figura 1. Multi-Layer Perceptron. Adaptado de: [Géron 2017]

Cada camada de um modelo de *deep learning* é constituída por um otimizador, quantidade de neurônios, um neurônio *bias* (que está sempre inferindo um valor constante, como mostrado na Figura 1), e uma função de ativação.

Otimizador é uma função aplicada no treinamento do modelo que visa diminuir as perdas. Ou seja, busca diminuir o erro do valor predito no momento atual do treinamento e do valor real. O otimizador é um dos principais elementos de uma rede neural, tendo em vista que ele é o responsável por ajustar os pesos em cada neurônio. Um dos principais otimizadores é o Adam [Kingma and Ba 2014], que se destaca devido a uma série de fatores, como: eficiência computacional, é adequado para trabalhar com grande quantidade de dados, possui baixa quantidade de consumo de memória, velocidade.

A função de ativação é uma função responsável pela excitação/inibição dos neurônios na camada. A ativação significa que o neurônio foi ativado. Nela o objetivo é calcular os pesos e então verificar se o neurônio será excitado ou não. Essas funções podem variar a saída; algumas funções, como *sigmoid*, possuem valores de saída entre 0 e 1, outras variam de 0 até infinito, e também saídas binárias (0 ou 1).

2.2. Guaraná

Guaraná [Frantz et al. 2016] é uma plataforma desenvolvida com o objetivo de auxiliar a realização de integração de aplicações. Ele realiza a integração a partir da troca de mensagens entre as aplicações. O Guaraná proporciona uma DSL, um SDK, e um motor.

A DSL permite modelar soluções de integração com um alto nível de abstração. Seus construtores estão apresentados na Figura 2. São eles: aplicação, tarefa, slot, processo de integração, porta de entrada, porta de saída, porta de solicitação, e porta de resposta. Uma aplicação é o software que será integrado; a Tarefa é o elemento que implementa uma atividade atômica que é executada sob as mensagens; o Slot é responsável por interligar as tarefas, permitindo o processamento assíncrono do processo de integração; o Processo de Integração implementa um *workflow* de Tarefas dessincronizadas por Slot. Esse processo se comunica com os meios externos a partir de portas; a Porta de Entrada é um canal de comunicação que permite ao processo de integração obter informações de uma aplicação, enquanto que a Porta de Saída é responsável por enviar informações às aplicações; a Porta de Solicitação é utilizada por uma tarefa para enviar solicitações e

receber respostas de/para uma aplicação; por fim, a Porta de Resposta é usada por uma tarefa para receber solicitações e responder de/para uma aplicação [Frantz 2012].

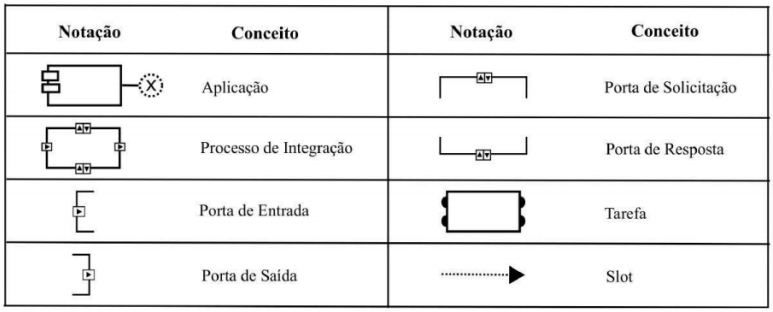


Figura 2. Descrição dos conceitos da DSL. Fonte: [Frantz et al. 2015]

O SDK consiste em um conjunto de classes que permitem transformar o modelo conceitual em código executável. O motor de execução é a parte responsável por executar o código resultante da transformação do modelo conceitual. Um elemento fundamental durante a execução é o número de *threads*. Engenheiros de Softwares necessitam configurar o motor de execução, sendo uma das configurações a quantidade de *threads*. No Guaraná, cada tarefa é executada de forma sequencial, ou seja, para executar a tarefa T2 da mensagem X é necessário que a T1 da mesma mensagem ter sido executada. Devido a sua política de tarefas, é possível que diversas mensagens sejam executadas paralelamente, pois cada *thread* pode executar a mesma tarefa para diferentes mensagens [Frantz 2012].

