**Instituto de Pesquisas Tecnológicas do Estado de São Paulo**

Marcelo de Rezende Martins

**Uso de redes neurais recorrentes na detecção de padrões de erros cometidos por alunos novatos durante a aprendizagem de programação**

São Paulo

2018

Marcelo de Rezende Martins

Uso de redes neurais recorrentes na detecção de padrões de erros cometidos por alunos novatos durante a aprendizagem de programação

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia da Computação

Data da aprovação \_\_\_\_/\_\_\_\_\_/\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof.Dr. Marco Aurélio Gerosa (Orientador)

USP – Universidade de São Paulo

Membros da Banca Examinadora:

Prof. Dr. Marco Aurélio Gerosa (Orientador)

USP – Universidade de São Paulo

Prof. Dr. YYYY (Membro)

ZZZZ.

Prof. WWWW (Membro)

ZZZZZZ

Marcelo de Rezende Martins

Uso de redes neurais recorrentes na detecção de padrões de erros cometidos por alunos novatos durante a aprendizagem de programação

Pesquisa apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, como requisito para a obtenção do título de Mestre em Engenharia da Computação.

Área de Concentração: Engenharia de Software

Orientador: Prof. Dr. Marco Aurélio Gerosa

São Paulo

Abril/2018

Ficha Catalográfica  
Elaborada pelo Centro de Informação Tecnológica do   
Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT

# 

# **A555p**

PEDIR POR EMAIL PARA MARIA DARCI [mdarci@ipt.br](mailto:mdarci@ipt.br) ou telefone 3767 4139.

05-63 CDU 004.41(043)

**Resumo**

Reumo

Palavras-chave: redes neurais recorrentes, aprendizagem de máquina, engenharia de atributos, dificuldades, erros, ciências da computação e educação

Keywords: recurrent neural networks, machine learning, feature engineering, difficulties, errors, ai for education, computer science education.

LISTA DE FIGURAS

[Figura 1 - Processo de revisão sistemática 24](#_Toc512882887)

LISTA DE QUADROS

No table of figures entries found.

LISTA DE ABREVIATURAS E SIGLAS

MOOC – Curso online, massivo e aberto (Massive Open Online Course)

SUMÁRIO

[**LISTA DE FIGURAS** 10](#_Toc512693368)

[**LISTA DE QUADROS** 11](#_Toc512693369)

[**SUMÁRIO** 12](#_Toc512693370)

[1 Introdução 13](#_Toc512693371)

[1.1 Apresentação e contextualização do problema 16](#_Toc512693372)

[1.2 Objetivo 19](#_Toc512693374)

[1.3 Contribuições Esperadas 20](#_Toc512693375)

[1.4 Método de trabalho 20](#_Toc512693376)

[1.5 Organização do trabalho 20](#_Toc512693377)

[1.6 Cronograma 21](#_Toc512693378)

[2. Conceitos e Trabalhos Relacionados 21](#_Toc512693379)

[2.1 Aprendizagem de Máquina 21](#_Toc512693382)

[2.2 Deep Learning 22](#_Toc512693383)

[2.3 Redes Neurais Recorrentes 22](#_Toc512693384)

[2.4 Trabalhos Relacionados 22](#_Toc512693385)

[3 Modelos Para Análise de código-fonte 22](#_Toc512693386)

[4 Arquitetura do Sistema para Detecção de Erros 22](#_Toc512693387)

[5 Conclusão 23](#_Toc512693389)

# Introdução

Em 1936, Alan Turing publicou um artigo chamado *On Computable Numbers, with an application to the Entscheidungsproblem* (TURING, 1936, p. 230—265) no qual descreveu uma máquina teórica capaz de resolver qualquer problema que pudesse ser descrito através de instruções pré-definidas numa fita de papel. Este artigo tornaria-se a base para a computação e invenção dos computadores. Durante a segunda guerra, Turing construiu uma máquina capaz de quebrar a maioria do código Morse secreto usado nas comunicações pela força naval alemã. Segundo alguns especialistas, a guerra teria durado mais tempo se não fosse a máquina inventada por Alan Turing para quebrar a criptografia das mensagens. A humanidade deu um grande salto graças ao avanço dos computadores desde o fim da segunda guerra até os dias de hoje. Este salto tecnológico permitiu o avanço em diversas áreas como saúde, telecomunicações, aviação, engenharia, financeira etc.

