AVALIANDO AS REDES NEURAIS RECORRENTES NA COMPLEMENTAÇÃO DE CÓDIGO

Matheus Feijó Ferreira Guimarães[1], **Carine Geltrudes Webber**[2]

[1] Universidade de Caxias do Sul Área do Conhecimento de Ciência Exatas e Engenharias

{mffguimaraes,cgwebber}@ucs.br

Abstract. *Code completion is a desirable feature for Integrated development environment. Several are the challenges encountered in its implementation. Languages present innumerable classes, methods, interfaces and resources. Therefore, recommending or indicating commands, instructions implies analyzing countless possibilities. The code completion functionality allows for better assertiveness during the development process. In terms of applied methods, the systematic literature review strongly indicated the use of natural language processing techniques, among which neural models showed a high rate of use. The recurrent neural network stand out for their ability to consider previous values when generating the prediction, this ability was strongly indicated as fruitful for code completion. This project aims to evaluate variations of recurrent neural network for completion tasks for the Python programming language focused on recommendations of application programming interface . For this purpose, several variations of recurrent neural network will be tested.*

Resumo. *A complementação de código é uma funcionalidade desejável a ambientes de desenvolvimento integrados. Diversos são os desafios encontrados na sua implementação. As linguagens apresentam inúmeras classes, métodos, interfaces e recursos. Por isso, recomendar ou indicar comandos, instruções implica em analisar inúmeras possibilidades. A funcionalidade de complementação de código permite uma melhor assertividade durante o processo de desenvolvimento. Em termos de métodos aplicados, a revisão sistemática da literatura indicou fortemente o uso de técnicas de processamento de linguagem natural, dentre elas os modelos neurais apresentaram alta taxa de uso. As redes neurais recorrentes destacam-se pela capacidade de considerar valores anteriores ao gerar a predição, essa capacidade foi fortemente indicada como proveitosa para a complementação de código. Este projeto visa avaliar variações de redes neurais recorrentes para tarefas de complementação para a linguagem de programação Python focadas em recomendações para interface de programação de aplicações. Sendo para isto testados diversas variações de redes neurais recorrentes.*

1. Introdução

O processo de desenvolvimento de software normalmente faz uso de *frameworks*, bibliotecas e Interface de Programação de Aplicações (API). Esses recursos contribuem para a produtividade de profissionais e estudantes da área de desenvolvimento de software, possibilitando o desenvolvimento e a manutenção de softwares com maior qualidade. Contudo, o seu uso acarreta em uma grande quantidade de informações para serem

lembradas. Para ilustrar, a *Standard Edition of the Java Development Kit (version* 1.6) contém 3.777 classes e interfaces[1]. Conhecê-las e usá-las, além de seus métodos, não é uma tarefa trivial. Por isso, muitos desenvolvedores empregam ferramentas ou recursos de recomendação e complementação de código [Hou and Pletcher 2011].

Os recursos de complementação de código normalmente apresentam uma lista de sugestões de possíveis métodos ou funções das API. Mesmo assim, em alguns casos, o número de sugestões é elevado. Isso pode ocorrer devido a fatores, dentre eles o grande número de possibilidades na API, ou ainda, a filtragem ou ordenação ineficiente, prejudicando a assertividade do método. Nesses casos, muitas vezes o programador precisa interromper a sua tarefa de programação, consultar manuais e materiais externos, para então retornar a sua produtividade. Com isso, há uma perda de foco, concentração e tempo. Neste sentido, a necessidade de melhoria dos métodos de complementação existentes é evidente.

A Figura 1 exemplifica uma recomendação gerada por complementação de código em linguagem de programação *Python*. Nela observa-se um código simples, que faz uso de uma variável *a* que pode assumir o tipo *string* ou inteiro. Na última linha do código fonte apresentado, ao digitar-se *a.* ocorre uma chamada ao método de complementação de código que deve realizar recomendações compatíveis com os dois tipos possíveis de inferência. Demonstra-se desta forma que a recomendação do código não considera apenas a última ocorrência da variável *a*, mas sim um trecho do código maior podendo compreender todo o ciclo de vida da variável. Os métodos utilizados atualmente para complementação de código são bem variados. Observa-se métodos baseados em frequência dos termos, outros em proximidade dos termos ou existindo ainda os preditivos. Recentemente novos modelos de complementação de código, inspirados por modelos estruturais e *machine learning* (ML), estão sendo construídos [Luo 2017].

O objetivo deste trabalho é desenvolver um estudo exploratório para identificar as técnicas oriundas da Inteligência Artificial (IA) para a complementação de código, avaliar e aplicar uma delas no contexto da linguagem de programação *Python* [2]. Para isso, organizou-se este artigo em 6 seções. A seção 2 apresenta o processo de revisão sistemática realizado. A partir dele evidenciou-se o uso das redes neurais recorrentes, descritas na seção 3. A seção 4 apresenta o método empregado neste trabalho. Por fim, as seções 5 e 6 detalham resultados e conclusões do estudo.

2. Revisão Sistemática da Literatura

A fim de mapear o estado da arte na área sobre complementação de código, procedeu-se com uma revisão sistemática da literatura [Okoli et al. 2019]. Seguindo o referido processo do método de revisão sistemática, iniciou-se definindo como propósito da pesquisa os métodos de inteligência artificial utilizados para a implementação de um sistema de complementação de código. Após esta etapa, realizou-se o processo de busca de bibliografias, que foi realizado no portal de periódicos da *CAPES*[3]. Aplicou-se então como filtro para a busca bibliográfica a seguinte palavra-chave: *code completion*, em conjunto com o filtro de ano de publicação no intervalo de 2016 a 2021.