3. Trabalhos Relacionados

Em [Nemirovsky et al. 2017], os pesquisadores propuseram um escalonador de CPU heterogêneo, que visa maximizar o *throughput* a partir do uso de redes neurais artificiais, sendo que obtiveram uma melhoria de 25% a 31% em relação aos escalonadores heterogêneos convencionais de CPU e memória intensiva. Embora o trabalho dos pesquisadores tenha sido de otimização com técnica de inteligência artificial, ele se diferencia deste artigo pois este está dentro do contexto de integração de aplicações.

Uma estratégia de programação que usa um Algoritmos Genéticos para escalonar tarefas heterogêneas em processadores heterogêneos para minimizar o tempo total de execução foi proposta por [Page and Naughton 2005]. Embora seja um problema de otimização e tenham resolvido ele com inteligência artificial, não foi utilizado no contexto de integração de aplicações e também não foi utilizada a técnica de *deep learing*.

No trabalho [Beer and Hassan 2018] os pesquisadores utilizaram redes neurais artificiais na prevenção de ataques SOA em *web services* RESTful. Embora o trabalho dos pesquisadores seja no campo de integração de aplicações e tenha utilizado inteligência artificial, ele se difere deste pois foi no contexto de segurança e não de otimização.

4. Metodologia

A pesquisa realizada neste artigo é considerada aplicada, tendo em vista que busca gerar conhecimento para aplicações práticas voltados a resoluções de problemas específicos. Neste artigo utilizou-se o método científico indutivo com o objetivo de generalizar a resposta de qual técnica possui maior assertividade a partir de experimentos sob um conjunto

de dados específico. Essa generalização é verificada pois não foram realizados experimentos com todas as combinações possíveis. Seus objetivos de estudo são explicativos, pois busca realizar o entendimento dos resultados encontrados. Os procedimentos são experimentais sob parâmetros controlados, juntamente com uma abordagem quantitativa.

Para realizar a implementação dos modelos utilizou-se a linguagem *Python*, juntamente com a biblioteca *TensorFlow* [Abadi et al. 2016]. O desenvolvimento foi separado da seguinte forma: i) análise e pré-processamento do *dataset*; ii) implementação do problema com a biblioteca *TensorFlow*; iii) execução e coleta dos resultados.

5. Objeto de Estudo

A Figura 3 apresenta a solução de um problema encontrado em um café, que é utilizado como *benchmark* no contexto de integração de aplicações. O processo de integração da solução Café inicia com os pedidos inseridos na integração pela porta de entrada denominada *Orders* e posteriormente encaminhados a tarefa *T1* do tipo *Splitter* que dividirá o pedido original em dois pedidos distintos, um contendo bebidas quentes e outro geladas. Em seguida, os pedidos gerados são encaminhados para a tarefa *T2* do tipo *Dispacher*, que encaminhará os pedidos para os seus respectivos balcões, bebidas geladas e bebidas quentes. Após essa etapa é então iniciado o processo assíncrono pelos baristas, sendo possível realizar o preparo tanto da bebida quente, quanto da bebida gelada.