Com o aumento do poder de processamento computacional dobrando a cada 18 meses, Lei de Moore, novos avanços em tecnologia surgem como a Internet da Coisas (IoT), cidades inteligentes (Smart Cities), Indústria 4.0. Todas tem um ponto em comum: computação úbiqua, uso de inteligência artificial e big data. E um outro ponto é a automação. Governos em todo o mundo estão preocupados com o advento da inteligência artificial e automação, pois muitos especialistas dizem que haverá um desemprego em massa. A maioria dos empregos que tem uma tarefa repetitiva, que não exige criatividade e interação social, serão automatizados. Em sua maioria, são empregos exercidos por pessoas com baixa qualificação (SMIT, 2016, p. 26). Segundo um relatório da OCDE, este desemprego em massa não ocorrerá tão rapidamente, pois o computador consegue fazer somente tarefas bem definidas e a maioria dos trabalhos não tem uma definição clara das tarefas a serem feitas (ARNTZ. Melanie; GREGORY. Terry; ZIERAHN. Ulrich., 2016, p. 22).

Em um item todos os governos concordam: há uma necessidade em preparar a nova geração para esta nova era. Hoje há um defícit de programadores em todo o mundo. E com o advento do IoT, Indústria 4.0, a demanda por desenvolvedores e programadores irá aumentar ainda mais. Governos de todo o mundo estão empenhados em um projeto de ensino de programação no currículo básico das escolas para atender a esta demanda. Alan Perlis disse em 1960 que programação deveria ser ensinada a todos. O modo de pensar "algoritmicamente" é útil para diversos problemas em nosso cotidiano. E com a popularização e a presença em massa dos computadores em nosso cotidiano, a importância de saber programar aumenta. Governos e empresas multinacionais juntaram forças para ensinar programação à populacao através de iniciativas como o code.org, udacity, scratch etc.

Mas aprender a programar não é uma tarefa fácil. Diversos estudos apontam para a alta taxa de evasão dos alunos dos cursos de Ciência da Computação (WATSON, Christopher; LI, Frederick W.B., 2014, p. 39—44). E outros estudos apontam a alta taxa de reprovação por parte dos alunos nos cursos de introdução à programação (WATSON, Christopher; LI, Frederick W.B., 2014, p. 39—44; BOSSE, Yorah; GEROSA, Marco Aurélio, 2015, p. 1389--1397). Segundo um estudo preliminar feito por Bosse (2015), a taxa de reprovação nos cursos introdutórios de programação ultrapassam a taxa de 30% somente na Universidade de São Paulo. A reprovação no curso de introdução à programação é um dos fatores que contribuem para a evasão do curso. Segundo os dados do censo do Ensino Superior divulgado pelo MEC em 2016, a taxa de evasão média para cursos de bacheralado é em torno de 22%, enquanto para os cursos de bacharelado em Computação, a taxa de evasão supera 28%.

Segundo Kalelioglu (2016), o pensamento computacional é um processo para resolução de problemas, envolve diversos modelos mentais como abstração, reconhecimento de padrões, decomposição, generalização etc. Além da necessidade de desenvolver o pensamento computacional, é importante saber quais são as principais dificuldades encontradas pelos alunos.

Segundo um estudo feito por Brown e Altadmri (2017), a maioria dos instrutores e professores não conseguem identificar quais são os erros mais frequentes cometidos pelos estudantes. A partir dos erros, é possível elaborar materiais e alterar o ritmo da aula para um melhor aproveitamento do aluno.

Com o aumento do poder computacional, crescente uso de ferramentas de aprendizagem de máquina e a grande oferta de repositórios de código-fontes como Github, por exemplo, nasce o termo Big Code (BHATIA, Sahil; SINGH, Rishabh, 2016). Big Code refere-se ao conhecimento adquirido a partir da análise dos repositórios de código fonte através de ferramentas de aprendizagem de máquina. Construir ferramentas que sejam capazes de aprender a partir dos repositórios de código fonte é um novo desafio. Com o conhecimento que podemos adquirir a partir da análise dos repositórios, podemos construir ferramentas para analisar a arquitetura de um projeto, verificar se o código fonte ou o nome do método está de acordo com a convenção, detectar erros de execução de forma antecipada.

Com as ferramentas de aprendizagem de máquina, é possível também identificar os erros mais comuns cometidos pelos alunos. Esta técnica é utilizada para dar uma resposta automática para o aluno, permitindo que ele receba uma avaliação mais rápida que a dada pelo professor ou monitor. Diversos estudos apontam redes neurais como uma técnica promissora na criação de ferramentas para detectar e corrigir erros de sintaxe, erros de semântica (WANG, 2017) . Um texto publicado no site da Microsoft, mostrou como o uso do algoritmo de *Deep Learning* foi eficaz para identificar as correções necessárias no código fonte submetido pelo aluno.

## Apresentação e contextualização do problema

A oferta de cursos MOOC para ensinar programação aumentou exponencialmente nos últimos 10 anos (adicionar referência). Há diversos fatores que contribuiram para isto. Entre eles a popularização do acesso à internet banda larga e a crescente demanda por programadores ao redor do mundo (adicionar referência). As plataformas MOOC mais populares hoje que ensinam programação são Coursera, EDx, Udacity e Codecademy. Nestas plataformas são disponibilizados aos alunos vídeos, textos e também exercícios para praticar o conteúdo visto. Há outras plataformas que são um repositório de exercícios. O HackerRank e o URIOnlineJudge, por exemplo, contém milhares de exercícios sobre uma variedade enorme de tópicos de computação.

O ensino a distância tem vários desafios e um deles é como reter o aluno. A porcentagem de abandono dos cursos online é alto, quando comparado com os cursos presenciais (verificar se isso é verdade). Entre os motivos da desistência do curso tem a falta de *feedback* e a demora na correção (acrescentar mais motivos e referências). Os exercícios são uma importante ferramenta no aprendizado do aluno. E a correção dos exercícios é uma oportunidade para que o aluno saiba a causa e o motive pelo qual errou.

Segundo XXXX, nos cursos de programação MOOC, a correção e o *feedback* do exercícios são feitos da seguinte maneira:

* Correção automática
* Semi-automática
* Manual

Na correção automática, a nota do aluno é calculada de acordo com a porcentagem de testes que o exercício acertou (é verdade isso?). Neste caso, o *feedback* é providenciado a partir de um mapeamento dos testes e possíveis erros associados (achar referências....). Já na semi-automática, o professor e/ou monitor corrigem os exercícios individualmente. No momento em que o professor provê um *feedback* a um aluno, o sistema já verifica códigos-fontes similares e já envia o mesmo *feedback* para outros alunos (adicionar a referência). Já na manual, os exercícios são corrigidos e *feedbacks* são providenciados individualmente pelo professor e/ou monitor.

Neste contexto, surgiram soluções de correção automática e semi-automática para plataformas MOOC utilizando inteligência artificial. Segundo um estudo XXXX, as principais soluções de correções de exercícios de programação automático são baseados em clustering, utilizando o algoritmo KNN, por exemplo (adicionar mais exemplos, referências). Recentemente, surgiram soluções utilizando aprendizagem de máquina profundo, como o sk\_p, DeepFix, SynFix entre outros para identificar erros de programação (WANG, 2017). Tanto o SynFix quanto o DeepFix utlizam redes recorrentes. E ambos utilizam sequência de *tokens* como forma de representação da entrada de dados para treinar a rede neural (refazer este trecho usando o artigo do WANG).

Segundo Wang (2017), as redes recorrentes mostraram-se promissoras na identificação e correção de erros em códigos-fontes. Porém, a representação dos programas através de uma sequência de *tokens* dificulta a aprendizagem pela rede neural. Segundo Wang (2017), a diferença na sintaxe de um programa correto e errado é de apenas poucos caracteres, na maioria dos casos. Esta diferença mínima, dificulta a rede neural construir o melhor modelo para diferenciar os programas corretos e programas errados.

Wang (2017) propôs uma forma de representar o

Esta dissertação tentará responder a seguinte pergunta:

1. Como as redes neurais recorrentes podem ser utilizadas para identificar padrões de erros de semântica cometidos por alunos?

## Objetivo

O uso de algoritmo de aprendizagem para correção automática de exercícios não é novidade. Porém, boa parte dos estudos foram feitos para plataformas MOOC, no qual a maioria dos exercícios são avaliados por meio de uma base de testes e a nota representa a porcentagem de testes que o exercício passou. Neste estudo, identificaremos os padrões de erros por meio do uso do algoritmo de redes neurais aplicado num conjunto de exercícios previamente avaliados por monitores e professores. A avaliação feita por monitores e instrutores leva em consideração se o aluno entendeu conceitos como instruções de repetição, instruções de decisão, chamada de métodos, recursão, uso de vetores, ponteiros, passagem de parâmetros etc. Este tipo de avaliação é diferente das avaliações baseadas em porcentagem de testes.

O objetivo da pesquisa é verificar o uso das redes neurais recorrentes como técnica para detecção de padrões de erros cometidos por alunos previamente analisados por monitores e instrutores. O escopo deste estudo será limitado inicialmente a uma classe de problema. Normalmente, os exercícios feitos pelos alunos visam reforçar um conceito visto em sala de aula. Por exemplo, o propósito de um programa para verificar se um ano é bissexto é reforçar os conceitos de instrução de decisão, operadores booleanos. Já um programa para exibir um calendário reforça os conceitos de instrução de decisão, instrução de repetição, operadores booleanos, formatação da saída, porém não exige o conhecimento de matrizes, por exemplo. Neste estudo, as redes neurais recorrentes serão utilizadas para detectar padrões de erros de lógica em códigos-fontes feitos em Python, inicialmente.

## Contribuições Esperadas

Conforme mencionado anteriormente, o contexto do uso de redes neurais recorrentes na detecção de padrões de erros será a principal contribuição deste trabalho. Contribuições secundárias serão o levantamento de padrões de erros de lógica cometidos por alunos novatos durante a aprendizagem de programação, definição de classe de problemas e exercícios, definição dos atributos que serão levados em consideração para o treinamento, análise e validação do uso de redes neurais recorrentes para detecção de padrões de erros no contexto deste trabalho.

## Método de trabalho

## Organização do trabalho

## Cronograma

# Conceitos e Trabalhos Relacionados



## Aprendizagem de Máquina

## Redes Neurais

## Aprendizagem de Máquina Profundo

## Big Code

## Trabalhos Relacionados

Para identificar os trabalhos relacionados, foi conduzido uma revisão sistemática. Segundo Bapuji et al. (2007 apud WALKER, 2010), a revisão sistemática aumenta a qualidade do processo de revisão estabelecendo uma revisão sistemática, transparente e reproduzível da literatura.

Segundo Walker (2010), uma revisão sistemática deve incluir objetivos claros, ser reproduzível, ter uma busca ampla e inclusiva, quer dizer, a seleção deve ser baseada no mérito do trabalho, a fim de reduzir o viés cognitivo. Além disso, deve incorporar um método de síntese para organizar a literatura.

Para apoiar a revisão sistemática foi utilizado a ferramenta StArt desenvolvido pelo Laboratório de Pesquisa em Engenharia de Software da Universidade Federal de São Carlos (UFSCar). De acordo com Kitchenham e Charters (2007 apud FABRI et al., 2016), a revisão sistemática é composta por três fases: Planejamento, Condução e Publicação dos resultados.

Na fase de planejamento, é definido o objetivo da revisão e é criado o protocolo contendo as informações importantes da revisão sistemática. Na fase de condução, é identificada e feita a seleção dos estudos primários relevantes, extração e síntese dos dados. E, por fim, na fase de publicação dos resultados, é feito a descrição, divulgação e avaliação dos resultados relatados (FABRI et al., 2016).

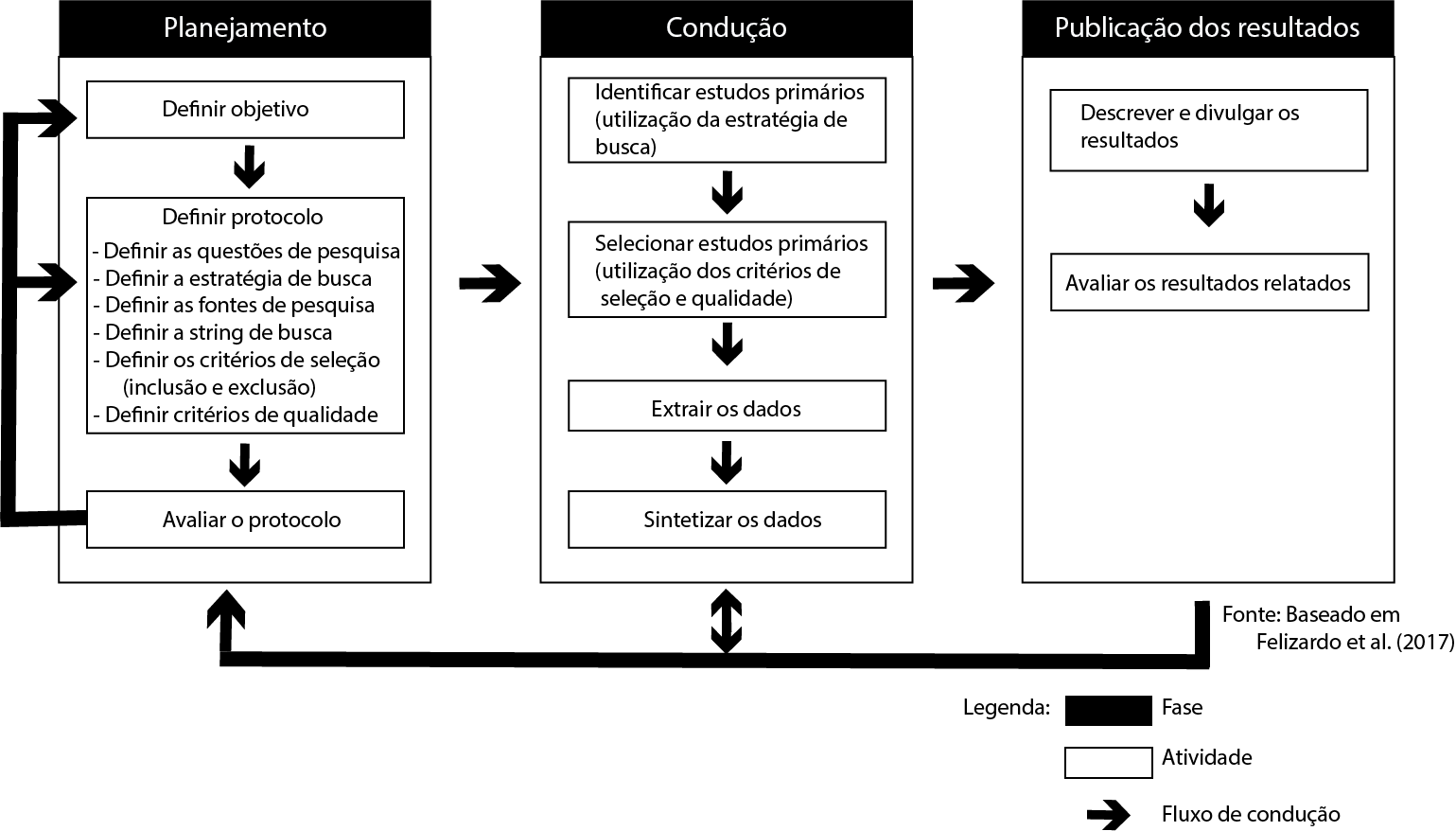


Figura 1 - Processo de revisão sistemática

­

# Modelo Para Código-Fonte

# RNN

# GRU

# Modelo

# Arquitetura do Sistema para Detecção de Erros



# Conclusão