Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Avaliação do Uso de Bibliotecas de Aprendizagem de Máquina para Integração com Simuladores de Redes Ópticas Elásticas

Mikael Marques Mello

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. André Costa Drummond

Brasília
2020

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Avaliação do Uso de Bibliotecas de Aprendizagem de Máquina para Integração com Simuladores de Redes Ópticas Elásticas

Mikael Marques Mello

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. André Costa Drummond (Orientador)
CIC/UnB

Prof. Dr. Eduardo Adilio Pelinson Alchieri
CIC/UnB

Prof. Dr. Marcelo Antonio Marotta
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 7 de dezembro de 2020

Dedicatória

Dedico este trabalho à minha querida e amada noiva, Bruna Barros, por sempre estar ao meu lado e ter sido uma fonte interminável de amor e carinho desde que a conheci, incentivando-me a ser uma pessoa melhor. Os últimos dois anos e os próximos que virão não poderiam ser melhores.

Dedico à minha mãe, Rosicler, por sempre ter feito o possível e o impossível para que nada me faltasse, todas minhas conquistas só foram possíveis graças à sua dedicação em dar tudo pelos seus filhos.

Dedico ao meu pai, Gilvan, por sempre ter me fornecido os meios e oportunidades que eu precisava para alcançar meus objetivos, minhas conquistas também não seriam possíveis sem seu enorme apoio.

Agradecimentos

Primeiramente, gostaria de agredecer ao meu orientador, André Costa Drummond, por todo o auxílio e prestatividade desde o começo do ano pré-pandemia, e pela incrível didática de ensino que pude prestigiar ao assistir aulas da disciplina Redes de Computadores.

Gostaria de agradecer também aos professores e professoras Carla Castanho, Cláudia Nalon, Flávio Moura, Jorge Lucero e Marcus Lamar, pela maravilhosa qualidade de ensino em disciplinas que considero favoritas na minha graduação e felizmente tive o prazer de estudá-las com professores excelentes.

Agradeço imensamente à comunidade de Maratona de Programação, a qual acredito que minha participação foi crucial para o meu bom desempenho acadêmico e profissional. Agradecimentos especiais aos professores Guilherme Ramos, Edson Alves, Vinícius Borges e diversos outros, por todo o trabalho duro que mantém a comunidade local de programação competitiva viva e por se encarregarem de divulgarem e incentivarem esta prática pela UnB e em outras instituições de ensino, consideravelmente aumentando o nível da nossa universidade e da região. Além disso, agradeço ao Matheus Pimenta, por sua dedicação em incentivar tantos alunos, eu incluso, a participarem desta incrível experiência.

Por fim, agradeço também a diversos amigos que conheci durante esta caminhada e que sempre estiveram presentes para me ajudar com diversas questões acadêmicas e profissionais: Gabriel Nunes, Giovanni Guidini, Thiago Veras, Léo Moraes, Gabriel Levi, José Leite, Luis Felipe e Gabriel Taumaturgo.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Redes ópticas elásticas são consideradas por pesquisadores uma das melhores soluções atuais para lidar com o crescente tráfego global de dados, provendo diversos benefícios ao serem comparadas com alternativas tradicionais, sendo a principal vantagem a possibilidade de alocar recursos físicos de forma flexível, o que viabiliza estratégias mais eficientes de alocação. Pelo fato de estratégias tradicionais de alocação de recursos não serem apropriadas para o cenário flexível, diversos estudos sobre novas estratégias de alocação têm sido desenvolvidos. Devido à complexidade de redes ópticas elásticas, tais estudos utilizam simulações para validar seus resultados. Recentemente, o uso de aprendizagem de máquina como ferramenta para o desenvolvimento de estratégias tem crescido, entretanto simuladores atuais da literatura não possuem suporte para uso de modelos de aprendizagem de máquina em suas implementações. Este trabalho apresenta uma análise qualitativa e quantitativa de bibliotecas de aprendizagem de máquina populares na literatura com o objetivo de definir direções e recomendações para futuras implementações, e realiza contribuições ao *Optical Network Simulator* (ONS) com implementações de funcionalidades para dar suporte ao uso de modelos de aprendizagem de máquina em simulações.

Palavras-chave: redes ópticas elásticas, aprendizagem de máquina

Abstract

Elastic optical networks are currently considered one of the best solutions in regards to handling the increasing global data traffic, providing several benefits compared to traditional alternatives. The flexibility in resources allocation allows it to be done in an adaptable and efficient manner. However, traditional resource allocation algorithms are not appropriate for a flexible scenario, resulting in several studies aiming to develop new allocation strategies. Because of the complexity of elastic optical networks, these studies often use simulations to validate their results. Recently, the use of machine learning as a tool for the development of strategies has been growing, although simulators currently found in the literature do not have support for using machine learning models in their implementations. This work presents a qualitative and quantitative analysis of popular machine learning libraries in the literature in order to define directions and recommendations for future implementations, and contributes to the *Optical Network Simulator* (ONS) by implementing features supporting the use of machine learning models in simulations.

Keywords: elastic optical networks, machine learning

Sumário

Lista de Tabelas

Lista de Abreviaturas e Siglas

API *Application Programming Interface.*

CEONS *Complex Elastic Optical Network Simulator.*

COMNET-UnB Laboratório de Redes de Computadores da Universidade de Brasília.

CUDA *Compute Unified Device Architecture.*

DL *Deep Learning.*

EON *Elastic Optical Networking.*

EONs *Elastic Optical Networks.*

ML *Machine Learning.*

N-WDM *Nyquist WDM.*

OAWG *Optical Arbitrary Waveform Generation.*

OFDM *Orthogonal Frequency Division Multiplexing.*

ONNX *Open Neural Network Exchange.*

ONS *Optical Network Simulator.*

RMLSA *Routing, Modulation Level and Spectrum Allocation.*

RSA *Routing and Spectrum Assignment.*

WDM *Wavelengh-Division Multiplexing.*

Capítulo 1

Introdução

Historicamente, o tráfego de internet global cresce de forma exponencial, possuindo uma taxa composta de crescimento anual de 45% nos anos 2000 [2] e aproximadamente 30% nos anos 2010 [3, 4, 5, 6, 7, 8, 9, 10]. Uma fatia significativa deste crescimento anual se deve ao tráfego de dados em redes móveis, cujas taxas de crescimento anuais na última década tem variado entre 50% a 60% e são motivadas pelo crescente número de assinaturas de *smartphones* e o volume de dados consumido por assinatura, este alavancado principalmente pelo crescente consumo de conteúdos de vídeo [11]. Além disso, circunstâncias especiais podem incentivar a população a aumentar seu uso de internet: Em Abril de 2020, a empresa Akamai relatou um crescimento de 30% do tráfego global em apenas um mês, aproximadamente dez vezes a taxa de crescimento esperada, atribuindo o pico de crescimento às mudanças de estilo de vida causadas pela pandemia do COVID-19 [12].

A infraestrutura responsável por lidar com tamanho tráfego é composta por redes ópticas que têm sido tradicionalmente *rígidas* e *homogêneas*, isto é, redes baseadas em *Wavelengh-Division Multiplexing* (WDM), ou multiplexação por divisão de comprimento de onda. Redes baseadas em WDM oferecem a possibilidade de estabelecer conexões com comprimentos de onda fixos e com uma taxa de *bits* fixa, em que os canais são modulados com um formato comum e espaçados por uma distância fixa de 50 GHz [13, 14]. O crescimento exponencial de demanda motivou a indústria a focar esforços em aumentar a capacidade destas redes, resultando na evolução das taxas de *bits* permitidas de inicialmente 10 Gb/s para 40 Gb/s e por fim 100 Gb/s [15]. Entretanto, este tipo de rede conta com alguns problemas:

- **Baixa adaptabilidade**. A flexibilidade destas redes é limitada pela configuração do *hardware*, tornando o processo de atualizar ou modificar a rede para adaptar-se às mudanças de demanda ou de condições de rede desafiador [13].

- **Baixa eficiência espectral**. O desenho da rede deve garantir que o caminho óptico mais longo (pior caso) seja transmitido com qualidade suficiente. Como os comprimentos de onda são fixos e homogêneos, a maioria das conexões irá possuir comprimentos muito menores do que o pior caso, gerando um problema de ineficiência onde há faixas de comprimentos de onda não utilizadas [14, 15].
- **Limite de futuros avanços**. Com taxas de bits maiores que 100 Gb/s, melhorias na eficiência espectral ao aumentar o número de *bits* por símbolo se torna cada vez mais difícil devido ao limite de Shannon, além do fato de que aumentar a taxa para além de 100 Gb/s é um desafio por si só [14].

Por estas dificuldades, o conceito de *Elastic Optical Networking* (EON), ou redes ópticas elásticas, foi introduzido como um modo de oferecer uma utilização eficiente dos recursos ópticos disponíveis, sendo capaz de acomodar taxas de *bits* que variam desde algumas dezenas de Gb/s até a magnitude de *terabits* por segundo [13, 16], além da alocação adaptável de recursos de *hardware* e espectrais de acordo com a demanda do tráfego [14]. Na literatura, os termos "flexível", "elástico", "flexgrid ou flexigrid", "gridless"e "adaptável"são usados intercambiavelmente. Esta alocação flexível é permitida graças ao uso de tecnologias como *Orthogonal Frequency Division Multiplexing* (OFDM), *Nyquist WDM* (N-WDM) e *Optical Arbitrary Waveform Generation* (OAWG) [13]. EONs têm sido amplamente aceitas como uma das melhores soluções com arquiteturas de rede flexíveis e capazes de alocar recursos de forma flexível [17].

Quanto ao design e otimização de *Elastic Optical Networks* (EONs), um dos principais desafios em seu desenvolvimento se trata da alocação eficiente de recursos. Algoritmos e ferramentas de planejamento de rede convencionais (WDM) não podem ser aplicados devido à natureza flexível das redes. Assim, de modo a aproveitar completamente a flexibilidade disponibilizada pelos avanços de tecnologias na camada física, novos algoritmos de alocação de recursos têm sido explorados [13, 18, 19, 20, 21]. Por exemplo, com flexibilidade apenas no número de subportadoras disponíveis para alocação, devem ser usadas técnicas de roteamento e alocação de espectro, *Routing and Spectrum Assignment* (RSA). Caso haja flexibilidade na seleção do formato de modulação, então técnicas de roteamento e atribuição de espectro com modulação adaptativa, *Routing, Modulation Level and Spectrum Allocation* (RMLSA), devem ser escolhidas.

A avaliação de desempenho de sistemas de comunicação óptica é um desafio para os pesquisadores. A dinamicidade e complexidade, especialmente em redes ópticas elásticas, torna inviável uma modelagem analítica precisa e o uso de ambientes reais para medições torna a avaliação bastante custosa, devido principalmente aos equipamentos e ferramentas envolvidos. Assim, a simulação é a alternativa disponível para atividades de teste,

validação e avaliação de novos mecanismos de controle para o ambiente de redes ópticas [22].

Diferentes ferramentas de simulação de redes ópticas elásticas foram desenvolvidas para auxiliar os pesquisadores a implementar, testar e analisar novos algoritmos ou soluções de problemas diversos na área. Como por exemplo, *Optical Network Simulator* (ONS) [22], *ElasticO++* [23], *Complex Elastic Optical Network Simulator* (CEONS) [24] e *Net2Plan* [25].

Recentemente, com a crescente popularização do uso de *Machine Learning* (ML), ou aprendizagem de máquina, na academia, estudos sobre a aplicação de ML para solução de problemas relacionados às EONs também têm se popularizado.

Machine Learning é o estudo de algoritmos de computação que se auto-otimizam de acordo com um critério de desempenho, usando dados de exemplo ou a própria experiência [26, 27]. Em sua forma mais básica, o método de ML constitui-se em coletar dados relevantes ao domínio do problema a ser resolvido para serem usados por um algoritmo de aprendizagem, o modelo. Este modelo é definido com parâmetros iniciais que são otimizados automaticamente a partir do consumo dos dados coletados. O objetivo do modelo pode ser tanto preditivo, para realizar predições no futuro sobre dados potencialmente desconhecidos; descritivo, para obter-se conhecimento novo acerca dos dados; ou ambos [27, 28].

Dentre recentes usos de ML em pesquisas relacionadas às EONs, pode-se citar a pesquisa de Yu et al., que desenvolveu uma estratégia RSA baseada em *Deep Learning* (DL) [17], ou aprendizagem profunda, um subgrupo de ML. Guilherme et al. desenvolveram um modelo de DL capaz de identificar estratégias RSA em EONs com 98% de acurácia [29]. Outras pesquisas acerca do uso de ML em problemas na área de EONs podem ser encontradas em [30] e [31].

Como o crescimento do uso de ML em pesquisas no campo de EONs é recente, simuladores de redes ópticas elásticas disponíveis na literatura ainda não possuem integração com modelos de ML para uso durante as simulações. O presente trabalho visa analisar na literatura atual as bibliotecas de ML capazes de importar modelos pré-treinados, especificamente no campo de aprendizagem profunda como o desenvolvido por Yu et al. [17], no contexto de integrá-las à execução de simulações de EONs.

1.1 Problema

Devido ao recente crescimento do uso de ML em pesquisas no campo de EONs, simuladores de redes ópticas elásticas disponíveis na literatura ainda não possuem suporte nativo à execução de modelos de ML para uso durante as simulações. Assim, pesquisas que

envolvem o uso de ML para a solução de problemas como RSA não possuem um *framework* definido para analisar o desempenho de seus algoritmos ou soluções, sendo necessárioas soluções *ad-hoc* para a análise de resultados.

1.2 Objetivos

O objetivo geral deste trabalho é possibilitar que pesquisas que envolvem ML em EONs usem simuladores de EONs para a análise, validação e comparação de resultados de modelos desenvolvidos, dado que um dos objetivos de simuladores é fornecer uma *framework* comum para a realização destas atividades.