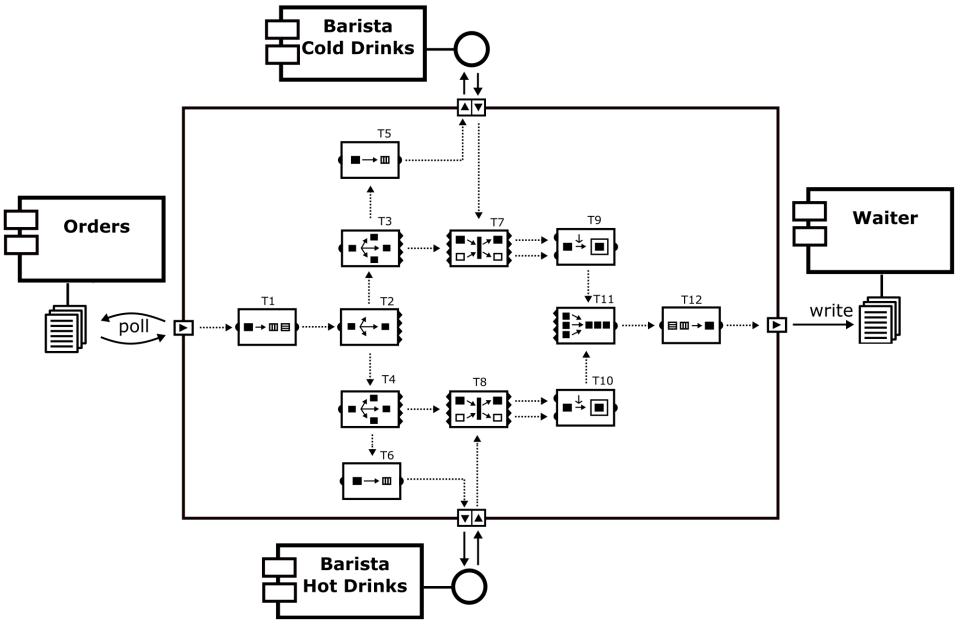


Figura 3. Solução Café implementada pelo Guaraná DSL. Fonte: [Frantz 2012]

Após passar pela tarefa *T2* o pedido é encaminhado a tarefa *T3* do tipo *Replicator*, que gerará duas cópias do pedido recebido por ela, uma será encaminhada a tarefa *T5*, *Translator* e a outra à tarefa *T7*, *Correlator*. As tarefas do tipo *Translator* aplicam modificações no pedido recebido de maneira a criar um novo pedido que possa ser lido pela aplicação destino, que no modelo é chamada de *Barista Cold Drinks*, que será responsável pela preparação do pedido do cliente. Tarefas *Correlator* fazem com que a cópia da mensagem só seja encaminhada a tarefa seguinte no momento em que o *Correlator* receber duas ou mais mensagem com o mesmo identificador.

A tarefa *T9* é do tipo *Context Based Enricher* que adicionará a bebida ao pedido que foi correlacionado pela tarefa *T7*. Posteriormente, será encaminhado para a tarefa *T11*, um *Merger* que juntará as mensagens de diferentes slots em apenas um, e então encaminhará para tarefa *T12*, do tipo *Aggregator*, que juntará as mensagens do slot em uma única mensagem, essa mensagem será composta pelo pedido e as bebidas, e então será encaminhado ao garçom para realizar a entrega.

6. Experimentos

Nesta Seção são apresentados os experimentos. São discutidos os seguintes assuntos: questão de pesquisa e hipóteses, variáveis, ambiente de execução, execução e coleta dos dados, resultados, discussão dos resultados, e ameaças a validade.

6.1. Questão de Pesquisa e Hipóteses

Neste experimento, busca-se responder como o modelo de *deep learning* se comporta em predições a partir do mesmo conjunto de dados de treinamento utilizado a variação de seus parâmetros. A hipótese H_0 é de que *Deep Learning não possui assertividade alta para a predição de mensagens processadas pela plataforma de integração Guaraná*, enquanto que H_1 busca apresentar que *Deep Learning apresenta uma alta assertividade na predição de mensagens processadas pela plataforma de integração Guaraná, a fim de ser possível a sua utilização como base para a definição da quantidade ideal de threads*.

6.2. Variáveis

As variáveis independentes deste experimento são responsáveis pelas configurações realizadas nos modelos: quantidade de neurônios e camadas. A fim de realizar a combinação entre as variações das variáveis, foram realizadas as combinações das variações. A quantidade de camadas variou de um a seis, enquanto que para a quantidade de neurônios utilizou-se a variação exponencial de base dois, iniciando com o expoente dois e finalizando em oito, sendo os valores 2 e 256 o mínimo e o máximo, respectivamente.

As saídas de cada modelo representam a assertividade que ele teve na predição da quantidade de mensagens processadas, ou seja, as variáveis dependentes que foram utilizadas neste experimento para realizar a comparação entre os modelos. Foi utilizada a métrica de avaliação *r-squared*, que tem sua variação ligada diretamente ao quão corretas/erradas foram as predições, o valor retornado estará na escala de zero e um.

