PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

DIONATRÃ FOLLE KIRCHOFF

AVALIAÇÃO DE TÉCNICAS DE APRENDIZADO DE MÁQUINA PARA PREVISÃO DE CARGAS DE TRABALHO APLICADAS PARA OTIMIZAR O PROVISIONAMENTO DE RECURSOS EM NUVENS COMPUTACIONAIS

Porto Alegre
2019

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

AVALIAÇÃO DE TÉCNICAS DE APRENDIZADO DE MÁQUINA PARA PREVISÃO DE CARGAS DE TRABALHO APLICADAS PARA OTIMIZAR O PROVISIONAMENTO DE RECURSOS EM NUVENS COMPUTACIONAIS

DIONATRÃ FOLLE KIRCHOFF

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. César A. F De Rose

Porto Alegre
2019

Ficha Catalográfica

K58a Kirchoff, Dionatrã Folle

Avaliação de técnicas de aprendizado de máquina para previsão de cargas de trabalho aplicadas para otimizar o provisionamento de recursos em nuvens computacionais / Dionatrã Folle Kirchoff . – 2019.

88.

Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. César Augusto Fonticielha De Rose.

1. previsão de carga de trabalho. 2. computação em nuvem. 3. eficiência de recursos. I. De Rose, César Augusto Fonticielha. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Salete Maria Sartori CRB-10/1363

Dionatrã Folle Kirchoff

AVALIAÇÃO DE TÉCNICAS DE APRENDIZADO DE MÁQUINA PARA PREVISÃO DE CARGAS DE TRABALHO APLICADAS PARA OTIMIZAR O PROVISIONAMENTO DE RECURSOS EM NUVENS COMPUTACIONAIS

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação, Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul.

Aprovado em 29 de Março de 2019.

BANCA EXAMINADORA:

Prof. Dr. Lucas Mello Schnorr (PPGC/UFRGS)

Prof. Dr. Tiago Coelho Ferreto (PPGCC/PUCRS)

Prof. Dr. César Augusto Fonticielha De Rose (PPGCC/PUCRS - Orientador)

AVALIAÇÃO DE TÉCNICAS DE APRENDIZADO DE MÁQUINA PARA PREVISÃO DE CARGAS DE TRABALHO APLICADAS PARA OTIMIZAR O PROVISIONAMENTO DE RECURSOS EM NUVENS COMPUTACIONAIS

RESUMO

A computação em nuvem transformou a forma de provisionar recursos computacionais nos últimos anos, oferecendo vários benefícios em relação aos sistemas tradicionais como escalabilidade e alta disponibilidade. No entanto, ainda existem algumas oportunidades a serem exploradas, especialmente na área de alocação e dimensionamento proativo de recursos. Como a carga de trabalho pode flutuar muito nestes ambientes, o provisionamento excessivo é uma prática comum para evitar quedas repentinas de Qualidade de Serviço (QoS) que podem resultar em violações de Acordo de Nível de Serviço (SLA), mas ao custo de um aumento nos custos de provisionamento e consumo de energia. A previsão de carga de trabalho é uma das estratégias pelas quais a eficiência e o custo operacional de uma nuvem podem ser melhorados. Saber antecipadamente a demanda de um serviço permite a alocação prévia de recursos suficientes para manter a QoS e evitar violações de SLA. Esta dissertação apresenta as vantagens e desvantagens de três técnicas de previsão de carga de trabalho usualmente aplicadas no contexto da computação em nuvem. São comparados os algoritmos de aprendizado de máquina ARIMA, MLP e o GRU em diferentes configurações, para ajudar os administradores a escolher o modelo preditivo mais adequado e eficiente para seu problema específico. O resultado das avaliações apresenta que os algoritmos estudados são equivalentes quanto à precisão obtida neste contexto, mas apresentam diferenças importantes em sua aplicação, de forma que este trabalho auxilia na escolha da melhor técnica para o cenário em questão.

Palavras-Chave: previsão de carga de trabalho, computação em nuvem, eficiência de recursos.

EVALUATION OF MACHINE LEARNING TECHNIQUES TO PREDICT WORKLOADS APPLIED TO OPTIMIZE COMPUTATIONAL CLOUD RESOURCE PROVISIONING

ABSTRACT

Cloud computing has transformed the means of resource provisioning in recent years with several benefits over traditional systems, like scalability and high availability. However, there are still some opportunities, especially in the area of proactive resource allocation and scaling. Since demand may fluctuate heavily in certain environments, over-provisioning is a common practice to avoid abrupt Quality of Service (QoS) drops that may result in Service Level Agreement (SLA) violations, but at the price of an increase in provisioning costs and energy consumption. Workload prediction is one of the strategies by which efficiency and operational cost of a cloud can be improved. Knowing demand in advance allows the allocation of sufficient resources to maintain QoS and avoid SLA violations. This paper presents the advantages and disadvantages of three workload prediction techniques usually applied in the context of cloud computing. We compare ARIMA, MLP and GRU under different configurations, and although all three strategies have similar accuracy results in this context, they present important differences in preparation and execution. This work helps system administrators in choosing the more appropriate and efficient predictive model for their specific problem.

Keywords: workload prediction, cloud computing, resource efficiency.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

A computação em nuvem ganhou mercado devido as vantagens oferecidas como prioridade na disponibilidade dos serviços, na escalabilidade e na promessa de custos de infraestrutura mais baixos [25]. Os aplicativos corporativos de grande escala baseados em componentes que aproveitam os recursos da nuvem esperam garantias de Qualidade de Serviço (QoS) e de Acordo de Nível de Serviço (SLA) entre o cliente e os provedores de serviços. No contexto da computação em nuvem, as estratégias de *auto-scaling* prometem garantir as propriedades de QoS aos aplicativos, ao mesmo tempo em que se utilizam eficientemente dos recursos. Apesar das vantagens percebidas de *auto-scaling*, apresenta-se complexo obter todo o seu potencial devido aos vários desafios decorrentes da necessidade de estimar com precisão o uso de recursos, o que é influenciado pela variação imprevisível dos padrões de carga de trabalho do cliente. Muitos pesquisadores buscam estratégias para adaptar melhor os ambientes para os aplicativos usando algoritmos que realizam o aprendizado baseados em experiências [25] [31] [35]. Os autores afirmaram que modelos preditivos podem ser aplicados sobre comportamentos de carga de trabalho, possibilitando que a estratégia de *auto-scaling* aloque com precisão os recursos necessários para satisfazer as políticas de QoS e o SLA [2].

Os algoritmos de predição de carga de trabalho pertencem a um problema de regressão, o que significa que as variáveis são estimadas ao longo do tempo [18]. Dessa forma, a fase de aprendizado em um algoritmo de Aprendizado de Máquina (AM) não requer uma infraestrutura robusta para ser executada, ao contrário dos problemas de classificação, os quais precisam de infraestrutura de alto preço para computar uma coleção massiva de dados. Amiri et al. [2] reuniram estudos relacionados à previsão de carga de trabalho em nuvens. Esses estudos são responsáveis por lidar com problemas semelhantes em complexidade, porém não possuem a clareza do desempenho e da precisão do modelo que são essenciais para a manutenção de QoS e SLA.

O provisionamento suficiente de recursos em uma nuvem pode variar com base nas características da aplicação. Consequentemente, com o objetivo de avaliar um cenário real com oscilações na demanda de recursos e, justificando a necessidade de provisionamento de recursos proativos. Nesta dissertação foi optado por abordar cargas de trabalho provenientes de aplicações web, pois possuem demandas ditadas por eventos externos que caracterizam o consumo de recursos em uma frequência não linear, sendo um cenário ideal para testar a efetividade das previsões de carga.

Existem diversas abordagens que procuram dar suporte ao provisionamento de recursos; no entanto, os algoritmos de AM têm ganho cada vez mais espaço nessa área, tornando-se objeto de estudo nos mais recentes trabalhos devido a sua aderência na resolução dos problemas [2]. Desta maneira, entender as capacidades e as necessidades

de cada algoritmo de AM mostra-se um diferencial que pode acelerar o desenvolvimento da implantação de soluções proativas. Portanto o trabalho desenvolvido nesta dissertação apresenta uma análise de custo benefício entre técnicas para a previsão de carga de trabalho em aplicações Web. Para comparar os algoritmos foram utilizados os modelos guiados por AM, os quais são adotados em estudos recentes [2]: *Autoregressive Integrated Moving Average* (ARIMA), *Multi Layer Perceptron* (MLP) e *Gated Recurrent Unit* (GRU).

1.1 Objetivos

Com o propósito de auxiliar na escolha de Algoritmos de AM utilizadas por estratégias proativas, tem-se, dessa maneira, o objetivo principal deste trabalho: a investigação dos prós e contras entre os modelos ARIMA, MLP e GRU, aplicando algoritmos de *Tuning* para explorar configurações de cada modelo.

1.2 Estrutura

A presente dissertação está organizada da seguinte forma: Capítulo 2 descreve os conceitos que correlacionam os tópicos relacionados a carga de trabalho e as respectivas abordagens utilizadas em soluções proativas; Capítulo 3 caracteriza os métodos de Aprendizado de máquina e suas definições; Capítulo 4 relata todos os experimentos executados ao longo desta pesquisa; Capítulo 5 descreve os trabalhos relacionados a esta pesquisa; Capítulo 6 descreve os resultados alcançados e os planos para as etapas futuras deste estudo.

2. CONTEXTUALIZAÇÃO E ESTADO DA ARTE

2.1 Contextualização

A computação em nuvem é um modelo que permite o acesso de rede onipresente e conveniente a um conjunto compartilhado de recursos computacionais configuráveis que podem ser rapidamente provisionados e liberados [30]. Esta tecnologia, permite com que as empresas produzam e disponibilizem seus serviços rapidamente. Possibilitando, dessa maneira, a configuração de uma máquina física para hospedar múltiplas máquinas virtuais e permitindo a implantação de diversas aplicações. Além disso, fornecendo recursos para os usuários armazenar e processar seus dados em centros de dados de terceiros.

De acordo com Nanda [33], em geral, os fornecedores de software de aplicativos estão migrando cada vez mais para as plataformas da nuvem, dependendo do modelo de serviço não requer conhecimento específico com a instalação e com a manutenção de uma infraestrutura de servidor físico. No entanto, a adoção da nuvem ganhou mercado devido a prioridade na disponibilidade dos serviços e na garantia de nível de serviço (SLA) oferecido aos clientes [25].

Portanto, o desempenho desses aplicativos é um fator chave para a receita das empresas. As lentidões ou interrupções de aplicativos levam a clientes insatisfeitos e, consequentemente, ao encerramento de contratos. Desta forma, a responsabilidade dos provedores de nuvem é de suprir a demanda requisitada para fornecer recursos suficientes aos respectivos clientes. No entanto, administradores precisam realizar uma análise entre os custos operacionais contratados e sua real demanda. Considerando a Figura 2.1, dentro de um cenário de escalonamento de recursos baseado em Hwang et al. [21] ao alocar mais recursos que o necessário para a execução das aplicações em nuvem, temos a ocorrência do conceito de *over-provisioning*, onde as aplicações não sofrem penalidades. No entanto os custos e energia são desperdiçados, pois estamos alocando mais recursos do que o necessário. O conceito de *under-provisioning*, deve-se pela quantidade insuficiente de recursos exigida pela demanda das aplicações, logo, as aplicações sofrem penalidades de QoS e quebra de SLA causada pelo atraso no ajuste de infraestrutura em decorrência de ações reativas.

No contexto de migração de máquinas virtuais, existe contribuições significativas destinadas a redução da quantidade de Máquinas Físicas (MFs) em execução através da consolidação de maquinas virtuais. No entanto, tais migrações, também trazem consumo extra de energia causado pela transmissão de imagens da máquina virtual e operações de *stop/resume*. Devido a estas migrações o tempo de resposta aumenta consideravelmente resultando em um efeito negativo no desempenho da máquina virtualizada [43]. Com estas

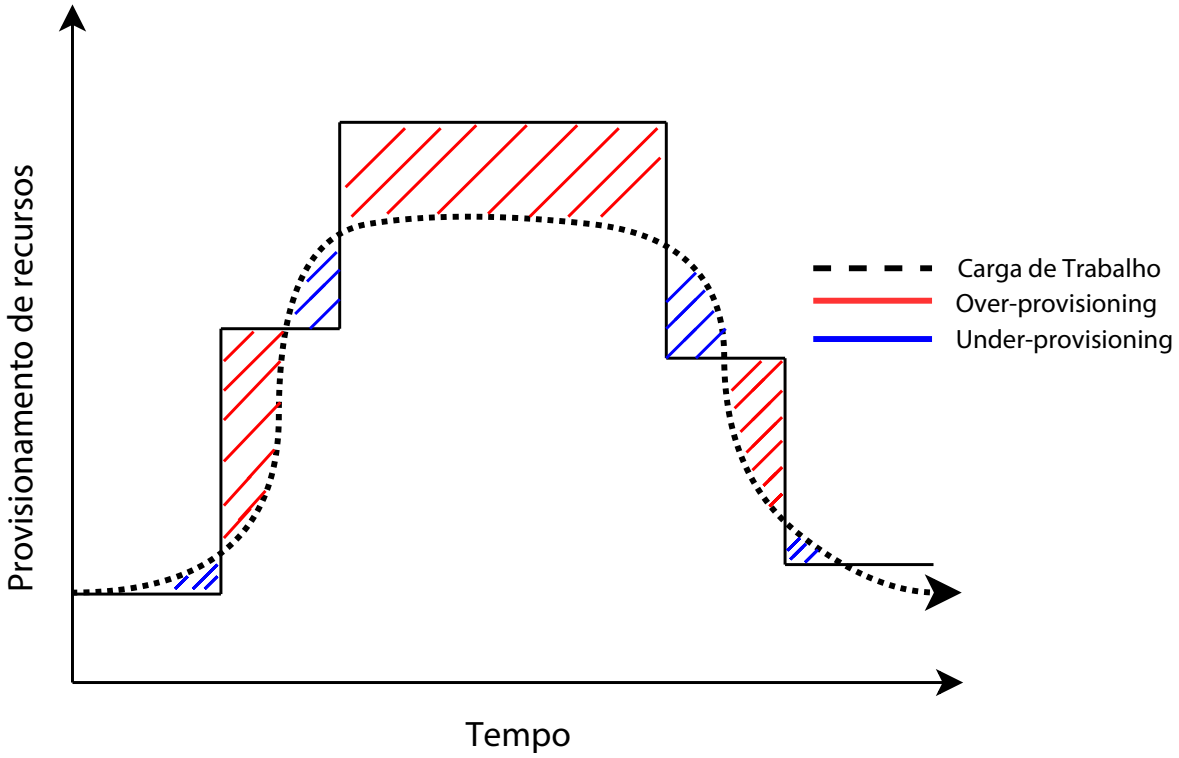

Figura 2.1 – Provisionamento reativo de recursos adaptado de Hwang et al.[21]

informações, para Xu et al.[43] ainda é um desafio realizar o escalonamento de máquinas virtuais visando reduzir o consumo de energia e a mitigação da degradação do desempenho.

Para Wei et al. [42] também há um trade-off entre custo e SLA para provedores de nuvem. Para lidar com a grande quantidade de solicitações dos usuários, algumas organizações utilizam a abordagem de alocação de máquinas, onde procuram manter MFs suficientes para suas demandas, e, consequentemente, não ocasionar a quebra de SLA. No entanto, a manutenção de muitas MFs resulta no efeito de *over-provisioning*. Além disso, existe a necessidade de aguardar a ativação de MFs, no qual pode causar impactos no tempo de resposta, sendo um desafio encontrar o número adequado de máquinas ativas.

Com o intuito de solucionar estes problemas, há estudos que procuram explorar estratégias para prever o comportamento de aplicações por meio do uso de algoritmos de aprendizado de máquina [25]. Amiri and Mohammad-Khanli [2] afirmaram que modelos preditivos podem ser aplicados com base em dados históricos de carga de trabalho para alocar com precisão os recursos necessários para satisfazer as políticas de QoS e o SLA. Desta forma, a Seção a seguir contextualiza as características de previsão de carga de trabalho.

2.2 Caracterização de previsão de Carga de Trabalho

A carga de trabalho pode ser adquirida pela coleta dos dados da utilização de recursos computacionais, aplicações e serviços, por exemplo. Estes dados podem ser utilizados como entrada para técnicas de previsão para descobrir comportamentos futuros e demandas de infraestrutura. Na computação em nuvem, a previsão do comportamento de aplicativos é uma etapa essencial para o gerenciamento eficiente de recursos [2].

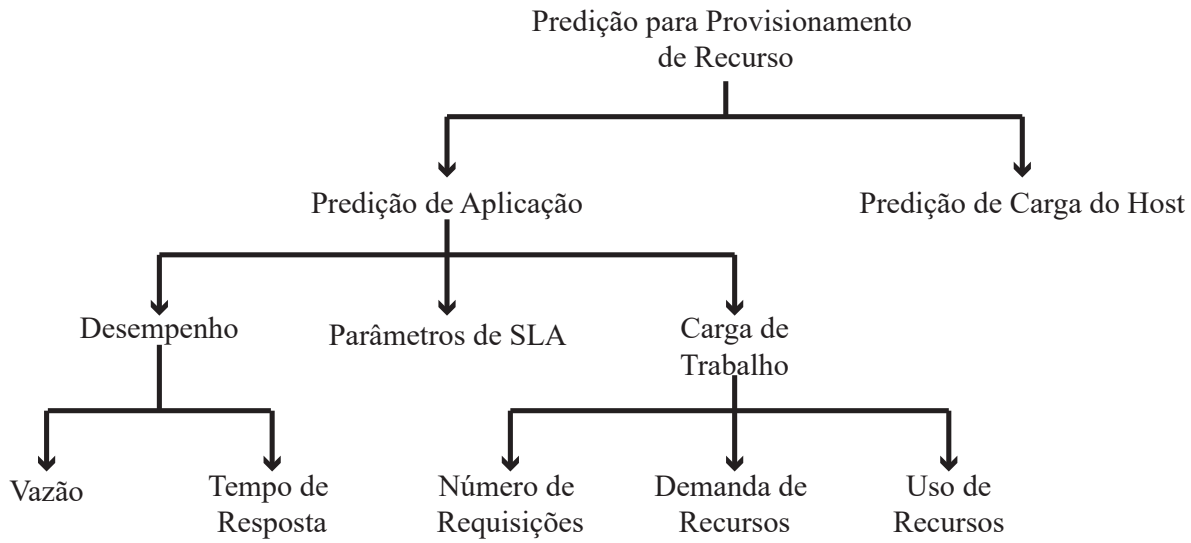

Figura 2.2 – Diferentes níveis de predição por Amiri e Mohammad-Khanli[2]

Amiri e Mohammad-Khanli [2] pesquisaram o conceito de *carga de trabalho* e descobriram que este conceito pode ser interpretado de maneiras diferentes na literatura, como pode ser observado na Figura 2.2. Em algumas referências como Liang et al. [27] e Kumar et al. [25], eles consideram que a carga de trabalho de um aplicativo é equivalente ao número de solicitações para o aplicativo. Nesta abordagem, o número futuro de requisições é a saída do método de previsão.

No caso de Jiang et al. [23] a carga de trabalho é interpretada como a demanda futura de máquinas virtuais (VMs). Para Garg et al. [16] a utilização de recursos das VMs é considerada como carga de trabalho. Eles assumiram que cada aplicativo é encapsulado dentro de uma VM. No contexto da previsão de desempenho de aplicações, Manvi e Krishna Shyam [29] mencionam que os valores de taxa de transferência e tempo de resposta, são previstos com base no recurso alocado para a aplicação. Nessa situação, o objetivo principal é determinar a melhor alocação de recursos para obter o nível de desempenho desejado.

Leitner et al. [26] preveem parâmetros de SLA de aplicativos. Os autores fazem uso da previsão de violação de SLA de acordo com a carga de trabalho ou com os recursos alocados para as aplicações. Em outras palavras, os dados que se referem a dados típicos

de QoS (por exemplo, tempos de resposta, disponibilidade, carga do sistema) e dados de instância de processo (por exemplo, identificadores de clientes, produtos encomendados) podem ser utilizados para previsão. A predição de carga do *host* é usada por Yang et al. [44] com o foco de melhorar a utilização de recursos na nuvem.