[1]Linguagem de programação Java, disponível em http://www.oracle.com/java

[2]Linguagem de programação *Python*, disponível em http://www.python.org

[3]https://www-periodicos-capes-gov-br.ezl.periodicos.capes.gov.br

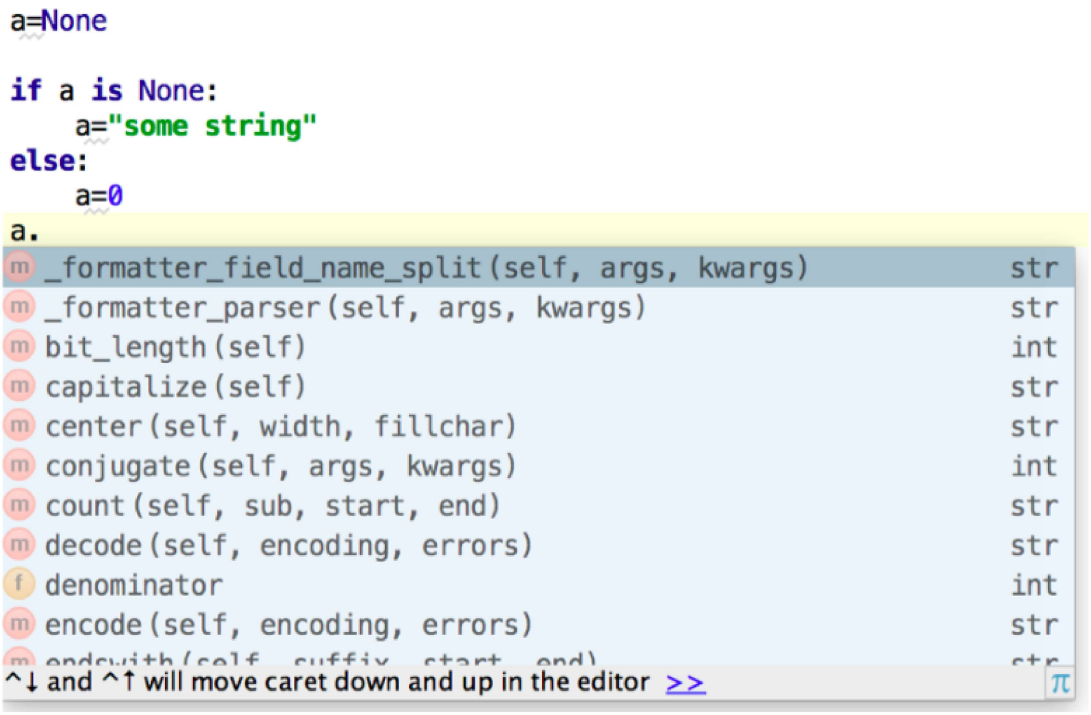

Figura 1. Exemplo de Complementação de Código.

Aplicados os filtros, obteve-se 89 trabalhos que foram selecionados de acordo com a leitura dos seus títulos e resumos. Após a nova filtragem, identificou-se que somente 15, dos 89 trabalhos, apresentavam um escopo relacionado ao propósito definido. Em análise mais aprofundada identificou-se que 8 trabalhos apresentavam escopo não fortemente relacionado ao problema de complementação de código(geração de código), sendo portanto descartados. Aos 7 trabalhos restantes foram aplicadas as etapas de extração e síntese (leitura e elaboração de ficha).

A partir das etapas de extração e síntese, tabularam-se as técnicas que obtiveram melhores resultados na implementação de sistemas de complementação de código. Na tabela 1 são apresentados os autores dos trabalhos selecionados, as principais técnicas e siglas empregadas. Dos 7 artigos analisados, 5 obtiveram melhores resultados empregando as redes neurais recorrentes. Por meio destes modelos identifica-se também os melhores resultados descritos na literatura.

Tabela 1. Principais Técnicas Utilizadas na Complementação de Código

Referência	Técnica	Sigla
[Rahman et al. 2020b]	Long Short Term Memory	LSTM
[Rahman et al. 2020a]	Long Short Term Memory com mecanismo de atenção	LSTM-AM
[D'Souza et al. 2016]	Best Matching Object	BMO
[Das 2015]	Feed-Forward com mecanismos de atenção	FFNN-AM
[Chen et al. 2019]	Tree-based Long Short-Term Memory Networks	Tree-LSTM
[Svyatkovskiy et al. 2019]	Long Short Term Memory	LSTM
[Alon et al. 2019]	Long Short Term Memory	LSTM

Tendo em vista os resultados favoráveis dos trabalhos empregando redes neurais

recorrentes na complementação de código, considerou-se esta abordagem como a mais promissora. Por esta razão, ela constituiu o foco deste artigo, sendo avaliada para fins de uma tarefa de complementação de código.

3. Conceituação das Redes Neurais Recorrentes

O ML compreende a construção de modelos que possuam a capacidade de representar o conhecimento a partir de um conjunto de dados (*dataset*). O processo de construção de modelos a partir de dados é apresentado na (Figura 2). Possuindo as etapas de seleção, pré-processamento e transformação de dados, aplicação da técnica de ML e por fim a interpretação do conhecimento. A seleção consiste na escolha dos dados. O pré-processamento consiste nas etapas para preparar os dados na etapa de seleção, por exemplo, eliminação de ruídos ou outro filtro. A transformação consiste na manipulação dos dados que vai produzir um *dataset*. A aplicação consiste na escolha da técnica de aprendizado em razão da natureza do problema a ser resolvido. E, por fim, a interpretação e avaliação dos resultados [Fayyad et al. 1996].