6.3. Ambiente de Execução

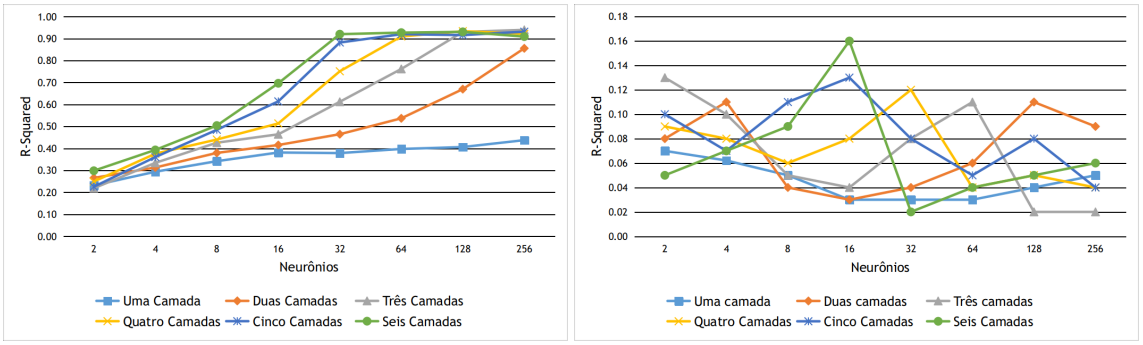
A Execução foi realizada em um computador com o processador Intel(R) Core(TM) $i3-3217U$ CPU, que opera na frequência de 1.80 GHz, possui 2 núcleos e 4 *threads*, e 4 GB de memória. O Sistema Operacional utilizado foi o Deepin/Debian $4.15.0-30$, 64 bits.

6.4. Execução e Coleta de Dados

Cada modelo foi executado 25 vezes e coletadas suas assertividades, a fim de obter as médias. Alguns modelos não foram capazes de realizar o aprendizado dos dados, e então realizou-se a exclusão dos valores no cálculo. Os dados foram gerados a partir de cada execução do modelo e então salvos em um arquivo CSV constando todas as repetições da combinação dos parâmetros. A quantidade de arquivos gerados é representado pela equação: Quantidade Camadas x Quantidade de Variações de Neurônios (6x8): 48.

6.5. Resultados

A assertividade de um modelo é o indicador de quão bom é o modelo, sendo assim os resultados utilizados para análise dos dados foram obtidos a partir da média da assertividade de cada modelo. A Figura 4a apresenta as assertividades com os modelos de *deep learning*, enquanto que a Figura 4b apresenta os desvios-padrões dos modelos. Sendo separados pela quantidade de camadas e neurônios. A visualização dos dados deve ser realizada a partir da quantidade de camadas (cor e formato) e de neurônios (eixo x). A assertividade está no eixo y e deve ser utilizada para realizar a comparação dos modelos.



(a) Assertividade dos Modelos **(b) Desvio Padrão dos Modelos**

Figura 4. Assertividade e Desvio Padrão dos modelos

Com os resultados obtidos, é possível observar que todos os modelos com três ou mais camadas e pelo menos 128 neurônios tiveram a assertividade acima de 90%, enquanto que ao utilizar apenas menos camadas as assertividades foram menores. Todos os modelos apresentaram baixa assertividade com dois neurônios. Também é visível o crescimento dos assertos dos modelos conforme o incremento de suas variáveis.

6.6. Discussão

Na Figura 4a, é possível verificar a tendência de incremento da assertividade em relação a quantidade de neurônios, o mesmo ocorre para a quantidade de camadas. O crescimento é explicado devido a maior possibilidade de interconexões entre os neurônios, permitindo assim melhores ajustes de seus pesos. O modelo de uma camada que obteve a maior assertividade foi o modelo com 256 neurônios, obtendo 0.44 de assertividade em seus testes. O modelo mais próximo foi o modelo de 128 neurônios, que obteve a assertividade de 0.41. Enquanto isso, o modelo de apenas 2 neurônios obteve o pior desempenho, de 0.23. Além disso, é possível verificar que o comportamento do desvio padrão foi mais suave em relação aos modelos de mais camadas, mostrando que a variação média da assertividade esteve mais balanceada em relação aos modelos com mais camadas.

Os modelos de duas camadas obtiveram um crescimento de semelhante ao crescimento exponencial, havendo uma diferença considerável da assertividade entre os modelos de 2 e 256 neurônios, sendo possível verificar um crescimento da assertividade com o incremento da quantidade de neurônios nos modelos. O melhor resultado desse modelo foi de 0.85 enquanto que o pior foi de 0.26, resultando na diferença de 0.59. Existe ainda uma diferença de 0.19 de assertividade entre os modelos de 128 (0.67) e 256 (0.86) neurônios. Essa diferença não é vista entre os outros modelos com essa quantidade de camadas, entretanto ela pode ser explicada pois o modelo de 256 neurônios possui o dobro

de neurônios se comparado ao outro modelo, também é possível verificar que há um valor de 0.11 no desvio padrão do modelo de 128 neurônios, enquanto que o modelo de 256 neurônios teve 0.09, ou seja, os modelos tiveram uma variação próxima.