Este objetivo geral será atingido por meio dos seguintes objetivos específicos:

- Estudar as bibliotecas de código existentes na literatura capazes de executar modelos de ML;
- Analisar qualitativamente bibliotecas e realizar uma seleção contendo escolhas apropriadas para o caso de uso de integração com simuladores de EONs;
- Avaliar de forma quantitativa o desempenho das bibliotecas com métricas e parâmetros pré-definidos;
- Analisar os resultados das avaliações e elencar as bibliotecas recomendadas para diferentes casos de uso;
- Realizar implementações de integração de ML com o ONS como prova de conceito e contribuição para a comunidade acadêmica.

1.3 Contribuições

A partir das análises qualitativas e quantitativas das bibliotecas de *machine learning* populares na literatura, este trabalho contribui com diversas recomendações acerca de como adicionar suporte para uso de modelos de aprendizagem de máquina em simuladores de EONs, considerando diversos casos de uso de acordo com as necessidades do usuário.

Adicionalmente, são feitas contribuições ao repositório do simulador ONS, onde são realizadas as seguintes implementações: implementação de funcionalidades que permitem a execução de modelos de forma nativa em Java, com as bibliotecas ONNX Runtime e Deeplearning4j; e implementação de uma funcionalidade que permite a comunicação do ONS com um sistema externo por meio de HTTP, possibilitando a executação de modelos de ML de forma abstrata ao ONS. É esperado que estas implementações sejam a base para futuros avanços na integração entre ML e simulações de EONs.

1.4 Organização do trabalho

O trabalho está organizado em 6 capítulos com os seguintes propósitos:

- Capítulo 2 - Introdução de conceitos considerados essenciais para o entendimento do trabalho. Serão descritos o uso de simuladores para pesquisas no campo de redes ópticas elásticas, conceitos de ML usados ao longo do trabalho e a metodologia de avaliação de desempenho utilizada neste trabalho.
- Capítulo 3 - Descrição das bibliotecas de ML populares na literatura, sendo explicadas suas principais características e o racional para a consideração delas ou não na análise de desempenho.
- Capítulo 4 - Descrição da avaliação de desempenho das bibliotecas selecionadas, como métricas utilizadas, carga de trabalho, a máquina em que os testes foram realizados, entre outros, além da análise dos resultados da avaliação.
- Capítulo 5 - Considerando o apresentado nos capítulos 3 e 4, serão descritas as recomendações para o suporte do uso de ML em simuladores de acordo com diferentes casos de uso, e as contribuições feitas para o *Optical Network Simulator* (ONS).
- Capítulo 6 - Serão apresentados os resultados obtidos nas avaliações, as recomendações de futuras implementações e as contribuições realizadas ao *Optical Network Simulator* (ONS).

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta conceitos básicos fundamentais para o melhor entendimento do trabalho. Na Seção 2.1 são introduzidos breves conceitos acerca de simulações de EONs e a magnitude de trabalho normalmente envolvida. Na Seção 2.2 são introduzidos conceitos fundamentais de aprendizagem de máquina e uma breve descrição de suas principais categorias. Por fim, na Seção 2.3 é apresentada a metodologia de avaliação de desempenho utilizada neste trabalho.

2.1 Usos de Simulações de EONs

A instalação e gerenciamento de redes ópticas elásticas envolve o uso de dispositivos óticos e infraestruturas de comunicação normalmente caros. Além disso, a maioria dos equipamentos utilizados são protegidos por leis de propriedade intelectual e não disponibilizam meios para modificações na programação de seu funcionamento.

Pelo alto custo, o uso de equipamentos e redes reais para o desenvolvimento de pesquisas na área de redes ópticas elásticas se tornou inviável, tornando necessária a busca por outras alternativas. Sua dinamicidade e complexidade não permite uma modelagem analítica precisa [22], já a simulação se mostra uma excelente alternativa pelo fato de permitir que pesquisadores validem e avaliem de forma fácil e barata o uso de novos protocolos e algoritmos [32].

Para auxiliar pesquisadores com ferramentas que agilizam suas pesquisas, diversos simuladores de redes ópticas elásticas têm sido publicados nos últimos anos. Como por exemplo, *Optical Network Simulator* (ONS) [22], *ElasticO++* [23], *Complex Elastic Optical Network Simulator* (CEONS) [24] e *Net2Plan* [25]. Também é válido mencionar a existência de simuladores de redes ópticas não-elásticas, como SimulNet [32] e SIMTON [33].

Dentre os diferentes simuladores de EONs existem vastas diferenças de arquitetura, funcionalidades e portabilidade. Entretanto, todas seguem um mesmo fluxo básico: dado uma requisição e o estado atual da rede, um algoritmo de alocação é executado para que seja decidido se a requisição será aceita ou não, e caso aceita quais recursos lhe serão alocados.

Em pesquisas que utilizam o ONS para a realização das simulações, é comum que apenas os resultados finais de um relatório envolvam a execução de dezenas de simulações, possivelmente cem ou mais.

Em uma simulação executada pelo *Optical Network Simulator* (ONS), é comum que o número de requisições processadas chegue até a magnitude de centenas de milhares, podendo chegar a milhões. Em [34], Leia et al. utilizam simulações que geram 10^5 eventos por execução em um estudo que propõe um novo algoritmo RSA. Costa et al. também utilizam simulações que geram 10^5 eventos por execução em [35], onde propõem um novo algoritmo RMLSA.

Assim, a execução de simulações para a elaboração dos resultados finais de um trabalho pode envolver um total de execuções perto da magnitude de 10^8. Além disso, um processo de pesquisa envolve múltiplas iterações na etapa de desenvolvimento, acarretando em constantes repetições de diversas simulações.

Devido a esta alta quantidade de eventos processados em simulações, é importante que algoritmos usados para processar estes eventos sejam eficientes. Por isto, ao buscar adicionar suporte de execução de modelos de aprendizagem de máquina em simuladores, é importante considerar o desempenho das tecnologias usadas de modo a não se tornarem um gargalo no desempenho geral de simulações.

2.2 Aprendizagem de Máquina

Machine Learning (ML), ou aprendizagem de máquina é o estudo de algoritmos de computação que se auto-otimizam de acordo com um critério de desempenho, usando dados de exemplo ou a própria experiência [26, 27].

Em sua forma mais básica, o método de ML é a coleta de um grande número de dados do domínio do problema a ser resolvido e o uso deles em um algoritmo de aprendizagem, o modelo. Este modelo é definido com parâmetros iniciais que são otimizados com o consumo dos dados. O objetivo do modelo pode ser tanto preditivo, para realizar predições no futuro sobre dados potencialmente desconhecidos, descritivo, para obter-se conhecimento novo acerca dos dados, ou ambos.

Há dois grandes desafios no campo de ML: em primeiro lugar, são necessários algoritmos eficientes para o problema de otimização do modelo inicial, de modo que a fase

de treinamento seja completada em tempo viável; em segundo lugar, uma vez que um modelo tenha sido aprendido, sua representação e solução algorítimica também devem ser eficientes [27, 28].

Normalmente, o campo de ML é dividido em três principais categorias: *supervised learning*; *unsupervised learning*; e *reinforcement learning*.

2.2.1 Aprendizado supervisionado

Em inglês, *supervised learning* é a categoria em que o conjunto de dados de entrada (*training set*) possui um mapeamento para o comportamento esperado, rotulado por um "supervisor"[27]. O objetivo é aprender uma regra geral que mapeie os valores de entrada para os respectivos valores esperados de saída. Os valores de saída podem ser contínuos (problemas de regressão) ou discretos (problemas de classificação) [36].

2.2.2 Aprendizado não-supervisionado

Em inglês, *unsupervised learning* é a categoria em que existe apenas o conjunto de dados de entrada e o objetivo é encontrar regularidades presentes nos mesmos [27]. Este tipo de aprendizagem é capaz de desempenhar várias tarefas, porém a mais comum é *clustering* [36].

Clustering é o processo de agrupar dados de modo que a similaridade de dados nos grupos (*clusters*) é alta, porém a similaridade de dados entre grupos diferentes é baixa. Esta similaridade é tipicamente expressada como uma função de distância, que depende do tipo de dados presente no conjunto [36].

Dentre os usos de aprendizagem não-supervisionada, pode-se destacar análise de redes sociais, agrupamento de genes e pesquisas de mercado como análises de aplicações bem-sucedidas [36].

2.2.3 Aprendizado por reforço

Em inglês, *reinforcement learning* é a categoria em que o algoritmo de aprendizagem tem como objetivo aprender uma política de ações que maximizem a recompensa em um dado ambiente [27].

O paradigma de *reinforcement learning* permite que agentes explorem possíveis ações e refinem seu comportamento utilizando apenas uma avaliação, conhecida como recompensa, tendo como objetivo maximizar seu desempenho de longo prazo [36].

Esta técnica é comumente usada em aplicações como robótica, área de finanças como decisões de investimentos e gerenciamento de estoque [36].

2.3 Avaliações de Desempenho

Avaliações são importantes na busca pelo melhor desempenho de um sistema com os recursos disponíveis. Seus resultados auxiliam tanto nas decisões de escolhas entre diferentes sistemas ou simplesmente entender o funcionamento de um sistema já existente. Devido à grande diversidade de sistemas, não existe um procedimento padrão comum em que seja possível analisar eficientemente um sistema qualquer, sendo necessário conhecer o sistema a ser avaliado e escolher as métricas, carga de trabalho e técnicas de avaliação apropriadas [1].

Uma simulação executada por simuladores de EONs costuma envolver dezenas de milhares de eventos, como requisições de conexões. Tamanha magnitude do número de eventos representa a importância de um bom desempenho na execução de modelos, como por exemplo em propostas de soluções para problemas de alocação de recursos (e.g. RSA) cujos modelos seriam executados em cada evento. Assim, uma análise quantitativa do desempenho de bibliotecas de ML é importante na busca por uma solução de integração de ML com simuladores que seja flexível de acordo com as necessidades de cada pesquisa e que possua um bom desempenho de modo a acelerar a obtenção de resultados.

Raj Jain em seu livro [1] propôs uma abordagem sistemática para a realização de análises de desempenho, descrita passo a passo nas próximas subseções.

2.3.1 Definição dos objetivos do estudo e do sistema

A primeira etapa se trata de definir o objetivo do estudo e os limites do sistema a ser analisado. A definição do sistema consiste em delinear os limites do sistema, como por exemplo uma biblioteca de serialização em JSON cujos serviços são funções para serializar e desserializar um conjunto de dados.

Definir o sistema e os objetivos do estudo é importante pois as métricas de desempenho e as cargas de trabalho usadas na análise de desempenho depende da definição do sistema.

2.3.2 Listar serviços e resultados

Cada sistema fornece um conjunto de serviços, como as funções da biblioteca de serialização em JSON previamente mencionadas. Os serviços fornecidos pelo sistema possuem um conjunto de possíveis resultados que podem ou não ser desejáveis (i.e. erros). A listagem dos serviços e de seus resultados considerados é importante para definir as métricas e cargas de trabalho da análise de desempenho.

Critério	Modelagem Analítica	Simulação	Medição
1. Estágio	Qualquer	Qualquer	Pós-protótipo
2. Tempo necessário	Pequeno	Médio	Variado
3. Ferramentas	Analistas	Ling. de programação	Instrumentação
4. Acurácia	Baixa	Moderada	Variada
5. Comparações	Fácil	Moderado	Difícil
6. Custo	Baixo	Médio	Alto
7. Vendabilidade	Baixo	Média	Alta

Tabela 2.1: Critérios para a seleção de uma técnica de avaliação de desempenho [1].

2.3.3 Selecionar métricas

As métricas são os critérios usados para avaliar o desempenho de um sistema ou comparar alternativas de sistema. Normalmente, as métricas são relacionadas à velocidade, acurácia e disponibilidade de serviços.

2.3.4 Listar parâmetros e fatores

A lista de parâmetros representa os componentes de uma análise de desempenho que podem afetar o desempenho. Esta lista pode ser dividida entre parâmetros de sistema, que podem incluir parâmetros de ambos tipos *software* e *hardware* e não variam entre instâncias do sistema e parâmetros de carga de trabalho, que podem variar entre instâncias, como por exemplos requisições de usuário.

Os fatores são os parâmetros que são variados durante a avaliação de desempenho, como por exemplo quantidade de usuários ou tipos de requisição, e os valores que cada um dos fatores pode assumir são chamados de níveis.

2.3.5 Escolher a técnica de avaliação

Há três métodos de avaliação de desempenho: modelagem analítica, simulação e medição [1]. A escolha de uma destas técnicas depende de vários critérios a serem considerados de acordo com o sistema a ser avaliado, como por exemplo o estágio do sistema, tempo disponível para a análise e a acurácia requerida. A Tabela 2.1 lista de forma resumida as considerações para a escolha de uma técnica de avaliação de desempenho.

2.3.6 Selecionar carga de trabalho

A carga de trabalho consiste de uma lista de requisições de serviço do sistema e pode ser representada de diferentes modos de acordo com a técnica de avaliação escolhida.

Em avaliações realizadas por modelagem analítica, a carga de trabalho pode ser definida como a probabilidade de várias requisições. Para simulações, pode ser uma lista de requisições de usuário coletadas de um sistema real. Para medições, podem ser *scripts* automatizados representando usuários no sistema real. É de fundamental importância que a carga de trabalho represente casos de uso reais do sistema.

2.3.7 Desenho de experimentos

Ao definir a lista de fatores é necessário decidir uma sequência de experimentos que forneça o máximo de informações com o mínimo de esforço necessário. Existem várias formas de desenhar experimentos sendo os 3 mais comuns o desenho simples, desenho fatorial completo e desenho fatorial fracionário.