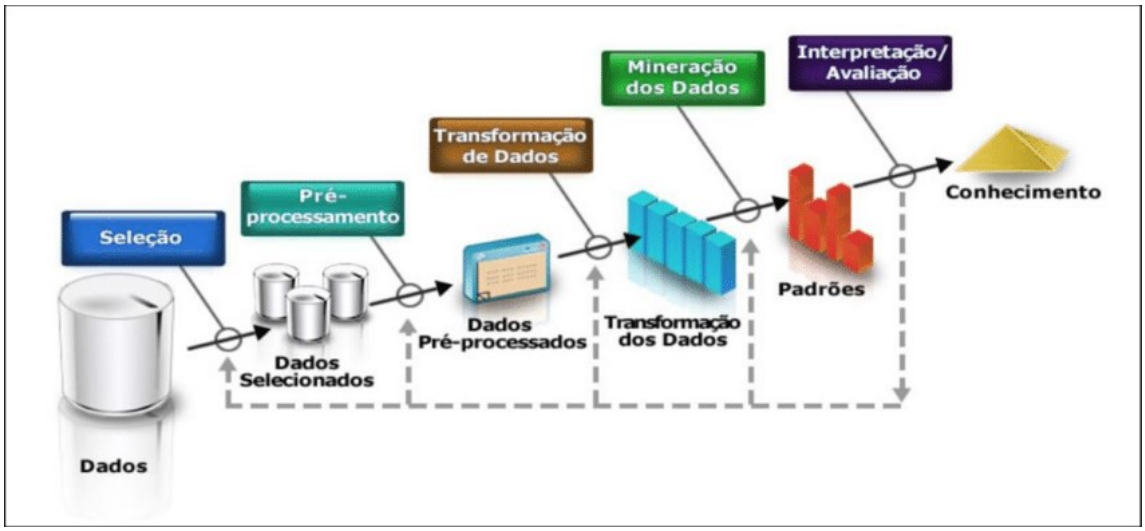

Figura 2. Processo de *Machine Learning*

Já redes neurais artificiais são técnicas computacionais que apresentam um modelo matemático inspirado no mecanismo biológico dos neurônios do cérebro humano e que adquirem conhecimento através da experiência. Elas compõem o corpo de métodos da área de *deep learning*, sendo o *deep learning* uma subárea do ML. O *deep learning* é uma evolução das redes neurais [Academy 2021]. Essa evolução parte da aplicação de um alto número de camadas ocultas para a rede neural. A consequência direta da adição de mais camadas é a criação de camadas sem relação direta entre a entrada e a saída. Assim, a cada camada adicionada espera-se que a relação entre entrada e saída seja mais complexa. Permitindo assim, a criação de neurônios que reconhecem características específicas de um determinado problema. Existem diversas arquiteturas de redes neurais, tais como: *feedforward*, recorrentes, auto-organizáveis entre outras [Haykin 2007]. Sendo utilizadas conforme sua aplicação. Entre as aplicações de redes neurais encontram-se reconhecimento de padrões, processamento de linguagem natural, reconhecimento de sentimentos, entre outras.

As redes neurais recorrentes, ao contrário das redes neurais tradicionais, além de processar dados da entrada atual, processam as entradas anteriores ao longo do tempo. Para exemplificar, pode-se ver o trabalho desses algoritmos em aplicativos de teclados virtuais. À medida que ocorrem as digitações, o sistema tenta prever as próximas palavras. As redes neurais recorrentes são essencialmente uma sequência de redes neurais que reintegram novas informações umas das outras nas suas entradas conforme representado na Figura 3. Na figura, cada célula recorrente (s) recebe novas entradas de dados (x) e o estado do instante anterior, gerando assim um célula de saída (o). Por exemplo, se a sequência que nos interessa é uma sentença de 5 palavras, a rede seria desdobrada em uma rede neural de 5 camadas, uma camada para cada palavra. No entanto, se o uso de uma sequência de informações de um *dataset* for muito longa, o algoritmo terá dificuldades para o transporte de informações entre camadas. Assim, para a análise de sequências longas utilizando redes neurais recorrentes, há o risco que algumas partes relevantes sejam descartadas, perdendo informações que podem alterar o resultado e interpretação, [Academy 2021].