Um crescimento linear é apresentado na variação de 16 até 128 neurônios nos modelos de três camadas, variando de 0.46 para 0.93. A maior assertividade desses modelos foi do modelo de 256 neurônios, com 0.94. Próximo ao modelo de 256 neurônios, esteve o modelo de 128 neurônios, que obteve a assertividade de 0.93, apresentando a primeira estabilização da assertividade entre os modelos. A menor assertividade dos modelos dessa quantidade de camadas, foi do modelo com apenas 2 neurônios, que possui apenas 0.22. É possível verificar o alto desvio padrão com 2 e 64 neurônios, mostrando que os valores variaram mais em relação aos outros modelos com essa quantidade de camadas.

Assim como nos modelos de três camadas, a assertividade dos modelos de quatro camadas tenderam a estabilização após um determinado número de neurônios, que nesse modelo foi de 64, havendo três modelos que ficaram acima de 90% de assertividade, os modelos com 64, 128 e 256 neurônios. A maior assertividade do modelo de quatro camadas ocorreu quando teve 128 neurônios disponíveis para suas interconexões, obtendo o valor de 0.93, enquanto que com 256 neurônios a assertividade ficou em 0.90. Entretanto, é possível verificar que com 32 neurônios houve um elevado desvio padrão, mostrando que houve uma variação maior com essa configuração.

Com cinco camadas é possível visualizar um crescimento da assertividade dos modelos com até 32 neurônios. Sendo que três modelos estiveram acima de 0.90 de assertividade, os modelos de 64, 128 e 256. Entretanto, o modelo de 32 neurônios esteve próximo dessa assertividade, obtendo 0.88. A menor assertividade ficou em 0.22 e a maior em 0.93, com 2 e 256 neurônios, respectivamente. Também é visível a diferença da assertividade entre os modelos de 16 e 32 neurônios. Já o desvio padrão se mostrou instável em relação aos outros modelos, sendo que o maior foi de 0.13 com 16 neurônios, o que pode justificar a diferença em relação ao modelo de 32 neurônios.

Por fim, utilizando seis camadas, é possível verificar que quatro modelos estiveram acima de 0.90 de assertividade, os modelos de 32, 64, 128 e 256 neurônios, obtendo o *r-squared* de 0.91, 0.92, 0.93 e 0.90, respectivamente. O modelo de menor assertividade foi o modelo com 2 neurônios, que obteve 0.29 de assertividade. Além disso, o desvio padrão se mostrou instável, sendo que com 16 e 32 neurônios obteve os valores de 0.16 e 0.02, respectivamente, sendo o maior e o menor valor entre todos os modelos.

Observando os resultados, é possível observar que o modelo de maior assertividade foi o de 3 camadas e 256 neurônios. Além disso, foi um dos modelos que menos variou a assertividade, com 0.02 de desvio padrão. Portanto, é possível verificar que embora outros modelos tiveram mais camadas, eles também tiveram maior variação da assertividade, ou seja, um desvio padrão maior.

6.7. Ameaças à Validade

Este estudo se mostrou válido pois com ele foi possível encontrar qual modelo se melhor adéqua na realização de predições da quantidade de mensagens que serão processadas pela plataforma de integração guaraná. Utilizando os modelos de *deep learning* pode-se realizar predições com alto nível de confiabilidade, utilizar as predições durante a execução da plataforma de integração e realizar a alteração da quantidade de *threads* em tempo

real. Ao realizar a alteração das variáveis independentes dos modelos de *deep learning*, é verificado que as variáveis dependentes se alteraram de forma positiva.

Neste estudo realizou-se a utilização de um *dataset* de uma experimentação da solução de integração café, sendo assim, não é possível generalizar os resultados obtidos para todas as soluções de integração. Portanto, essa é a principal ameaça aos resultados.