Esta Seção apresentou diferentes perspectivas no que é conhecido por carga de trabalho, sendo aplicadas soluções proativas em cenários contextualizados. Como o foco desta dissertação é baseada na utilização de aplicações web como *benchmark*, podemos concluir que a carga de trabalho ideal para esse contexto também corresponde as requisições de aplicação conforme adotado pelos autores [27][25].

A próxima sessão descreve as metodologias utilizadas na literatura para a previsão das cargas de trabalho.

2.3 Taxonomia de previsão de carga de trabalho

Existem esforços de pesquisa para descobrir abordagens para encontrar o comportamento do aplicações. Com base nesta informação, Amiri e Mohammad-Khanli [2] propuseram uma taxonomia geral para modelos de previsão de aplicação. Segundo os autores, os modelos de previsão podem ser divididos em quatro grupos; esses grupos incluem métodos como orientados por tabela, teoria de controle, teoria das filas e técnicas de aprendizagem de máquina conforme Figura 2.3

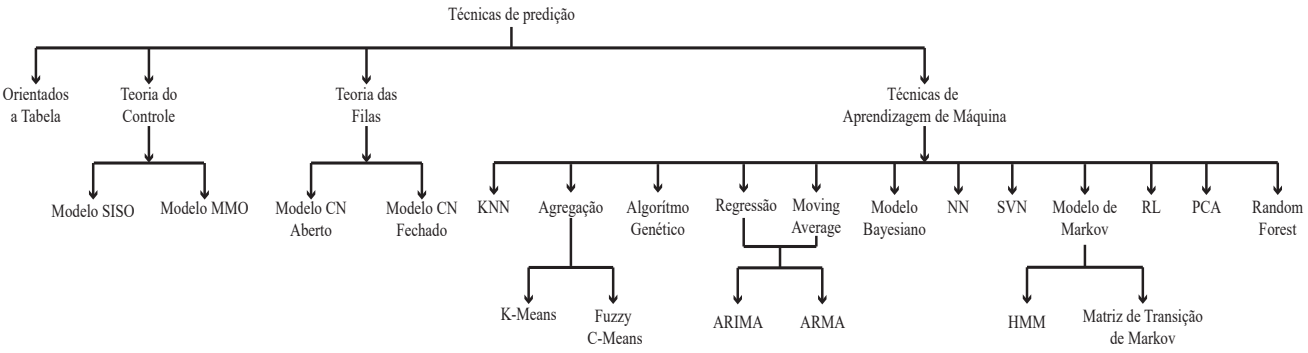

Figura 2.3 – Taxonomia dos métodos de previsão por Amiri and Mohammad-Khanli[2]

2.3.1 Método Orientados a Tabela

Em métodos orientados a tabela, o comportamento do aplicativo monitorado é registrado em uma tabela para diferentes valores da intensidade da carga de trabalho e

diferentes quantidades de recursos alocados [2]. A interpolação é um método de construção de novos pontos de dados usada para obter valores não registrados na tabela. Essa abordagem tem algumas limitações, como baixa escalabilidade devido ao número de aplicativos, estados diferentes da alocação de recursos e diferentes tipos de cargas de trabalho [4]. A construção da tabela consome tempo, e demanda a necessidade de diversos experimentos para preencher a tabela. Portanto, esse método é considerado obsoleto conforme mencionam os autores Amiri e Mohammad-Khanli [2].

2.3.2 Método Teoria de Controle

Nos métodos de controle, o objetivo é controlar recursos compartilhados entre aplicativos em nuvem [2]. O método de Teoria de controle tem um mecanismo para trabalhar com mudanças imprevisíveis e perturbações em sistemas usando *feedback* [46].

Este tipo de sistemas pode ser dividido em dois grupos, controle de *loop* aberto e sistemas de controle de *loop* fechado [2]: Em sistemas de controle aberto, o desempenho da aplicação (saída) depende dos recursos alocados (os sinais de entrada), e o sinal de saída não afeta a entrada para controlar os recursos alocados para a aplicativo. Em sistemas de controle em *loop* fechado há um *loop* de *feedback* que compara a execução do aplicativo com o desempenho desejável. Assim, o controlador ajusta os recursos alocados de acordo com sua meta de desempenho.

Um controle de *loop* fechado padrão conforme ilustrado na Figura 2.4 baseado em Zhu et al. [46], refere-se ao sistema que está sendo controlado como o sistema de destino, que possui um conjunto de métricas de interesse (referente as medidas de saída) e um conjunto de botões de controle (referenciado como entrada de controle).

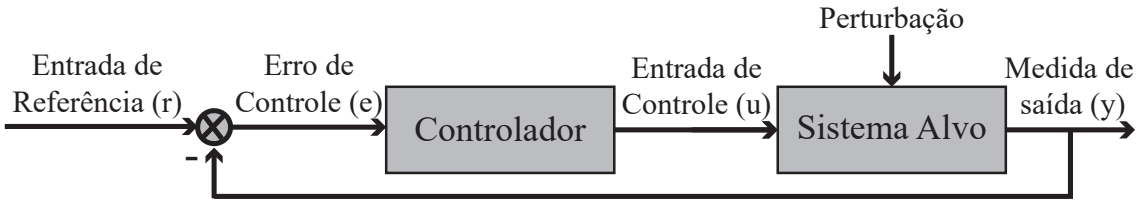

Figura 2.4 – Loop de controle de feedback padrão Zhu et al.[46].

Além disso, como mencionado por Zhu et al. [46] existem algumas limitações para essa abordagem, como as inter-relações de não-linearidade em sistemas de computação, o que torna a modelagem mais difícil de ser entendida e aplicada na prática. Além disso, impõe uma limitação na velocidade com que as cargas de trabalho ou o comportamento do sistema podem mudar. Existem problemas com a granularidade de tempo, dificultando a projeção de controladores que respondam a mudanças em escalas de tempo mais curtas.

2.3.3 Método Teoria das Filas

O método de teoria das filas (TF) pode ser usado para prever o desempenho do aplicativo e modelar o relacionamento entre a carga de trabalho e os critérios de desempenho [2]. Na TF, cada servidor é alocado para o aplicativo como um sistema de filas, conforme mencionado por Urgaonkar et al. [41], que investiga essa abordagem com aplicativos de Internet de multicamadas.

Os trabalhos vão de uma fila e chegam em outra. Os métodos TF possuem parâmetros como a taxa de chegada de solicitações e os requisitos médios de recursos de solicitações que devem ser especificados [41].

O método TF aberto ilustrado pela Figura 2.5a as demandas podem ser externas. Assim, o número de *jobs* no sistema varia com o tempo [19]. No TF fechado ilustrado pela Figura 2.5b, existe uma população constante na rede e nenhuma fonte externa. No entanto, esta abordagem possui fontes limitadas ao que tange os tópicos de previsão de carga de trabalho [2]. É possível que o método tenha sido substituído para uma implementação mais direta. Na pesquisa de Amiri e Mohammad-Khanli [2] também não há informação sobre novas pesquisas neste cenário.

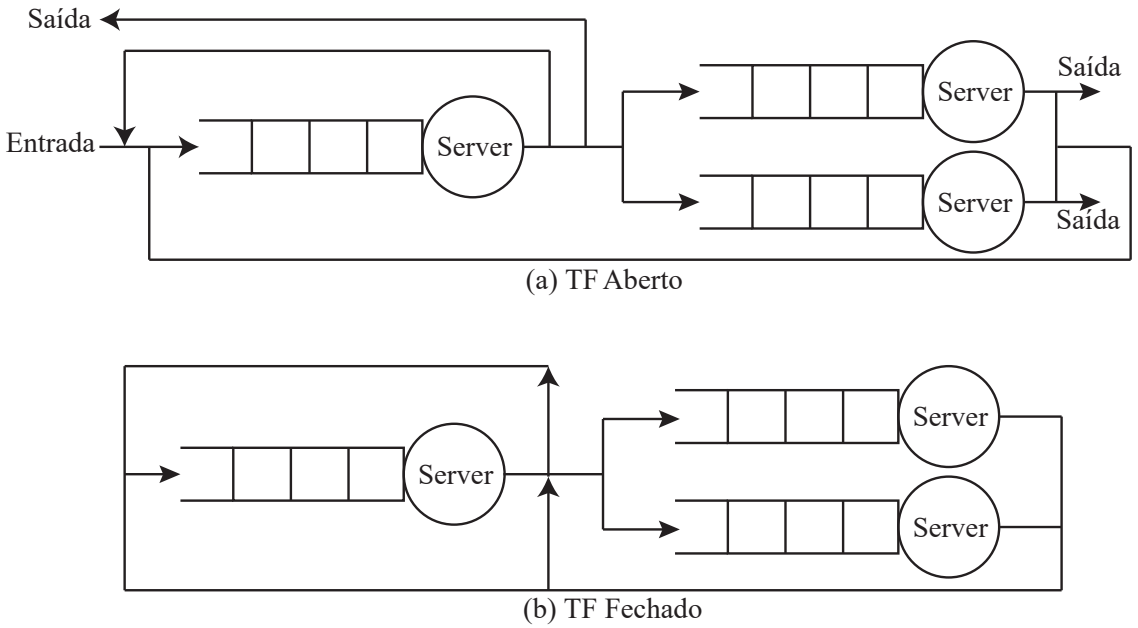

Figura 2.5 – Um exemplo de modelos TF abertas e fechadas, Amiri and Mohammad-Khanli[2]

2.3.4 Métodos de *Aprendizado de Máquina*

Uma das abordagens recentemente pesquisadas para a previsão de aplicações corresponde aos métodos de AM. Esses métodos são capazes de prever o comportamento da aplicação em diferentes dimensões, como, por exemplo, previsão de recursos [23], Violação de SLA [26], desempenho de aplicações e o tempo de execução de *jobs* [2].

Com base nos conceitos mencionados por Gollapudi [18], abaixo estão descritos diferentes categorias de problemas em que os algoritmos de AM podem resolver, conforme categorizado pela Figura 2.6:

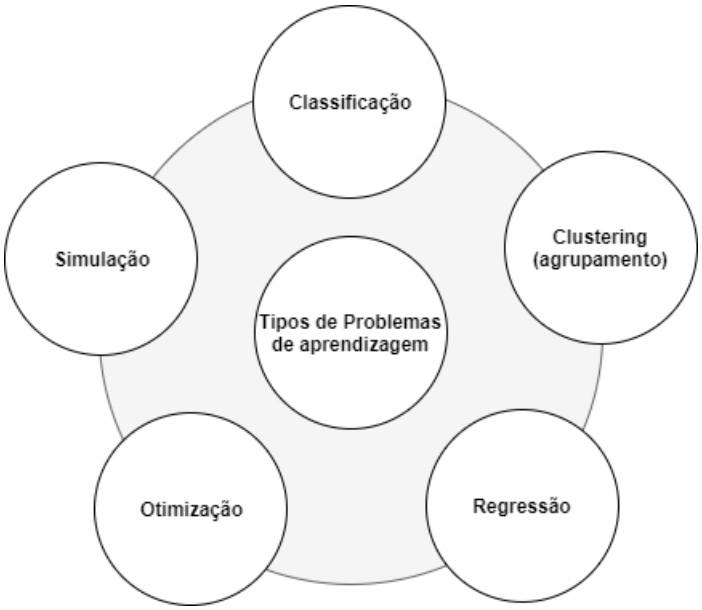

Figura 2.6 – Tipos de problemas de aprendizagem
Fonte: baseado em GOLLAPUDI, 2016, p. 16.

- **Classificação:** é uma maneira de identificar uma técnica de agrupamento para um determinado conjunto de dados. Dependendo de um valor do atributo de destino ou saída. Todo conjunto de dados pode ser qualificado para pertencer à uma determinada classe. Essa técnica tem como objetivo identificar padrões de comportamento de dados, tratando-se de um mecanismo de discriminação. As técnicas de classificação nem sempre se tratam de diferenciações binárias como sim/não, masculino/feminino, mas, também, pode ser definida como abaixo da média, na média, acima da média, bom, ótimo. Essas classificações dependem da definição de problema nas quais estão baseadas.
- **Clustering (agrupamento):** este, diferente do problema de classificação supracitado onde o usuário especifica qual é a medida de classificação que se pretende extrair. Na categoria de *clustering* são fornecidos dados e se espera encontrar padrões interessantes que possam agregar inteligência, ou seja, isso é uma análise de classificação

que não inicia com um objetivo definido (bom / ruim, alto / baixo), e, sim, explora novas opções que podem ser extraídas por meio dos *datasets*(conjuntos de dadosntos de) fornecidos.

- **Predição ou Regressão:** similar a classificação, os problemas de predição também estão interessados em identificar o caminho que as coisas vão seguir no futuro. Estas informações são derivadas de experiências passadas, ou seja, os *datasets* são compostos por dados históricos. Neste caso, existe a necessidade de prever o futuro através da regressão. Essas classes de aprendizado são usualmente empregadas para resolver problemas que podem ocorrer no futuro, como crescimento e queda de bolsa de valores, estoques e estimativas de vendas entre outras.

- **Simulação:** além das técnicas supracitadas, pode existir situações em que os dados podem ser incertos. Para resolver tais problemas é necessário simular uma grande quantidade de alternativas. Portanto, os cientistas de dados combinam uma ou mais das técnicas anteriores para resolver problemas como criação de drogas, buscas na Web, extração de informações, compreender o comportamento de clientes, entre outros.

- **Otimização:** em termos gerais, é um mecanismo para fazer algo melhor ou definir um contexto para uma solução que o torne adequado. Ao contrário do caso de simulações em que há incerteza associada à entrada de dados, na otimização existem dados de entrada sobre as dependências de relações entre atributos.

Os algoritmos de AM são aplicados em problemas que não podem ser resolvidos através de algoritmos específicos. No entanto, se houver dados históricos disponíveis é possível prever ou derivar tarefas usando essas técnicas de AM, entretanto, conforme exposto nos itens acima é necessário definir quais tipos de problemas precisam ser resolvidos [18]. Em outras palavras, com base no armazenamento do comportamento dos aplicativos ao longo de um período de tempo, estas informações podem ser utilizadas como dados de entrada para os algoritmos de AM, tornando possível a construção de modelos que realizem a classificação, simulação e até mesmo a previsão do comportamento futuro de aplicações [2].

Desta forma, é compreensível que a abordagem de AM esteja sendo amplamente adotada para estabelecer modelos de previsão motivados pelo poder de trabalhar com cargas de trabalho não lineares, consequentemente, promovendo resultados promissores [25]. Portanto, com base nas declarações supracitadas, escolhemos as técnicas de AM para explorar durante o desenvolvimento desta dissertação.

3. PREVISÃO DE CARGA COM APRENDIZADO DE MÁQUINA

3.1 Contextualização de Aprendizado de Máquina

Atualmente diversas áreas da computação vem aderindo ao uso de técnicas de AM, isso proporcionado pelo alto poder computacional disponível na atualidade, devido a capacidade de resolução de diversos tipos de problemas. Sendo assim, esta dissertação busca resolver problemas de previsão de carga de trabalho, portanto aborda um tipo específico de algoritmos de AM conhecido por algoritmos supervisionados, ou seja, precisam consumir dados históricos para induzir um modelo que seja capaz de fazer previsões. Desta forma, utilizaremos dados históricos da demanda de aplicações como entrada para nossos algoritmos. Os conjuntos de dados de entrada nesse contexto também são chamados de conjuntos de dados "rotulados". Algoritmos classificados nesta categoria concentram-se em estabelecer uma relação entre os atributos de entrada e saída e usar esse relacionamento especulativamente para gerar uma saída para novos pontos de dados de entrada [18].

O aprendizado no contexto de AM pode ser caracterizado de forma simples, como dados históricos ou observações utilizadas para realizar previsões ou derivar tarefas. A seguir são descritas algumas considerações que definem um problema de aprendizagem [18]:

- Fornecer uma definição do que a "aprendizagem" deve aprender e a necessidade de aprender.
- Definir os requisitos de dados e as fontes dos dados.
- Definir se o método de aprendizado deve operar no conjunto de dados na íntegra ou em um subconjunto.

Para compreender cada tipo de aprendizagem será necessário ter o conhecimento deste processo, no qual envolve a construção e validação de modelos que resolvam um problema com o máximo de precisão.

Para que ocorra a aprendizagem de máquina são considerados dois tipos de *datasets*. O primeiro *dataset é* geralmente preparado manualmente, onde os dados de entrada e de saída estão disponíveis e preparados. Neste caso, cada entrada tem um ponto de dados de saída que será usado de forma supervisionada para construir a regra [18]. O segundo conjunto de dados representa os de entrada, o qual é usado para prever o resultado esperado, afim de testar se o modelo conseguiu aprender durante a fase de treinamento.

Como primeira etapa, os dados podem ser divididos em dois (*dataset* de treino e testes) ou três partes, caracterizando os *datasets* de treinamento, validação e testes. Não

existe uma regra fixa que define um percentual de cada *dataset*. Podendo ser 70-10-20, 50-25-25, ou qualquer outra combinação de valores. [18].

O *dataset* de treinamento é utilizado para aprender ou construir por exemplo, um preditor ou classificador. O *dataset* de validação refere-se aos exemplos de dados que são verificados contra o preditor/classificador incorporado, esses podendo ajudar a sintonizar a precisão da saída.

Existem tipicamente três fases para realizar a aprendizagem de máquinas [18]:

- **Fase 1 - Fase de treinamento:** esta é a fase onde os dados de treinamento são usados para treinar um modelo emparelhando a entrada fornecida com a saída esperada. O resultado desta fase é o próprio modelo de aprendizagem.
- **Fase 2 - Fase de validação e teste:** esta fase é utilizada para medir o quão bom o modelo de aprendizagem foi treinado, e estima as propriedades do modelo, tais como medidas de erro, precisão e outros. Esta fase utiliza o conjunto de dados de validação e a saída é um modelo de aprendizado sofisticado.
- **Fase 3 - Fase de aplicação:** nesta fase, o modelo está sujeito à dados do mundo real para os quais os resultados devem ser derivados.

A Figura 3.1 ilustra o processo de como a aprendizagem pode ser aplicada para prever o comportamento, com base nos tópicos supracitados.

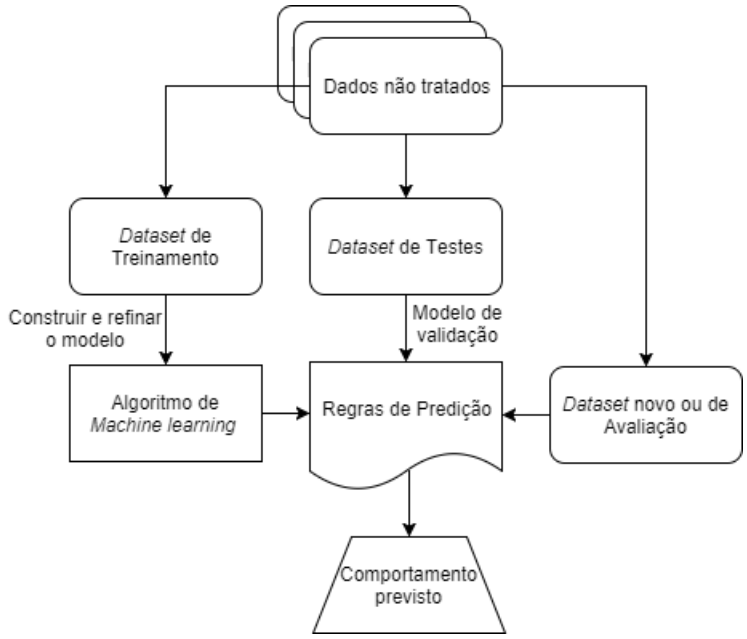

Figura 3.1 – Exemplo de processo de aprendizagem
Fonte: baseado em GOLLAPUDI, 2016, p. 6.