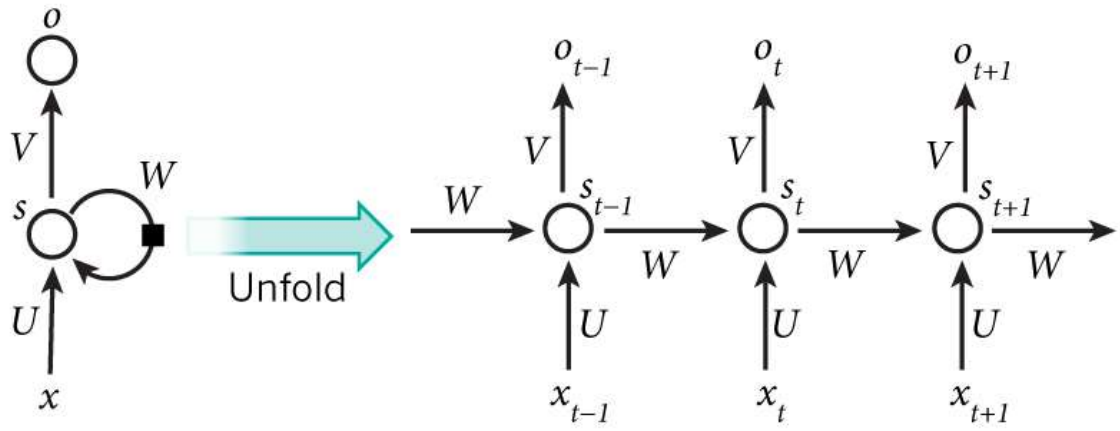

Figura 3. Ilustração de uma Rede Neural Recorrente sendo Desdobrada

Já as redes neurais Long Short Term Memory (LSTM) são uma variação de redes neurais recorrentes. Elas possuem a capacidade de aprender dependências de longo prazo, ao contrário de redes neurais recorrentes comuns. A célula de uma rede neural LSTM aplicam mecanismos denominados portões internos para regular o fluxo de informações. Tais mecanismos são divididos em *forget gate*, *input gate* e *output gate*. O *forget gate* toma decisões sobre quais partes da célula ainda são relevantes ao contexto. O *input gate*, por sua vez, decide quais afirmações de longo prazo devem ser adicionadas ao estado da célula. E por fim o *output gate* que decide, quais informações do estado atual da célula devem ser apresentadas na saída. A Figura 4 ilustra a comparação entre uma célula de uma rede neural recorrente comum e uma célula de uma rede neural LSTM. Para exemplificar o modelo preditivo, considera-se um experimento que busca prever as próximas palavras de um texto. Na frase ”(...) eu trabalho com redes neurais com meu professor, (?) ”, ao ser digitado a palavra ”professor”os passos dos portões podem ser traduzidos em: Esquecer o sujeito ”eu”(forget gate); Lembrar o sujeito ”meu professor”(input gate); E utilizar a célula de estado para lembrar com o que o professor trabalha (output gate).

Redes neurais LSTM apresentam variações de forma estrutural, tais como, Bi-

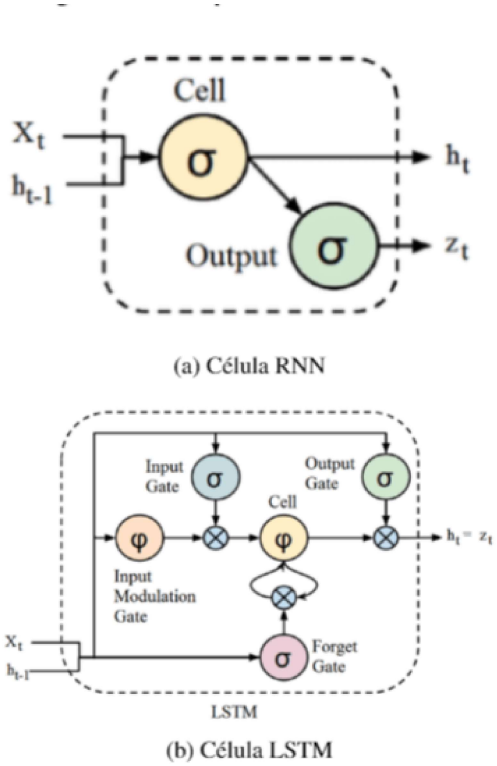

Figura 4. Comparativo RNN x LSTM.

directional Long Short Term Memory (BiLSTM) e Tree-based Long Short Term Memory (Tree-LSTM). A variação bidirecional utiliza duas redes LSTM para treinamento. A primeira utiliza a sequência de entrada de forma normal, enquanto a segunda utiliza a sequência de entrada de forma invertida, podendo com isto fornecer contexto adicional para o treinamento. A adição de contexto pode permitir um treinamento mais rápido e preciso, segundo [Brownlee 2021]. Por outro lado, a variação Tree-LSTM organiza a rede neural na topologia de árvore. Esta organização permite a uma rede neural relacionar e analisar dependências entre as informações [Tai et al. 2015]. Aplicações de Tree-LSTM incluem classificação de sentimento e relacionamento semântico.

As redes neurais Gated Recurrent Unit (GRU) são outra variação das redes neurais recorrentes. Semelhante a LSTM, as redes GRU visam resolver o problema da dissipação do gradiente, amenizando assim, o esquecimento do contexto para sequências longas que redes neurais recorrentes comuns enfrentam. Utiliza-se dois mecanismos denominados portões para gerenciar as informações da célula. O *reset game* determina quais informações são esquecidas pela célula. Já o *update gate* determina quais informações devem ser descartadas e quais devem gerar a nova saída.

A Figura 5 apresenta uma forma alternativa de ilustração de uma célula LSTM e de outra célula GRU. As funções relativas aos portões, operações ponto a ponto e concatenação de vetores, para cada variação de rede neural recorrente, também são apresentadas na imagem. De forma geral, as redes neurais GRU possuem tempo de treinamento menor que as redes neurais LSTM [Phi 2018]. A partir de testes e experimentos com os dados é que pode-se inferir qual modelo de rede recorrente é o mais adaptado a

cada contexto.