Para que este artigo possa responder a questão de pesquisa e testar as hipóteses, algumas etapas foram realizadas. Levantamento do referencial teórico; as informações sobre o objeto de estudo são apresentadas, seguido pelas configurações utilizadas, e também os diferentes cenários de execução são apresentadas. Por fim, é realizado a execução de diferentes combinações para representar diferentes modelos que possam ser utilizados.

Para apresentar resultados que possuam conclusões corretas, diversos cuidados foram levados em consideração. A quantidade de execução de cada modelo, que representa um número que permite realizar análise e conclusões de forma confiável. As alterações das variáveis independentes foram realizadas a fim de verificar com maior intensidade as alterações dos resultados. Por fim, a separação dos dados de treinamento foi feita de forma randômica, para que não tenha causado resultados inconsistentes.

7. Conclusão

Muitos desafios são encontrados e resolvidos pelas plataformas de integração, como a definição do número ideal de *threads*. Neste artigo, utilizou-se modelos de *deep learning* para realizar a predição da quantidade de mensagens que seriam processadas a partir da taxa de entrada e da quantidade de *threads* configuradas pela plataforma de integração Guaraná. A técnica *r-squared* foi utilizada para mensurar os modelos. O objetivo do artigo foi encontrar a configuração ideal para um modelo de *deep learning* na realização da predição do número de mensagens que seriam processadas a partir da taxa de entrada e da quantidade de *threads* disponíveis. Com esse modelo, um gerenciador de *threads* automático e em tempo de execução poderia ser construído, para que fosse possível uma melhor definição do número de *threads* que fosse capaz de executar todas as mensagens que chegassem, além de não sobrecarregar o servidor. A partir das distintas configurações de modelos, é possível verificar que o aumento da quantidade de camadas e neurônios é capaz de aumentar a precisão das predições, até chegar a estabilidade de, no máximo, 94%. Por outro lado, a baixa assertividade do modelo de apenas uma camada também é notável, pois o número de interconexões entre os neurônios é baixo.

Agradecimentos

À Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS) (termo de outorga número 17/2551-0001206-2) e Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

Referências

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. In *12th USENIX Symp. on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283.

Beer, M. I. and Hassan, M. F. (2018). Adaptive security architecture for protecting restful web services in enterprise computing environment. *Service Oriented Computing and Applications*, 12(2):111–121.

Dossot, D., d'Emic, J., and Romero, V. (2014). *Mule in action*. Manning Publications Co.

Fisher, M., Partner, J., Bogoevici, M., and Fuld, I. (2012). *Spring integration in action*. Manning Publications Co.

Frantz, R. Z. (2012). *Enterprise application integration: an easy-to-maintain model-driven engineering approach*. PhD thesis, Universidad de Sevilla.

Frantz, R. Z., Corchuelo, R., Basto-Fernandes, V., Rosa-Sequeira, F., Roos-Frantz, F., and Arjona, J. L. (2020). A cloud-based integration platform for enterprise application integration: a model-driven engineering approach. *Software: Practice and Experience. (in-press)*, 1(1):1–25.

Frantz, R. Z., Corchuelo, R., and Roos-Frantz, F. (2016). On the design of a maintainable software development kit to implement integration solutions. *Journal of Systems and Software*, 111(1):89–104.

Frantz, R. Z., Sawicki, S., Roos-Frantz, F., Yevseyeva, I., and Emmerich, M. (2015). On using markov decision processes to model integration solutions for disparate resources in software ecosystems. In *Intl. Conf. on Enterprise Information Systems (ICEIS)*, pages 260–268.

Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. "O'Reilly Media, Inc.".

Hohpe, G. and Woolf, B. (2004). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional.

Ibsen, C. and Anstey, J. (2018). *Camel in action*. Manning Publications Co.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. In *3rd Intl. Conf. for Learning Representations (ICLR)*, pages 1–13.

Manikas, K. (2016). Revisiting software ecosystems research: A longitudinal literature study. *Journal of Systems and Software*, 117:84–103.

Nemirovsky, D., Arkose, T., Markovic, N., Nemirovsky, M., Unsal, O., and Cristal, A. (2017). A machine learning approach for performance prediction and scheduling on heterogeneous cpus. In *29th Intl. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 121–128.

Page, A. J. and Naughton, T. J. (2005). Framework for task scheduling in heterogeneous distributed computing using genetic algorithms. *Artificial Intelligence Review*, 24(3):415–429.