No entanto, a configuração inicial de um algoritmo de AM é considerada complexa, pois existem diversos parâmetros que devem ser ajustados para que o modelo de AM funcione da maneira adequada ao problema proposto. Portanto, existem conceitos e definições técnicas que precisam ser levadas em consideração ao construir modelos, como seguem [13]:

- **Amostra/observação:** número de elementos de um conjunto de dados. Em nosso caso quantidade de requisições realizadas por minuto.
- **Neuron:** neurônio artificial é um componente que recebe entrada de uma ou mais fontes, sendo outros neurônios ou entrada de dados de um *dataset*. O neurônio multiplica cada uma dessas entradas por um peso. Em seguida, adiciona as multiplicações e passa a soma para uma função de ativação.
- **Epoch:** geralmente definido como "uma passagem sobre todo o conjunto de dados", usado para separar o treinamento em fases distintas.
- **Batch/batch size:** um conjunto de *N* amostras expostas ao modelo, como exemplo, 20 amostras em um *dataset*, e *batch size* com tamanho de 5 amostras, para expor todo o *dataset* ao modelo, será necessário 4 batchs *(batch 5 x 4 = 20 amostras)*, e este ciclo caracteriza uma *Epoch*(Época). As amostras em um *batch* são processadas independentemente em paralelo.
- **Dropout:** consiste em configurar aleatoriamente uma taxa de fração de unidades de entrada para 0 a cada atualização durante o tempo de treinamento, o que ajuda a evitar o *overfitting*.

Os itens descritos acima são chamados de parâmetros ou hiperparâmetros. Eles estão presentes na maioria dos modelos de redes neurais e precisam estar ajustados para que os modelos ofereçam previsões coerentes. Para cada aplicação ou nova carga de trabalho o processo de ajuste dos hiperparâmetros deverá ser realizado novamente.

Para que seja possível verificar a qualidade e efetividade das previsões, os modelos são avaliados com base em métricas predefinidas, ou seja, para cada tipo de problema existem métricas especificas. Portanto, na seção seguinte serão abordadas algumas métricas especificas utilizadas para a avaliação da qualidade dos preditores.

3.2 Métricas de avaliação

A avaliação de um algoritmo de AM supervisionado é realizada por meio da análise do desempenho do preditor gerado por ele durante a previsão de novos dados, os quais não foram apresentados previamente em sua fase de treinamento [15].

Existem métricas especificas para cada tipo de problema, no entanto, esta seção aborda métricas especificas destinadas a avaliação de problemas de regressão, onde são obtidas a partir da comparação dos valores das previsões dos modelos com os valores reais/observados [18]. Os valores resultantes destas métricas são sempre não-negativos e valores próximos de zero são melhores, exceto a métrica R^2, que será detalhada ao longo desta Seção [35].

3.2.1 Erro Quadrático Médio (MSE)

Para calcular o MSE, primeiro elevamos o quadrado da diferença entre os valores observados e previstos de cada registro. Então, calculamos o valor médio desses erros quadrados. Se o valor previsto do registro i^{th} for P_i e o valor real for A_i, dividido por n que corresponde a um número de observações, então o MSE é a equação 3.1 [15]:

$$MSE = \frac{\sum_{i=1}^{n}(P_i - A_i)^2}{n} \tag{3.1}$$

3.2.2 Erro Quadrático Médio da Raiz (RMSE)

Também é comum usar a raiz quadrada do MSE chamada RMSE que representa o desvio padrão da amostra das diferenças entre os valores previstos e os valores observados (chamados residuais) [35]. Matematicamente é representado pela fórmula 3.2:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(P_i - A_i)^2}{n}} \tag{3.2}$$

3.2.3 Erro Absoluto Médio (MAE):

MAE é a média da diferença absoluta entre os valores previstos e os valores observados. O MAE é uma pontuação linear, o que significa que todas as diferenças individuais são ponderadas igualmente na média, conforme equação 3.3 [18].

$$MAE = \frac{\sum_{i=1}^{n}|P_i - A_i|}{n} \tag{3.3}$$

A distinção entre o MAE e o RMSE, é dado pela minimização do erro ao quadrado sobre um conjunto de números que resulta em encontrar sua média e minimizar os

resultados do erro absoluto na localização de sua mediana. Essa é a razão pela qual o MAE é robusto para outliers, enquanto o RMSE não é. No entanto, mesmo depois de ser mais complexo e enviesado para desvios mais altos, o RMSE ainda é a métrica padrão de muitos modelos, porque a função de perda definida em termos de RMSE é suavemente diferençável e facilita a execução de operações matemáticas.

3.2.4 R ao quadrado ou R^2:

R ao quadrado, originalmente empregada pelas áreas da estatística, é utilizada para avaliar a precisão de modelos matemáticos. Dentro do contexto de AM esta métrica estima até que ponto as variáveis independentes selecionadas explicam a variabilidade das variáveis dependentes [11]. Os valores de referência variam entre 0 e 1, ou seja, um modelo com R^2 : 0.05 é considerado insuficiente se comparado com outro que seja 0.83. Esta métrica indica o quão preciso é o modelo de AM [11]. Diferente da métrica de RMSE que é utilizado para comparar a precisão da previsão alcançada. A respectiva métrica é representada pela equação 3.4.

$$\hat{R}^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y}_i)^2} = 1 - \frac{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y}_i)^2} \tag{3.4}$$

onde $\hat{Y}_i$ denota o valor previsto de Y_i e $\bar{Y}$ denota a média da amostra de Y.

As seções a seguir descrevem três técnicas de aprendizado de máquina que mostraram resultados promissores em trabalhos relacionados em cenários semelhantes ao que foi explorado nesta pesquisa [25][31][35]. Eles serão comparados uns aos outros em nossos experimentos no Capítulo 4.

3.3 Técnicas de Aprendizado de Máquina

Na literatura diversos algoritmos de AM estão sendo propostos com o objetivo de otimizar a previsão de cargas de trabalho, no entanto, alguns algoritmos específicos têm adquirido resultados promissores nesta área, como o modelo ARIMA utilizada por [10], ANN proposta por [25] e GRU implementada por [34]. Por conseguinte, escolhemos estes algoritmos para realizar a comparação de seus respectivos prós e contras. A seção seguinte abordará cada uma das técnicas utilizadas durante o desenvolvimento desta dissertação.

3.3.1 ARIMA

Uma série temporal é uma sequência de observações feitas ao longo do tempo, que geralmente são dependentes umas das outras [8]. Assim, entender como essa dependência funciona é crucial para encontrar uma técnica que possa se ajustar aos dados analisados e prever etapas futuras. Entre vários métodos disponíveis para essa finalidade, existe o modelo ARIMA, que representa a classe de modelos estatísticos para análise e previsão de series temporais. De acordo com Messias et al. [31], ARIMA significa Média Móvel Integrada Auto Regressiva e pode ser aplicada em diversas áreas que demandam séries temporais [4]. O acrônimo ARIMA(p, d, q) é composto de três partes que se referem a modelos de previsão mais simples, cada um sendo adequado para detectar um comportamento específico [17], como segue:

- **AR (p):** O primeiro deles é o componente (Autoregressão), que é responsável pelo modelo da dependência entre cada observação e suas observações defasadas. Se *p* é 0, denota que não há padrão de autorregressão no modelo respectivo.

- **I(d):** Para aproveitar as propriedades estatísticas necessárias para ajustar o ARIMA, a "estacionariedade"é uma característica desejável. Portanto, o segundo componente (Integrado) representa o uso da diferenciação (isto é, subtraindo uma observação no tempo *t* do *d* anterior) para transformar a série temporal em estacionária, mas quando *d* é 0, uma vez que implica que a série já esteja estacionária.

- **MA (q):** A última parte é responsável por identificar a dependência entre observações e erros residuais, com base em uma média móvel da ordem *q* (ou seja, um intervalo de *q* unidades de tempo).

De acordo com Box & Jenkins [8], o modelo ARIMA(p, d, q) usado para a previsão de séries temporais é estabelecido como:

$$\Phi_p B(1-B)^d Z_t = \Theta_0 + \Theta_q(B) a_t \tag{3.5}$$

Onde: p, d, q, t são inteiros positivos, Z_t é a variável de interesse da série temporal $B^i(Z_t) = Z_{t-i}$ = z t-i . B é chamado o operador lag, Φ_p são os parâmetros do componente Autorregressivo de ordem p, $\Theta_0 + \Theta_q$ são os parâmetros da Média Móvel do componente de ordem q , $B(1-B)^d$ é o componente Integrado de d grau, a_t é o componente aleatório, também conhecido como ruído do modelo [10][31][8].

3.3.2 ANN

As Redes Neurais Artificiais (ANN) foram inspiradas na estrutura das redes neurais biológicas [18] e pertencem a uma classe de técnicas de correspondência de padrões. A ANN tem um comportamento *feedforward*, formando um gráfico acíclico direcionado. Segundo Russel e Norvig [38], as redes neurais são compostas de nós ou unidades conectadas por links diretos, as propriedades da rede são determinadas por sua topologia e pelas propriedades dos neurônios.

A Figura 3.2 mostra um modelo composto de três camadas principais que é conhecido como *Multi Layer Perceptron* (MLP): camadas de entrada, ocultas e de saída.

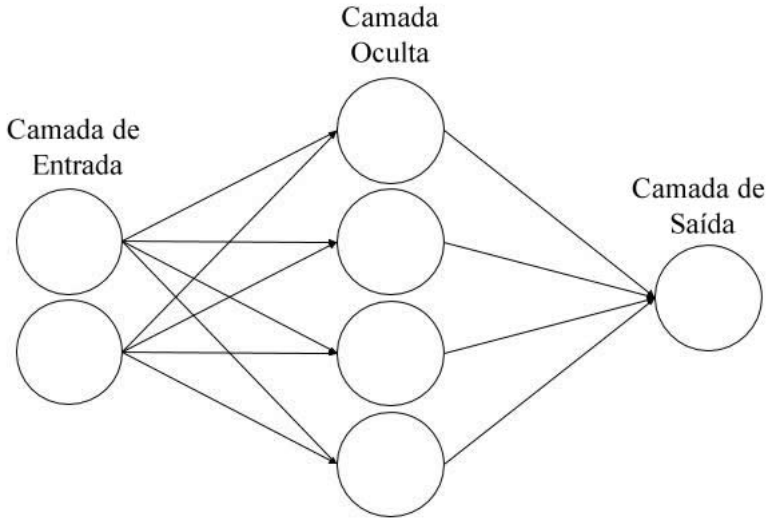

Figura 3.2 – Rede Neural Artificial com três camadas

A camada de entrada é onde a informação externa é recebida (apenas números), a camada oculta é onde cada operação do neurônio é calculada (através das funções de ativação) e a camada de saída é onde a solução de previsão é fornecida [45].

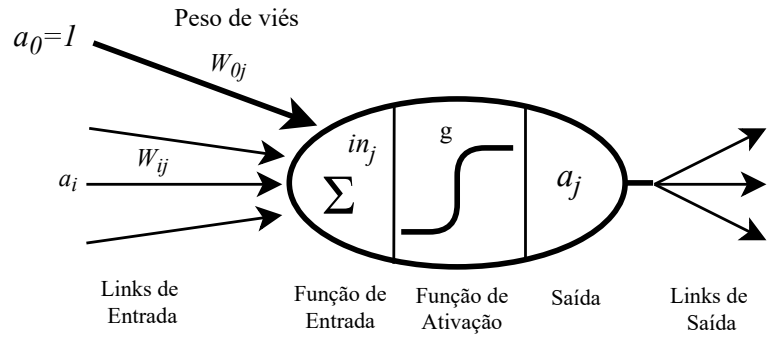

Figura 3.3 – Modelo simplificado de um neurônio, adaptado de Russel e Norvig [38]

A Figura 3.3 mostra um modelo matemático simples para um neurônio [38]. Um link da unidade *i* para unidade *j* serve para propagar a ativação a_i de *i* para *j*. também tem

um peso numérico $w_{i,j}$ associado a ele, que determina a força e o sinal da conexão. Assim como nos modelos de regressão linear, cada unidade tem uma entrada simulada $a_0 = 1$ com um peso associado $w_{0,j}$. Cada unidade j primeiro calcula uma soma ponderada de suas entradas [38]:

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i \tag{3.6}$$

Em seguida, aplica-se uma função de ativação g a essa soma para derivar a saída:

$$a_j = g(in_j) = \left(\sum_{i=0}^{n} w_{i,j} a_i \right) \tag{3.7}$$

Depois de definir a estrutura da ANN, como o número de camadas e neurônios, a rede deve ser treinada, assim refinando a saída e a função de erro para dar respostas confiáveis.

3.3.3 GRU

A abordagem das Redes Neurais Recorrentes (RNN) tem o mesmo processo de construção que a ANN. No entanto, os RNNs têm conexões entre as unidades formando um ciclo dirigido para que possam "lembrar" de coisas importantes sobre a entrada que receberam, o que permite que sejam precisas na previsão do que vem a seguir. Geralmente esta abordagem é adotada para tarefas de curto prazo, como reconhecimento de fala, compreensão de linguagem, processamento de linguagem natural (NLP), aplicações em reconhecimento de sequência, produção ou predição de séries temporais [12], [3]. Todavia, os RNNs simples são difíceis de treinar para capturar dependências de longo prazo de conjuntos de dados sequenciais longos, porque o gradiente geralmente desaparece após várias etapas [3]. Portanto, Cho et al. [12], propôs o modelo *Gated Recurrent Unit* (GRU), esse modelo possui duas camadas de portas (gates) conforme ilustra a Figura 3.4: redefinir *gates* e atualizar *gates* [12].

A camada oculta GRU no momento t é definida seguindo-se [39].

$$r_t = \sigma(x_t W_{xr} + h_{t-1} W_{hr} + b_r) \tag{3.8}$$

$$z_t = \sigma(x_t W_{xz} + h_{t-1} W_{hz} + b_r) \tag{3.9}$$

$$\tilde{h}_t = f(x_t W_{xh} + (r_t \odot h_{t-1}) W_{hh} + b_h) \tag{3.10}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{3.11}$$

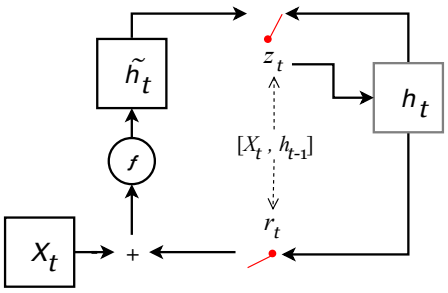

Figura 3.4 – Gated Recurrent Unit, adaptado de Tjandra et al. [39]

Onde $\sigma(\cdot)$ é uma função de ativação sigmoide, r_t, z_t são redefinidos e atualizado os *gates*, $\tilde{h}_t$ são os valores de camada oculta candidata e h_t são os valores da camada oculta no tempo-t. Os portões de reinicialização determinam qual valor de camada oculta anterior é útil para gerar a camada oculta candidata atual. Os *gates* de atualização mantêm os valores de camadas ocultas anteriores ou são substituídas por novos valores de camadas ocultas candidatas [39].

3.4 *Tuning* de Hiperparâmetros

Os algoritmos de aprendizado podem ser aplicados em diversos contextos. Porém, para que os resultados sejam eficientes é necessário realizar o ajuste dos hiperparâmetros do modelo de AM. Os hiperparâmetros correspondem a variáveis diretamente relacionadas aos dados de treinamento, como exemplo, a definição de quantos neurônios cada camada do modelo vai possuir, e a quantidade de amostras do *dataset* que será exposto ao modelo a cada iteração de treinamento, ou seja, são variáveis de configuração dos modelos de AM [32], conforme previamente introduzido na Seção 3.1.

Segundo Diaz et al. [14] um grande desafio na concepção de sistemas de AM é determinar a melhor estrutura e parâmetros para a rede neural, baseado nos dados disponíveis que serão usados para construir o modelo. Os hiperparâmetros podem ser escolhidos com base em regras heurísticas e ou ajustados manualmente para acelerar a descoberta de configurações. Esta etapa é crucial para a implementação de um algoritmo de aprendizado eficaz, no entanto, é conhecido por ser um processo oneroso. A abordagem usual é tentar diversos conjuntos de hiperparâmetros e escolher aquele que executa os melhores resultados [1].

A expressão utilizada por alguns autores: *No Free Lunch Theorem* mais conhecido como "não existe almoço grátis", corresponde a uma afirmação de que não existe um algoritmo de aprendizagem único que, em qualquer domínio, induza sempre o preditor mais preciso [1]. A definição dos hiperparâmetros é ainda um problema em aberto na área da aprendizagem de máquina, mas ao longo dos anos foram desenvolvidas algumas aborda-

gens que procuram minimizar o processo de definição dos hiperparâmetros. Ao longo desta Seção serão descritas algumas abordagens utilizadas.

3.4.1 Técnicas de *Tuning* de Hiperparâmetros

Grid Search:

Grid Search é um algoritmo utilizado para a otimização de hiperparâmetros que consiste em uma pesquisa em grade. Portanto, é preciso definir um conjunto de valores de parâmetros, onde o modelo será treinado para todas as combinações de parâmetros possíveis, e no final vai indicar a melhor combinação. Segundo Bergstra et al. [6], o *grid search* é a estratégia mais utilizada para a utilização de hiperparâmetros. No entanto, este método é uma boa escolha quando o modelo pode ser treinado rapidamente, pois as execuções em grade são exponenciais no número de dimensões da pesquisa, nos quais tornam irrelevantes para um determinado conjunto de dados [32]. A Figura 3.5 ilustra a pesquisa em *grid* para a descoberta dos parâmetros do modelo ARIMA, com valores do parâmetro *p* variando entre (0, 1, 2, 4, 6, 8, 10), *d* e *q* entre 0 a 3. O método realiza a avaliação de todas as possibilidades e no final indica a opção mais precisa.

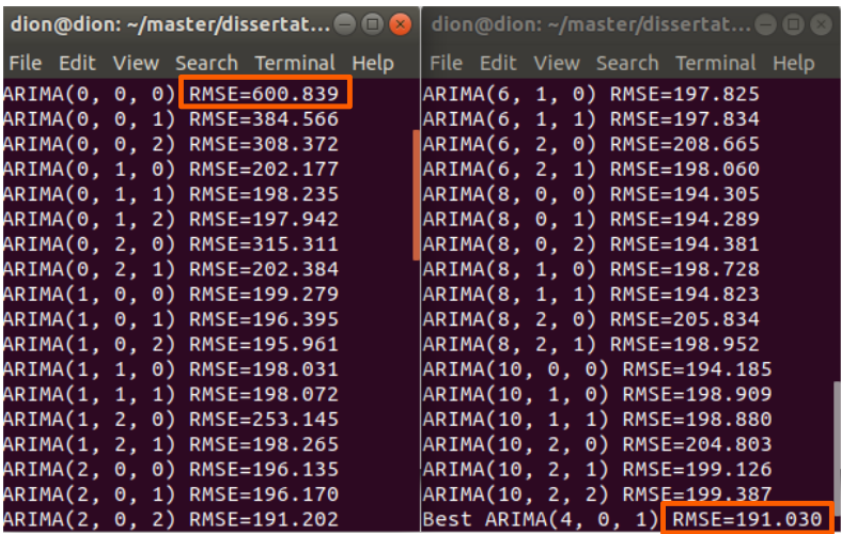

Figura 3.5 – Exemplo de pesquisa em *grid* para o modelo ARIMA. [39]

Random Search:

Corresponde a um método que pertence aos campos de otimização estocástica e otimização global. O funcionamento desta estratégia é amostrar soluções de todo o espaço de pesquisa usando uma distribuição de probabilidade uniforme. Desta forma, ao invés de testar todas as combinações possíveis, são selecionados randomicamente os conjuntos de

parâmetros. Cada amostra futura é independente das amostras que vêm antes da amostra atual [9]. Este método pode retornar uma aproximação razoável da solução ótima dentro de um tempo razoável sob baixa dimensionalidade do problema. Olhando a perspectiva do pior caso, uma vez que a pesquisa aleatória não tem memória e pode ser recapturada cegamente, não existe garantias do alcance de uma solução ótima para o problema [9].

Hand *tuning*:

Consiste no processo de definição de hiperparâmetros, treinamento e teste de forma manual realizada pelo desenvolvedor, utilizando o conhecimento empírico, baseado nas experiências de implementações anteriores. Podendo ser combinada com outras técnicas como exemplo: *Grid Search* e *Random Search*.