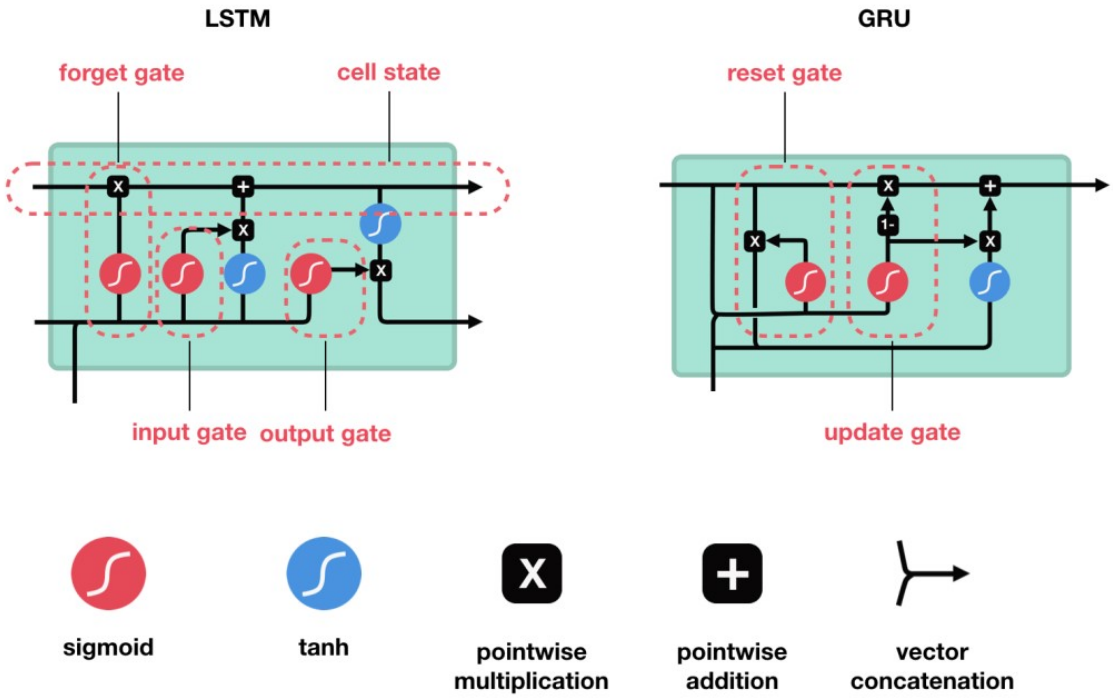

Figura 5. Ilustração de uma Célula LSTM e GRU.

4. Materiais e Método

Os recursos e processos utilizados para a construção de modelos para a complementação de código são descritos nas subseções seguintes. A subseção 4.1 apresenta os materiais e recursos utilizados. Já a subseção 4.2 apresenta o método utilizado para construção e avaliação dos modelos.

4.1. Materiais

Python [4] é uma linguagem de código livre, com suporte a múltiplos paradigmas. Ela pode ser aplicada em diferentes plataformas. Possui uma indentação caracterizada pelo uso de tabulação ou espaços, facilitando a identificação dos blocos de códigos bem como seu nível hierárquico. Tal forma facilita sua leitura e possíveis manutenções que tenham de ser realizadas ao longo da construção código. Outra característica marcante é sua ampla coleção de bibliotecas nativas, contendo métodos e funções básicas que atendem desde o acesso a bancos de dados até a interface gráficas de interação com o usuário. Além do suporte a bibliotecas de amplo uso *Python* conta com um amplo repositório de bibliotecas para ML e *deep learning* (*TensorFlow*[5], Pandas[6], *Keras*[7], *NumPy*[8], entre outras) [Santana 2019]. Graças a estas características, *Python* foi escolhida como linguagem de programação a ser utilizada na construção dos modelos avaliados.

[4]Disponível em https://www.anaconda.org
[5]Disponível em https://www.tensorflow.org
[6]Disponível em https://pandas.pydata.org
[7]Disponível em https://keras.io
[8]Disponível em https://numpy.org

O *TensorFlow* é uma plataforma completa de código aberto, destinada para tarefas de ML e e *deep learning*. Desenvolvido inicialmente pela *Google Brain*[9] a plataforma possui um ecossistema abrangente e flexível de ferramentas, bibliotecas e recursos da comunidade que permite aos pesquisadores levar adiante ML de última geração e a implementação de aplicativos com a mesma. Apresentando suporte à implementação de redes neurais. Com a atenção voltada para ao *deep learning*, a biblioteca *Keras*, tem suas aplicações desenvolvidas para trabalhar com redes neurais, com suporte a execução na plataforma *TensorFlow*. Desenvolvida inicialmente com o objetivo de permitir experimentos rápidos com os algoritmos de *deep learning*. Além disso, a biblioteca contém as diretivas necessárias para realizar uma prototipagem rápida e fácil, suporte a Convolutional Neural Network (CNN) e LSTM. *Gensim*[10] é uma biblioteca *Python* de código aberto gratuita para representar documentos como vetores semânticos. Possuindo suporte a abordagem de *word2vec*.

O *Colaboratory* ou *Colab* é um produto do *Google Research* que permite a execução de código *Python* arbitrário pelo navegador sendo adequado para a construção de modelos de ML, análise de dados e educação. Baseado no *Jupyter*[11] permite a criação e compartilhamento de notebooks em conjunto com a sua facilidade de configuração. Por estas razões foi escolhido como ambiente de execução. A Figura 6 apresenta uma ilustração do ambiente e recurso utilizados. Na imagem são apresentados 2 blocos. O primeiro bloco corresponde a implementação do modelo utilizando para isso recursos como o *Keras* e *TensorFlow*. Já o segundo bloco apresenta valores referentes ao treinamento de um modelo construído.