Em um desenho simples, o primeiro experimento é realizado com uma configuração comum do sistema e um fator é variado de cada vez. Este desenho não é eficiente estatisticamente e se os fatores interagirem entre si, pode levar a conclusões erradas.

Em um desenho fatorial completo, todas as combinações possíveis de todos os níveis de todos os fatores são considerados. A principal vantagem é que todas as combinações são examinadas permitindo a análise do impacto de cada fator, incluindo fatores secundários e suas interações. A principal desvantagem é o custo do estudo pelo alto número de experimentos a serem realizados. É comum a utilização de técnicas para reduzir o número de combinações e consequentemente o custo.

Desenhos fatorial fracionário são utilizados quando o número de experimentos necessário para a realização do fatorial completo é muito alto. Este desenho consiste em escolher um subconjunto (fração) das combinações de um fatorial completo, uma vez que é comum haverem experimentos redundantes em um desenho fatorial completo.

2.3.8 Apresentação de resultados

Nesta etapa, os resultados são apresentados de forma consistente de acordo com o público-alvo, preferencialmente com auxílio de elementos visuais como gráficos e tabelas.

Capítulo 3

Análise de Bibliotecas

Pela visão de auxiliar pesquisadores em suas integrações de modelos de ML com simuladores de redes ópticas elásticas, as análises aqui feitas devem buscar o aumento de produtividade do pesquisador. Para isto, as tecnologias de ML analisadas serão avaliadas de forma qualitativa, sendo esta avaliação guiada pelas seguintes questões:

- Há algum custo para usar a tecnologia?
- O código da tecnologia é aberto?
- A tecnologia é ativamente mantida por *maintainers* e/ou pela comunidade?
- O uso da tecnologia é amplamente documentado?
- A instalação e uso da tecnologia é simples?
- A tecnologia permite a execução de modelos pré-treinados?
- A tecnologia permite a execução de modelos pré-treinados com outras tecnologias?

Nesta análise, apenas tecnologias com licenças de código aberto serão consideradas de modo que os pesquisadores tenham livre acesso às recomendações e sejam capazes de manipulá-las em seus projetos, caso seja necessário. Dentre essas, apenas tecnologias disponíveis para uso em programas Java ou Python serão consideradas.

A linguagem Java é amplamente utilizada para a implementação de simuladores de redes ópticas elásticas, como em [22], [24] e [25], por isto, a integração de modelos de ML de forma embutida no simulador permite uma integração de melhor desempenho ao excluir-se a necessidade de comunicação com outros sistemas para executar os modelos.

A linguagem Python é a mais usada e priorizada para desenvolvimento em ML entre trabalhadores da área [37]. Assim, uma integração de simuladores com um sistema independente escrito em Python, responsável por executar os modelos, pode estar sob

Biblioteca	Python GPU	Python CPU	Java GPU	Java CPU
OpenCV	-	X	-	-
PyTorch	-	-	-	-
Scikit Learn	-	-	-	-
TensorFlow	X	X	-	-
TensorFlow Lite	-	X	-	-
ONNX Runtime	X	X	X	X
Deeplearning4j	-	-	X	X
MXNet	-	-	-	-
Caffe2	-	-	-	-
Paddle Lite	-	-	-	-

Tabela 3.1: Bibliotecas de ML e seus respectivos casos de uso selecionados para avaliação quantitativa.

um maior domínio de um pesquisador de ML na área de EONs, apesar da sobrecarga no tempo total de execução de modelos, graças ao tempo tomado apenas para comunicação entre os sistemas.

As tecnologias também devem ser fáceis de serem instaladas, configuradas e manipuladas de acordo com as necessidades de cada pesquisador. Para isto, é fundamental que suas APIs sejam bem documentadas e que a instalação exija o mínimo de modificações de configurações da máquina, externas ao simulador.

Por fim, é ideal que a análise considere a flexibilidade das bibliotecas à respeito dos diversos tipos de modelos de *machine learning*, uma vez que pesquisas sobre o uso de ML em EONs são diversas e podem utilizar diferentes técnicas [31].

O resultado da análise detalhada nas próximas seções pode ser observado de forma resumida na tabela 3.1

3.1 OpenCV

OpenCV (Open Source Computer Vision Library) é uma biblioteca de código aberto voltada para visão computacional e aprendizagem de máquina, construída para fornecer uma infraestrutura comum para aplicações de visão computacional e acelerar o uso de percepção de máquina em produtos comerciais [38].

Apesar do foco principal de OpenCV ser visão computacional, a biblioteca possui um módulo de redes neurais profundas e interfaces para as linguagens Python e Java. Adicionalmente, também é possível realizar a importação de modelos serializados em diversos formatos, como Darknet [39], Torch7 [40], ONNX [41] e TensorFlow [42]. A API

da biblioteca é extensamente documentada, porém com poucos tutoriais sobre o uso do módulo de DNNs, evidenciando o foco em problemas de visão computacional.

No quesito facilidade de instalação e configuração os resultados foram variados: Em Python, para a execução de modelos com apenas o uso da CPU a instalação se resume a instalar o pacote `opencv-python-headless`, a versão sem dependências de bibliotecas de interfaces de usuário gráficas, e está pronto para uso. Porém, se há interesse em utilizar a GPU na execução dos modelos, o processo de instalação se torna bastante complexo, sendo necessário compilar manualmente a biblioteca considerando diversas configurações do ambiente da máquina, de modo que a criação de uma solução generalizada se torne inviável.

Em Java, o processo de instalação é complexo independente do uso ou não de GPU, sendo necessário o mesmo processo de compilação manual do projeto considerando configurações da máquina, não havendo nenhuma integração com gerenciadores de pacotes populares como *Maven* ou *Gradle*.

Assim, pelas dificuldades presentes na instalação, o único uso de OpenCV considerado é o de Python com as execuções sendo realizadas apenas pela CPU da máquina, sem uso de GPU.

3.2 PyTorch

PyTorch é uma biblioteca de ML de código aberto que provê uma plataforma de pesquisas em aprendizagem profunda, oferecendo máxima flexibilidade e velocidade [43], sendo considerada a biblioteca de aprendizagem profunda que mais cresce no mundo [44].

A biblioteca possui suporte para Python e conta com ampla documentação de diversas práticas. Na análise da biblioteca, não foram encontrados métodos nativos de conversão de modelos de outros formatos para serem inferidos com o uso de PyTorch e por este motivo a biblioteca foi descartada.

3.3 Scikit Learn

Scitkit-learn é uma biblioteca de código-aberto desenvolvida em Python que integra diversos algoritmos de ML para problemas supervisionados e não-supervisionados de média escala com ênfase em trazer ML para não-especialistas [45]. É uma biblioteca extremamente popular, principalmente entre iniciantes na área de ML, com desenvolvimento ativo na comunidade.

A biblioteca, com suporte apenas à Python, possui extensa documentação de modo a melhor auxiliar iniciantes. Pelo seu foco em simplicidade, a biblioteca não possui suporte

para uso de GPU em treinos ou execuções de modelos. Para importação e exportação de modelos, a biblioteca conta apenas com serialização nativa de Python por meio da biblioteca *pickle*, não sendo possível importar modelos de outros formatos.

Apesar da facilidade de se desenvolver modelos de ML com *Scikit Learn*, o uso da biblioteca foi descartado pela falta de suporte à importação de modelos pré-treinados salvos em formatos diversos.

3.4 TensorFlow

TensorFlow é uma interface para a expressão de algoritmos de ML e uma implementação para a execução de tais algoritmos [46]. Além da biblioteca principal (TensorFlow Core), a organização possui vários produtos para a execução de algoritmos de ML em larga-escala (TensorFlow Extended), execução de modelos em sistemas embarcados, mobile ou de baixa potência (Tensorflow Lite) e execução de modelos em JavaScript, para ser usado em browsers ou em Node.js (TensorFlow.js).

TensorFlow é uma das bibliotecas mais populares na área de ML, sua biblioteca principal possui ampla documentação além de diversos guias e tutoriais desenvolvidos pela comunidade. Está em ativo desenvolvimento e tem o apoio de diversas empresas de tecnologia reconhecidas.

O uso de TensorFlow Extended está fora do escopo desta pesquisa pelo objetivo de integração com a execução de um simulador e não um sistema de larga-escala. TensorFlow Lite será avaliado de forma mais profunda na Seção 3.6 e TensorFlow.js também está fora do escopo por ser exclusivo para JavaScript. Restando apenas considerar a biblioteca principal, que possui suporte para Python, C, Go e Java.

O suporte para Java está em uma fase de transição, onde a biblioteca antiga está deprecada para uso e a biblioteca nova ainda está em fase experimental. O uso da biblioteca em fase experimental é complexo e não existe quase nenhuma documentação, sendo encontrada até uma recomendação de ler os testes unitários da implementação para aprender como usar certos elementos da API [47]. Por estes motivos, o uso de TensorFlow em Java foi descartado.

O suporte para Python é o principal foco de TensorFlow, de modo que a biblioteca para Python satisfaz todos os requisitos de nossa análise qualitativa: possui ótima documentação; é capaz de importar modelos HDF5 e é possível converter outros modelos para HDF5; possui fácil utilização; entre outros. Logo, será considerada para execução de modelos com e sem GPU, utilizando a biblioteca Keras como a API de alto nível para implementação do programa de simulação.

3.5 Keras

Keras [48] é uma biblioteca de *deep learning* de alto nível, desenvolvida usando a plataforma Tensorflow como base, sendo seu foco principal a capacidade de fácil experimentação.

A biblioteca é uma interface de alto nível para a criação e execução de modelos, instalada por meio da biblioteca do TensorFlow, sendo então utilizada como interface para a implementação do programa de simulação responsável por avaliar o desempenho da biblioteca TensorFlow.

3.6 TensorFlow Lite

TensorFlow Lite [49] é uma *framework* de aprendizagem profunda para execução de modelos em dispositivos. Se trata da versão da popular *framework* TensorFlow que é projetada para execuções em dispositivos com menor poder computacional.

A importação de gráficos é limitada apenas a arquivos do tipo TFLite, porém existem ferramentas para realizar a conversão de formatos comuns em Tensorflow, como Keras e SavedModel. Atualmente, a biblioteca não fornece suporte à execução com uso de GPUs NVIDIA [50].

Tendo os fatores acima em consideração, a instalação e uso da biblioteca são simples e amplamente documentados para programas Python. Para a instalação, é necessário instalar a versão do pacote específica para a versão do interpretador Python instalado na máquina, sendo possível configurar uma detecção automática. A biblioteca não possui versões para uso em programas Java que não sejam voltados para Android.

Apesar da limitação de importação de modelos e a impossibilidade de uso em programas Java, TensorFlow Lite será avaliado de forma quantitativa pelo seu foco específico de execução rápida de modelos em dispositivos de borda.

3.7 ONNX Runtime

A ONNX Runtime [51] se trata de um acelerador de treinamento e execução de modelos de ML salvos em formatos ONNX [41]. *Open Neural Network Exchange* (ONNX) é um formato aberto construído para ser uma representação comum de modelos de ML, possuindo amplo suporte da comunidade de modo que diversas bibliotecas populares de aprendizagem de máquina oferecem a capacidade de exportar modelos em formato ONNX.

A ONNX Runtime é desenvolvida com foco em suporte cross-plataforma, fornecendo uma API comum para diversas linguagens como C, C++, Java e Python. Seu desenvolvi-

mento é apoiado por diversas empresas como Microsoft, Facebook, IBM, Intel, NVIDIA, entre outras.

Graças ao foco de implementações cross-plataforma, a instalação das bibliotecas para Java e Python são simples e exigem pouca ou nenhuma configuração. Em ambos os casos, é necessário apenas instalar o respectivo pacote caso a intenção seja executar os modelos usando apenas a CPU. Se houver interesse em usar uma GPU NVIDIA, é necessário instalar a versão da *runtime* compatível com a versão de CUDA instalada.

A documentação da biblioteca possui poucos guias ou tutoriais mas conta com diversos exemplos de código em Python para operações comuns e um exemplo de código em Java, além da completa referência dos pacotes de ambas versões da biblioteca.

Apesar da documentação limitada, a ONNX Runtime será considerada na avaliação de desempenho pela alta portabilidade da biblioteca, sendo ideal para atender aos diferentes casos de uso de pesquisas de ML em EONs.

3.8 Deeplearning4j

Eclipse Deeplearning4j [52] é uma biblioteca de código-aberto para aprendizagem de máquina distribuída, disponível para Java e Scala. Seu desenvolvimento está ativo e é conduzido pela empresa Konduit.

A biblioteca possui ampla documentação da API e diversos tutoriais que explicam os diferentes usos da biblioteca. É possível importar modelos no formato ONNX, HDF5 e TensorFlow, além de seu próprio formato.

A instalação da biblioteca é extremamente simples para uso de CPU e se dá por meio do gerenciador de pacotes Maven, sendo necessário apenas adicionar algumas linhas na configuração do projeto para que seja possível executar o Deeplearning4j. Para uso de GPU, apenas máquinas com placas de vídeo NVIDIA e com CUDA configurado são suportadas. Neste caso, o processo de instalação é quase o mesmo, tendo como diferença o identificador da biblioteca que depende da versão da GPU instalada na máquina.

A biblioteca Deeplearning4j será avaliada em um programa Java com uso de ambas CPU e GPU.

3.9 MXNet

A biblioteca MXNet [53] é uma *framework* de código-aberto para aprendizagem profunda que permite a definição, o treinamento e a execução de redes neurais profundas em diversas plataformas, estando atualmente em desenvolvimento, em processo de incubação pela Apache Incubator. Se trata de uma biblioteca bastante versátil, com suporte à diversas

linguagens, incluindo Java e Python, e importação de modelos em formato ONNX para execução.

Entretanto, o suporte para Java é limitado primeiramente no quesito disponibilidade de bibliotecas. Alguns arquivos binários presentes no pacote possuem licenças incompatíveis com a licença Apache 2, resultado na retirada dos pacotes de Java do repositório Maven. Assim, para utilizar uma versão atualizada da biblioteca é necessário compilar o código-fonte. Além disso, a documentação para Java também é quase vazia, tendo apenas dois tutoriais simples e a publicação da referência da API. Pela dificuldade de instalação e a falta de tutoriais e suporte da comunidade, o uso de MXNet para Java foi descartado.

A versão em Python da biblioteca MXNet é bastante completa, contando com diversos tutoriais para diferentes casos de uso, além de prover uma fácil instalação por meio do gerenciador de pacotes *pip*. Entretanto, em testes iniciais para avaliar a viabilidade de uso da biblioteca, foi descoberto que a biblioteca dispara erros ao tentar executar o modelo usado pelo autor para a comparação de desempenho entre as bibliotecas. Após não haver sucesso em resolvê-los, o uso de MXNet para Python também foi descartado.

3.10 Outras tecnologias

Além das bibliotecas previamente mencionadas, outras bibliotecas foram brevemente avaliadas mas descartadas por motivos desclassificatórios, como Paddle Lite, cuja documentação está disponível apenas em chinês, a biblioteca Caffe2 disponível para Python que foi descontinuada em favor de PyTorch, entre outras não disponíveis para Python ou Java ou sem atual expressão na literatura.

Capítulo 4

Avaliação de Desempenho

Este capítulo tem como objetivo apresentar a metodologia da avaliação de desempenho das diferentes bibliotecas de ML selecionadas.

4.1 Metodologia

A avaliação de desempenho seguirá a abordagem sistemática para a realização de avaliações de desempenho, proposta por Raj Jain e descrita na Seção 2.3.

Em primeiro lugar, o objetivo desta análise consiste em comparar diversas bibliotecas de aprendizagem de máquina, utilizadas em programas Java e Python, de modo a escolher a melhor alternativa no quesito velocidade na execução de modelos de ML.

O sistema a ser avaliado consiste de um módulo de programação, programado em Python ou Java de acordo com a análise realizada no Capítulo 3, que fornece um serviço de execução de modelos pré-treinados e carregados em memória, onde ao receber dados de entrada de um tipo especificado pelo modelo, executa o modelo com a entrada recebida e retorna o resultado da execução do modelo.

Nesta análise, erros e falhas durante a execução do serviço não serão consideradas. Em cada execução do serviço, serão registrados o tempo total de execução e o resultado da mesma.

A principal métrica de desempenho será o tempo de execução de um modelo. Este tempo de execução é medido ao consultar o tempo de sistema pré e pós execução, sendo o tempo de execução o módulo da diferença dos resultados. Este método de medição do tempo de execução foi escolhido devido a complexidade de medir tempos de operações na GPU, onde é necessário o uso de instrumentações específicas de cada biblioteca, nem sempre disponíveis como no caso de ONNX Runtime para Java. Além disso, os resultados das execuções realizadas por cada alternativa serão comparados entre si para verificar se

há perda ou ganho de precisão do modelo, de acordo com o formato em que se encontra serializado e a biblioteca de ML utilizada para a execução.

Definidos os objetivos, sistema, serviços, resultados e métricas, os parâmetros são listados a seguir:

- Velocidade da CPU da máquina;
- Velocidade e funcionalidades da GPU da máquina, em execuções de modelo que utilizam a GPU na execução;
- Formato de serialização em que o modelo está salvo;
- Biblioteca utilizada para a execução do modelo;
- Uso ou não da GPU para a execução do modelo;
- Número, tamanho e valor dos dados de entrada de uma execução de modelo;
- O modelo executado;
- Número e tamanho dos dados de saída de uma execução de modelo;
- Ambiente de execução, ou *runtime environment* em que o programa é executado.

Dentre estes parâmetros, foram selecionados três fatores:

- Ambiente de execução. Serão executados programas escritos em Python, executados com a *runtime* Python 3.8.5, e em Java, executados com a *runtime* OpenJDK 11.0.7+10.
- Biblioteca utilizada para a execução do modelo. Como avaliado na Seção 3, serão consideradas as bibliotecas ONNX Runtime, Tensorflow Lite, TensorFlow, OpenCV e Deeplearning4j.
- Uso de GPU. Haverão experimentos com uso de GPU na execução dos modelos e sem o uso de GPU.
- Formato de serialização em que o modelo está salvo. Onde será utilizado preferencialmente o arquivo original do modelo, em formato HDF5, e outros arquivos convertidos quando necessário, em formatos ONNX e TensorFlow Lite.

A experimentação será a técnica de avaliação de desempenho escolhida. Apesar de o objetivo de estudo deste trabalho se tratar do uso de ML em simuladores, as bibliotecas terão seus respectivos desempenhos medidos em programas isolados. Assim, as diferentes alternativas de sistemas a serem comparadas serão implementadas e instrumentadas na

execução da análise de desempenho, sendo cada uma das implementações um programa desenvolvido em Python ou Java.

Para a experimentação, será utilizado um modelo representativo de pesquisas existentes acerca do uso de ML em EONs, desenvolvido pelo Laboratório de Redes de Computadores da Universidade de Brasília (COMNET-UnB). Neste trabalho, o modelo utilizado se trata de um classificador de estratégias de RSA em EONs, baseado em *deep learning*. Este modelo recebe como entrada o estado de uma EON e tem como saída a classificação da estratégia RSA em utilização, de acordo com o estado. O modelo possui como saída um inteiro no intervalo $[0, 2]$, uma classificação da estratégia de alocação identificada a partir do estado como ruim (0), média (1) ou boa (2).

A carga de trabalho é formada por 97301 requisições onde os dados de cada entrada correspondem a estados de rede coletados em execuções do ONS, sendo cada estado representado por uma matriz de 86 linhas e 320 colunas, a representação da topologia USANet com 24 nós e 86 enlaces em que cada enlace contém 320 *slots*. Esta entrada é representada por um vetor de 27520 valores de ponto flutuante de precisão simples, possuindo um tamanho total de aproximadamente 107,5 kB. Esta carga de trabalho é o mesmo conjunto de dados utilizado para treinamento e validação do modelo usado nesta análise e similar às utilizadas em outras pesquisas do Laboratório de Redes de Computadores da Universidade de Brasília (COMNET-UnB), como em [29]. As primeiras 3000 requisições possuem o propósito de *aquecer* os ambientes de execução, isto é, garantir que todo o código a ser executado já está compilado pelo ambiente de execução, que os recursos apropriados já foram alocados, que o lixo de memória inicial da aplicação já foi coletado, entre outros. Portanto, os resultados das primeiras 3000 requisições serão descartados na análise final.

Os experimentos a serem realizados consistem em um desenho fatorial fracionário, onde diversas combinações foram descartadas por incompatibilidade de fatores, como o uso de Deeplearning4j em Python, ou motivos diversos detalhados no Capítulo 3. Assim, serão realizadas as seguintes combinações de experimentos, como resumido na tabela 3.1:

- Java, Deeplearning4j, sem GPU, modelo original no formato HDF5 (TensorFlow);
- Java, Deeplearning4j, com GPU, modelo original no formato HDF5 (TensorFlow);
- Java, ONNX Runtime, sem GPU, modelo convertido para o formato ONNX;
- Java, ONNX Runtime, com GPU, modelo convertido para o formato ONNX;
- Python, OpenCV, sem GPU, modelo convertido para o formato ONNX;
- Python, Tensorflow Lite, sem GPU, modelo convertido para o formato TFLite;
- Python, Tensorflow, sem GPU, modelo original no formato HDF5 (TensorFlow);

- Python, Tensorflow, com GPU, modelo original no formato HDF5 (TensorFlow);
- Python, ONNX Runtime, sem GPU, modelo convertido para o formato ONNX;
- Python, ONNX Runtime, com GPU, modelo convertido para o formato ONNX.

4.2 Ambiente de Execução

Para a realização das medições, 10 programas foram desenvolvidos considerando todas as combinações descritas anteriormente. Estes programas realizam o mesmo conjunto de tarefas: 1. inicializar o sistema ao carregar o modelo; 2. inicializar procedimentos necessários para a execução da carga de trabalho; 3. carregar a carga de trabalho usada como entrada do modelo; 4. realizar as chamadas de serviço com as entradas carregadas para executar o modelo, registrando o tempo real tomado pela execução e o resultado. Cada programa trata de medir apenas o intervalo de tempo em que a execução do modelo ocorre. A execução de um dos programas foi orquestrada por *scripts* auxiliares feitos em Python, responsáveis por instalar dependências, e executar os programas de simulação.

As simulações foram realizadas em uma máquina com processador Intel Core i3-8100, placa de vídeo GeForce RTX 2060 e memória RAM de 32 GB (2x16GB 3000Mhz DDR4). Trechos e arquivos do código-fonte dos programas de simulação são mostrados no Apêndice A.

4.3 Resultados

Devido a outros processos de sistema sendo executados durante a medição, o desempenho das bibliotecas foi ocasionalmente afetado por fatores externos, resultando em picos de tempo de execução ao longo das medições. Para mitigar este fenômeno, cada experimento foi executado com a carga de trabalho 5 vezes. Após as execuções, a métrica final de tempo de execução para cada uma das 94301 entradas foi definida como a mediana dos 5 tempos de execução coletados. A mediana foi escolhida como índice pelo fato de os dados serem quantitativos, o total das execuções não ser de interesse e que a média seria altamente enviesada pela presença destes picos.

Para cada um dos experimentos, foram calculados os dados de média das amostras das execuções, o desvio padrão, o mínimo e o máximo, exibidos na Tabela 4.1. Devido ao alto número de amostras (94301), é possível estimar a média populacional com um alto intervalo de confiança. A Tabela 4.2 exibe a estimativa da média populacional com um intervalo de confiança de 99.98%.

ling.	biblioteca	GPU	média (µs)	desvio padrão (µs)	mínimo (µs)	máximo (µs)
Python	OpenCV	sem	1773.14	12.42	1729	1972
	TensorFlow	sem	2988.96	19.54	2940	3332
		com	2885.55	18.72	2827	3219
	TensorFlow Lite	sem	2083.75	12.21	2043	2218
	ONNX Runtime	sem	2106.01	12.31	2060	2276
		com	1767.84	10.62	1729	1902
Java	ONNX Runtime	sem	2366.02	153.21	2322	23961
		com	2067.64	12.67	2041	2383
	Deeplearning4j	sem	763.44	69.23	722	11749
		com	1026.51	33.67	990	1629

Tabela 4.1: Estatísticas descritivas acerca da amostra do tempo de execução do modelo para cada um dos experimentos.

ling.	biblioteca	GPU	intervalo de confiança (µs)
Python	OpenCV	sem	(1772.99, 1773.29)
	TensorFlow	sem	(2988.72, 2989.20)
		com	(2885.32, 2885.78)
	TensorFlow Lite	sem	(2083.60, 2083.90)
	ONNX Runtime	sem	(2105.86, 2106.16)
		com	(1767.71, 1767.97)
Java	ONNX Runtime	sem	(2364.16, 2367.88)
		com	(2067.49, 2067.79)
	Deeplearning4j	sem	**(762.60, 764.28)**
		com	(1026.10, 1026.92)

Tabela 4.2: Estimativa da média populacional, o tempo de execução do modelo, com 99.98% de confiança.

De acordo com os dados presentes na Tabela 4.2, é possível afirmar que o desempenho da biblioteca Deeplearning4j sem GPU apresenta o melhor desempenho com o modelo utilizado, seguido por sua alternativa que utiliza a GPU na execução dos modelos, em média aproximadamente 34% mais lenta.

Como elencado na seção 2.1, o desenvolvimento de uma pesquisa pode exigir o processamento de um grande número de eventos, podendo ser na ordem de 10^9 eventos em casos extremos porém comumente 10^8. Assim, considerando que o modelo seja executado em cada processamento de evento, 1 milissegundo a menos no tempo de sua execução representa uma economia de 10^5 segundos, ou em torno de 28 horas, de tempo total de simulações durante o desenvolvimento de uma pesquisa. Estes números demonstram a importância de escolher bibliotecas com foco em desempenho.

Após Deeplearning4j, as duas combinações com melhor desempenho correspondem à ONNX Runtime executada em Python com o uso de GPU, seguida por OpenCV executada em Python sem o uso de GPU. Nesta comparação, apesar da biblioteca ONNX Runtime apresentar um melhor desempenho em média, a diferença entre o intervalo de confiança das duas alternativas é de apenas 5 microssegundos, uma quantidade de tempo negligenciável ao considerar o tempo total de execução e o contexto de simuladores de EONs em que o modelo está inserido.

O uso de ONNX Runtime em Java com GPU é o quarto com maior desempenho. Aproximadamente 200 microssegundos mais lento que sua alternativa executada em Python. Esta combinação é seguida por TensorFlow Lite e por ONNX Runtime em Python sem GPU. Aqui, o fenômeno de desempenhos claramente melhores mas negligenciáveis se repete, onde ONNX Runtime em Java com GPU é apenas 40 microssegundos em média mais rápido do que sua alternativa em Python sem GPU.

A execução de ONNX Runtime em Java sem GPU é aproximadamente 260 microssegundos mais lenta que sua alternativa em Python, similar à diferença observada na comparação com uso de GPU. Nas execuções de programa Java, pode-se observar uma variância consideravelmente maior quando se usa a CPU para execução de modelos, resultando em tempos quase tão altos quanto 24 milissegundos para uma única execução do modelo. Este resultado indica que programas Java tendem a serem mais impactados por interrupções como coleta de lixo de memória.

A biblioteca TensorFlow apresenta o pior desempenho, sendo quase quatro vezes mais lenta sem o uso de GPU, ao ser comparada com Deeplearning4j. Este resultado demonstra o fato da biblioteca *core* de TensorFlow não ser otimizada para execução de modelos, sendo esta tarefa delegada para outros produtos da organização TensorFlow, como TensorFlow Lite que obteve um desempenho significativamente melhor.

Além disso, é possível observar que a ONNX Runtime possui melhor desempenho geral

em Python ao ser comparada com Java. Uma hipótese plausível diz respeito à maior popularidade da biblioteca para Python, o que resulta em maiores esforços de engenharia e consequentemente melhores desempenhos. A priorização de suporte à Python não é um caso isolado, como observado no Capítulo 3, onde bibliotecas como TensorFlow e OpenCV, cujas bibliotecas desenvolvidas para uso em Java possuem grandes limitações ao serem comparadas com suas alternativas para Python.

Não é possível realizar análises conclusivas acerca dos benefícios de uso ou não da GPU para execução de modelos.

Os resultados mostram um comportamento inconsistente acerca da comparação de desempenho entre o uso de CPU ou GPU para execução de modelos. A hipótese inicial se tratava de um possível gargalo durante a transferência de dados para a GPU. Ao utilizar a GPU para a execução de modelos, o valor de entrada deve ser transferido da CPU para a GPU e o tempo tomado por esta transferência de dados pode ser significativo na medição do tempo total de execução. Neste trabalho, cada execução do modelo deve transferir 107,5 KB para a GPU. Foram usados testes realizados por Harris em [54], adaptados para usar o parâmetro de 107,5 KB de tamanho a ser copiado, de modo a observar o tempo tomado pela transferência de dados.

A máquina em que os testes foram realizados demonstrou ser capaz de transferir consistentemente entre 4 GB a 5 GB de dados por segundo, divididos em payloads de 107,5 KB. Assim, a transferência de dados para a GPU é responsável por no máximo 30 microsegundos do tempo de execução e por este motivo, no momento não é possível realizar análises conclusivas acerca dos benefícios de uso ou não da GPU para execução deste modelo. Novos estudos são necessários para explicar a diferença de comportamento da biblioteca Deeplearning4j ao ser comparada com outras.

Por fim, os resultados das execuções do modelo em cada uma das entradas foi consistente entre todas as diferentes combinações, tornando possível a conclusão de que, para este modelo, a conversão entre diferentes formatos de serialização não gerou perdas de precisão aparentes.

Capítulo 5

Contribuições ao ONS

A área de aprendizagem de máquina é extensa e diversa, modelos são em sua maioria desenvolvidos de forma personalizada para um problema único que buscam resolver. Esta diversidade é refletida na atual literatura sobre o uso de ML em EONs. Pesquisas como [31] e [36] mostram as diversas aplicações em diferentes problemas de EONs, apesar da pesquisa na área estar em sua infância.

Ao propor soluções que busquem auxiliar pesquisadores no uso de ML em simuladores, é necessário considerar os diversos casos de uso e suas características, fundamentais para definir a importância de fatores como desempenho do tempo de execução, facilidade de execução, disponibilidade da GPU, entre outros. Para isto, este trabalho focará suas contribuições ao ONS em duas categorias, realizar a integração de ML com simuladores com focos separados em ambas flexibilidade, ou facilidade de uso do usuário, e melhores desempenhos em execuções de modelos.

5.1 Flexibilidade

No campo de aprendizagem de máquina, é comum que haja um constante refinamento de modelos em desenvolvimento, havendo várias iterações de testes e modificações de parâmetros de modo a maximizar o desempenho do modelo, avaliado em métricas apropriadas. Nestes casos, uma integração em que o pesquisador precisa constantemente exportar o modelo para carregá-lo em um simulador acaba se tornando trabalhosa. Além de exportar o modelo, ainda seria necessário modificar o simulador, possivelmente desenvolvido em uma linguagem não dominada pelo pesquisador, para que colete métricas relevantes ao processo de desenvolvimento do modelo.

Assim, a implementação de interfaces genéricas de comunicação em simuladores é uma boa alternativa para pesquisadores que desejam manter o controle sobre o ambiente de

desenvolvimento e execução de seus modelos de aprendizagem de máquina e ainda assim integrá-los facilmente com simuladores.

Nesta solução, o simulador é programado para executar um modelo por meio de uma interface de comunicação pré-definida, como por exemplo chamadas em HTTP para um servidor configurado, e o pesquisador é responsável por implementar a interface no sistema que executa o modelo, neste exemplo um servidor HTTP. Assim, o pesquisador mantém o controle sobre o ambiente deste modelo, sendo capaz de coletar quaisquer métricas relevantes de execução como os valores de entrada e saída, tempo de execução, entre outros, e sem precisar constantemente exportar o modelo e carregá-lo no simulador a cada rodada de testes.

Sua principal desvantagem consiste em maiores tempos de execução do modelo, do ponto de vista do simulador. Isto acontece pela sobrecarga de operações para transferir os dados de entrada e saída do modelo entre processos além da latência presente na comunicação entre sistemas. Esta sobrecarga adicional de tempo de execução pode ser significativa ou não, dependendo de vários fatores acerca do estudo sendo realizado e do modelo sendo desenvolvido.

5.1.1 Implementação de modelos no ONS com foco em flexibilidade

No ONS, este objetivo foi realizado por meio da implementação da interface `Model` e da classe `HttpRemoteModel`.

A interface `Model`, exibida no Apêndice B.1, é uma definição genérica adaptável para quaisquer tipos de entradas e saídas de um modelo, definindo a existência de um método específico para execução do modelo.

A classe `HttpRemoteModel` implementa a interface `Model`. Esta classe ao ser instanciada com uma URL, implementa o método de execução do modelo de modo que a execução do modelo é despachada para um sistema externo por meio de uma requisição HTTP POST para a URL definida. Esta requisição envia em seu corpo a entrada do modelo, serializada em JSON, e deve receber como resposta um valor do tipo de saída, também serializado em JSON.

Como prova de conceito, um simples servidor HTTP foi desenvolvido para servir a execução de modelos em Python com a biblioteca ONNX Runtime, chamado por uma instância da classe `HttpRemoteModel` apropriadamente configurada com a URL local. Ao receber uma requisição, o servidor HTTP envia os dados de entrada para o modelo utilizado para medições no Capítulo 4. O resultado da execução é enviado como resposta de volta ao ONS, que pode então tratar a classificação de acordo com seus objetivos, para

o teste os resultados foram apenas imprimidos na tela. As implementações podem ser vistas nos Apêndices B.4 e B.5.

Com este tipo de integração, estas são as etapas necessárias para a execução de modelos em uma simulação do ONS:

- Implementar um simples servidor HTTP que ouve requisições do tipo POST;
- Integrar o servidor HTTP de modo que requisições do tipo POST sigam o seguinte comportamento: o corpo da requisição possui a entrada do modelo, o modelo será executado com a entrada, e por fim a resposta da requisição possuirá a saída do modelo.
- Definir no ONS os tipos de entrada e saída do modelo, do ponto de vista do simulador (e.g. implementação de classes com propriedades específicas);
- Instanciar um objeto da classe `HttpRemoteModel` e especificar na instanciação os tipos de entrada, saída e a URL em que o modelo está sendo servido por um servidor HTTP externo;
- Executar o modelo e utilizar sua saída onde for apropriado.

Assim, pesquisadores podem concentrar seus esforços de desenvolvimento em ambientes de desenvolvimento de aprendizagem de máquina, exigindo esforço mínimo de importação e exportação de modelos para o ONS.

5.2 Desempenho

A solução com foco em desempenho busca diminuir o tempo total de execução de modelos, o que torna importante que os modelos possam ser executados de forma nativa pelo simulador uma vez que isto elimina a sobrecarga de tempo por comunicação externa.

Para simuladores desenvolvidos em Java, vimos no Capítulo 4 que para o modelo selecionado, executá-lo com a biblioteca Deeplearning4j sem o uso de GPU exibiu o melhor desempenho. A biblioteca ONNX Runtime, apesar de mais lenta, também é atrativa por permitir a importação de modelos no formato ONNX, um formato popular e capaz de ser o alvo de conversão de diversos outros formatos existentes na literatura.

Em Python, a biblioteca ONNX Runtime com o uso de GPU exibiu um desempenho tão bom quanto OpenCV sem o uso de GPU. Ambas combinações possuem suas vantagens e desvantagens em relação a facilidade e flexibilidade no uso: OpenCV é voltada para uso na área de visão computacional, o que torna o uso dela em outros contextos sub-óptimo se ela não for a melhor escolha no quesito desempenho; ONNX Runtime precisa do uso

da GPU para ter um desempenho tão bom quanto OpenCV com este modelo, porém tem suporte para fácil configuração de uso da GPU e uma API mais amigável.

5.2.1 Implementação de modelos no ONS com foco em desempenho

Foram feitas contribuições ao ONS para permitir o carregamento e execução nativa de modelos de aprendizagem de máquina durante a execução de uma simulação no ONS. Assim, pesquisadores podem realizar implementações personalizadas de seus modelos utilizando tanto a biblioteca ONNX Runtime como a biblioteca Deeplearning4j para carregamento e execução dos modelos.

A estrutura utilizada para implementação da funcionalidade nativa tem como base a classe abstrata `NativeModel`, que implementa os métodos da interface `Model` com as seguintes adições: quatro métodos abstratos diferentes responsáveis por carregar o modelo a partir de diferentes formatos de entrada, como arquivos ou *streams*; implementação do método `predict` definido na interface `Model`, onde primeiro é conferido se o modelo está carregado em memória e se sim, é realizada a chamada para um novo método protegido nomeado `_predict`, que deve ser implementado pelas classes específicas de modelos para de fato executar o modelo.

Para adicionar o suporte ao uso de ambos Deeplearning4j e ONNX Runtime, foram implementadas duas classes, `Dl4jNativeModel` e `OnnxNativeModel`. Estas classes são responsáveis apenas por implementar os métodos de carregamento dos modelos, levando em conta as especificações de cada biblioteca e os tipos de serialização suportados.

Como prova de conceito, o modelo classificador utilizado no Capítulo 4 foi implementado em Java para o ONS utilizando ambas bibliotecas previamente mencionadas, sob as classes `Dl4jClassifier` e `OnnxClassifier`. Estas classes extendem `Dl4jNativeModel` e `OnnxNativeModel`, respectivamente, e precisam apenas implementar o método de execução responsável por receber a entrada, executar o modelo e retornar a saída.

Assim como no teste da classe `HttpRemoteModel`, estes modelos foram integrados a uma simulação do ONS e executados em cada requisição de rede, tendo como entrada o estado atual da rede. Os resultados foram então impressos para a validação manual dos resultados e da integração bem-sucedidos. As implementações das classes relacionadas a execuções nativas de modelos podem ser encontradas no Apêndice B.3.

Com este tipo de integração, estas são as etapas necessárias para a execução de modelos em uma simulação do ONS:

- Definir no ONS os tipos de entrada e saída do modelo, do ponto de vista do simulador (e.g. implementação de classes com propriedades específicas);

- Implementar uma classe específica para o modelo:

  Decidir se utilizará Deeplearning4j ou ONNX Runtime;

  Criar uma classe nova para o modelo que extenderá ou `OnnxNativeModel` ou `Dl4jNativeModel`;

  Implementar o método `_predict` da classe recém-criada, responsável por receber dados de entrada, executar o modelo utilizando a biblioteca escolhida e retornar a saída do modelo.

- Instanciar um objeto da classe recém-criada;
- Carregar o modelo em memória utilizando o método `load` do objeto recém-instanciado, sendo necessário definir o caminho do modelo;
- Executar o modelo e utilizar sua saída onde for apropriado.

Com esta abordagem, pesquisadores podem tomar proveito de modelos pré-treinados e usá-los em seus algoritmos em desenvolvimento, possuindo diversas vantagens como bom desempenho e nenhuma necessidade de dependências externas.

Com estas contribuições, espera-se que atuais usuários do ONS possam utilizar modelos de ML pré-treinados em suas simulações, e que pesquisadores de ML na área de EONs sintam-se incentivados a utilizar o ONS para auxiliar-los em suas pesquisas.

Capítulo 6

Conclusão

Com a crescente demanda de tráfego global, *Elastic Optical Networking* (EON) surgiu como uma nova solução para o gerenciamento eficiente de recursos em redes ópticas. Uma das principais ferramentas de pesquisa em EONs são os simuladores, ideais pelo baixo custo e baixa complexidade de implementação.

Pelo crescente desenvolvimento de ambas áreas de *Elastic Optical Networks* (EONs) e *Machine Learning* (ML), o estudo das aplicações de algoritmos de aprendizagem de máquina em redes ópticas elásticas tem se popularizado. Entretanto, pelo fato de esta área de pesquisa ainda estar em seus estágios iniciais, há lacunas a serem preenchidas na academia, como por exemplo a falta de suporte para execução de modelos de ML em simuladores de EONs.

Este trabalho teve como objetivo analisar a literatura atual de bibliotecas de ML para propor soluções a respeito da implementação de suporte dos simuladores à execução de ML.

Para isso, fez-se uma análise qualitativa de diversas bibliotecas populares, filtrando-as por critérios como facilidade de instalação e configuração, documentação, e disponibilidade. Em seguida, o desempenho das bibliotecas selecionadas foi avaliado tendo como critério o tempo de execução de um modelo pré-selecionado.

Na análise de desempenho, a biblioteca Deeplearning4j apresentou o melhor desempenho ao ser comparada com outras alternativas. Em ambientes de execução Python, ONNX Runtime (com GPU) e OpenCV (sem GPU) apresentaram os melhores desempenhos.

Este trabalho por fim apresentou recomendações e direções acerca de implementações de suporte de aprendizagem de máquina para simuladores de redes ópticas elásticas, onde interfaces genéricas de comunicação foram recomendadas para casos de uso que exigem alta flexibilidade na integração de modelos, como por exemplo modelos ainda sendo desenvolvidos, e bibliotecas nativas de alto desempenho foram recomendadas, como Deeplearning4j para simuladores em Java, para casos de uso que exigem alto desempenho.

Por fim, foram apresentadas contribuições de implementações para adicionar ao *Optical Network Simulator* (ONS) o suporte para uso de modelos de aprendizagem de máquina em suas simulações. Estas contribuições cobrem ambos casos em que ou a flexibilidade ou o desempenho são priorizados, sendo fornecido o código-fonte por meio de um repositório hospedado na plataforma GitHub.

Referências

[1] Jain, R.: *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1991, ISBN 978-0-471-50336-1. ix, 9, 10

[2] Le Rouzic, E.: *Network evolution and the impact in core networks*. Em *36th European Conference and Exhibition on Optical Communication*, páginas 1–8, 2010. 1

[3] Cisco, T: *Cisco visual networking index: Forecast and methodology, 2010–2015*. Growth Lakeland, 2011. 1

[4] Cisco, T: *Cisco visual networking index: Forecast and methodology, 2011–2016*. Growth Lakeland, 2012. 1

[5] Cisco, T: *Cisco visual networking index: Forecast and methodology, 2012–2017*. Growth Lakeland, 2013. 1

[6] Cisco, T: *Cisco visual networking index: Forecast and methodology, 2013–2018*. Growth Lakeland, 2014. 1

[7] Cisco, T: *Cisco visual networking index: Forecast and methodology, 2014–2019*. Growth Lakeland, 2015. 1

[8] Cisco, T: *Cisco visual networking index: Forecast and methodology, 2015–2020*. Growth Lakeland, 2016. 1

[9] Cisco, T: *Cisco visual networking index: Forecast and methodology, 2016–2021*. Growth Lakeland, 2017. 1

[10] Cisco, T: *Cisco visual networking index: Forecast and methodology, 2017–2022*. Growth Lakeland, 2018. 1

[11] Ericsson: *Q2 2020 update*. Ericsson Mobility Report, Sep 2020. `https://www.ericsson.com/4a4e5d/assets/local/mobility-report/documents/2020/emr-q2-update-03092020.pdf`. 1

[12] McKeay, Martin: *The building wave of internet traffic*. Akamai Security Intelligence & Threat Research Blog, Apr 2020. `https://blogs.akamai.com/sitr/2020/04/the-building-wave-of-internet-traffic.html`. 1

[13] Tomkos, I., S. Azodolmolky, J. Solé-Pareta, D. Careglio e E. Palkopoulou: *A tutorial on the flexible optical networking paradigm: State of the art, trends, and research challenges*. Proceedings of the IEEE, 102(9):1317–1337, 2014. 1, 2

[14] Jinno, M.: *Elastic optical networking: Roles and benefits in beyond 100-gb/s era.* Journal of Lightwave Technology, 35(5):1116–1124, 2017. 1, 2

[15] Jorge López Vizcaíno, Yabin Ye e I. T. Monroy: *Energy efficiency in elastic-bandwidth optical networks.* Em *2011 International Conference on the Network of the Future*, páginas 107–111, 2011. 1, 2

[16] Tomkos, I., E. Palkopoulou e M. Angelou: *A survey of recent developments on flexible/elastic optical networking.* Em *2012 14th International Conference on Transparent Optical Networks (ICTON)*, páginas 1–6, 2012. 2

[17] Yu, J., B. Cheng, C. Hang, Y. Hu, S. Liu, Y. Wang e J. Shen: *A deep learning based rsa strategy for elastic optical networks.* Em *2019 18th International Conference on Optical Communications and Networks (ICOCN)*, páginas 1–3, 2019. 2, 3

[18] Christodoulopoulos, K., I. Tomkos e E. A. Varvarigos: *Elastic bandwidth allocation in flexible ofdm-based optical networks.* Journal of Lightwave Technology, 29(9):1354–1366, 2011. 2

[19] Klinkowski, M. e K. Walkowiak: *Routing and spectrum assignment in spectrum sliced elastic optical path network.* IEEE Communications Letters, 15(8):884–886, 2011. 2

[20] Costa, Lucas R. e André C. Drummond: *Novo algoritmo rmlsa com roteamento multihop em redes Ópticas elásticas.* Em *Anais do XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBC, 2016. `http://www.sbrc2016.ufba.br/downloads/SessoesTecnicas/152171.pdf`. 2

[21] Costa, Lucas R. e André C. Drummond: *Novo esquema para provisão de modulação adaptativa em redes Ópticas elásticas.* Em *Anais do XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, Porto Alegre, RS, Brasil, 2017. SBC. `https://sol.sbc.org.br/index.php/sbrc/article/view/2662`. 2

[22] Costa, Lucas R, Léia S de Sousa, Felipe R de Oliveira, Kaio A da Silva, Paulo JS Júnior e André C Drummond: *Ons: Simulador de eventos discretos para redes ópticas wdm/eon.* Salão de Ferramentas do Simpósio Brasileiro de Redes de Computadores-SBRC, 2016. 3, 6, 12

[23] Tessinari, Rodrigo Stange, Bart Puype, Didier Colle e Anilton Salles Garcia: *Elastico++: An elastic optical network simulation framework for omnet++.* Optical Switching and Networking, 22:95 – 104, 2016, ISSN 1573-4277. `http://www.sciencedirect.com/science/article/pii/S1573427716300571`. 3, 6

[24] Aibin, M. e M. Blazejewski: *Complex elastic optical network simulator (ceons).* Em *2015 17th International Conference on Transparent Optical Networks (ICTON)*, páginas 1–4, 2015. 3, 6, 12

[25] Pavon-Marino, P. e J. Izquierdo-Zaragoza: *Net2plan: an open source network planning tool for bridging the gap between academia and industry.* IEEE Network, 29(5):90–96, 2015. 3, 6, 12

[26] Mitchell, Thomas M.: *Machine Learning*. McGraw-Hill, Inc., USA, 1ª edição, 1997, ISBN 0070428077. 3, 7

[27] Alpaydin, E.: *Introduction to Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2020, ISBN 9780262043793. `https://books.google.com.br/books?id=tZnSDwAAQBAJ`. 3, 7, 8

[28] Simeone, O.: *A very brief introduction to machine learning with applications to communication systems*. IEEE Transactions on Cognitive Communications and Networking, 4(4):648–664, 2018. 3, 8

[29] Silva, Guilherme, Lucas R. Costa e André C. Drummond: *Classificador baseado em aprendizado profundo para identificação de estratégias de alocação de espectro em redes Ópticas elásticas*. Em *Anais do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBC, 2020. `https://sol.sbc.org.br/index.php/sbrc/article/download/12293/12158`. 3, 21

[30] Choudhury, P. D. e T. De: *Recent developments in elastic optical networks using machine learning*. Em *2019 21st International Conference on Transparent Optical Networks (ICTON)*, páginas 1–3, 2019. 3

[31] Zhang, Yongjun, Jingjie Xin, Xin Li e Shanguo Huang: *Overview on routing and resource allocation based machine learning in optical networks*. Optical Fiber Technology, 60:102355, Dec 2020. 3, 13, 26

[32] Palmieri, F., U. Fiore e S. Ricciardi: *Simulnet: a wavelength-routed optical network simulation framework*. Em *2009 IEEE Symposium on Computers and Communications*, páginas 281–286, 2009. 6

[33] Chaves, Daniel, Helder Pereira, Carmelo Bastos-Filho e Joaquim Martins Filho: *Simton: A simulator for transparent optical networks*. Journal of Communication and Information Systems, 25:1–10, abril 2010. 6

[34] Sousa, Léia de e André Drummond: *Políticas de escalonamento para transferências de dados em massa inter centro de dados utilizando roteamento e alocação de espectro*. Em *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, páginas 459–472, Porto Alegre, RS, Brasil, 2019. SBC. `https://sol.sbc.org.br/index.php/sbrc/article/view/7379`. 7

[35] Costa, Lucas, Ítalo Brasileiro e André Drummond: *Novo rmlsa com tonificação de circuito e ciente da qualidade de transmissão com baixa margem em redes Ópticas elásticas*. Em *Anais do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, páginas 197–210, Porto Alegre, RS, Brasil, 2020. SBC. `https://sol.sbc.org.br/index.php/sbrc/article/view/12283`. 7

[36] Musumeci, F., C. Rottondi, A. Nag, I. Macaluso, D. Zibar, M. Ruffini e M. Tornatore: *An overview on application of machine learning techniques in optical networks*. IEEE Communications Surveys Tutorials, 21(2):1383–1408, 2019. 8, 26

[37] Wilcox, Mark, Stijn Schuermans, Christina Voskoglou e Alexandre Sabolevski: *State of the developer nation 12th edition - q1 2017*, Mar 2017. https://www.developereconomics.com/resources/reports/state-of-the-developer-nation-q1-2017. 12

[38] *Opencv - about.* https://web.archive.org/web/20201104030111/https://opencv.org/about/, Accessed: 2020-11-08. 13

[39] *Darknet: Open source neural networks in c.* https://web.archive.org/web/20201101011701/https://pjreddie.com/darknet/, Accessed: 2020-11-08. 13

[40] *Torch | scientific computing for luajit.* https://web.archive.org/web/20201101135030/https://torch.ch/, Accessed: 2020-11-08. 13

[41] *Onnx.* https://web.archive.org/web/20201105074026/https://onnx.ai/, Accessed: 2020-11-08. 13, 16

[42] *Tensorflow.* https://web.archive.org/web/20201108012745/https://www.tensorflow.org/, Accessed: 2020-11-08. 13

[43] PyTorch: *What is pytorch?* https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html#what-is-pytorch. 14

[44] Gugger, Sylvain e Jeremy Howard: *Practical deep learning for coders.* https://web.archive.org/web/20201103183225/https://course.fast.ai/, Accessed: 2020-11-08. 14

[45] Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot e E. Duchesnay: *Scikit-learn: Machine learning in Python.* Journal of Machine Learning Research, 12:2825–2830, 2011. 14

[46] Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu e Xiaoqiang Zheng: *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. https://www.tensorflow.org/, Software available from tensorflow.org. 15

[47] *Getting start tutorial - issue #68 - tensorflow/java*, Oct 2020. https://github.com/tensorflow/java/issues/68#issuecomment-704622036. 15

[48] Chollet, François *et al.*: *Keras.* https://keras.io, 2015. 16

[49] *Tensorflow lite | ml for mobile and edge devices.* https://web.archive.org/web/20201104084527/https://www.tensorflow.org/lite, Accessed: 2020-11-08. 16

[50] *Is tf lite optimized for nvidia gpu's and intel cpus?* http://web.archive.org/web/20201108062538/https://github.com/tensorflow/tensorflow/issues/34536, Accessed: 2020-11-08. 16

[51] *Onnx runtime.* https://web.archive.org/web/20201107081602/https://microsoft.github.io/onnxruntime/, Acessed: 2020-11-10. 16

[52] *Deeplearning4j.* https://web.archive.org/web/20201103204920/https://deeplearning4j.org/, Accessed: 2020-11-08. 17

[53] *Apache mxnet | a flexible and efficient library for deep learning.* https://web.archive.org/web/20201108093437/https://mxnet.apache.org/versions/1.7.0/, Accessed: 2020-11-10. 17

[54] Harris, Mark: *How to optimize data transfers in cuda c/c.* NVIDIA Developer Blog, Dec 2012. https://developer.nvidia.com/blog/how-optimize-data-transfers-cuda-cc/. 25

Apêndice A

Códigos usados para medição

A seguir são exibidos os principais arquivos e trechos de código utilizados na medição de desempenho dos classificadores implementados. O repositório completo pode ser encontrado na URL `https://github.com/mikaelmello/tcc`.

A.1 Medições em Java

Listing A.1: Dependências Maven comuns a todas medições em Java

```
<dependency>
    <groupId>commons-cli</groupId>
    <artifactId>commons-cli</artifactId>
    <version>1.4</version>
</dependency>
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.6</version>
</dependency>
```

Listing A.2: Classe principal de execução das medições em Java

```
package br.unb.cic;

import com.google.gson.Gson;
import org.apache.commons.cli.*;

import java.io.FileInputStream;
```

```
import java.io.FileWriter;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.zip.ZipInputStream;

public class Main {
    public static void main(String[] args) throws Exception {
        // create the command line parser
        var parser = new DefaultParser();

        var options = new Options();
        options.addRequiredOption("i", "input-path", true, "");
        options.addRequiredOption("o", "output-path", true, "");
        options.addRequiredOption("m", "model-path", true, "");


        var line = parser.parse(options, args);

        var inputPath = line.getOptionValue("i");
        var outputPath = line.getOptionValue("o");
        var modelPath = line.getOptionValue("m");

        var classifier = new Classifier(modelPath);
        var zis = new ZipInputStream(new FileInputStream(inputPath));

        var outputGson = new Gson();
        var outputList = new ArrayList<>();

        var zipEntry = zis.getNextEntry();
        while (zipEntry != null) {
            var bytes = zis.readAllBytes();
            var gson = new Gson();
            var workload = gson.fromJson(new String(bytes), Workload.class
                ↪ );

            for (int i = 0; i < workload.count; i++) {
                var input = workload.inputs.get(i);
```

```
                var data = input.data;

                var startTime = System.nanoTime();

                var output = classifier.classify(data);

                var stopTime = System.nanoTime();
                var usDuration = (stopTime - startTime) / 1000;

                var outputObj = new Output();
                outputObj.o = output;
                outputObj.d = usDuration;
                outputObj.du = "us";
                outputObj.eo = input.expected_output;
                outputObj.iid = input.id;
                outputList.add(outputObj);

                if (outputList.size() % 1000 == 0) {
                    System.out.print(String.format("\r%d␣registered...",
                        ↪ outputList.size()));
                }
            }

            zipEntry = zis.getNextEntry();
        }
        zis.closeEntry();
        zis.close();

        System.out.println("\nSaving!");
        var writer = new FileWriter(outputPath);
        outputGson.toJson(outputList, writer);
        writer.flush();
        writer.close();

        System.out.println("Done!");
    }
}
```

A.1.1 ONNX Runtime

A biblioteca ONNX Runtime exige modificações no código para uso ou não de GPU. Para uso de GPU, a dependência `onnxruntime_gpu` deve ser utilizada, caso contrário, `onnxruntime`. Além disso, a sessão criada por meio da API da ONNX Runtime deve adicionar explicitamente a GPU (dispositivo CUDA) para uso.

Listing A.3: Dependências Maven para uso de ONNX Runtime sem GPU

```
<dependency>
    <groupId>com.microsoft.onnxruntime</groupId>
    <artifactId>onnxruntime</artifactId>
    <version>1.5.2</version>
</dependency>
```

Listing A.4: Dependências Maven para uso de ONNX Runtime com GPU

```
<dependency>
    <groupId>com.microsoft.onnxruntime</groupId>
    <artifactId>onnxruntime_gpu</artifactId>
    <version>1.4.0</version>
</dependency>
```

Listing A.5: Classificador implementado em Java utilizando ONNX Runtime sem GPU

```
package br.unb.cic;

import ai.onnxruntime.NodeInfo;
import ai.onnxruntime.OnnxTensor;
import ai.onnxruntime.OrtEnvironment;
import ai.onnxruntime.OrtException;
import ai.onnxruntime.OrtSession;
import ai.onnxruntime.OrtSession.Result;

import java.nio.file.Paths;
import java.util.Collections;

import static java.lang.Math.round;

public class Classifier {
   private final OrtSession session;
```

```
    private final OrtEnvironment environment;
    private final String inputName;

    public Classifier(String modelDir) throws Exception {
        var modelPath = Paths.get(modelDir, "model.onnx").toString();
        environment = OrtEnvironment.getEnvironment();
        session = environment.createSession(modelPath);
        inputName = session.getInputNames().iterator().next();
    }

    public int classify(float[][] testData) throws OrtException {
        try (OnnxTensor test = OnnxTensor.createTensor(environment,
            ↪ testData);
             Result output = session.run(Collections.singletonMap(inputName
                ↪ , test))) {
            var x = (float[][])output.get(0).getValue();
            if (x[0][0] >= x[0][1] && x[0][0] >= x[0][2]) return 0;
            if (x[0][1] >= x[0][0] && x[0][1] >= x[0][2]) return 1;
            return 2;
        }
    }
}
```

Listing A.6: Classificador implementado em Java utilizando ONNX Runtime com GPU

```
package br.unb.cic;

import ai.onnxruntime.NodeInfo;
import ai.onnxruntime.OnnxTensor;
import ai.onnxruntime.OrtEnvironment;
import ai.onnxruntime.OrtException;
import ai.onnxruntime.OrtSession;
import ai.onnxruntime.OrtSession.Result;

import java.nio.file.Paths;
import java.util.Collections;

import static java.lang.Math.round;
```

```
public class Classifier {
   private final OrtSession session;
   private final OrtEnvironment environment;
   private final String inputName;

    public Classifier(String modelDir) throws Exception {
        var modelPath = Paths.get(modelDir, "model.onnx").toString();
        environment = OrtEnvironment.getEnvironment();
        int gpuDeviceId = 0; // The GPU device ID to execute on
        var sessionOptions = new OrtSession.SessionOptions();
        sessionOptions.addCUDA(gpuDeviceId);
        session = environment.createSession(modelPath, sessionOptions);
        inputName = session.getInputNames().iterator().next();
    }

    public int classify(float[][] testData) throws OrtException {
        try (OnnxTensor test = OnnxTensor.createTensor(environment,
            ↪ testData);
             Result output = session.run(Collections.singletonMap(inputName
                ↪ , test))) {
            var x = (float[][])output.get(0).getValue();
            if (x[0][0] >= x[0][1] && x[0][0] >= x[0][2]) return 0;
            if (x[0][1] >= x[0][0] && x[0][1] >= x[0][2]) return 1;
            return 2;
        }
    }
}
```

A.1.2 Deeplearning4j

A biblioteca Deeplearning4j não exige modificações no código para uso ou não de GPU, sendo esta variável controlada por meio da dependência instalada. Caso se queira usar a GPU, a dependência `nd4j-cuda-<cuda_version>-platform` deve ser utilizada, caso contrário, `deeplearning4j-core`.

Listing A.7: Dependências Maven para uso de Deeplearning4j sem GPU

```
<dependency>
```

```
    <groupId>org.nd4j</groupId>
    <artifactId>nd4j-native-platform</artifactId>
    <version>1.0.0-beta7</version>
</dependency>
<dependency>
    <groupId>org.deeplearning4j</groupId>
    <artifactId>deeplearning4j-core</artifactId>
    <version>1.0.0-beta7</version>
</dependency>
<dependency>
    <groupId>org.deeplearning4j</groupId>
    <artifactId>deeplearning4j-modelimport</artifactId>
    <version>1.0.0-beta7</version>
</dependency>
```

Listing A.8: Dependências Maven para uso de Deeplearning4j com GPU

```
<dependency>
    <groupId>org.nd4j</groupId>
    <artifactId>nd4j-native-platform</artifactId>
    <version>1.0.0-beta7</version>
</dependency>
<dependency>
    <groupId>org.nd4j</groupId>
    <artifactId>nd4j-cuda-10.1-platform</artifactId>
    <version>1.0.0-beta7</version>
</dependency>
<dependency>
    <groupId>org.deeplearning4j</groupId>
    <artifactId>deeplearning4j-modelimport</artifactId>
    <version>1.0.0-beta7</version>
</dependency>
```

Listing A.9: Classificador implementado em Java utilizando Deeplearning4j

```
package br.unb.cic;

import org.deeplearning4j.nn.modelimport.keras.KerasModelImport;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
```

```
import org.nd4j.linalg.factory.Nd4j;

import java.nio.file.Paths;

public class Classifier {
    private final MultiLayerNetwork model;

    public Classifier(String modelDir) throws Exception {
        var modelPath = Paths.get(modelDir, "model.h5").toString();
        this.model = KerasModelImport.importKerasSequentialModelAndWeights
            ↪ (modelPath);
    }

    public int classify(float[][] testData) {
        var input = Nd4j.create(testData);
        var x = model.output(input);
        if (x.getColumn(0).getDouble(0) >= x.getColumn(1).getDouble(0) &&
            ↪ x.getColumn(0).getDouble(0) >= x.getColumn(2).getDouble(0))
            ↪ return 0;
        if (x.getColumn(1).getDouble(0) >= x.getColumn(0).getDouble(0) &&
            ↪ x.getColumn(1).getDouble(0) >= x.getColumn(2).getDouble(0))
            ↪ return 1;
        return 2;
    }
}
```

A.2 Medições em Python

A Listagem A.10 exibe o arquivo principal de execução das medições em Python. Algumas medições utilizam uma versão levemente modificada onde o argumento `gpu` do programa é ignorado.

Listing A.10: Arquivo principal de execução das medições em Python

```
import time
import os
import argparse
import json
```

```
from classifier import Classifier
from zipfile import ZipFile


def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument("--model-path", "-m", required=True, type=str)
    parser.add_argument("--input-path", "-i", required=True, type=str)
    parser.add_argument("--output-path", "-o", required=True, type=str)
    parser.add_argument("--gpu", "-g", action="store_true", default=False)

    args = vars(parser.parse_args())
    return args


class Input:
    def __init__(self, input_path):
        self.input_zip = ZipFile(input_path)
        self.to_open = self.input_zip.namelist()
        self.open = False

    def get_next(self):
        while True:
            if not self.open:
                if not self.to_open:
                    return

                filename = self.to_open[0]
                self.to_open = self.to_open[1:]
                content = self.input_zip.read(filename)
                self.cur_file = json.loads(content.decode("utf8"))
                self.open = True
                self.cur_index = 0

            inputs = self.cur_file["inputs"]
            count = self.cur_file["count"]
            payload = inputs[self.cur_index]
```

```
            self.cur_index += 1
            if self.cur_index == count:
                self.open = False

            yield payload


class Output:
    def __init__(self, output_path):
        self.output_path = output_path
        self.results = []

    def register_result(self, td, output, inp):
        output_payload = {
            "iid": inp["id"],
            "o": output,
            "eo": inp["expected_output"],
            "d": td,
            "du": "us",
        }

        self.results.append(output_payload)

    def save(self):
        with open(self.output_path, "w") as f:
            json.dump(self.results, f, separators=(",", ":"))


def classify(classifier, data):
    start = time.time()
    res = classifier.classify(data)
    end = time.time()

    microseconds = int((end - start) * 1e6)
    return (res, microseconds)
```

```
if __name__ == "__main__":
    args = parse_args()

    model_path = os.path.join(args["model_path"], "model.onnx")
    input_path = args["input_path"]
    output_path = args["output_path"]

    classifier = Classifier(model_path, args["gpu"])
    data = Input(input_path)
    output = Output(output_path)

    count = 0
    for inp in data.get_next():
        if count % 1000 == 0:
            print(f"\r{count} registered...", end="")
        result = classify(classifier, inp["data"])
        output.register_result(result[1], result[0], inp)
        count += 1

    print("\nSaving!")
    output.save()
    print("Done!")
```

A.2.1 OpenCV

A biblioteca OpenCV exige *builds* customizadas do código-fonte para ser compatível com o uso de GPU. Por este motivo foram realizadas apenas medições com uso de CPU. A versão *headless* da biblioteca foi utilizada pois não era necessária a utilização de nenhuma funcionalidade visual.

Listing A.11: Dependências PIP para uso de OpenCV sem GPU

```
opencv_python_headless==4.4.0.44
numpy==1.18.5
```

Listing A.12: Classificador implementado em Python utilizando OpenCV

```
import cv2
```

```
import numpy as np


class Classifier:
    def __init__(self, model_path: str, gpu: bool):
        self.net = cv2.dnn.readNetFromONNX(model_path)
        self.net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)

        if gpu: # should not be used
            self.net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
        else:
            self.net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)

    def classify(self, values: list):
        self.net.setInput(np.array(values))
        prediction = self.net.forward()

        return int(np.argmax(prediction))
```

A.2.2 TensorFlow

Em TensorFlow, o código é o mesmo para executar com o uso de GPU ou não. O uso de GPU é controlado por meio da variável de ambiente `CUDA_VISIBLE_DEVICES`, igual a -1 quando a GPU não deve ser utilizada e 0 quando a GPU de ID 0 está disponível para ser utilizada.

Listing A.13: Dependências PIP para uso de TensorFlow

```
numpy==1.18.5
tensorflow==2.3.1
```

Listing A.14: Classificador implementado em Python utilizando TensorFlow

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import load_model


class Classifier:
```

```
    def __init__(self, model_path: str):
        self.model = load_model(model_path)

    def classify(self, values: list):

        input_data = np.asarray(values, dtype=np.float32)
        output_data = self.model.__call__(
            tf.convert_to_tensor(input_data), training=False
        )
        return int(np.argmax(output_data))
```

A.2.3 TensorFlow Lite

A instalação da biblioteca TensorFlow Lite depende da versão de Python utilizada para testes. Por este motivo, foram adicionadas condições no arquivo de dependências para que seja compatível com pelo menos 3 diferentes versões de Python.

Listing A.15: Dependências PIP para uso de TensorFlow Lite sem GPU

```
numpy==1.18.5
tensorflow==2.3.1
https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp36-cp36m-
    ↪ linux_x86_64.whl ; python_version == '3.6'
https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp37-cp37m-
    ↪ linux_x86_64.whl ; python_version == '3.7'
https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp38-cp38-
    ↪ linux_x86_64.whl ; python_version == '3.8'
```

Listing A.16: Classificador implementado em Python utilizando TensorFlow Lite

```
import tensorflow as tf
import numpy as np


class Classifier:
    def __init__(self, model_path: str):
        self.interpreter = tf.lite.Interpreter(model_path=model_path)
        self.interpreter.allocate_tensors()
        self.input_tensor = self.interpreter.get_input_details()[0]["index
            ↪ "]
```

```
        self.output_tensor = self.interpreter.get_output_details()[0]["
            ↪ index"]

    def classify(self, values: list):
        input_data = np.array(values, dtype=np.float32)
        self.interpreter.set_tensor(self.input_tensor, input_data)
        self.interpreter.invoke()
        output_data = self.interpreter.get_tensor(self.output_tensor)
        return int(np.argmax(output_data))
```

A.2.4 ONNX Runtime

Para o uso de ONNX Runtime em Python, o código necessário para execução dos modelos é o mesmo seja para uso de GPU ou não. Entretanto, é necessário o uso de uma biblioteca diferente chamada `onnxruntime-gpu` caso se queira utilizar a GPU. Além disso, é necessário se atentar à versão da biblioteca `onnxruntime-gpu` para que ela seja compatível com a versão da CUDA instalada na máquina, como a máquina utilizada na avaliação de desempenho possuía a CUDA versão 10.1 instalada, a versão 1.4.0 da biblioteca foi utilizada ao invés da versão 1.5.2, como no uso de CPU.

Listing A.17: Dependências PIP para uso de ONNX Runtime sem GPU

```
numpy==1.18.5
onnxruntime==1.5.2
onnx==1.7.0
```

Listing A.18: Dependências PIP para uso de ONNX Runtime com GPU

```
numpy==1.18.5
onnxruntime-gpu==1.4.0
onnx==1.7.0
```

Listing A.19: Classificador implementado em Python utilizando ONNX Runtime

```
import numpy as np
import onnxruntime as rt


class Classifier:
    def __init__(self, model_path: str):
```

```
        self.session = rt.InferenceSession(model_path)

    def classify(self, values: list):
        input_name = self.session.get_inputs()[0].name
        prediction = self.session.run(None, {input_name: values})

        return int(np.argmax(prediction))
```

Apêndice B

Modificações de código no ONS

Neste apêndice são exibidos trechos de códigos referentes às contribuições feitas ao código-fonte do *Optical Network Simulator* (ONS). O repositório completo do projeto pode ser encontrado na URL `https://github.com/comnetunb/ons-maven`.

B.1 Interface genérica de modelos

A interface Model (B.1) foi criada com o objetivo de representar uma interface genérica que deve ser utilizada para a representação de qualquer modelo, em qualquer biblioteca. Assim, outros módulos do código podem executar modelos de aprendizagem de máquina de forma abstraída, sem precisar detalhar detalhes do modelo além dos tipos de entrada e saída esperados.

A interface possui dois parâmetros genéricos, nomeados `IT` (*InputType*, ou tipo de entrada) e `OT` (*OutputType*, ou tipo de saída). Estes parâmetros devem representar os tipos de entrada e saída definidos pelo modelo que seria utilizado

A interface possui dois métodos: implementações de `predict` devem receber como parâmetro uma variável `input` (entrada) do tipo `IT` e devem retornar um valor do tipo `OT`. A definição de `predict` também indica que a execução do método pode lançar exceções. O método `getModelFramework` (obter a *framework* do modelo), deve retornar um valor do tipo `ModelFramework`, explicado na Seção B.2. A implementação deste método deve apenas retornar o valor correspondente à *framework* utilizada para a execução do modelo.

Listing B.1: Implementação da interface Model

```
package ons.util.ml;

public interface Model<IT, OT> {
    OT predict(IT input) throws Exception;
```

```
    ModelFramework getModelFramework();
}
```

B.2 Valores constantes

Foram criadas duas classes do tipo `enum`, ou enumeração, em que são definidos valores constantes para as *frameworks* ou bibliotecas usadas, no *enum* `ModelFramework` (B.2), e valores constantes para diferentes tipos de formatos de modelos, como ONNX e HDF5, no *enum* `ModelFormat` (B.3).

Listing B.2: Implementação do *enum* ModelFormat

```
package ons.util.ml;

public enum ModelFormat {
    ONNX,
    HDF5,
};
```

Listing B.3: Implementação do *enum* ModelFramework

```
package ons.util.ml;

public enum ModelFramework {
    ONNX_RUNTIME,
    DEEPLEARNING4J,
    UNKNOWN,
};
```

B.3 Modelos nativos

Foram escritas implementações de classes para a execução de modelos de forma nativa, isto é, modelos que são executados pelo programa do ONS.

A classe `NativeModel`, exibida na Listagem B.4, implementa a interface `Model` com as seguintes adições: definição de 4 métodos públicos abstratos nomeados `load`, responsáveis por carregar o modelo em memória; definição de um método protegido abstrato nomeado `_predict`, responsável por executar de fato o modelo; implementação do método público `predict`, definido pela interface `Model`, com o objetivo de garantir que o modelo foi carregado antes de executar de fato o modelo por meio do método `_predict`.

Listing B.4: Implementação da classe abstrata NativeModel

```
package ons.util.ml;

import java.io.InputStream;

public abstract class NativeModel<IT, OT> implements Model<IT, OT> {
    protected final ModelFramework modelFramework;
    protected boolean ready;

    protected NativeModel(final ModelFramework modelFramework) {
        this.modelFramework = modelFramework;
        this.ready = false;
    }

    public abstract void load(String path) throws Exception;
    public abstract void load(String path, ModelFormat format) throws
        ↪ Exception;
    public abstract void load(InputStream inputStream) throws Exception;
    public abstract void load(InputStream inputStream, ModelFormat format
        ↪ ) throws Exception;

    protected abstract OT _predict(IT input) throws Exception;

    public OT predict(IT input) throws Exception {
        if (!ready) {
            throw new IllegalStateException("Must␣load␣local␣model␣before␣
                ↪ predicting");
        }

        return _predict(input);
    }

    public ModelFramework getModelFramework() {
        return modelFramework;
    }
}
```

A Listagem B.5 exibe a implementação da classe `OnnxNativeModel`, responsável por

implementar os métodos de carregamento de modelos do formato ONNX em memória utilizando a biblioteca ONNX Runtime. A Listagem B.6 se refere à implementação da classe `Dl4jNativeModel`, responsável por implementar os mesmos métodos que a anterior, porém com o apoio da biblioteca Deeplearning4j.

Listing B.5: Implementação da classe OnnxNativeModel

```
package ons.util.ml;

import ai.onnxruntime.OrtEnvironment;
import ai.onnxruntime.OrtException;
import ai.onnxruntime.OrtSession;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

public abstract class OnnxNativeModel<IT, OT> extends NativeModel<IT, OT>
    ↪ {
    protected OrtSession session;
    protected OrtEnvironment environment;
    protected String inputName;

    public OnnxNativeModel() {
        super(ModelFramework.ONNX_RUNTIME);
    }

    public void load(String path) throws IOException, OrtException {
        load(path, ModelFormat.ONNX);
    }

    public void load(String path, ModelFormat format) throws IOException,
        ↪ OrtException {
        InputStream targetStream = new FileInputStream(path);
        load(targetStream, format);
    }

    public void load(InputStream inputStream) throws IOException,
        ↪ OrtException {
```

```
        load(inputStream, ModelFormat.ONNX);
    }

    public void load(InputStream is, ModelFormat format) throws
        ↪ IOException, OrtException {
        if (format != ModelFormat.ONNX) {
            throw new UnsupportedOperationException("Unsupported format");
        }

        environment = OrtEnvironment.getEnvironment();
        session = environment.createSession(is.readAllBytes());
        inputName = session.getInputNames().iterator().next();
        ready = true;
    }
}
```

Listing B.6: Implementação da classe Dl4jNativeModel

```
package ons.util.ml;

import org.deeplearning4j.nn.modelimport.keras.KerasModelImport;
import org.deeplearning4j.nn.modelimport.keras.exceptions.
    ↪ InvalidKerasConfigurationException;
import org.deeplearning4j.nn.modelimport.keras.exceptions.
    ↪ UnsupportedKerasConfigurationException;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

public abstract class Dl4jNativeModel<IT, OT> extends NativeModel<IT, OT>
    ↪ {
    protected MultiLayerNetwork model;

    public Dl4jNativeModel() {
        super(ModelFramework.DEEPLEARNING4J);
    }
```

```
    public void load(String path) throws IOException,
        ↪ InvalidKerasConfigurationException,
        ↪ UnsupportedKerasConfigurationException {
        load(path, ModelFormat.HDF5);
    }

    public void load(String path, ModelFormat format) throws IOException,
        ↪  UnsupportedKerasConfigurationException,
        ↪ InvalidKerasConfigurationException {
        InputStream targetStream = new FileInputStream(path);
        load(targetStream, format);
    }

    public void load(InputStream inputStream) throws IOException,
        ↪ InvalidKerasConfigurationException,
        ↪ UnsupportedKerasConfigurationException {
        load(inputStream, ModelFormat.HDF5);
    }

    public void load(InputStream is, ModelFormat format) throws
        ↪ UnsupportedKerasConfigurationException, IOException,
        ↪ InvalidKerasConfigurationException {
        if (format == ModelFormat.HDF5) {
            this.model = KerasModelImport.
                ↪ importKerasSequentialModelAndWeights(is);
            ready = true;
        } else {
            throw new UnsupportedOperationException("Unsupported␣format");
        }
    }
}
```

As Listagens B.7 e B.8 exibem as implementações dos classificadores que utilizam Deeplearning4j e ONNX Runtime respectivamente. Nelas, é possível observar como ambas herdam de suas respectivas classes que herdam de `NativeModel` e necessitam implementar apenas uma simples função específica do modelo.

Listing B.7: Implementação da classe Dl4jClassifier

```
package ons.tools;

import ons.util.ml.Dl4jNativeModel;
import org.nd4j.linalg.factory.Nd4j;

public class Dl4jClassifier extends Dl4jNativeModel<float[][], Number> {
    protected Integer _predict(float[][] testData) {
        var input = Nd4j.create(testData);
        var x = this.model.output(input);
        if (x.getColumn(0).getDouble(0) >= x.getColumn(1).getDouble(0) &&
            ↪ x.getColumn(0).getDouble(0) >= x.getColumn(2).getDouble(0))
            ↪ return 0;
        if (x.getColumn(1).getDouble(0) >= x.getColumn(0).getDouble(0) &&
            ↪ x.getColumn(1).getDouble(0) >= x.getColumn(2).getDouble(0))
            ↪ return 1;
        return 2;
    }
}
```

Listing B.8: Implementação da classe OnnxClassifier

```
package ons.tools;

import ai.onnxruntime.OnnxTensor;
import ai.onnxruntime.OrtException;
import ai.onnxruntime.OrtSession;
import ons.util.ml.OnnxNativeModel;

import java.util.Collections;

public class OnnxClassifier extends OnnxNativeModel<float[][], Number> {
    public Integer _predict(float[][] testData) throws OrtException {
        try (OnnxTensor test = OnnxTensor.createTensor(this.environment,
            ↪ testData);
             OrtSession.Result output = this.session.run(Collections.
                ↪ singletonMap(this.inputName, test))) {
            var x = (float[][])output.get(0).getValue();
```

```
            if (x[0][0] >= x[0][1] && x[0][0] >= x[0][2]) return 0;
            if (x[0][1] >= x[0][0] && x[0][1] >= x[0][2]) return 1;
            return 2;
        }
    }
}
```

B.4 Modelos remotos

Para permitir a execução de modelos de forma flexível, como elencado na Seção 5.1, a classe `HttpRemoteModel` (B.9) foi implementada. Ela implementa a interface `Model` de modo que a execução do modelo é realizada por meio de uma requisição HTTP POST: os dados de entrada são serializados em JSON, a requisição é feita, a resposta em JSON é desserializada e retornada para onde a função foi chamada.

Listing B.9: Implementação da classe Dl4jNativeModel

```
package ons.util.ml;

import com.google.gson.Gson;
import org.apache.http.client.HttpClient;
import org.apache.http.client.HttpResponseException;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;

public class HttpRemoteModel<IT, OT> implements Model<IT, OT> {
    private final HttpClient httpClient;
    private final String endpointUrl;
    private final Gson gson;

    public HttpRemoteModel(String endpointUrl) {
        this.httpClient = HttpClientBuilder.create().build();
        this.endpointUrl = endpointUrl;
        this.gson = new Gson();
    }
```

```
    private class Request {
        IT input;
    }

    private class Response {
        OT output;
    }

    public OT predict(IT input) throws Exception {
        var request = new Request();
        request.input = input;

        var httpPost = new HttpPost(endpointUrl);
        var entity = new StringEntity(gson.toJson(request));

        httpPost.setEntity(entity);
        httpPost.setHeader("Content-type", "application/json");

        var response = httpClient.execute(httpPost);

        var statusCode = response.getStatusLine().getStatusCode();

        if (200 > statusCode || statusCode >= 300){
            throw new HttpResponseException(statusCode, response.
                ↪ getStatusLine().getReasonPhrase());
        }

        var jsonString = EntityUtils.toString(response.getEntity());
        var responseObj = gson.fromJson(jsonString, Response.class);

        return (OT)responseObj.output;
    }

    public ModelFramework getModelFramework() {
        return ModelFramework.UNKNOWN;
    }
}
```

B.5 Servidor HTTP para servir execuções de modelos

Para realizar testes acerca do funcionamento da implementação das classes descritas no Apêndice B.4, um simples servidor HTTP foi implementado, exibido na Listagem B.10. Seu papel é reduzido a ouvir por requisições POST, ler e desserializar o corpo da requisição em formato JSON, executar o modelo utilizando o corpo da requisição como entrada, responder a requisição com o resultado do modelo serializado em JSON. O modelo é executado pela biblioteca ONNX Runtime, como pode ser visto na Listagem B.11.

Listing B.10: Simples servidor HTTP para testes de integração

```
from http.server import BaseHTTPRequestHandler, HTTPServer
from classifier import Classifier
import json
import os

classifier = Classifier(os.getenv("MODEL_PATH"))


class Handler(BaseHTTPRequestHandler):
    def do_POST(self):
        content_length = int(self.headers["Content-Length"])
        post_data = self.rfile.read(content_length)

        inp = json.loads(post_data.decode("utf-8"))["input"]
        res = classifier.classify(inp)

        self.send_response(200)
        self.send_header("Content-type", "application/json")
        self.end_headers()
        self.wfile.write(json.dumps({"output": res}).encode("utf-8"))


def run(server_class=HTTPServer, port=8080):
    server_address = ("", port)
    httpd = server_class(server_address, Handler)
    try:
        httpd.serve_forever()
```

```
    except KeyboardInterrupt:
        pass
    httpd.server_close()


if __name__ == "__main__":
    from sys import argv

    if len(argv) == 2:
        run(port=int(argv[1]))
    else:
        run()
```

Listing B.11: Classificador usado nas medições e pelo servidor HTTP

```
import numpy
import onnxruntime as rt


class Classifier:
    def __init__(self, model_path: str):
        self.session = rt.InferenceSession(model_path)

    def classify(self, values: list):
        input_name = self.session.get_inputs()[0].name
        prediction = self.session.run(None, {input_name: values})

        return int(numpy.argmax(prediction))
```