Bayesian optimization:

Para esta otimização é utilizado o Processo Gaussiano com Função de Aquisição. A ideia por trás do Processo Gaussiano é que para cada entrada x temos saída $y = f(x)$, onde f é uma função estocástica. Esta função corresponde a saída de uma distribuição gaussiana. Além disso, podemos dizer que cada entrada x tem uma distribuição gaussiana associada. O que significa que para cada entrada o processo x gaussiano definiu-se a média μ e desvio padrão σ para alguma distribuição gaussiana. O processo Gaussiano seleciona um conjunto de parâmetros com base na melhor observação, no entanto, as redes neurais geralmente envolvem a randomização na inicialização de pesos durante o processo de treinamento. A execução da rede neural com os mesmos parâmetros pode levar a pontuações diferentes, o que significa que a nossa melhor pontuação nem sempre pode ocorrer [36].

Tree of Parzen Estimators ou TPE:

Os estimadores *Parzen* estruturados em árvore (TPE) corrigem as desvantagens do processo gaussiano. De acordo com Madrigal et al. [28] cada iteração TPE coleta novas observações e no final da iteração, o algoritmo decide qual conjunto de parâmetros deve tentar em seguida. O TPE desenha amostras uniformemente no espaço de configuração, portanto, não requer estimativas iniciais ou conjuntos de treinamento. Então, as amostras são avaliadas de acordo com a função de custo. A abordagem divide as amostras em dois grupos com base em sua pontuação. O primeiro grupo tem todas as amostras que melhoram a estimativa atual do *score*, enquanto o segundo contém o restante, portanto, ambos os grupos são usados para modelar a probabilidade do melhor conjunto de hiperparâmetros [28].

O algoritmo TPE (Tree of Parzen Estimators) está incluído na biblioteca *Hyperopt*, onde é possível definir um intervalo de busca para cada hiperparâmetro da rede neural de interesse, conforme destacado na Figura 3.6. As árvores lidam naturalmente com parâmetros condicionais, são rápidas de treinar e podem ser atualizadas incrementalmente em dados online [7] [5].

```
87  def model(X_train, Y_train, X_test, Y_test):
88
89      lookback = 1
90      model=Sequential()
91      model.add(Dense({{choice([30,60,80])}}, input_dim=lookback, activation='relu'))
92      model.add(Dropout(0.2))
93      model.add(Dense({{choice([10,30,60])}}, activation='relu'))
94      model.add(Dropout(0.2))
95      model.add(Dense(1))
96
97      model.compile(loss='mean_squared_error', optimizer="adam", metrics=['mae'])
98
99      model.fit(X_train, Y_train,
100                batch_size={{choice([130,180])}},
101                nb_epoch={{choice([130,200])}},
102                verbose=2,
103                validation_data=(X_test, Y_test))
104     score, mae = model.evaluate(X_test, Y_test, verbose=0)
105     print('Test mae:', mae)
106     return {'loss': -mae, 'status': STATUS_OK, 'model': model}
```

Figura 3.6 – Exemplo de implementação do modelo MLP com o uso da biblioteca Hyperopt para otimização do modelo com o método TPE

Considerações na escolha de Algoritmos de *Tuning* de Hiperparâmetros

Conforme descrito anteriormente, existem esforços na literatura como [5][36][9][32] que abordam diferentes métodos para realizar a otimização dos hiperparâmetros das Redes Neurais. Estas abordagens oferecem resultados satisfatórios que possibilitam a implementação de diversas soluções de AM. Os algoritmos de *tuning* de hiperparâmetros, ajudam a encontrar parâmetros consistentes, mas não garantem que os mesmos sejam realmente os ideais para o respectivo modelo, exigindo com que os desenvolvedores executem os algoritmos diversas vezes, e com diferentes parametrizações, para que sejam encontrados conjuntos de parâmetros ideais para o problema proposto.

Como exemplo, ao utilizar o método de *grid search* para encontrar os hiperparâmetros de uma rede neural, não é compensador, devido ao tempo de execução elevado para executar o cruzamento de todos os valores de hiperparâmetros existentes na rede neural. Portanto, trata-se de um processo exponencial na medida que novos parâmetros são adicionados. Por conseguinte, seria necessário realizar novas execuções para encontrar um conjunto de hiperparâmetros ideal. No contexto do modelo ARIMA, pelo fato de possuir poucos hiperparâmetros, o uso do método de *grid search* acaba compensando, pois o modelo exige apenas a definição de três parâmetros com valores finitos de 0 a 10.

Dessa forma, ao iniciar no processo de otimização de hiperparâmetros, é recomendável realizar uma análise de qual abordagem é mais adequada para a resolução do

problema alvo, sendo assim, proporcionando a economia de tempo e esforço para a adequação de modelos de AM.

4. AVALIANDO AS TÉCNICAS DE APRENDIZAGEM DE MÁQUINA EMPREGADAS EM AMBIENTES DE NUVEM

Conforme mencionado por Singh e Chana [37] a computação em nuvem suporta aplicações com diversas características, sejam escaláveis, intensivas em dados, intensivas em rede, intensivas em processamento, com fluxos de trabalho científicos, e até mesmo múltiplas camadas.

Dentro deste contexto, as arquiteturas de software foram evoluindo para suportar a demanda de processamento e acessos, como exemplo, temos as aplicações web que possuem diversas camadas, onde cada camada é responsável por executar uma tarefa específica. Por exemplo, como manipular a autenticação do usuário, acessar o banco de dados, acessar os mais variados serviços. Além disso, as camadas podem depender de outras camadas para serem executadas e devem estabelecer comunicação através da rede para responder a cada solicitação. Essa abordagem de desenvolvimento foi remodelada para suportar a demanda de escalabilidade. Dado as características de uso, as aplicações web recebem requisições de serviço, onde variam amplamente com base em fatores como a hora do dia e eventos inesperados, os quais podem acionar multidões, como exemplo, promoções, feriados e tendências de mercado [20].

Diante da não linearidade[1] no uso das aplicações web, algumas abordagens precisam ser implementadas para que não ocorra penalidades de QoS e nem quebra de SLA. Para Huang et al. [20] o planejamento de capacidade é um método clássico para determinar o número de recursos para o requisito de QoS fornecido. No entanto, os autores também explicam que o planejamento de capacidade é uma decisão de longo prazo e praticamente estática, portanto, os recursos são determinados pela taxa máxima de solicitação no caso das aplicações web para que sejam evitadas a cobrança de multas excessivas. Desta forma, a taxa máxima de solicitações de aplicativos podem ser estimadas de acordo com um modelo de previsão baseado em dados históricos, isso com o objetivo de transformar uma abordagem estática em uma abordagem proativa [20]. Partindo dessa colocação Kumar et al. [25] também menciona que a previsão da carga de trabalho é uma das possibilidades pelas quais a eficiência e os custos operacionais de uma nuvem podem ser melhorados.

Portanto, com o objetivo de avaliar cargas de trabalho não lineares e que permitam explorar as características das técnicas de AM em estudo, optamos por executar em nossos experimentos cargas de trabalho referentes a aplicações web. Essas cargas de trabalho também são exploradas em trabalhos relacionados na literatura como [25] e [10]. Ao longo das seções serão descritas as etapas realizadas durante a análise de custo benefício entre as três técnicas que são utilizadas para realizar a previsão de carga de trabalho

[1] Não linearidade representa a falta de sazonalidade ou padrões no uso das aplicações, ou seja, o acesso das aplicações podem ser promovidas por eventos externos, como exemplo, promoções, noticias, tendências mercadológicas.

encontradas na literatura, as quais alcançaram resultados promissores na resolução do problema proposto. As técnicas abordadas correspondem as siglas ARIMA, MLP e GRU, sob diferentes configurações conforme definidas no Capítulo 3.

4.1 Ambiente dos experimentos

Os experimentos foram executados em uma máquina equipada com um processador Intel Core I7-6500U com 4 núcleos e velocidade de *clock* de 2.50 GHz e 16 GB de memória. Para a execução dos algoritmos desenvolvidos nesta dissertação foi utilizado a linguagem Python 2.7, e as seguintes bibliotecas utilizadas na área de AM:

- Keras[2]
- SKlearn[3]
- Hyperopt[4]
- Statsmodels[5]
- Matplotlib [6]
- Pandas[7]
- Numpy[8]

Também utilizou-se o Google *Colaboratory*[9], o qual se apresenta como uma ferramenta para educação e pesquisa em AM. Esse sendo um ambiente de notebook Jupyter que não requer configuração para utilização.

Para definir o *benchmark* utilizado em nossos experimentos analisou-se as características dos *benchmarks* que são usados em trabalhos relacionados, como [25] o *dataset* possui apenas os HTTP *requests* da NASA encontrados em [22]. Escolhemos este *dataset* justamente por representar a não linearidade das requisições HTTP de aplicações web, diferente da utilização de dados sintéticos produzidos por *benchmarks* de simulação.

O HTTP da NASA contém dois *traces* separados de dois meses contendo todas as solicitações HTTP para o servidor WWW do Centro Espacial Kennedy da NASA na Flórida. Os *traces* são armazenados em arquivos ASCII com uma linha por solicitação.

[2]https://keras.io/
[3]https://scikit-learn.org
[4]http://hyperopt.github.io/hyperopt/
[5]https://www.statsmodels.org/stable/index.html
[6]https://matplotlib.org/
[7]https://pandas.pydata.org/
[8]http://www.numpy.org/
[9]https://colab.research.google.com

Foram gerados *datasets* com diferentes intervalos de tempo de acordo com a progressão (1, 5, 10, 15 até 60), onde cada número representa o intervalo de tempo em minutos. Os diferentes *datasets* foram definidos com o objetivo de verificar a aderência dos modelos ao executar previsões em diferentes escalas de tempo. Para preparar os *datasets* utilizou-se a operação de soma, consequentemente, cada amostra corresponde a soma de requisições feitas dentro do período de tempo estipulado conforme ilustrado pela tabela 4.1 que representa o *dataset* correspondente as previsões dos próximos 60 minutos.

Tabela 4.1 – Amostra do *dataset* com granularidade de 60 minutos.

Tempo 60min	**Requisições**
00:59:58	3565
01:59:56	3004
02:59:56	2268
03:59:58	1734
04:59:58	1482

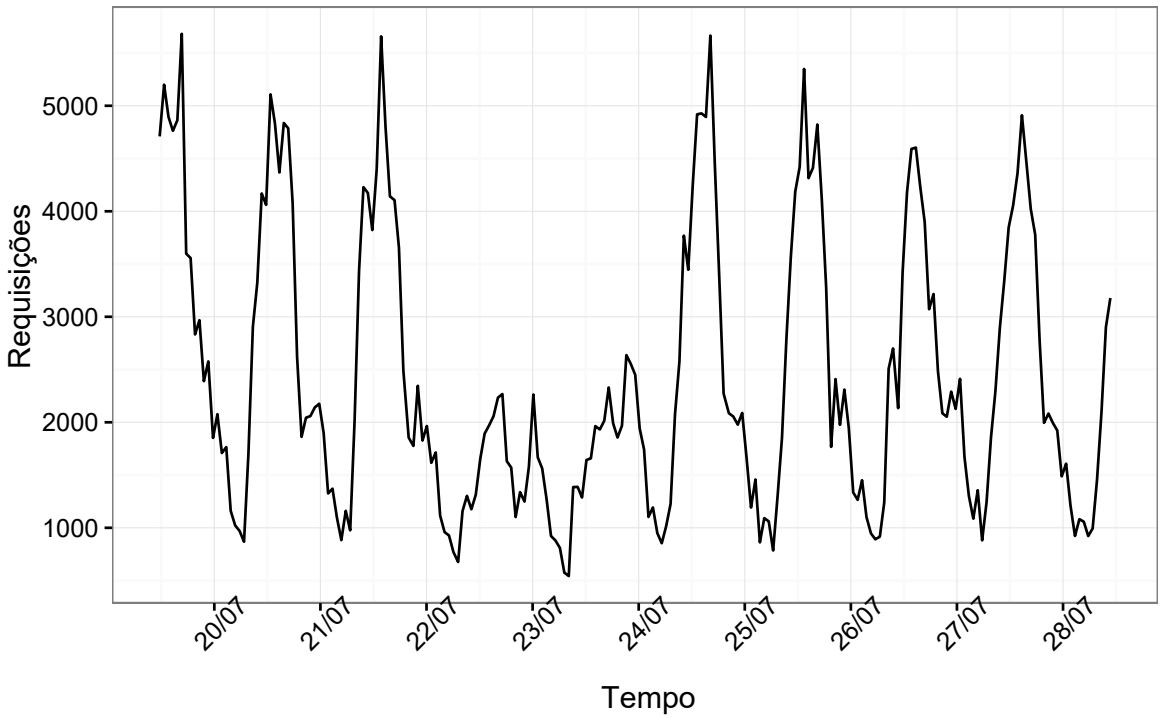

Figura 4.1 – Amostra do conjunto de dados HTTP da NASA com um intervalo de previsão de 60 minutos

Também empregou-se o *dataset* que corresponde as requisições HTTP do site Wikipédia com o objetivo de demonstrar a aderência dos modelos com diferentes cargas de trabalho; esse correspondendo aos dados históricos do mês de setembro de 2007 [40]. Assim como no *dataset* da NASA, também utilizou-se a operação soma para a geração dos *datasets* nos diferentes intervalos de previsão.

Após aquisição e preparação dos dados, se delimitou os *datasets* em duas partes, conforme mencionado na Seção 3.1 que descreve a divisão do *dataset* em dados de treinamento e dados de teste [18] antes de atribuir os dados históricos para os modelos preditivos. Portanto, todos os *datasets* foram divididos em duas partes, foi aplicado cerca de 60% dos respectivos *datasets* para a fase de treinamento e, os outros 40% para a fase de testes, conforme ilustrado pela Figura 4.2. O maior percentual de dados é atribuído para o *dataset* de treinamento para que os modelos tenham mais conteúdo para extrair padrões durante o processo de aprendizado.

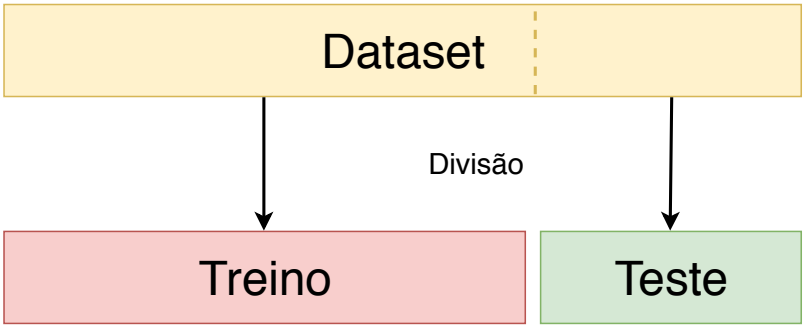

Figura 4.2 – Proporção da divisão do *dataset* de treino e teste

4.2 Avaliação dos modelos de aprendizagem de máquina

Com o objetivo de realizar a análise de custo benefício entre os modelos ARIMA, MLP e GRU disponíveis nos Apêndices (A,B,C), foram desenvolvidas avaliações para evidenciar as particularidades de cada modelo. Por conseguinte, se executou cada avaliação sumarizada na tabela 4.2 com o respectivo *Dataset* em diferentes intervalos de tempo, conforme mencionado na seção anterior. As subseções seguintes detalham cada avaliação realizada neste trabalho.

Tabela 4.2 – Experimentos executados nos modelos de AM

Avaliações realizadas	**Dataset**
4.2.1 Avaliação de estratégias de *tuning* para os modelos preditivos	NASA
4.2.2 Avaliação da influência da quantidade de dados históricos para o treinamento	NASA
4.2.3 Avaliação da capacidade de generalização no modelo ARIMA	NASA
4.2.4 Avaliação do tempo de execução dos modelos	NASA
4.2.5 Análise geral dos modelos com dataset NASA	NASA
4.2.6 Análise da precisão dos modelos de AM com *dataset* da Wikipédia	Wikipédia

4.2.1 Avaliação de estratégias de *tuning* para os modelos preditivos

As avaliações efetuadas nesta subseção consistem na execução dos algoritmos de *tuning* para os modelos preditivos, com o objetivo de explorar melhores combinações de hiperparâmetros.

Para o modelo ARIMA, os parâmetros $ARIMA(p, d, q)$ foram definidos com o resultado da execução do método *grid search* [32], o qual estima a melhor configuração de forma iterativa de revisão de diagnósticos, conforme detalhado na seção 3.4 e no código fonte do método, que pode ser consultado no Apêndice D. A tabela 4.3 sumariza os hiperparâmetros retornados pela execução do método *Grid search*.

Tabela 4.3 – Hiperparâmetros utilizados nos experimentos

Intervalo de Prev.(min)	**ARIMA(p,d,q)**
1	(0, 1, 2)
5	(2, 0, 1)
10	(0, 1, 1)
15	(0, 2, 1)
20	(1, 0, 2)
25	(4, 0, 1)
30	(2, 0, 2)
35	(4, 0, 1)
40	(8, 0, 1)
45	(2, 0, 2)
50	(2, 0, 1)
55	(10, 0, 2)
60	(2, 1, 2)

Conforme descrito na Seção 3.4, o método *grid search* é ideal para modelos de AM com poucos parâmetros, exemplificado pelo modelo ARIMA, pois possui o tempo de execução elevado para realizar o cruzamento do maior número de hiperparâmetros existentes, assim como em outros modelos preditivos, por exemplo, as redes neurais MLP e GRU. Sendo assim, o método *grid search* apresenta processos exponenciais em relação ao número de parâmetros de entrada.

A parametrização de uma rede neural no caso dos modelos MLP e GRU não é uma atividade trivial; portanto, para dar apoio a esta atividade foi utilizado um algoritmo de otimização conhecido por *Tree of Parzen Estimator* (TPE), conforme descrito na Seção 3.4. A codificação desse algoritmo seguiu na premissa de desenvolvimento padrão das redes neurais MLP (Apêndice E) e GRU (Apêndice F) em conjunto com a biblioteca Hyperopt, os hiperparâmetros receberam um intervalo de valores ao invés de um valor fixo. Desta forma, o otimizador executa suas funções procurando o conjunto de parâmetros ideais referente

ao espaço amostral definido. A tabela 4.4, sumariza os hiperparâmetros retornados pela execução do método TPE.

Tabela 4.4 – Hiperparâmetros utilizados nos experimentos

MLP				
Intervalo de Prev.(min)	**Batch size**	**N. Epochs**	**Input Layer**	**Hidden layer**
1	200	130	100	10
5	110	180	50	25
10	200	130	100	10
15	130	200	60	30
20	110	190	50	16
25	130	200	60	30
30	130	200	60	30
35	130	200	60	30
40	130	200	60	30
45	130	200	60	30
50	130	200	60	30
55	130	200	30	60
60	130	130	30	30
GRU				
Intervalo de Prev.(min)	**Batch size**	**N. Epochs**	**Input Layer**	**Hidden layer**
1	140	150	8	64
5	130	130	128	64
10	140	100	16	32
15	140	190	5	40
20	140	200	50	45
25	180	200	10	80
30	140	190	5	40
35	180	200	10	80
40	130	130	256	32
45	140	150	8	64
50	130	130	60	60
55	130	130	128	64
60	130	130	128	64

Um dos pontos positivos dessa abordagem é sua execução, a qual se apresenta relativamente ágil. Assim, permitindo que o desenvolvedor realize novas execuções com diferentes espaços amostrais em busca do conjunto de hiperparâmetros mais precisos. Essa abordagem não requer estimativas iniciais ou conjuntos de treinamento, assim facilitando a obtenção de conjuntos de hiperparâmetros. A Tabela 4.5 contém os resultados da métrica RMSE que estima as previsões otimizadas dos três modelos preditivos em estudo.

Conforme Figura 4.3, os gráficos em linhas mostram uma visão mais detalhada das previsões realizadas no intervalo de previsão de 30 minutos, confirmando que as três técnicas tiveram a habilidade de trabalhar com a não linearidade da carga de trabalho de teste.

Tabela 4.5 – RMSE após *tuning* de hiperparâmetros

Intervalo de Prev.(min)	MLP	GRU	ARIMA
1	15,95	16,35	14,27
5	53,22	52,62	47,86
10	94,64	92,28	86,98
15	133,73	131,13	135,08
20	178,62	171,76	168,39
25	204,61	197,32	191,03
30	250,52	241,44	237,73
35	273,33	268,29	263,43
40	337,76	322,61	316,6
45	383,42	367,21	360,48
50	408,77	399,3	380,1
55	540,16	466,63	439,89
60	546,64	523,11	503,67

A Figura 4.4 representa a comparação da precisão da métrica RMSE entre os três modelos após o processo de tuning. Como observou-se, a precisão entre os modelos com intervalos de previsão baixos não possui diferenças; no entanto, nos intervalos mais longos há tendência que os modelos tenham diferenças de precisão. Levando em consideração as pequenas diferenças de precisão encontradas durante os experimentos, podemos concluir que ambos os modelos oferecem resultados equivalentes dentro das condições criadas no nosso ambiente de testes.

Apesar da pouca diferença entre as previsões, o método TPE não superou a precisão do modelo ARIMA, após o processo de *tuning* com o método *grid search*. Porém, vale ressaltar que o uso desse método é conhecido pelo consumo de tempo. Para a obtenção dos hiperparâmetros em todos os intervalos de previsão de acordo com a tabela 4.3, o método ficou em estado de processamento aproximadamente três semanas consecutivas para encontrar os hiperparâmetros para todos os intervalos de previsão.

Para os modelos MLP e GRU os hiperparâmetros considerados no algoritmo de otimização TPE foram: *batch size*, *epoch*, *units/neurons* da primeira e segunda camada. Além de existirem estudos que comprovem a utilidade do método TPE como [5] e [28], também acreditamos que seja válido realizar novas execuções com diferentes espaços amostrais referente aos hiperparâmetros, com o objetivo de explorar resultados mais apurados. Também podemos considerar que a utilização do método foi vantajoso, visto que, o uso do método manual de *tuning* desprende mais tempo e esforço da parte do desenvolvedor em relação a execução do método TPE, o qual encontrou valores de precisão equivalentes em questão de poucos minutos. Além disso, o método TPE representa uma abordagem automatizada que, pode ser facilmente aplicada para diferentes cargas de trabalho.

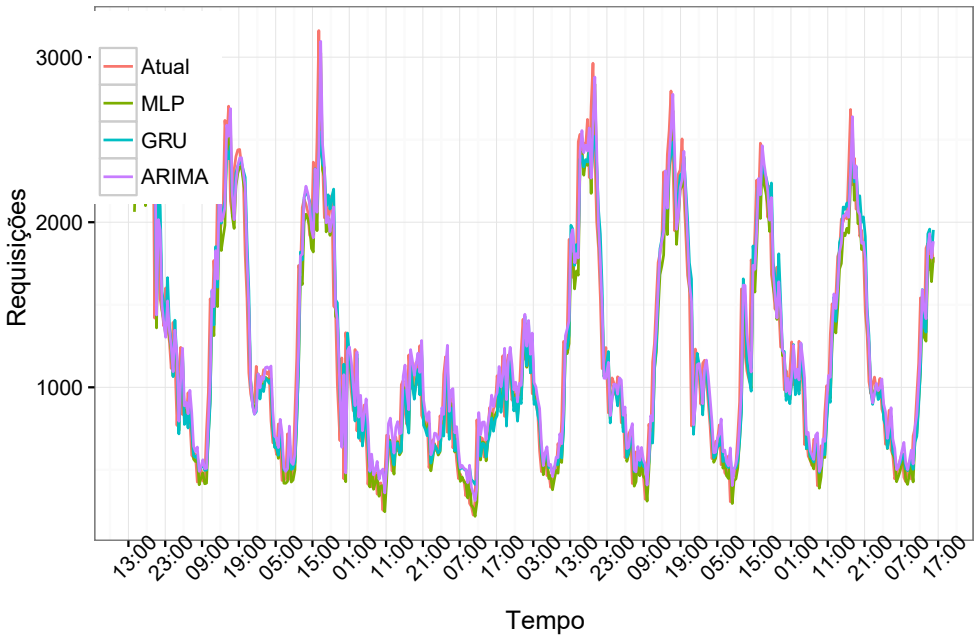

Figura 4.3 – Previsões com intervalo de 30 minutos

Na literatura, além das otimizações com hiperparâmetros, alguns autores como [25] e [31] fazem uso de algoritmos genéticos e/ou evolutivos com o objetivo de modificar a forma de treinamento das redes neurais, buscando otimizar o desempenho de seus algoritmos e adaptar para os seus respectivos contextos.

Baseado nas colocações realizadas nessa Seção, fica evidente que o *tuning* não é um processo trivial, porém ao fazer uso desses algoritmos de otimização é possível alcançar resultados que possam ser precisos o suficiente para a resolução do problema alvo.

4.2.2 Avaliação da influência da quantidade de dados históricos para o treinamento

Para induzir modelos proativos é necessário o uso de dados históricos para treinar os modelos de AM. No entanto, em determinadas situações é difícil adquirir dados históricos suficientes para realizar as previsões de carga, como exemplo, aplicações implantadas recentemente ou sem monitoramento prévio.

Desse modo, o objetivo da análise desta Subseção é medir o quão sensível as técnicas são em relação aos dados históricos. Procurou-se, consequentemente, avaliar a quantidade de dados históricos suficientes para que, o modelo consiga prever sem perder a qualidade das previsões, ou seja, sem aumentar o valor da métrica de RMSE. Executou-se os algoritmos de AM cerca de 10 vezes, no entanto, em cada execução reduziu o *dataset* de treinamento em 10%, conforme ilustrado pela figura 4.5.

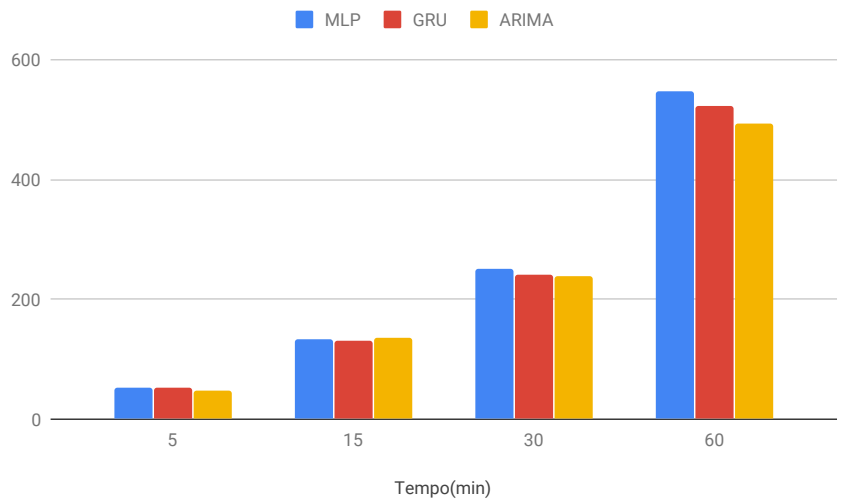

Figura 4.4 – Representação da diferença da precisão do RMSE entre os modelos após o processo de tuning

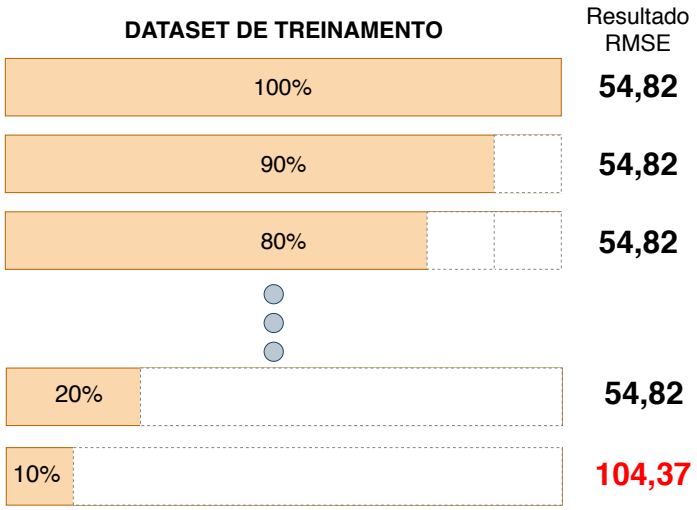

Figura 4.5 – Restrição do *dataset* de treinamento

Como resultado a Tabela 4.6 ilustra os experimentos que não apresentaram perda de qualidade das previsões. Para fins de comparação a tabela apresenta apenas os intervalos de tempo de 5, 15, 30 e 60 minutos.

A coluna *Dias hh:mm*, corresponde ao período em que os dados históricos precisam ser acumulados para oferecer amostras suficientes para o treinamento dos modelos de AM.

Para obter o valor da coluna *Dias hh:mm* foram feitos os seguintes passos:

- Dividido o número de amostras da coluna (*Amostras*) pela quantidade de amostras da coluna (*Interv. de Prev.*) acumuladas no período de 60 minutos(como exemplo: Em 60 min. existem 4 amostras de 15min, ou 2 de 30 min);

- O resultado do passo 1 então é multiplicado por 60 para extrair o valor de tempo em minutos referente a quantidade de amostras da coluna (*Amostras*);
- Converter o valor do passo 2 em dias, horas e minutos.

Tabela 4.6 – Número mínimo de amostras para previsões

Técnicas	Interv. de Prev.	RMSE	Amostras T.	Perc.%	Amostras	hh:mm	Dias hh:mm
MLP							
	5	54,52	5295	10%	530	44:10	1d 20:10
	15	136,20	1763	10%	177	44:15	1d 20:15
	30	249,80	880	10%	89	44:30	1d 20:30
	60	546,90	438	10%	44	44:00	1d 20:00
GRU							
	5	52,84	5295	20%	1060	88:20	3d 16:20
	15	134,33	1763	50%	883	220:45	9d 04:45
	30	243,11	880	30%	618	309:00	12d 21:00
	60	519,09	438	40%	176	176:00	7d 08:00
ARIMA							
	5	47,91	5296	20%	1060	88:20	3d 16:20
	15	135,63	1764	10%	177	44:15	1d 20:15
	30	239,95	881	50%	441	220:30	9d 04:30
	60	491,84	439	70%	132	132:00	5d 12:00

Como podemos ver na Tabela 4.6, o modelo MLP teve destaque neste experimento, pois ele conseguiu manter a eficiência das previsões na métrica RMSE com a menor quantidade de amostras de dados históricos. Dessa maneira, podemos considerar que dados históricos de aproximadamente 1 dia 20 horas e 30 minutos sejam suficientes para treinar o modelo em qualquer granularidade de tempo, considerando nosso ambiente de testes.

O modelo GRU, no entanto, apresentou a necessidade de maior quantidade de dados para a fase de treinamento, em comparação aos demais modelos para prover previsões adequadas. O modelo apresenta uma dependência em relação a quantidade de amostras no hiperparâmetro *batch size*, no qual influência os resultados das previsões. Para previsões dos próximos 5 minutos serão necessários dados de aproximadamente 3 dias e 16 horas e 20 minutos, e para os próximos 60 minutos serão necessários dados históricos de 7 dias e 8 horas, por exemplo.

Durante as execuções do modelo ARIMA tivemos falhas no experimento ao retirar alguns percentuais do *dataset*. Este padrão ocorreu em vários intervalos de previsão, ou seja, caso não exista dados suficientes é necessário realizar o treinamento novamente dos hiperparâmetros, para que o modelo se ajuste à serie temporal submetida. Consequentemente, pode-se concluir que, o modelo não se adapta automaticamente as cargas de trabalho em comparação as demais técnicas.

4.2.3 Avaliação da capacidade de generalização no modelo ARIMA

Realizou-se a comparação das três técnicas na Subseção 4.2.1 e se verificou que, os modelos MLP e GRU obtiveram resultados satisfatórios e equiparáveis ao ARIMA. De forma complementar, no experimento anterior Subseção 4.2.2 constatou-se que o modelo ARIMA não induziu o modelo de previsão adequadamente com a redução da quantidade de amostras na série temporal, caracterizando, dessa forma, a necessidade de atualizar os hiperparâmetros.

Tabela 4.7 – Comparação de RMSE na execução do ARIMA com hiperparâmetros fixos

Intervalo de Prev.	**ARIMA(Tuning)**	**ARIMA(0,1,1)**	**ARIMA(2, 0, 2)**	**ARIMA(0, 2, 1)**
5	47,86	48,64	47,88	54,67
10	86,98	86,98	86,22	**95,14**
15	135,08	125,95	**erro**	135,08
20	168,39	169,77	**erro**	178,88
25	191,03	197,86	191,22	**202,22**
30	237,73	**247,35**	237,73	**248,35**
35	263,43	**275,48**	263,48	**274,74**
40	316,6	**338,65**	317,47	**335,91**
45	360,48	**385,49**	360,48	**380,14**
50	380,1	**406,9**	380,47	**411,51**
55	439,89	**475,18**	**442,59**	**481,95**
60	503,67	**530,03**	**491,15**	**541,45**

Portanto, averiguou-se em mais detalhe a capacidade do ARIMA induzir um modelo preditivo ao fixar 3 conjuntos de hiperparâmetros. Na Tabela 4.7, a coluna *ARIMA(Tuning)* representa os valores de RMSE com *tuning* para todas as granularidades de tempo, sendo uma cópia dos valores da coluna ARIMA da Tabela 4.5. As demais colunas representam a execução de todos os intervalos de previsão com a respectiva configuração descrita. A coluna *ARIMA(0,1,1)* corresponde aos hiperparâmetros específicos para a granularidade de 10 minutos, a coluna *ARIMA(0, 2, 1)* a granularidade de 15 minutos e a coluna *ARIMA(2, 0, 2)* a granularidade de 30 minutos.

O propósito desse experimento era confirmar que o modelo ARIMA não realiza a generalização das previsões ao executar com parâmetros fixos, conforme capacidade identificada nos modelos MLP e GRU durante as avaliações da Subseção 4.2.2. Os dados em destaque na Tabela 4.7 correspondem a perda da precisão das previsões. Como pode ser verificado os resultados sofreram penalidades em consequência dos parâmetros.

A Figura 4.6 expõe as previsões realizadas com diferentes hiperparâmetros não sendo equivalentes, e demonstrando, também, que a previsão de intervalos mais longos tendem a agravar as previsões. A tendência dos preditores é perder a qualidade de acordo

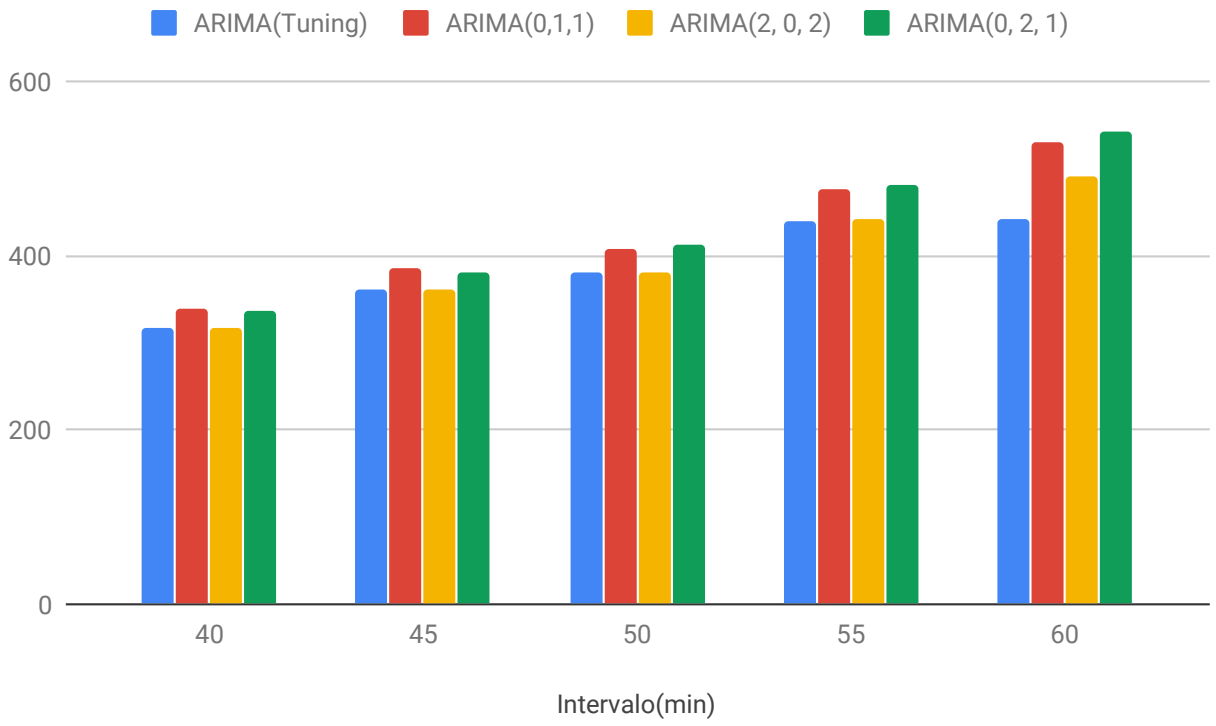

Figura 4.6 – Comparação da diferença do RMSE na execução de hiperparâmetros fixos

com as mudanças das cargas de trabalho, exigindo, assim, novas atualizações dos hiperparâmetros. Logo, a utilização de preditores com parâmetros desajustados tendem a não apresentar previsões confiáveis, as quais não representam as cargas de trabalho reais.

Ao optar pelo uso do modelo ARIMA para estratégias de escalonamento, considerando o problema da não linearidade das cargas de trabalho das aplicações web e a mudança do comportamento de aplicações, implica-se em uma atenção maior onde o uso de parâmetros desajustados podem não proporcionar previsões tão confiáveis quanto a parâmetros ajustados. Isso se deve ao fato que, o modelo não consegue se adaptar automaticamente sem a atualização dos hiperparâmetros, fenômeno este observado nos modelos MLP e GRU.

4.2.4 Avaliação do tempo de execução dos modelos

A finalidade deste experimento é evidenciar a agilidade de execução de cada modelo, diante da situação em que o tempo de resposta pode influenciar na qualidade das estratégias em que os modelos forem aplicados. Desta maneira, as amostras das execuções presentes na tabela 4.8 correspondem ao tempo de execução exigido para prever todas as amostras dos respectivos *datasets* de teste.

Tabela 4.8 – Comparativo do tempo de execução dos modelos no formato hh:mm:ss.m

Intervalo de Prev.(min)	MLP	GRU	ARIMA
5	00:00:00.014795	00:00:00.057586	00:58:47.485485
15	00:00:00.005369	00:00:00.012579	00:03:33.975461
30	00:00:00.002928	00:00:00.007393	00:03:37.539076
60	00:00:00.001527	00:00:00.006171	00:01:05.701869

Como pode ser observado na Tabela 4.8, os experimentos com a granularidade de tempo de 5 minutos possuem valores mais altos, isso devido a quantidade de amostras serem mais elevada que as demais. Além disso, fica evidente o menor custo de tempo das redes neurais MLP e GRU em comparação ao modelo ARIMA, porque as redes neurais após a fase de treinamento fazem o uso do modelo "aprendido" para a execução das previsões. O maior custo de execução do modelo ARIMA se deve pela utilização da série temporal para toda a previsão realizada; desta forma, a velocidade de execução é comprometida.

Assim sendo, ao avaliar o tipo de modelo a ser aplicado em uma estratégia proativa é importante verificar o quesito de tempo de execução, para que a demora causada pelos algoritmos de AM não interfira na estratégia adotada.

4.2.5 Análise geral dos modelos com *dataset* NASA

Com o intuito de realizar uma avaliação geral das três técnicas na utilização do *dataset* da NASA foram sumarizadas algumas especificações de cada modelo na Tabela 4.9. A coluna *RMSE Treino* representa a avaliação das previsões durante o processo de treinamento dos modelos, exceto o modelo ARIMA que não possui fase de treinamento, pois utiliza toda a série temporal a cada previsão realizada. A coluna *RMSE Teste* representa a avaliação das previsões realizadas com dados ainda não vistos pelo modelo. A métrica R^2 conforme descrita na Seção 3.2, avalia especificamente o modelo e o quão ele é preciso.

Ao observar o comportamento das métricas de treino e teste do modelo GRU podemos notar que a métrica de RMSE de treinamento é praticamente duas vezes mais precisa que os resultados do modelo MLP, característica a qual pode sugerir a ocorrência de *Overfitting*[10]. Por se tratar de uma Rede Neural Recorrente, dado as suas características, o modelo tende a memorizar informações, assim na fase de treinamento o modelo é exposto ao *dataset* diversas vezes, podendo, dessa forma, ser explicado este fenômeno da coluna *RMSE Teste* ser inferior ao modelo MLP. Como pode ser observado na fase de testes, o

[10]Overfitting: Quando um modelo se ajusta muito bem ao conjunto de dados de treinamento, mas se mostra ineficaz para prever novos resultados

Tabela 4.9 – Avaliações das técnicas

Técnica	**Intervalo de Prev.**	**RMSE Treino**	**RMSE Teste**	R^2
MLP				
	5	64,00	53,22	0,80
	15	151,62	133,73	0,85
	30	288,55	250,53	0,86
	60	717,69	546,64	0,84
GRU				
	5	26,68	53,39	0,80
	15	57,90	130,96	0,85
	30	126,64	243,53	0,87
	60	278,19	522,29	0,84
ARIMA				
	5	-	47,86	0,84
	15	-	135,08	0,84
	30	-	237,73	0,87
	60	-	490,86	0,86

modelo GRU demonstra previsões satisfatórias equiparáveis aos outros modelos. Também pode ser observado, apesar dos modelos terem algumas diferenças de valores na precisão das previsões, que os três modelos são equivalentes, ou seja, não possuem uma variância considerável que diferencie uma das outras. Pode, também, ser levado em consideração a métrica R^2 que ampara essa afirmação, onde todos os modelos possuem o resultado de precisão a partir de 80%.

Desta forma, podemos concluir que os modelos são equiparáveis dentro do nosso contexto de codificação e testes. No entanto, para a sua implantação em um ambiente de nuvem proativa deve ser levado em consideração as características do modelo, como tempo de execução, quantidade mínima de amostras, capacidade de generalização e precisão das previsões. Diferentes abordagens de implementação poderão ser tomadas, consequentemente. Essas características serão melhor descritas na Seção 4.3, a qual discute os resultados alcançados.

4.2.6 Análise da precisão dos modelos de AM com *dataset* da Wikipédia

Com o objetivo de validar a capacidade dos modelos de AM, foi executado novos experimentos com a carga de trabalho da Wikipédia especificado na Seção 4. Os experimentos foram realizados com os mesmos hiperparâmetros do *dataset* anterior. Contudo, o modelo ARIMA foi ajustado novamente para conseguir trabalhar com a nova carga de

trabalho devido a limitação de adaptação a novas cargas de trabalho, conforme descrito na Seção 4.2.3.

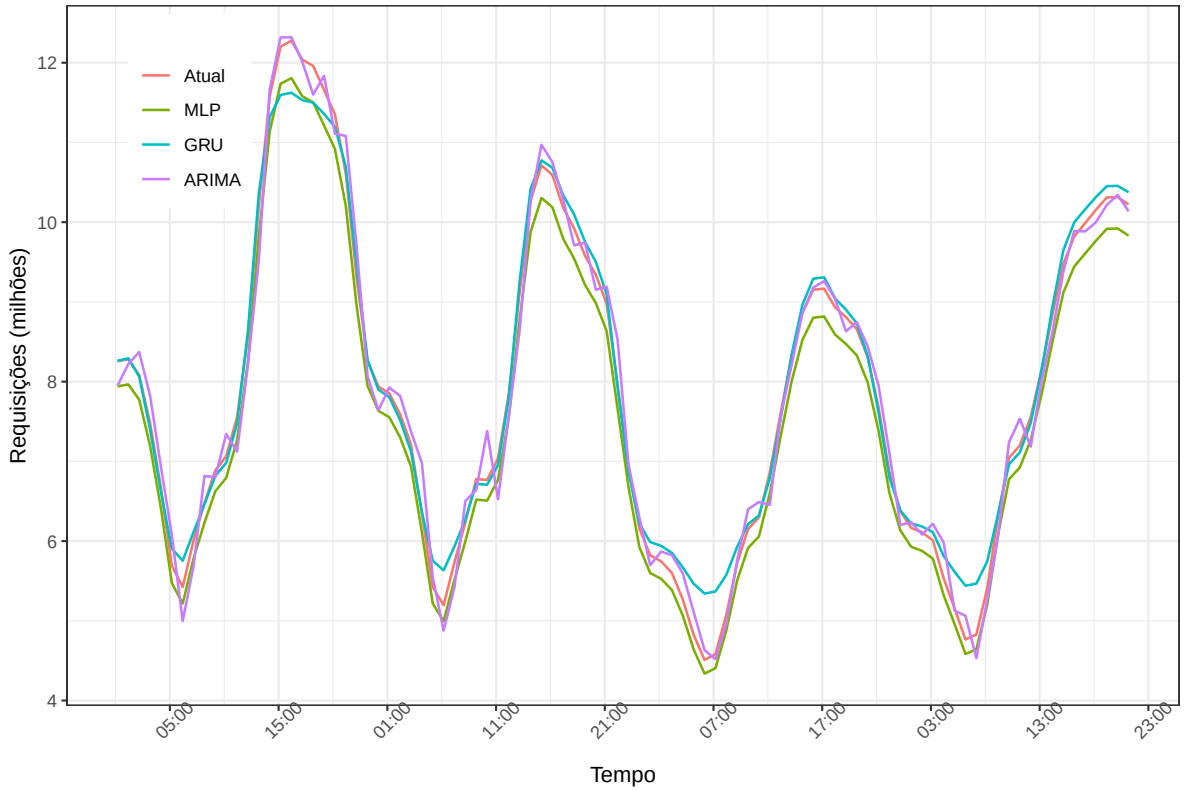

Figura 4.7 – Previsões com intervalo de tempo de 60 minutos referente a carga de trabalho da Wikipédia.

Tabela 4.10 – Precisão dos modelos baseado na métrica R^2

Intervalo de Prev.(min)	MLP	GRU	ARIMA
5	0.94	0.90	0.98
15	0.93	0.98	1.00
30	0.96	0.96	1.00
60	0.90	0.91	1.00

Como pode ser verificado na Figura 4.7, todos os modelos conseguiram realizar previsões precisas na nova carga de trabalho. Conforme exposto na Tabela 4.10, os modelos alcançaram mais de 90% de precisão nos diferentes intervalos de previsão.

A carga de trabalho neste contexto possui uma quantidade massiva de acessos em comparação a carga de trabalho da NASA, e, também, apresenta menos oscilações de comportamento. Assim, explicando a maior precisão R^2 alcançada pelos modelos em comparação a precisão de carga da NASA que caracteriza um contexto mais complexo. Portanto, podemos concluir que os três modelos são qualificados para trabalhar com diferentes cargas de trabalho, no que tange nosso ambiente de testes.

4.3 Discussão dos resultados

O desenvolvimento deste trabalho nos oportunizou a conhecer melhor sobre os conceitos e aplicações das técnicas de AM. Durante a execução dos experimentos, pudemos observar algumas características especificas de cada modelo estudado. Por conseguinte, a Tabela 4.11 representa um resumo das características que diferenciam os três modelos.

Tabela 4.11 – Características dos Modelos

Características	**ARIMA**	**MLP**	**GRU**
Tempo de execução		X	X
Cargas de trabalho não lineares	X	X	X
Menor RMSE entre os modelos	X		
Menor RMSE entre as Redes Neurais			X
Quantidade de amostras		X	
Generalização automática da Carga de Trabalho		X	X

- **Tempo de execução:** No quesito tempo de execução, o modelo ARIMA apresentou desvantagem, pois ao ser executado para a realização de uma previsão o modelo executa a leitura de toda a série temporal submetida ao modelo. Já os demais modelos estudados executam a fase de treinamento e previsões de teste em questão de segundos para todo o *dataset*.

- **Cargas de trabalho não lineares:** Sabe-se que a carga de trabalho de aplicações, principalmente aplicações web, são influenciadas por eventos externos, podendo receber uma quantidade massiva de requisições a qualquer momento caracterizando uma carga de trabalho não linear. Partindo desse pressuposto as três técnicas foram submetidas a este cenário e, como resultado ofereceram previsões precisas no contexto do nosso ambiente de testes.

- **Menor RMSE entre os modelos:** Quanto a precisão das previsões entre os três modelos avaliados, o ARIMA teve a métrica de RMSE mais baixo. Porém, conforme mencionado anteriormente, deve ser avaliada a relevância dos prós e contras desse modelo para o contexto proativo que o mesmo pode ser aplicado.

- **Menor RMSE entre as Redes Neurais:** Entre as Redes Neurais, o modelo GRU teve destaque na precisão das previsões em diferentes granularidades de tempo.

- **Quantidade de amostras:** Os experimentos realizados neste tópico consistiram no treinamento dos modelos com restrições de amostras em diferentes dimensões, tendo

como o objetivo de avaliar a quantidade mínima de amostras necessárias para realizar previsões satisfatórias dentro do contexto em estudo. Portanto, o MLP demonstrou uma capacidade notável de generalização conseguindo realizar previsões precisas em todas as granularidades de tempo com apenas 10% do *dataset*.

- **Generalização automática da Carga de Trabalho:** Neste tópico foi avaliado a capacidade de adaptação dos modelos quanto a capacidade de fazer previsões, mesmo com hiperparâmetros não adaptados especificamente para o *dataset* submetido. Neste quesito, os modelos MLP e GRU conseguiram realizar boas previsões, mesmo com os hiperparâmetros fixos para todos os *datasets* em que foram submetidos. Foram realizados os testes com o modelo ARIMA, porém em alguns casos o modelo não executou ou não alcançou previsões precisas.

Diante dos resultados obtidos durante a execução dos experimentos deste trabalho podemos concluir que, no que tange a precisão dos modelos, os três modelos estudados são equiparáveis. No entanto, existem prós e contras que precisam ser levados em consideração para que seja adotada uma estratégia proativa em um ambiente de nuvem. Uma vez que, cada modelo possui sua respectiva característica de implementação.

Ao optar pelo uso do modelo ARIMA, não será necessário realizar o processo de treinamento. Porém, deve ser atualizado com cautela cada um dos seus hiperparâmetros, no momento em que ocorrer mudanças de comportamento da carga de trabalho de forma drástica ou a proporção de tamanho da série temporal for alterada, para que o modelo ofereça previsões confiáveis. O tempo de resposta para esta previsão também não pode ser uma prioridade, visto que em comparação aos outros modelos esse tempo pode variar dependendo do tamanho da série temporal submetida ao modelo, conforme constatado na avaliação realizada na Seção 4.2.4.

Ao necessitar de um modelo com prioridade na precisão das previsões, que retorne os resultados de forma ágil e que seja adaptável a não linearidade da carga de trabalho sem requerer o ajuste preciso dos hiperparâmetros, o modelo GRU pode ser a opção ideal. Não obstante, é necessário uma quantidade de amostras significativa para o treinamento do modelo, sendo o equivalente a 2 semanas no contexto dos nossos experimentos avaliados na Seção 4.2.2.

Caso a necessidade seja utilizar um modelo que forneça previsões precisas com resultados em tempo real, adaptável a não linearidade da carga de trabalho e que a quantidade de dados históricos é limitada, podemos indicar o modelo MLP. Uma vez que, o modelo não exige grande quantidade de dados de treinamento para oferecer previsões precisas conforme exposto na Seção 4.2.2.

Com base nos pressupostos desta seção, o modelo GRU foi explorado junto com os demais algoritmos de AM. A partir dos experimentos foi evidenciado que ele é uma opção viável para agregar soluções proativas. Acreditamos que ele possa ser melhor explorado e

otimizado devido suas características de memória. Entretanto, avaliando os experimentos realizados após os testes preliminares alcançados, notamos que o modelo GRU precisa de mais dados de treinamento para realizar suas previsões. Indicamos o modelo MLP, por conseguinte, visando um modelo que ofereça previsões precisas e não necessite de muitos dados para performar em uma solução proativa. Conforme visto na seção anterior, ele além de demonstrar a flexibilidade de modelar diferentes cargas de trabalho, possui a característica de induzir adequadamente com poucos dados de treinamento que, dependendo das circunstâncias da aplicação, constitui um requisito diferencial.

5. TRABALHOS RELACIONADOS

Nesta seção, analisamos o atual estado da arte relacionado à previsão de carga de trabalho. Conforme mencionado na Seção 2.2, já existem diversos esforços de pesquisa para outros domínios problemáticos e personalizados para suas características específicas de carga de trabalho. Alguns deles fazem uso de preditores estatísticos simples, outros realizam o aprimoramento de técnicas de aprendizado de máquina para seus problemas. Ao longo da seção apresentamos as contribuições mais relevantes que possuem semelhanças com o que fizemos neste trabalho para a previsão de aplicações de computação em nuvem.

Kumar e Singh [25] apresentam um preditor de carga de trabalho na nuvem usando uma rede neural artificial (ANN) e algoritmos evolutivos. Essa escolha foi feita devido a capacidade que os algoritmos evolutivos possuem de explorar o espaço em múltiplas direções usando um conjunto de soluções [25]. Sua técnica de AM é treinada com solicitações HTTP alinhado a abordagem evolucionária para minimizar o efeito da escolha inicial da solução. Com isso, as técnicas de AM aprendem e extraem padrões da carga de trabalho, que podem ser utilizados para melhorar as decisões de dimensionamento de recursos. Sua implementação resultou em melhores resultados quando comparados com abordagens tradicionais de AM, como o treinamento com propagação reversa mais conhecido como *back propagation*. O modelo MLP com *back propagation* implementado em nosso trabalho corresponde ao modelo ANN utilizado pelos autores. No entanto, os autores utilizaram outra abordagem para realizar o processo de treinamento do modelo, que de acordo com os experimentos apresentados representam valores mais precisos em relação a abordagem do *back propagation*. Este trabalho, serve como parâmetro para realizarmos futuras melhorias em nosso modelo MLP. Contudo, não foi especificado as particularidades da nova abordagem como a quantidade de dados históricos necessários para explorar os padrões de carga.

Messias et al. [31] construíram um modelo que combina modelos de previsão de séries temporais com o uso de um algoritmo genético para escalonar automaticamente aplicativos da Web hospedados na infraestrutura de nuvem. As previsões são baseadas em cinco modelos: modelo ingênuo (Naive); modelo autoregressivo (AR); modelo de média móvel autorregressiva (ARMA); modelo de média móvel integrado autorregressivo (ARIMA); estendido modelo de suavização exponencial (ETS) [31]. Como nenhum desses métodos alcança o melhor resultado em todos os casos, os autores usam um algoritmo genético (GA) para combinar os benefícios de cada método de previsão individual e aumentar a precisão da previsão. Para avaliar a proposta, eles usaram três *logs* extraídos de servidores web reais. Diferente do trabalho proposto nesta dissertação, os autores não exploraram o funcionamento e o tempo de execução de sua abordagem, e, sim, executaram diversos modelos estatísticos focando na precisão das previsões. Os autores propuseram uma mé-

trica específica, desta maneira, não conseguimos realizar a comparação da abordagem utilizada.

Prevost et al. [35] introduz uma nova estrutura que combina previsão de carga de trabalho e modelos de transição de estado estocásticos. O objetivo deste trabalho é obter uma alocação ótima de recursos na nuvem, minimizando a energia consumida e, ao mesmo tempo, mantendo os níveis de desempenho necessários. Eles usaram ANN e AR para prever a carga de trabalho do aplicativo em data centers em nuvem. Os autores mostram que ambos os modelos podem prever adequadamente as cargas de trabalho futuras, mas os valores relativos do RMSE para o preditor de AR linear obtiveram resultados mais precisos do que os da ANN. Embora eles comparem uma ampla gama de intervalos de previsão, variando de 1 a 90 segundos, eles são todos relativamente curtos. Nos experimentos realizados em nosso trabalho, conseguimos identificar que o modelo ANN tem capacidade de generalização para modelar diversos tamanhos de carga, no entanto, o tamanho do intervalo de previsão utilizado pelos autores corresponde apenas previsões dos próximos segundos, influenciando com que o *dataset* representasse comportamento linear. Assim, favorecendo o modelo AR e, gerando um espaço de investigação que poderia ser explorado com intervalos de previsão maiores.

Calheiros et al. [10] apresenta a implementação de um módulo de previsão de carga de trabalho em nuvem para provedores de Software como serviço (SaaS) e seu impacto na QoS dos aplicativos em nuvem. A carga prevista é usada para provisionar dinamicamente as VMs em um ambiente de nuvem elástica. A respectiva técnica de previsão é baseada no modelo ARIMA. O sistema foi avaliado com dados históricos (*Traces*) reais de solicitações para os servidores da Web da *Wikimedia Foundation*. Os autores justificaram essa escolha com base em seus experimentos, onde o ARIMA apresentou melhores resultados de precisão do que os modelos de ANN mais complexos, visto que as cargas de trabalho utilizadas possuem comportamentos mais lineares. Para avaliar o impacto da precisão em relação à eficiência na utilização de recursos e QoS das solicitações do usuário, os experimentos foram realizados via simulação usando o kit de ferramentas *CloudSim*, que contém um simulador de eventos discretos e classes que permitem aos usuários modelar ambientes de nuvem. Em seu contexto de carga de trabalho, a simulação atinge uma precisão média de 91%, o que resulta em impacto mínimo na QoS. O trabalho relatado aderente com os resultados alcançados nesta dissertação, no que tange a precisão média das previsões do modelo ARIMA. No entanto, foram explorados apenas o intervalo de previsão de 60 minutos, diferente de nossas avaliações que procuraram explorar outras granularidades de tempo. O respectivo trabalho também evidenciou uma alternativa de ambiente de testes, no qual poderá ser utilizado para a execução dos experimentos em futuras extensões desta dissertação.

Nguyen et al. [34] projetaram e implementaram um modelo de algoritmos de previsão, usando empilhamento para prever séries temporais de carga de trabalho em sistemas

Cloud e Grid usando vários modelos lineares, Redes Neurais Recorrentes e *Autoencoder*. A Rede Neural Recorrente é utilizada para identificar dependências lineares e não lineares entre os dados nos grupos de vetores de entrada. Para a escolha do modelo RNN ideal no cenário proposto, foram testados três modelos e os candidatos foram *Identity Recurrent Neural Network* (IRNN), *Long Short Term Memory* (LSTM), e *Gated Recurrent Unit* (GRU). Os modelos RNN foram treinados para combinar diferentes previsões em um resultado, enquanto o *autoencoder* é usado para extrair recursos importantes para entradas no RNN. Como resultado dos experimentos, o modelo GRU teve o desempenho computacional mais rápido para treinar devido a estrutura menos complicada que o modelo LSTM, portanto, os autores decidiram usar o GRU para a camada RNN na previsão do modelo. Além de mostrar que o modelo alcança um menor NRMSE[1] médio nos 3 *datasets* em que foram explorados, esse supera os algoritmos dos componentes[2] com melhorias no NRMSE de 7,43% para 12,45%. Diante dos resultados alcançados em relação ao comparativo dos algoritmos da classe das RNN, esta pesquisa reafirmou a nossa escolha referente a implementação do modelo GRU em nossa análise. Porém, o estudo não explorou a quantidade de amostras que cada modelo de AM necessita para realizar suas previsões, e, também, os intervalos de previsão não foram indicados para que pudéssemos realizar as comparações entre os modelos RNN.

Embora os trabalhos relacionados descritos acima mostrassem bons resultados para os cenários em que estavam ajustados, os trabalhos não exploraram os tempos de execução, quantidade de amostras que cada algoritmo precisa para oferecer previsões precisas. Portanto, nessa dissertação estávamos interessados em verificar a eficiência das técnicas de previsão em um cenário realista, essa com diferentes intervalos de previsão e cargas de trabalho não lineares, como é o caso de aplicações modernas em nuvem. Com base nos respectivos trabalhos relacionados, selecionamos os modelos ARIMA, GRU e MLP e comparamos umas às outras nas mesmas condições, analisando algumas compensações em relação à precisão, aos esforços de treinamento e aos tempos de execução. Essas sendo características que podem influenciar na escolha dos algoritmos de AM ao implementar uma estratégia proativa com o objetivo de otimizar o provisionamento de recursos em nuvem. A tabela 5.1 sumariza os trabalhos relacionados descritos neste capítulo.

[1] Erro Quadrado Médio da Raiz Normalizado [34]
[2] Conjunto de algoritmos (Essemble)

Tabela 5.1 – Estrategias de previsão de carga dos trabalhos relacionados

Autores	**Técnicas**	**Resultados**
Jitendra and Singh[25]	ANN e evolução diferencial adaptativa	Abordagem evolucionária no treinamento do modelo para minimizar o efeito da escolha.
Messias et al. [31]	Naive, AR, ARMA, ARIMA, ETS e Algoritmos genéticos	Algoritmos Genéticos na combinação dos resultados dos modelos estatísticos.
Prevost et al. [35]	ANN, AR	AR resultou em melhores resultados devido a carga de trabalho em um período curto de tempo.
Calheiros et al. [10]	ARIMA	Uso do simulador CloudSim. Simulações atingiram uma precisão média de 91%, o que resulta em impacto mínimo na QoS em intervalos de 60 minutos.
Nguyen et al. [34]	GRU, IRNN, LSTM, Essemble	Comparação de modelos RNN e identificação do GRU como modelo relevante entre as técnicas de rede neural recorrente.

6. CONCLUSÃO

Nesta dissertação foram avaliadas características de modelos preditivos utilizados para a previsão de carga de trabalho. Avaliaram-se as vantagens e desvantagens de três modelos de previsão amplamente aplicadas no contexto de computação em nuvem para previsão de cargas em aplicações web.

Nossos resultados comparam o ARIMA, MLP e GRU em diferentes configurações para ajudar administradores a escolher o modelo preditivo mais adequado e eficiente para seu problema exclusivo. Algumas características especificas analisadas durante a avaliação dos modelos são destacadas abaixo:

- **Precisão:** Nossos experimentos mostram que as três técnicas, embora com métricas moderadamente distintas, apresentam precisões de modelo superior a 80% para o *dataset* da NASA e de 90% para o *dataset* da Wikipédia, ou seja, em termos de previsão as técnicas são equiparáveis.

- **Tempo de execução:** Quanto ao tempo de execução para previsão, os modelos MLP e GRU executaram as previsões da série temporal submetida em milésimos de segundos, ambos apresentando um comportamento similar independentemente da quantidade de entrada de dados. Por outro lado, o modelo ARIMA executou as previsões na faixa de minutos, aumentando o tempo de acordo com a quantidade de dados disponíveis na entrada. Sendo assim, este modelo apresenta um comportamento dependente do tamanho da entrada de dados para finalizar as previsões.

- **Generalização:** O modelo ARIMA não oferece previsões precisas para os casos em que os hiperparâmetros não estejam ajustados especificamente para o *dataset*. No entanto, os modelos MLP e GRU demonstraram capacidade de generalização ao trabalhar com as não linearidades de carga independente das definições dos hiperparâmetros. Essas não estando ajustadas especificamente para o *dataset* submetido.

- **Quantidade de amostras:** Ao avaliar a qualidade das previsões com a quantidade limitada de dados históricos na fase de treinamento, o modelo MLP demonstrou alta capacidade de adaptação, oferecendo previsões precisas com apenas 10% de dados históricos para cada intervalo de previsão testado.

- **Algoritmos de Tuning:** Foram explorados dois algoritmos de otimização de hiperparâmetros. O algoritmo *Grid search* foi aplicado no *tuning* do modelo ARIMA, o que proporcionou resultados precisos, no entanto, este processo pode perdurar por horas devido a relação da quantidade de hiperparâmetros a serem cruzados e a quantidade de amostras do *dataset*. Dessa forma, o *Grid Search* mostra-se ineficiente em cenários vinculados a logística de tempo. Para os algoritmos MLP e GRU foi utilizado

o algoritmo *Tree of Parzen Estimator* (TPE) que, pode ser adotado como uma ferramenta útil para a implantação de redes neurais com diferentes cargas de trabalho. Visto que, o baixo tempo de execução permite realizar diversos testes com diferentes espaços amostrais para encontrar bons conjuntos de hiperparâmetros.

Com base nas características mencionadas e nas considerações realizadas durante a discussão dos resultados da Seção 4.3, o modelo MLP se apresentou flexível e adaptável em diferentes cargas de trabalho, com capacidade de fornecer previsões precisas mesmo com a quantidade limitada de dados históricos para a fase de treinamento. Indicamos esse modelo para cenários de nuvem, porque essas características o tornam mais flexível e aplicável que as outras técnicas analisadas neste estudo.

Os scripts, códigos-fonte dos modelos em estudo e *traces* de carga de trabalho usados em nossos experimentos estão disponíveis no GitHub[1] para fins de reprodutibilidade. Também está disponível uma versão preparada para ser executada direto da ferramenta do Google *Colaboratory* que pode ser acessada pelo mesmo endereço.

Os resultados preliminares deste trabalho fazem parte de uma publicação realizada no *27th euromicro international conference on parallel distributed and network-based processing(PDP 2019)*[24]. O foco principal do artigo foi realizar uma comparação preliminar das três técnicas de AM alvo desta pesquisa.

Há possibilidade de realizar, em trabalhos futuros, uma análise de aderência destes algoritmos em outras cargas de trabalho, como diferentes tipos de aplicações e até mesmo recursos computacionais. Além disso, realizar uma análise de desempenho destes algoritmos em um ambiente de simulação de computação em nuvem. Possibilita-se, com base neste estudo realizado, o planejamento e desenvolvimento de um serviço que ofereça previsões de carga de trabalho baseada em aplicações, onde este possa ser acoplado em soluções de escalonamento reativo com o objetivo de transforma-lo em uma solução de escalonamento proativa.

[1] *https* : *//github.com/dionatrafk/model_evaluation*

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Alpaydin, E. “Introduction to machine learning”. MIT press, 2009, 76–79p.

[2] Amiri, M.; Mohammad-Khanli, L. “Survey on prediction models of applications for resources provisioning in cloud”, *Journal of Network and Computer Applications*, volume 82, Mar 2017, páginas 93–113.

[3] Bengio, Y.; Simard, P.; Frasconi, P. “Learning long-term dependencies with gradient descent is difficult”, *IEEE Transactions on Neural Networks*, volume 5–2, Mar 1994, páginas 157–166.

[4] Bennani, M. N.; Menasce, D. A. “Resource allocation for autonomic data centers using analytic performance models”. In: Second International Conference on Autonomic Computing, 2005, páginas 229–240.

[5] Bergstra, J.; Bardenet, R.; Kégl, B.; Bengio, Y. “Implementations of algorithms for hyper-parameter optimization”. In: NIPS Workshop on Bayesian optimization, 2011, página 29.

[6] Bergstra, J.; Bengio, Y. “Random search for hyper-parameter optimization”, *Journal of Machine Learning Research*, volume 13, Fev 2012, páginas 281–305.

[7] Bergstra, J.; Komer, B.; Eliasmith, C.; Warde-Farley, D. “Preliminary evaluation of hyperopt algorithms on hpolib”. In: ICML workshop on Automated Machine Learning, 2014.

[8] Box, G. E.; Jenkins, G. M.; Reinsel, G. C.; Ljung, G. M. “Time series analysis: forecasting and control”. John Wiley & Sons, 2015, 92–97p.

[9] Brownlee, J. “Clever algorithms: nature-inspired programming recipes”. Jason Brownlee, 2011, 30-32p.

[10] Calheiros, R. N.; Masoumi, E.; Ranjan, R.; Buyya, R. “Workload prediction using arima model and its impact on cloud applications’ qos”, *IEEE Transactions on Cloud Computing*, volume 3–4, Out-Dez 2015, páginas 449–458.

[11] Cameron, A. C.; Windmeijer, F. A. “An r-squared measure of goodness of fit for some common nonlinear regression models”, *Journal of Econometrics*, volume 77–2, Abr 1997, páginas 329–342.

[12] Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. “Learning phrase representations using rnn encoder-decoder for statistical machine translation”, *arXiv preprint arXiv:1406.1078*, Set 2014.

[13] Chollet, F. “Keras documentation”. Capturado em: https://keras.io, Jun 2018.

[14] Diaz, G. I.; Fokoue-Nkoutche, A.; Nannicini, G.; Samulowitz, H. “An effective algorithm for hyperparameter optimization of neural networks”, *IBM Journal of Research and Development*, volume 61–4, Set 2017, páginas 1–9.

[15] Faceli, K.; Lorena, A. C.; Gama, J.; Carvalho, A. C. P. d. L.; et al.. “Inteligência Artificial: Uma abordagem de aprendizado de máquina”. LTC-Livros Técnicos e Científicos Editora Ltda, 2011, 160p.

[16] Garg, S. K.; Toosi, A. N.; Gopalaiyengar, S. K.; Buyya, R. “Sla-based virtual machine management for heterogeneous workloads in a cloud datacenter”, *Journal of Network and Computer Applications*, volume 45, Out 2014, páginas 108–120.

[17] Geurts, M. D.; Ibrahim, I. “Comparing the box-jenkins approach with the exponentially smoothed forecasting model application to hawaii tourists”, *Journal of Marketing Research*, Mai 1975, páginas 182–188.

[18] Gollapudi, S. “Practical machine learning”. Packt Publishing Ltd, 2016, 3–22p.

[19] Honnappa, H.; Jain, R. “Strategic arrivals into queueing networks”. In: 48th Annual Allerton Conference on Communication, Control, and Computing, 2010, páginas 820–827.

[20] Huang, D.; He, B.; Miao, C. “A survey of resource management in multi-tier web applications”, *IEEE Communications Surveys & Tutorials*, volume 16–3, Jan 2014, páginas 1574–1590.

[21] Hwang, K.; Bai, X.; Shi, Y.; Li, M.; Chen, W.-G.; Wu, Y. “Cloud performance modeling with benchmark evaluation of elastic scaling strategies”, *IEEE Transactions on Parallel and Distributed Systems*, volume 27–1, Jan 2016, páginas 130–143.

[22] ITA. “Traces available in the internet traffic archive”. Capturado em: http://ita.ee.lbl.gov/html/contrib/, Set 2018.

[23] Jiang, Y.; Perng, C.-S.; Li, T.; Chang, R. N. “Cloud analytics for capacity planning and instant vm provisioning”, *IEEE Transactions on Network and Service Management*, volume 10–3, Mai 2013, páginas 312–325.

[24] Kirchoff, F. D.; Xavier, M. G.; Mastella, J.; De Rose, C. A. “A preliminary study of machine learning workload prediction techniques for cloud applications”. In: 27th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2019, páginas 253–260.

[25] Kumar, J.; Singh, A. K. “Workload prediction in cloud using artificial neural network and adaptive differential evolution”, *Future Generation Computer Systems*, volume 81, Abr 2018, páginas 41–52.

[26] Leitner, P.; Wetzstein, B.; Rosenberg, F.; Michlmayr, A.; Dustdar, S.; Leymann, F. “Runtime prediction of service level agreement violations for composite services”. In: Service-Oriented Computing. Service Wave Workshops, 2009, páginas 176–186.

[27] Liang, Q.; Zhang, J.; Zhang, Y.-h.; Liang, J.-m. “The placement method of resources and applications based on request prediction in cloud data center”, *Information Sciences*, volume 279, Set 2014, páginas 735–745.

[28] Madrigal, F.; Maurice, C.; Lerasle, F. “Hyper-parameter optimization tools comparison for multiple object tracking applications”, *Machine Vision and Applications*, volume 30, Mar 2019, páginas 269–289.

[29] Manvi, S. S.; Shyam, G. K. “Resource management for infrastructure as a service (iaas) in cloud computing: A survey”, *Journal of Network and Computer Applications*, volume 41, Mai 2014, páginas 424–440.

[30] Mell, P.; Grance, T. “The NIST definition of cloud computing”. Computer Security Division, Information Technology Laboratory, National, 2011, 2–3p.

[31] Messias, V. R.; Estrella, J. C.; Ehlers, R.; Santana, M. J.; Santana, R. C.; Reiff-Marganiec, S. “Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure”, *Neural Computing and Applications*, volume 27–8, Nov 2016, páginas 2383–2406.

[32] Müller, A. C.; Guido, S.; et al.. “Introduction to machine learning with Python: a guide for data scientists”. O’Reilly Media, Inc., 2016, 267–269p.

[33] Nanda, S.; Hacker, T. J.; Lu, Y. H. “Predictive model for dynamically provisioning resources in multi-tier web applications”, *International Conference on Cloud Computing Technology and Science, CloudCom*, Jan 2017, páginas 326–335.

[34] Nguyen, H. M.; Woo, S.; Im, J.; Jun, T.; Kim, D. “A workload prediction approach using models stacking based on recurrent neural network and autoencoder”. In: 18th International Conference on High Performance Computing and Communications, 2016, páginas 929–936.

[35] Prevost, J. J.; Nagothu, K.; Kelley, B.; Jamshidi, M. “Prediction of cloud data center networks loads using stochastic and neural models”. In: 6th International Conference on System of Systems Engineering, 2011, páginas 276–281.

[36] Rasmussen, C. E. “Gaussian processes in machine learning”. Springer, Berlin, Heidelberg, 2003, capítulo: 4, páginas 63–71.

[37] Singh, S.; Chana, I. “A survey on resource scheduling in cloud computing: Issues and challenges”, *Journal of Grid Computing*, volume 14–2, Jun 2016, páginas 217–264.

[38] Stuart, R.; Peter, N. “Inteligência Artificial: Tradução da 3a Edição”. Elsevier Brasil, 2016, 634–636p.

[39] Tjandra, A.; Sakti, S.; Manurung, R.; Adriani, M.; Nakamura, S. “Gated recurrent neural tensor network”. In: International Joint Conference on Neural Networks, 2016, páginas 448–455.

[40] Urdaneta, G.; Pierre, G.; van Steen, M. “Wikipedia workload analysis for decentralized hosting”, *Elsevier Computer Networks*, volume 53–11, Jul 2009, páginas 1830–1845.

[41] Urgaonkar, B.; Pacifici, G.; Shenoy, P.; Spreitzer, M.; Tantawi, A. “Analytic modeling of multitier internet applications”, *ACM Transactions on the Web*, volume 1–1, Mai 2007, página 2.

[42] Wei, L.; Foh, C. H.; He, B.; Cai, J. “Towards efficient resource allocation for heterogeneous workloads in iaas clouds”, *IEEE Transactions on Cloud Computing*, volume 6–1, Set 2018, páginas 264–275.

[43] Xu, X.; Wang, W.; Wu, T.; Dou, W.; Yu, S. “A virtual machine scheduling method for trade-offs between energy and performance in cloud environment”. In: International Conference on Advanced Cloud and Big Data, 2016, páginas 246–251.

[44] Yang, Q.; Peng, C.; Zhao, H.; Yu, Y.; Zhou, Y.; Wang, Z.; Du, S. “A new method based on psr and ea-gmdh for host load prediction in cloud computing system”, *The Journal of Supercomputing*, volume 68–3, Jun 2014, páginas 1402–1417.

[45] Zhang, G.; Patuwo, B. E.; Hu, M. Y. “Forecasting with artificial neural networks: The state of the art”, *International Journal of Forecasting*, volume 14–1, Mar 1998, páginas 35–62.

[46] Zhu, X.; Uysal, M.; Wang, Z.; Singhal, S.; Merchant, A.; Padala, P.; Shin, K. “What does control theory bring to systems research?”, *ACM Operating Systems Review*, volume 43–1, Jan 2009, páginas 62–69.

APÊNDICE A – MODELO ARIMA

Comando de entrada:

```
python arima.py trace5.csv 2 0 1
```

Resultado da execução:

```
python arima.py trace5.csv 2 0 1 - R2: 0.84, score: 2290.93 MSE (47.86 RMSE)
```

```
# -*- coding: utf-8 -*-
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt
import numpy as np
import math
import sys,os

# receive the parameters by command line
filename = str(sys.argv[1])
p = int(sys.argv[2])
d = int(sys.argv[3])
q = int(sys.argv[4])
#perc = float(sys.argv[5])
print p,d,q

#read the csv file
dataset = read_csv(filename, header=0, parse_dates=[0], index_col=0, squeeze=
    True)

# split into train and test sets
X = dataset.values
X = X.astype('float32')
size = int(len(X) * 0.67)
train, test = X[0:size], X[size:len(X)]
series = [x for x in train]
predictions = list()

training_size = len(train)

#test with less samples
#size = int(training_size * (perc /100))
#series = series[size:training_size:]

# walk-forward validation
```

```
38 print 'current, prediction'
39 for t in range(len(test)):
40   model = ARIMA(series, order=(p,d,q))
41   model_fit = model.fit(disp=0)
42   output = model_fit.forecast()
43   yhat = output[0]
44   predictions.append(yhat)
45   current = test[t]
46   series.append(current)
47   print('%.2f, %.2f' % (current,yhat))
48
49 # evaluate forecasts
50 score = mean_squared_error(test, predictions)
51 r2 = r2_score(test, predictions)
52 print ('R2: %.2f, Testscore: %.2f MSE (%.2f RMSE)' %(r2,score, math.sqrt(score))
       )
53
54 # Save into a file the configuration and evaluation executed
55 f=open("ARIMA_Execu es.txt", "a")
56 f.write("\n python arima.py %s %d %d %d - R2: %.2f, score: %.2f MSE (%.2f RMSE)"
       %(filename, p,d,q, r2,score, math.sqrt(score)))
57
58 #plot the execution
59 #pyplot.plot(test)
60 #pyplot.plot(predictions, color='red')
61 #pyplot.show()
```

APÊNDICE B – MODELO MLP

Comando de entrada:

```
python mlp.py trace5.csv 110 180 50 25
```

Resultado da execução:

```
python mlp.py trace5.csv 110 180 50 25 - R2: 0.80, Trainscore: 4096.38 MSE
    (64.00 RMSE), Testscore: 2832.34 MSE (53.22 RMSE)
```

```
# -*- coding: utf-8 -*-
import pandas as pd
import time
import matplotlib.pyplot as plt
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.metrics import mean_squared_error, r2_score
import math
import os, sys

def create_dataset(dataset, lookback=1):
    dataX, dataY = [], [] # create 2 empty list

    # go through the lenght of dataset, subtract the lookback and 1.
    #2 steps before the end of dataset, because we predict 1 step to the future
    for i in range(len(dataset)-lookback-1):
        a = dataset[i:(i+lookback),0]
        dataX.append(a)
        dataY.append(dataset[i+lookback,0]) # get the next value
    return np.array(dataX), np.array(dataY)

def save_model(model):
    filename = "data/" + "mlp" + ".h5"
    filename = "mlp" + ".h5"
    model.save(filename)

# receive the parameters by command line
filename, BATCH_SIZE, NB_EPOCHS, LAYER1, LAYER2 = str(sys.argv[1]), int(sys.argv
    [2]),int(sys.argv[3]),int(sys.argv[4]),int(sys.argv[5])

#perc = float(sys.argv[6])

np.random.seed(7)
# Data preparation
dataset = pd.read_csv(filename, usecols=[1], header=None)
dataset = dataset.values #convert to the array
```

```
37 dataset = dataset.astype('float32') # convert to float
38
39 # length of our data set
40 training_size = int(len(dataset)*0.67)
41 testing_size = len(dataset)-training_size
42
43 # split the data set
44 train, test = dataset[0:training_size,:], dataset[training_size:len(dataset),:]
45 #print(len(train),len(test))
46
47 # one time step to the future
48 lookback = 1
49 trainX, trainY = create_dataset(train, lookback)
50 testX, testY = create_dataset(test, lookback)
51
52 #size = int(training_size * (perc /100))
53 #trainX = trainX[0:training_size - size:]
54 #strainY = trainY[0:training_size - size:]
55
56 # create the model
57 model_name = 'requests_MLP'
58 model=Sequential()
59 model.add(Dense(LAYER1, input_dim=lookback, activation='relu'))
60 model.add(Dropout(0.2))
61 model.add(Dense(LAYER2, activation='relu'))
62 model.add(Dropout(0.2))
63 model.add(Dense(1))
64
65 #Compilation and training
66 #start = time.time()
67 model.compile(loss = 'mean_squared_error', optimizer = "adam")
68
69 #print "Compilation Time : ", time.time() - start
70 #start = time.time()
71
72 model.fit(trainX,trainY,epochs=NB_EPOCHS, batch_size=BATCH_SIZE, verbose=0)
73 #print "Training time : ", time.time() - start
74
75 #model.save("{}.h5".format(model_name))
76
77 # Making predictions
78 yhat = model.predict(trainX)
79 y_train = trainY
80
81 train_score = mean_squared_error(y_train, yhat)
82 print ('Trainscore: %.2f MSE (%.2f RMSE)' %(train_score, math.sqrt(train_score))
       )
83
```

```
84 yhat = model.predict(testX)
85 y_test = testY
86 r2 = r2_score(y_test,yhat)
87
88 # configuration executed
89 print('Filename:',filename)
90 print('BATCH_SIZE:',BATCH_SIZE)
91 print('NB_EPOCHS: ',NB_EPOCHS)
92 print('LAYER1: ',LAYER1)
93 print('LAYER2: ',LAYER2)
94
95 print('R2: ',r2)
96 test_score = mean_squared_error(y_test, yhat)
97 print ('Testscore %.2f' %(math.sqrt(test_score)))
98 #print ('%.2f , %.2f ,' %(math.sqrt(score),perc)) + filename
99
100 #Save into a file the command of executions and its scores
101 f=open("MLP_Execu es.txt", "a")
102 f.write("\n python mlp.py %s %d %d %d - R2: %.2f, Trainscore: %.2f MSE (%.2f
      RMSE), Testscore: %.2f MSE (%.2f RMSE)" %(filename, BATCH_SIZE,NB_EPOCHS,
      LAYER1,LAYER2, r2, train_score, math.sqrt(train_score),test_score, math.sqrt(
      test_score)))
103
104 #print 'actual, predicted'
105 #for i in range(0, len(y_test)):
106 #  print('%.2f, %.2f' % (y_test[i],yhat[i+1]))
107
108 #plot executions
109 #plt.plot(yhat[-100:], label='Predicted')
110 #plt.plot(y_test[-100:], label='Current')
111 #plt.legend()
112 #plt.grid()
113
114 #plt.ylabel('Requests')
115 #plt.xlabel('Samples 15 min')
116 #plt.show()
```

APÊNDICE C – MODELO GRU

Comando de entrada:

```
python gru.py trace5.csv 130 130 128 64
```

Resultado da execução:

```
python gru.py trace5.csv 130 130 128 64 – R2: 0.80,Trainscore: 712.00 MSE (26.68 RMSE), Testscore: 2850.82 MSE (53.39 RMSE)
```

```
# -*- coding: utf-8 -*-
import pandas as pd
from keras.layers.core import Dense, Dropout
from keras.layers.recurrent import GRU
from keras.models import Sequential, load_model
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
import math, time
import os, sys

def create_dataset(dataset, lookback=1):
    dataX, dataY = [], [] # create 2 empty lists

    # go through the lenght of dataset, subtract the lookback and 1. 2 steps before the end of dataset,
    #because we predict 1 step to the future
    for i in range(len(dataset)-lookback-1):
        a = dataset[i:(i+lookback),0]
        dataX.append(a)
        dataY.append(dataset[i+lookback,0]) # get the next value
    return np.array(dataX), np.array(dataY)

# receive the parameters by command line
filename, BATCH_SIZE, NB_EPOCHS, LAYER1, LAYER2 = str(sys.argv[1]), int(sys.argv[2]),int(sys.argv[3]),int(sys.argv[4]),int(sys.argv[5])
#perc = float(sys.argv[6])

# Dataset configuration
dataset = pd.read_csv(filename, usecols = [1], header=None)
dataset.columns = ["request"]
dataset = dataset.values #convert to the array
dataset = dataset.astype('float32') # convert to float

# length of our dataset
```

```
36 training_size = int(len(dataset)*0.67)
37 testing_size = len(dataset)-training_size
38
39 # split the data set
40 train, test = dataset[0:training_size:], dataset[training_size:len(dataset),:]
41
42 # one time step to the future
43 lookback = 1
44 trainX, trainY = create_dataset(train, lookback)
45 testX, testY = create_dataset(test, lookback)
46
47 # Scaling dataset
48 x_train, y_train = trainX, trainY
49 x_test, y_test = testX, testY
50
51 # scaling values for model
52 scaleX = MinMaxScaler()
53 scaleY = MinMaxScaler()
54
55 trainX = scaleX.fit_transform(x_train)
56 trainX = trainX.reshape((-1,1,1))
57
58 trainY = scaleY.fit_transform(y_train.reshape(-1,1))
59
60 testX  = scaleX.fit_transform(x_test)
61 testX = testX.reshape((-1,1,1))
62
63 testY  = scaleY.fit_transform(y_test.reshape(-1,1))
64
65 # creating model using Keras
66 model_name = 'requests_GRU'
67 model = Sequential()
68 model.add(GRU(units=LAYER1,
69               return_sequences=True,
70               input_shape=(1, 1)))
71 model.add(Dropout(0.2))
72 model.add(GRU(units=LAYER2))
73 model.add(Dropout(0.2))
74 model.add(Dense(1, activation='sigmoid'))
75
76 #test with less samples
77 #size = int(training_size * (perc /100))
78 #trainX = trainX[0:training_size - size:]
79 #trainY = trainY[0:training_size - size:]
80
81 # Compilation and training
82 #start = time.time()
83 model.compile(loss='mean_squared_error', optimizer='adam')
```

```
84
85 #print "Compilation Time : ", time.time() - start
86 #start = time.time()
87 model.fit(trainX,trainY,batch_size=BATCH_SIZE, epochs=NB_EPOCHS,
       validation_split=0.1, verbose=0)
88 #print "Training time : ", time.time() - start
89 #model.save("{}.h5".format(model_name))
90
91 # Making predictions
92 yhat = model.predict(trainX)
93 yhat = scaleX.inverse_transform(yhat)
94 y_test = scaleX.inverse_transform(trainY)
95
96 train_score = mean_squared_error(y_test, yhat)
97 print ('Trainscore: %.2f' %(math.sqrt(train_score)))
98
99 yhat = model.predict(testX)
100 yhat = scaleY.inverse_transform(yhat)
101 y_test = scaleY.inverse_transform(testY)
102
103 #RMSE score
104 test_score = mean_squared_error(y_test, yhat)
105 #R square score
106 r2 = r2_score(y_test,yhat)
107
108 print('Filename:',filename)
109 print('BATCH_SIZE:',BATCH_SIZE)
110 print('NB_EPOCHS: ',NB_EPOCHS)
111 print('LAYER1: ',LAYER1)
112 print('LAYER2: ',LAYER2)
113
114 print('R2: ',r2)
115
116 print ('Testscore: %.2f MSE (%.2f RMSE)' %(test_score, math.sqrt(test_score)))
117
118 # Save into a file the command of executions and its scores
119 f=open("GRU_Execu es.txt", "a")
120 f.write("\n python gru.py %s %d %d %d %d - R2: %.2f,Trainscore: %.2f MSE (%.2f
       RMSE), Testscore: %.2f MSE (%.2f RMSE)" %(filename, BATCH_SIZE,NB_EPOCHS,
       LAYER1,LAYER2, r2,train_score, math.sqrt(train_score), test_score, math.sqrt(
       test_score)))
121
122 #print 'actual,  predicted'
123 #for i in range(0, len(y_test)):
124 #  print('%.2f, %.2f' % (y_test[i],yhat[i+1]))
125
126 # plot some samples.
127 #plt.plot(yhat[-100:], label='Predicted')
```

```
128 #plt.plot(y_test[-100:], label='Current')
129 #plt.legend()
130 #plt.grid()
131
132 #plt.ylabel('Requests')
133 #plt.xlabel('Time')
134 #plt.show()'''
```

APÊNDICE D – MÉTODO DE *TUNING* DE HIPERPARÂMETRO: *GRID SEARCH*

Comando de entrada:

```
python grid_search.py trace25.csv
```

Resultado da execução:

O algoritmo executa todos os parâmetros do grid e indica a melhor configuração.

```
ARIMA(0, 0, 0) RMSE=600.839
ARIMA(0, 0, 1) RMSE=384.566
...
ARIMA(10, 2, 1) RMSE=199.126
ARIMA(10, 2, 2) RMSE=199.387
Best ARIMA(4, 0, 1) RMSE=191.030
```

```
# grid search ARIMA parameters for time series
import os, sys
import warnings
from math import sqrt
from pandas import Series
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error

filename = str(sys.argv[1])
log = open('log_grid.txt','w')
log.write(filename)

# evaluate an ARIMA model for a given order (p,d,q)
def evaluate_arima_model(X, arima_order):
  # prepare training dataset
  train_size = int(len(X) * 0.66)
  train, test = X[0:train_size], X[train_size:]
  history = [x for x in train]
  # make predictions
  predictions = list()
  for t in range(len(test)):
    model = ARIMA(history, order=arima_order)
    model_fit = model.fit(disp=0)
    yhat = model_fit.forecast()[0]
    predictions.append(yhat)
    history.append(test[t])
  # calculate out of sample error
  rmse = sqrt(mean_squared_error(test, predictions))
  return rmse

# evaluate combinations of p, d and q values for an ARIMA model
```

```
def evaluate_models(dataset, p_values, d_values, q_values):
  dataset = dataset.astype('float32')
  best_score, best_cfg = float("inf"), None
  for p in p_values:
    for d in d_values:
      for q in q_values:
        order = (p,d,q)
        try:
          rmse = evaluate_arima_model(dataset, order)
          if rmse < best_score:
            best_score, best_cfg = rmse, order
          print('ARIMA%s RMSE=%.3f' % (order,rmse))
          log.write('ARIMA%s RMSE=%.3f' % (order,rmse))
        except:
          continue
  print('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))
  log.write('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))


# load dataset
#series = Series.from_csv(filename, header=0)
series = Series.from_csv(filename, header=0, parse_dates=[0], index_col=0)
# evaluate parameters
p_values = [0, 1, 2, 4, 6, 8, 10]
d_values = range(0, 3)
q_values = range(0, 3)

warnings.filterwarnings("ignore")
evaluate_models(series.values, p_values, d_values, q_values)
log.close()
```

APÊNDICE E – MÉTODO DE *TUNING* DE HIPERPARÂMETRO: TPE PARA MLP

Comando de entrada:

```
python hypeas_mlp.py trace5.csv
```

Resultado da execução: No final da execução, o algoritmo de otimização retorna uma linha de comando com os hiperparâmetros otimizados, no formato de entrada do programa mlp.py

```
'python mlp.py trace5.csv 110 180 50 25'
```

```
# -*- coding: utf-8 -*-
#!pip install hyperas
from __future__ import print_function
from hyperopt import Trials, STATUS_OK, tpe
from hyperas import optim
from hyperas.distributions import choice, uniform
from hyperas.utils import eval_hyperopt_space, space_eval

from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import RMSprop

from keras.utils import np_utils
import os, sys
import pandas as pd
import numpy as np

def data():
    '''
    Data providing function:
    This function is separated from model() so that hyperopt
    won't reload data for each evaluation run.
    '''

    # good practice to set the seed state, starting point
    #receive command line parameter
    filename = str(sys.argv[1])
    dataset = pd.read_csv(filename, usecols=[1], header=None)
    dataset = dataset.values #convert to the array
    dataset = dataset.astype('float32') # convert to float

    # lenth of our data set
    training_size = int(len(dataset)*0.67)
    #testing_size = len(dataset)-training_size

```

```
    # split the data set
    train, test = dataset[0:training_size,:], dataset[training_size:len(dataset)
    ,:]

    # one time step to the future
    lookback = 1
    #dataX, dataY = [], [] # create 2 empty list
    #-----------------------------------------
    dataX_train, dataY_train = [], [] # create 2 empty list
    dataX_test, dataY_test = [], [] # create 2 empty list
    np.random.seed(7)

    for i in range(len(dataset)-lookback-1):
        a = dataset[i:(i+lookback),0]
        dataX_train.append(a)
        dataY_train.append(dataset[i+lookback,0]) # get the next value

        X_train, Y_train = np.array(dataX_train), np.array(dataY_train)

    for i in range(len(dataset)-lookback-1):
        a = dataset[i:(i+lookback),0]
        dataX_test.append(a)
        dataY_test.append(dataset[i+lookback,0]) # get the next value

        X_test, Y_test = np.array(dataX_test), np.array(dataY_test)
    #-----------------------------------------
    #X_train, Y_train = create_dataset(train, lookback)
    #X_test, Y_test = create_dataset(test, lookback)

    return X_train, Y_train, X_test, Y_test

def model(X_train, Y_train, X_test, Y_test):

    lookback = 1
    model=Sequential()
    model.add(Dense({{choice([30,60,80])}}, input_dim=lookback, activation='relu
    '))
    model.add(Dropout(0.2))
    model.add(Dense({{choice([10,30,60])}}, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1))

    model.compile(loss='mean_squared_error', optimizer="adam", metrics=['mae'])

    model.fit(X_train, Y_train,
              batch_size={{choice([130,180])}},
              nb_epoch={{choice([130,200])}},
              verbose=2,
```

```
82                 validation_data=(X_test, Y_test))
83     score, mae = model.evaluate(X_test, Y_test, verbose=0)
84     print('Test mae:', mae)
85     return {'loss': -mae, 'status': STATUS_OK, 'model': model}
86 
87 
88 if __name__ == '__main__':
89     trials=Trials()
90     best_run, best_model, space = optim.minimize(model=model,
91                                                  data=data,
92                                                  algo=tpe.suggest,
93                                                  max_evals=5,
94                                                  trials=trials,
95                                                  eval_space=True,
96                                                  return_space=True)
97     X_train, Y_train, X_test, Y_test = data()
98     print("Evalutation of best performing model:")
99     print(best_model.evaluate(X_test, Y_test))
100    print('')
101    print("Best performing model chosen hyper-parameters:")
102    print(best_run)
103    print('')
104    print(filename)
105    print('BATCH_SIZE =', best_run.get('batch_size'))
106    print('NB_EPOCHS =', best_run.get('nb_epoch'))
107    print('LAYER1 =',best_run.get('Dense'))
108    print('LAYER2 =',best_run.get('Dense_1'))
109    print('')
110
111    print("python gru.py %s %d %d %d %d" %(filename, best_run.get('batch_size'),
       best_run.get('nb_epoch'), best_run.get('Dense'), best_run.get('Dense_1')))
112
113    f=open("MLP_hyperparameters.txt", "a")
114    f.write("\n python mlp.py %s %d %d %d %d " %(filename, best_run.get('
       batch_size'), best_run.get('nb_epoch'), best_run.get('Dense'), best_run.get('
       Dense_1')))
115
116    '''for t, trial in enumerate(trials):
117        vals = trial.get('misc').get('vals')
118        print("Trial %s vals: %s" % (t, vals))
119        tmp = {}
120        for k,v in list(vals.items()):
121            tmp[k] = v[0]
122        print('')
123
124    print(eval_hyperopt_space(space, tmp))
125    '''
```

APÊNDICE F – MÉTODO DE *TUNING* DE HIPERPARÂMETRO: TPE PARA GRU

Comando de entrada:

```
python hypeas_gru.py trace5.csv
```

Resultado da execução: No final da execução, o algoritmo de otimização retorna uma linha de comando com os hiperparâmetros otimizados, no formato de entrada do programa gru.py

```
'python gru.py trace5.csv 130 130 128 64'
```

```
# -*- coding: utf-8 -*-
#!pip install hyperas
from __future__ import print_function
from hyperopt import Trials, STATUS_OK, tpe
from hyperas import optim
from hyperas.distributions import choice, uniform
from keras.layers.recurrent import GRU
from hyperas.utils import eval_hyperopt_space, space_eval

from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import RMSprop
from sklearn.preprocessing import MinMaxScaler
from keras.utils import np_utils
import os, sys
import pandas as pd
import numpy as np

#receive command line parameter
filename = str(sys.argv[1])

def data():

    #Data providing function:
    #This function is separated from model() so that hyperopt
    #won't reload data for each evaluation run.

    filename = str(sys.argv[1])
    dataset = pd.read_csv(filename, usecols=[1], header=None)
    dataset = dataset.values #convert to the array
    dataset = dataset.astype('float32') # convert to float

    # lenth of our data set
    training_size = int(len(dataset)*0.67)
    #testing_size = len(dataset)-training_size
```

```
36
37      # split the data set
38      train, test = dataset[0:training_size,:], dataset[training_size:len(dataset)
        ,:]
39
40      # one time step to the future
41      lookback = 1
42
43      #dataX, dataY = [], [] # create 2 empty list
44
45      dataX_train, dataY_train = [], [] # create 2 empty list
46      dataX_test, dataY_test = [], [] # create 2 empty list
47      np.random.seed(7)
48      for i in range(len(dataset)-lookback-1):
49          a = dataset[i:(i+lookback),0]
50          dataX_train.append(a)
51          dataY_train.append(dataset[i+lookback,0]) # get the next value
52
53          X_train, Y_train = np.array(dataX_train), np.array(dataY_train)
54
55      for i in range(len(dataset)-lookback-1):
56          a = dataset[i:(i+lookback),0]
57          dataX_test.append(a)
58          dataY_test.append(dataset[i+lookback,0]) # get the next value
59
60          X_test, Y_test = np.array(dataX_test), np.array(dataY_test)
61
62      # scaling values for model
63      scaleX = MinMaxScaler()
64      scaleY = MinMaxScaler()
65
66      X_train = scaleX.fit_transform(X_train)
67      X_train = X_train.reshape((-1,1,1))
68
69      Y_train = scaleY.fit_transform(Y_train.reshape(-1,1))
70
71      X_test  = scaleX.fit_transform(X_test)
72      X_test = X_test.reshape((-1,1,1))
73
74      Y_test  = scaleY.fit_transform(Y_test.reshape(-1,1))
75
76      return X_train, Y_train, X_test, Y_test
77
78  def model(X_train, Y_train, X_test, Y_test):
79      '''
80      Model providing function:
81      Create Keras model with double curly brackets dropped-in as needed.
82      Return value has to be a valid python dictionary with two customary keys:
```

```
83          - loss: Specify a numeric evaluation metric to be minimized
84          - status: Just use STATUS_OK and see hyperopt documentation if not
        feasible
85      The last one is optional, though recommended, namely:
86          - model: specify the model just created so that we can later use it
        again.
87      '''
88      lookback = 1
89
90      model = Sequential()
91      model.add(GRU(units={{choice([32,128,256])}},
92                    return_sequences=True,
93                    input_shape=(1, 1)))
94      model.add(Dropout(0.2))
95      model.add(GRU(units={{choice([16,32,64])}}))
96
97      model.add(Dropout(0.2))
98      model.add(Dense(1, activation='sigmoid'))
99
100     model.compile(loss='mean_squared_error', optimizer="adam", metrics=['mae'])
101
102     model.fit(X_train, Y_train,
103               batch_size={{choice([130,180])}},
104               nb_epoch={{choice([130,200])}},
105               verbose=2,
106               validation_data=(X_test, Y_test))
107     score, mae = model.evaluate(X_test, Y_test, verbose=0)
108     print('Test mae:', mae)
109     return {'loss': -mae, 'status': STATUS_OK, 'model': model}
110
111
112 if __name__ == '__main__':
113     trials=Trials()
114     best_run, best_model, space = optim.minimize(model=model,
115                                                  data=data,
116                                                  algo=tpe.suggest,
117                                                  max_evals=5,
118                                                  trials=trials,
119                                                  eval_space=True,
120                                                  return_space=True)
121     X_train, Y_train, X_test, Y_test = data()
122     print("Evalutation of best performing model:")
123     print(best_model.evaluate(X_test, Y_test))
124     print('')
125     print("Best performing model chosen hyper-parameters:")
126     print(best_run)
127     print('')
128     print(filename)
```

```
129
130     print('BATCH_SIZE =', best_run.get('batch_size'))
131     print('NB_EPOCHS =', best_run.get('nb_epoch'))
132     print('LAYER1 =',best_run.get('units'))
133     print('LAYER2 =',best_run.get('units_1'))
134     print('')
135     print("python hypeas_gru.py %s %d %d %d %d" %(filename, best_run.get('
        batch_size'), best_run.get('nb_epoch'),
136           best_run.get('units'), best_run.get('units_1')))
137
138     # Save into a file the command of executions and its scores
139     f=open("gru_hyperparameters.txt", "a")
140     f.write("\n python gru.py %s %d %d %d %d" %(filename, best_run.get('
        batch_size'), best_run.get('nb_epoch'), best_run.get('units'),      best_run.
        get('units_1')))
```

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br