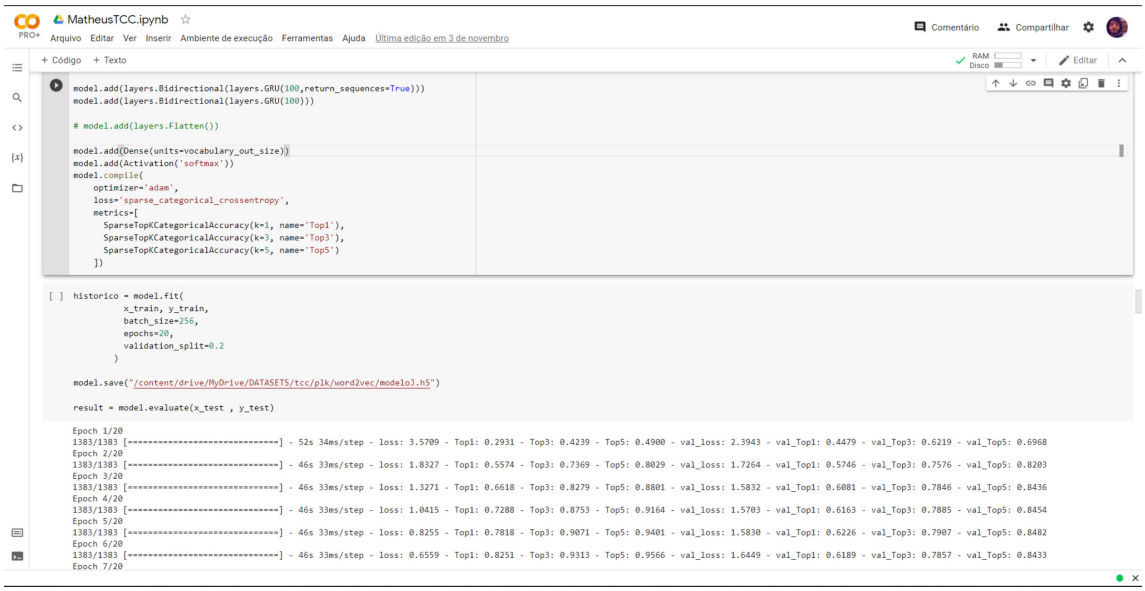

Figura 6. Exemplo de Notebook no Colab

[9]Informações sobre, disponível em: https://research.google/teams/brain
[10]Disponível em https://radimrehurek.com/gensi/
[11]Disponível em https://jupyter.org/

4.2. Método

O trabalho é uma pesquisa de natureza exploratória, a qual visa identificar e avaliar técnicas de complementação de código baseadas em redes neurais aplicadas a linguagem de programação *Python*. Para isso, o trabalho foi desenvolvido em cinco etapas sendo elas: (1) construção do *dataset*, (2) pré-processamento e transformação dos dados, (3) implementação do algoritmo para redes neurais recorrentes, (4) comparação e avaliação dos modelos gerados e, por último, (5) seleção do melhor modelo.

A construção do *dataset* iniciou-se pela utilização de um *dataset*[12] já existente sendo formado pelas Árvore sintática abstrata (AST) de programas *Python* coletados de repositórios GitHub com a remoção de arquivos duplicados, bifurcações de projeto, limitado a ASTs de no máximo 30.000 nós em conjunto com licenças permissivas e não virais. Com isso, obteve-se um *dataset* de 150.000 arquivos analisados, sendo divididos em 100.000 instâncias para treinamento e 50.000 instâncias para teste. A Figura 7 exemplifica o processo de conversão, onde cada nodo da AST corresponde a um objeto com as propriedades *type*, *value* e *children*.

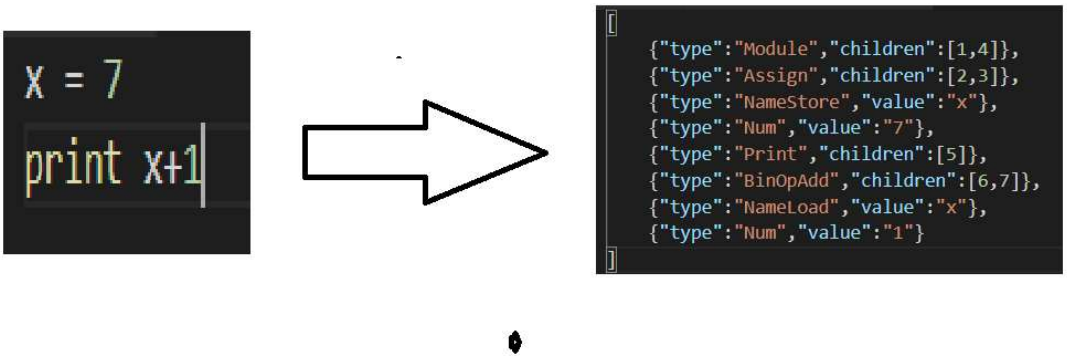

Figura 7. Conversão de Código para AST.

O pré-processamento e transformação dos dados consistiu-se na transformação das ASTs em sequências treináveis. O pré-processamento iniciou-se pela identificação dos *tokens* a partir das ASTs, sendo considerado como um *token* a propriedade *value* de um nodo. Caso não exista essa propriedade foi considerada a propriedade *type* do nodo como *token*. Após essa etapa, realizou-se uma busca em profundidade pelos nodos e, quando identificado um nodo com o *type* referente a carga de atributos(*attributeload*) iniciou-se a montagem de uma sequência de treinamento. A sequência é formada por T *tokens* anteriores ao *token* de carga de atributo, sendo T um número parametrizável. O resultado dessa sequência é dado pelo próximo *token* do tipo *nameload*. Este processo foi realizado nos *datasets* de treino e teste.

Iniciou-se a transformação dos dados pela construção dos vocabulários, sendo divididos em dois. O primeiro dos vocabulários foi constituído pelas sequências de *tokens* anteriores ao *token* de carga (incluindo este), aplicando a transformação para valores numéricos utilizando o método de *Word2Vec* da biblioteca gensim. Foi considerado somente o *token* com frequência igual ou superior a 5 (identificado por meio de testes). Já

[12]Disponível em https://www.sri.inf.ethz.ch/py150

o vocabulário de saída foi constituído pelos *tokens* posteriores ao *token* de carga, sendo mapeado para valores numéricos utilizando a biblioteca Tokenizer do Keras, tendo sido considerado somente *tokens* com frequência maior ou igual a 250 (identificado por meio de testes).

Definiu-se um conjunto de camadas a serem testadas para compor os modelos de redes neurais recorrentes. As camadas inicialmente escolhidas foram: entrada de dados, processamento dos *tokens* (*Embedding*), redes neurais recorrentes (LSTM, BiLSTM, GRU e BidirectionalGated Recurrent Unit (BiGRU)) e classificação do resultado. A camada de entrada e processamento dos *tokens* possui a tarefa de converter as sequências de *tokens* mapeados para vetores densos, de tamanho pré-determinado. Utiliza-se nesta tarefa a classe *Embedding*[13] em conjunto com valores do modelo de *word2vec*, construído na etapa de pré-processamento. Já a camada da rede neural, pode incluir uma ou mais redes LSTM, GRU, entre outras. E, por fim, a camada de saída classifica o resultado para um dos *tokens* mapeados no vocabulário de saída, sendo portanto uma implementação densamente conectada com função de ativação. A Figura 8 exemplifica um modelo possível de rede, seguindo a arquitetura mencionada.

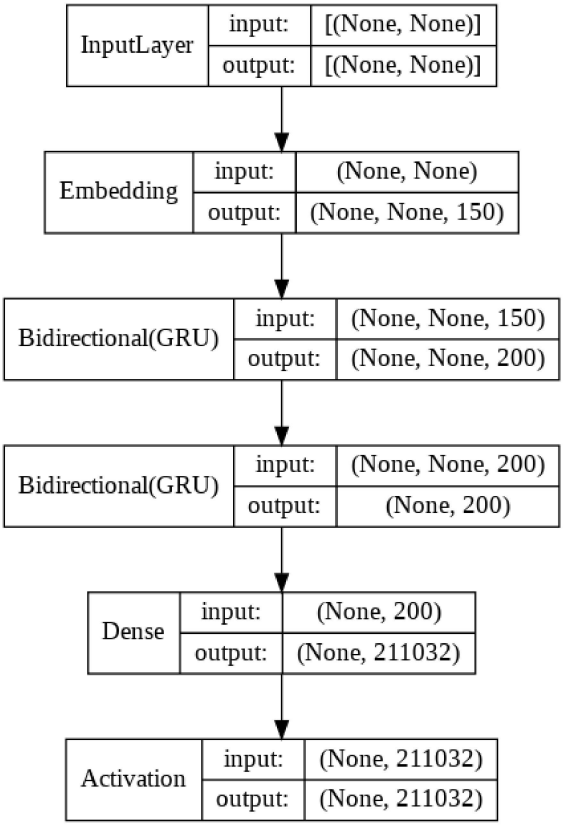

Figura 8. Modelo de Rede Neural para Complementação de Código.

Considerando as possibilidades de configurações de uma arquitetura *deep learning* para o problema da complementação de código, tem-se um espaço multidimensional de valores para os atributos. Explorar este espaço a fim de encontrar a melhor configuração

[13]https://keras.io/api/layers/core_layers/embedding/

Tabela 2. Hiperparâmetros Utilizados na Construção dos Modelos.

Hiperparâmetro	Valores Avaliados	Melhor Valor
Otimizador	Adam	Adam
Função de perda	Sparse categorical crossentropy, Categorical crossentropy,	Sparse categorical crossentropy
Frequência mínima para vocabulário de saída	100, 150, 250, 500	250
Frequência mínima para vocabulário de entrada	1, 5, 10, 50	5
Tamanho do batch	64, 128, 256, 512	256
Número de unidades ocultas por camada	100, 200	100
Dimensão do vetor de embedding	150, 300	150
Número de camadas de rede neural recorrente	2, 3, 5	2

é uma tarefa trabalhosa e computacionalmente cara. Para isso, utiliza-se o conceito de hiperparâmetros, que especificam arquiteturas e parametrizações a serem avaliadas (espaço de soluções). Em conjunto com a estrutura de camadas exemplificadas anteriormente foram identificados como melhores hiperparâmetros aqueles apresentados na Tabela 2. A aferição desses hiperparâmetros se deu através da experimentação de diferentes conjuntos e valores.

Como métricas de avaliação, utilizou-se as mesmas observadas em trabalhos relacionados ([Svyatkovskiy et al. 2019]). Foram calculadas a precisão para a lista de recomendações de *k* elementos, denominada *top-k*, como representada na fórmula 1. Tem-se $N_{\text{top-k}}$ denotando as recomendações relevantes dentro do conjunto *top-k* e Q denotando o número total de testes. Para este cálculo foi utilizada a classe *SparseTopKCategoricalAccuracy*[14], que calcula os valores de Top-1, Top-3 e Top-5 para os modelos avaliados.

$$Acc(k) = \frac{N_{\text{top-k}}}{Q} \tag{1}$$

Por fim, a comparação entre os modelos se deu pelo valor de *top-k* do *dataset* de avaliação, considerando uma lista com 5 recomendações. Os resultados finais são descritos na próxima seção.

5. Resultados

Seguindo o método proposto foram avaliados modelos de complementação de código utilizando as redes neurais LSTM[15], GRU[16], BiLSTM e BiGRU[17] e Recurrent Neural

[14]https://keras.io/api/metrics/accuracy_metrics/#sparsetopkcategoricalaccuracy-class
[15]https://keras.io/api/layers/recurrent_layers/lstm
[16]https://keras.io/api/layers/recurrent_layers/gru/
[17]https://keras.io/api/layers/recurrent_layers/bidirectional

Network (RNN)[18]. Para esta avaliação foram considerados como dados de treinamento 442378 sequências, construídas nas etapas de pré-processamento e transformação dos dados. E como dados de testes foram utilizados 215194 sequências construídas nas etapas anteriores. A Tabela 3 apresenta os resultados alcançados pelos modelos, considerando as métricas Top-1, Top-3 e Top-5.

Tabela 3. Acurácia para Modelos Construídos

Modelo	Top-1 acurácia	Top-3 acurácia	Top-5 acurácia
RNN	0,13	0,26	0,30
GRU	0,55	0,70	0,76
LSTM	0,57	0,71	0,77
BiLSTM	0,57	0,72	0,78
BiGRU	0,59	0,74	0,80

Entre os modelos construídos para a avaliação. O modelo utilizando as redes BiGRU apresentou melhores resultados, para fins de identificação este modelo será denominado Modelo A. A Figura 9 apresenta a estrutura do Modelo A, bem como a quantidade de parâmetros utilizados em suas camadas.

```
Model: "Modelo A"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, None, 150)         82569150

 bidirectional_4 (Bidirectio  (None, None, 200)         151200
 nal)

 bidirectional_5 (Bidirectio  (None, 200)               181200
 nal)

 dense_2 (Dense)              (None, 211032)            42417432

 activation_2 (Activation)    (None, 211032)            0

=================================================================
Total params: 125,318,982
Trainable params: 125,318,982
Non-trainable params: 0
_________________________________________________________________
```

Figura 9. *Summary* do Modelo A

A Tabela 4 apresenta um comparativo entre resultados alcançados pelo Modelo A

[18]https://keras.io/api/layers/recurrent_layers/simple_rnn/

e resultados alcançados por modelos mapeados na revisão sistemática. Sendo considerado como métricas a acuracidade no Top-1, Top-3 e Top-5.

Tabela 4. Comparativo entre o Modelo A e modelos mapeados na revisão sistemática.

Método	Top-1 acurácia	Top-3 acurácia	Top-5 acurácia
Alphabetic	0,36	-	0,47
Frequency	0,38	-	0,64
Frequency-if	0,40	-	0,67
Modelo A	0,59	0,74	0,80
Markov Chain	0,58	-	0,83
LSTM [Svyatkovskiy et al. 2019]	0,71	-	0,92

Sendo os métodos mapeados na revisão os seguintes: *Alphabetic* método baseado na ordenação em ordem alfabética; *Frequency* e *Frequency-if* métodos baseados na frequência em que uma recomendação aconteceu anteriormente no corpus de treinamento; *Markov Chain* método baseado na probabilidade de estados anteriores; e, por fim, modelos baseado em redes neurais.

6. Conclusão

Neste trabalho foram construídos e avaliados sistemas preditivos partindo de métodos *machine learning* e *deep learning* com ênfase em métodos de redes neurais recorrentes. Segue-se a aplicação desses métodos para problemas de complementação de código para a linguagem de programação *Python*, sendo esse o objetivo principal do trabalho.

A utilização de redes neurais recorrentes para construção de sistemas voltados à complementação de código nos modelos construídos apresentou um aumento na assertividade da complementação em comparativo a modelos não preditivos. Esse aumento na assertividade em conjunto com resultados alcançados na literatura são indicadores positivos para a utilização das redes neurais recorrentes em tarefas de complementação de código.

Durante a construção dos modelos, o elevado tempo de treinamento mostrou-se um fator adverso. Melhorias futuras para o trabalho devem visar a otimização do tempo de treinamento, a diversificação e enriquecimento dos dados utilizados no treinamento e a construção de novos modelos aproveitando o conhecimento construído.

Para finalizar, considera-se que o objetivo geral do trabalho foi alcançado. Partiu-se de um estudo exploratório e desenvolveu-se um modelo de aprendizado de máquina que produz recomendações para a complementação de código na linguagem de programação *Python*. Como trabalhos futuros indica-se o uso de métodos de aprendizado por transferência para ampliar a capacidade do modelo.

Referências

Academy, D. S. (2021). *Deep learning book*. Data Science Academy.

Alon, U., Sadaka, R., Levy, O., and Yahav, E. (2019). Structural language models for any-code generation. *CoRR*, abs/1910.00577.

Brownlee, J. (2021). *How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras*. Machine Learning Mastery.

Chen, C., Peng, X., Sun, J., Xing, Z., Wang, X., Zhao, Y., Zhang, H., and Zhao, W. (2019). Generative api usage code recommendation with parameter concretization. *Science China Information Sciences*, 62(9):192103.

Das, S. (2015). Contextual code completion using machine learning.

D'Souza, A. R., Yang, D., and Lopes, C. V. (2016). Collective intelligence for smarter API recommendations in python. *CoRR*, abs/1608.08736.

Fayyad, U., Piatffsky-Shapiro, G., and Smyth, P. (1996). The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27 – 34.

Haykin, S. (2007). *Redes Neurais: Princípios e Prática*. Artmed.

Hou, D. and Pletcher, D. M. (2011). An evaluation of the strategies of sorting, filtering, and grouping api methods for code completion. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 233–242.

Luo, C. (2017). A report on automatic code completion.

Okoli, C., Duarte, T. p. W. A., and Mattar, R. t. e. i. (2019). Guia para realizar uma revisão sistemática de literatura. *EaD em Foco*, 9(1).

Phi, M. (2018). *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. Towards Data Science.

Rahman, M., Watanobe, Y., and Nakamura, K. (2020a). Source code assessment and classification based on estimated error probability using attentive lstm language model and its application in programming education. *Applied Sciences*, 10:2973.

Rahman, M. M., Watanobe, Y., and Nakamura, K. (2020b). A neural network based intelligent support model for program code completion. *Scientific Programming*, 2020:7426461.

Santana, F. (2019). *Por que o Python é a Linguagem mais adotada na área de Data Science*.

Svyatkovskiy, A., Zhao, Y., Fu, S., and Sundaresan, N. (2019). Pythia: Ai-assisted code completion system. *CoRR*, abs/1912.00742.

Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks.