Um estudo preliminar sobre o uso de uma arquitetura deep learning para seleção de respostas no problema de recuperação de código-fonte

Marcelo de Rezende Martins[1], **Marco Aurélio Gerosa**[2]

[1]Instituto de Pesquisas Tecnológicas (IPT)
São Paulo – SP – Brazil

[2]Northern Arizona University
Flagstaff, AZ, US

rezende.martins@gmail.com, Marco.Gerosa@nau.edu

Abstract. *Code retrieval techniques aim to select a code snippet to solve a question given a set of possible solutions. This article presents a preliminary study about a new approach for code retrieval, applying an answer selection deep learning architecture. We present the preliminary results of a bidirectional LSTM with convolutional network model adapted for code retrieval. After 20 runs on a evaluation dataset, our model could achieve a mean MRR of* 0.60 ± 0.02 *and a top-1 accuracy up to 51%.*

Resumo. *Dado uma questão e um conjunto de trechos de código-fonte, recuperação de código-fonte ou code retrieval busca encontrar o código que soluciona a dada questão. Este artigo apresenta um estudo preliminar sobre uma nova abordagem para o problema de recuperação de código-fonte, utilizando uma arquitetura deep learning de seleção de respostas. Apresentamos os resultados preliminares do modelo de redes neurais recorrentes bidirecionais (bi-LSTM) com redes neurais convolucionais (CNN) adaptado para o problema de recuperação de código-fonte. Após 20 execuções na amostra de teste, nosso modelo conseguiu atingir uma média MRR de* $0,60 \pm 0,02$ *e a medida top-1 o máximo de 51%.*

1. Introdução

Segundo [Allamanis et al. 2015], recuperação de código-fonte (*code retrieval*) é um problema de recuperação de informação, onde dado uma questão ou uma descrição em linguagem natural e um conjunto de possíveis trechos de código-fonte, o objetivo é recuperar o trecho de código-fonte que solucione a questão ou seja mais relevante de acordo com a descrição. Já segundo [Lai et al. 2018], dado uma questão e um conjunto de possíveis respostas, ambos em linguagem natural, seleção de respostas ou *answer selection* busca identificar qual resposta consegue responder a pergunta corretamente.

O problema do *code retrieval* busca associar um texto em linguagem natural a um código-fonte. Esta associação tem diversas aplicações como busca de código-fonte a partir de uma consulta em linguagem natural, documentação e geração de programas a partir de uma especificação, por exemplo [Allamanis et al. 2018]. Em engenharia de software, a associação entre código-fonte e texto em linguagem natural pode auxiliar na

rastreabilidade de requisitos. Além disso, pode ajudar o desenvolvedor na geração de código-fonte. Um modelo pode gerar testes de unidade a partir de uma estória de usuário.

Partindo da hipótese inicial de que software é uma forma de comunicação humana e tem propriedades estatísticas similares a corpora de linguagem natural [Allamanis et al. 2018]. Neste artigo, propusemos uma nova abordagem para o problema de *code retrieval*. Abordamos o problema sob a perspectiva do problema de *answer selection*, problema conhecido em NLP (Natural Language Processing), i.e., utilizamos soluções inicialmente propostas para um conjunto de dados formado por perguntas e respostas em linguagem natural e aplicamos em um conjunto formado por pares de perguntas e trechos de código-fonte.

Inicialmente, avaliamos o desempenho de soluções de arquiteturas deep learning comumente utilizadas no problema de *answer selection*. Optamos pela solução de deep learning, mais especificamente uma arquitetura composta por uma rede neural recorrente com uma camada de rede convolucional, devido ao bom desempenho apresentado nos dados InsuranceQA [Feng et al. 2015], um conjunto de dados de referência para avaliação de modelos em *answer selection*.

Além disso, as soluções anteriores propostas para *answer selection* tipicamente baseavam-se em técnicas de pré-processamento para extrair características relevantes para auxiliar o modelo na predição, ferramentas externas e até ferramentas linguísticas [Lai et al. 2018]. Enquanto a definição típica para deep learning é que ele aprende *deep representations*, i.e., aprende múltiplos níveis de representações e abstrações dos dados. Ele mostrou-se capaz de representar imagens, textos, dados de diferentes contextos juntos em um mesmo modelo [Zhang et al. 2019]. No nosso caso, isto é um fator importante, pois o nosso intuito é criar um modelo capaz de representar um conjunto de dados formado por textos em linguagem natural e trechos de código-fonte.

Neste artigo apresentamos os resultados preliminares de um estudo sobre o uso de arquitetura deep learning de *answer selection* no problema do *code retrieval*. Inicialmente, utilizamos a arquitetura bi-LSTM com CNN proposta por [Tan et al. 2015]. Além de propor uma nova arquitetura para este problema, avaliamos o modelo utilizando os dados de entrada da base StaQC, criada por [Yao et al. 2018]. Esta base de dados é composta de milhares de pares de perguntas e trechos de código-fonte do StackOverFlow[1].

2. Método

Conforme mencionado anteriormente, neste trabalho abordamos o problema do *code retrieval* sob a perspectiva do *answer selection*. Utilizamos a arquitetura proposta por [Tan et al. 2015], que utiliza a função de classificação *pairwise* e uma arquitetura siamesa, de acordo com [Lai et al. 2018]. Para [Lai et al. 2018], o problema de *answer selection* pode ser analisado sob duas formas: a de aprendizado e a de arquitetura.

2.1. Forma de aprendizado

O problema de *answer selection* consiste em encontrar a resposta mais relevante dado uma questão. Ele pode ser abordado como uma problema de classificação, onde o objetivo é classificar com uma pontuação melhor as respostas mais relevantes de acordo com a questão.

[1] `https://www.stackoverflow.com`

[Tan et al. 2015] utilizaram o método *pairwise*, no qual o objetivo da função é classificar as respostas corretas com uma pontuação maior que a das incorretas. Dado uma questão, o método avalia o conjunto de repostas e aprende a classificar qual resposta é mais relevante para a questão. Por exemplo, o modelo proposto por [Tan et al. 2015] e utilizado como referência neste artigo, os dados de entrada para o treinamento são triplas (q_i, c_i^+, c_i^-), onde q_i é uma questão, c_i^+ é uma resposta correta, c_i^- é uma resposta incorreta. Seja a função de perda, *hinge*, definida como:

$$L = max(0, m - h_\theta(q_i, c_i^+) + h_\theta(q_i, c_i^-)) \quad (1)$$

Onde m é a margem. Se $h_\theta(q_i, c_i^+) - h_\theta(q_i, c_i^-) < m$ então L é positivo. Quando esta condição é satisfeita, a implicação é que o sistema classifica a resposta correta abaixo da resposta incorreta, ou a questão correta pontua um pouco acima da resposta incorreta. Por outro lado, se a resposta correta tem uma pontuação maior que a incorreta por uma margem acima ou igual a m (i.e., $h_\theta(q_i, c_i^+) - h_\theta(q_i, c_i^-) \geq m$, a função de perda é igual a zero. Em resumo, a função de perda incentiva a reposta correta a ter uma pontuação maior que a incorreta por uma certa margem [Lai et al. 2018].

[Tan et al. 2015] propuseram o uso da função de similaridade *cosine*. Esta função de similaridade é comumente utilizada em problemas de *answer selection* [Feng et al. 2015].

2.2. Arquitetura

[Tan et al. 2015] propuseram uma arquitetura que utiliza uma rede neural recorrente bidirecional, mais especificamente uma rede LSTM (biLSTM) [Hochreiter and Schmidhuber 1997] e uma camada pooling para construir a representação dos vetores de entrada. Ao final, o modelo utiliza a função de similaridade *cosine* para calcular a distância entre as representações.

A arquitetura de referência que iremos utilizar acrescenta uma estrutura de rede convolucional na saída da rede bi-LSTM. Os vetores de saída da rede bi-LSTM tornam-se vetores de entrada na na estrutura CNN. O CNN auxilia na síntese da representação dos vetores. A imagem ilustrativa da arquitetura pode ser visualizada na Figura 1.

3. Experimento

Para avaliar a arquitetura proposta por [Tan et al. 2015] composta por uma rede bi-LSTM com CNN no problema do *code retrieval*, utilizamos os dados disponibilizados por [Yao et al. 2018]. Em seu trabalho, [Yao et al. 2018] coletaram mais de **147 mil** pares de perguntas e trechos de código-fonte em Python e aproximadamente **119 mil** pares de perguntas e trechos de código-fonte em SQL.

Inicialmente, utilizamos uma amostra de 62.252 pares (q_i, c_i^+) em Python. Onde q_i é o título de uma questão no site StackOverFlow e c_i^+ corresponde a um trecho de código-fonte apontado como solução para a questão q_i. Destes 62.252 pares, excluímos 2.169 pares que foram anotados manualmente. Estes 2.169 pares (q_i, c_i^+) anotados manualmente serão divididos em duas amostras: *DEV* e *EVAL*.

Conforme a Tabela 2, utilizaremos 60.083 pares (q_i, c_i^+) para treinamento. Para obter o modelo, adotamos o mesmo procedimento proposto por [Iyer et al. 2016]. No

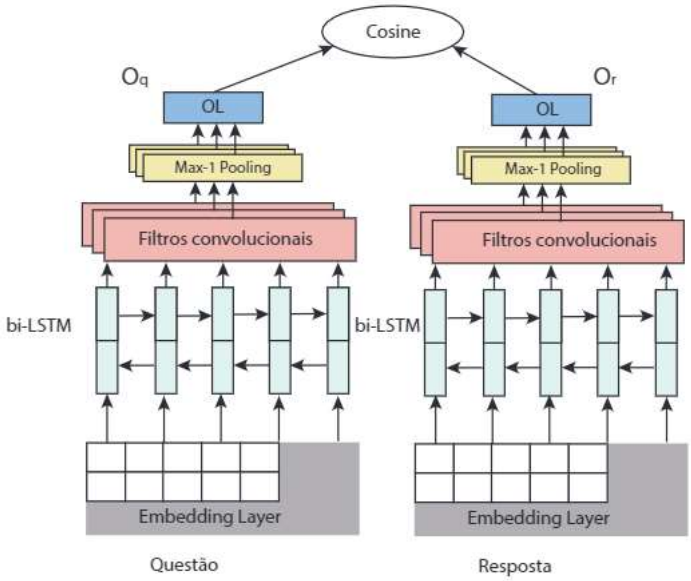

Figura 1. Figura da arquitetura proposta para o problema do *code retrieval*. Figura adaptada do [Tan et al. 2015].

Entrada	Tipo de Dado	Média	Desvio Padrão	Mínimo	25%	50%	75%	Máximo
Questão	Tamanho	51,60	18,57	13	38	49	62	150
	# de palavras	8,9	3,64	2	6	8	11	32
Código-Fonte	Tamanho	326	477	4	95	192	380	17.200
	# de palavras	48,84	65,79	0	16	31	58	3.170

Tabela 1. Estatística descritiva da amostra de 62.252 pares (q_i, c_i^+) em Python extraída do conjunto de dados disponibilizado por [Yao et al. 2018].

caso, o modelo será treinado, inicialmente, durante 80 épocas. Caso o *learning rate* fique abaixo de 0,001, o treinamento é interrompido. Ao final de cada época, o modelo é avaliado na amostra *DEV*.

A avaliação na amostra *DEV* consiste em avaliar a qualidade do modelo calculando o *Mean Reciprocal Rank* (MRR). Esta avaliação é feita da seguinte forma: Para cada par (q_i, c_i^+) na amostra *DEV*, selecionaremos outros 49 distratores $c^{'}$ da amostra de treinamento, onde $c^{'} \neq c_i^+$. Estes 50 pares (c_i, q_i) serão classificados de acordo com a função h_θ de similaridade. Posteriormente, o MRR de cada questão q_i é calculado. Ao final, calculamos a média do valor do MRR. O modelo de treinamento que obter a maior média MRR na amostra *DEV* será escolhido.

Após o término do treinamento e com o modelo com a maior média MRR na amostra *DEV* escolhido, a avaliação final é feita. Durante a avaliação final, a qualidade do modelo é avaliado na amostra *EVAL*. O procedimento é o mesmo adotado para avaliar o modelo na amostra *DEV*.

3.1. Configuração

Os dados foram representados como sequência de tokens. Utilizamos uma representação distribuída word2vec [Mikolov et al. 2013]. Diferente do *answer selection* proposto por [Tan et al. 2015] no qual ele cria apenas uma representação distribuída para a amostra

Amostras	Quantidade de (q_i, c_i^+)
Treinamento	60.083
DEV	1.085
EVAL	1.084
Total	**62.252**

Tabela 2. Divisão das amostras para treinamento, avaliação do modelo (DEV) e avaliação final (EVAL) conforme os critérios adotados por [Iyer et al. 2016].

inteira, nós geramos a representação distribuída para as questões e outra para os trechos do código-fonte.

Quanto ao código-fonte, foi feito um pré-processamento. Utilizamos a função disponibilizada por [Yao et al. 2018] que substitui literais numéricos e texto (*string*) por NUMBER e STRING. Os comentários são removidos e o nome das variáveis são substituídas por VAR.

Os parâmetros de execução foram os mesmos utilizados por [Tan et al. 2015]. Com exceção dos filtros na camada CNN, que reduzimos para o valor 100. O valor utilizado por [Tan et al. 2015] de 1.000, aumentou a capacidade do modelo, causando o *overfitting*.

4. Resultados preliminares

Os resultados do experimento podem ser visualizados na Tabela 3. Os resultados correspondem a média do MRR após 20 rodadas de execução utilizando o melhor modelo obtido a partir do treinamento. E o modelo foi avaliado na amostra *EVAL*.

Nestes resultados preliminares, comparamos o modelo bi-LSTM com CNN com outros dois modelos. O modelo *Embedding*, mais simples, é composto apenas por uma camada com a representação distribuída das questões e trechos de código-fonte. A saída é uma camada de *maxpool* e ao final a similaridade é calculada através da função *cosine*.

O modelo *CNN* é uma rede neural convolucional com uma camada *hidden* de entrada *HL*. Esta camada *hidden* é definida como $z = tanh(Wx + B)$. Onde W é a matriz de pesos; B é o vetor *bias*; x é o vetor de entrada; z é o resultado da função de ativação $tanh$. Este modelo utiliza também o *maxpool* e a mesma função de similaridade.

Imagens ilustrativas das arquiteturas dos modelos *Embedding* e *CNN* podem ser visualizadas nas Figura 2 e Figura 3, respectivamente.

A implementação dos modelos foi feito utilizando a biblioteca Keras. O código de implementação e o resultados preliminares estão disponíveis no repositório Git `https://github.com/mrezende/keras-language-modeling`. Além disso, os dados pré-processados podem ser visualizados no repositório `https://github.com/mrezende/stack_over_flow_python`.

De acordo com a Tabela 3, **bi-LSTM-CNN** obteve o melhor desempenho. Porém, CNN obteve um resultado muito próximo e o tempo de treinamento é muito menor. O tempo total de treinamento da rede bi-LSTM com CNN levou em torno de 48 minutos, utilizando uma GPU Tesla K80. Enquanto a arquitetura CNN levou em torno de 6s.

Em todos os modelos, utilizamos uma camada de *maxpool* e a função de simila-

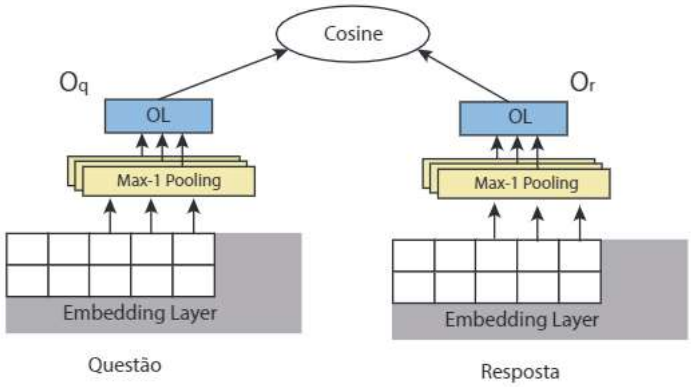

Figura 2. Figura da arquitetura *Embedding*. Figura adaptada do [Tan et al. 2015].

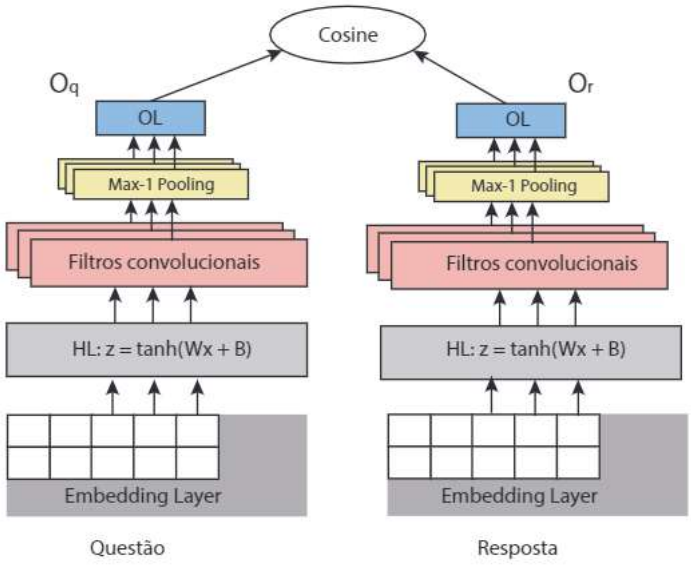

Figura 3. Figura da arquitetura *CNN*. Figura adaptada do [Tan et al. 2015].

ridade *cosine*. Utilizamos o valor de margem $0,009$ para a função de perda *hinge*, valor proposto por [Feng et al. 2015].

4.1. Ameaças à validade

Os dados utilizados no treinamento para obter o modelo final foram coletados automaticamente por [Yao et al. 2018]. [Yao et al. 2018] criaram um modelo composto por uma rede neural recorrente para obter os pares de questões e trechos de código-fonte automaticamente. E para treinar este modelo, foram utilizados os dados anotados manualmente. Estes dados anotados manualmente (amostras *DEV* e *EVAL*) são os mesmos utilizados na avaliação do nosso modelo.

Para minimizar o viés, utilizamos o mesmo procedimento de avaliação proposto por [Iyer et al. 2016]. Para cada par de (q_i, c_i^+), selecionamos aleatoriamente 49 distratores $c^{'}$ da amostra de treinamento, onde $c^{'} \neq c_i$.

Modelos	Resultados (MRR)
Embedding	$0,52 \pm 0,01$
CNN	$0,58 \pm 0,01$
bi-LSTM-CNN	**0,60 ± 0,02**

Tabela 3. Resultado preliminar do modelo bi-LSTM-CNN proposto em comparação a outros dois modelos (CNN e Embedding). Estes resultados foram obtidos a partir da amostra EVAL.

5. Conclusões

Neste trabalho, propusemos uma abordagem diferente para o problema do *code retrieval*. Nosso intuito foi abordar o problema do *code retrieval* sob a perspectiva do problema *answer selection*, já conhecido em NLP. E dado o bom desempenho dos modelos deep learning no contexto do *answer selection*, utilizamos o modelo proposto por [Tan et al. 2015].

Conforme apresentado na Seção 4, o modelo proposto por [Tan et al. 2015] apresentou um bom desempenho. O valor médio de MRR de $0,60 \pm 0,02$ é um valor expressivo. Este resultado preliminar serve como um indicativo para aprimorarmos o modelo para o problema do *code retrieval*. [Yao et al. 2018] obtiveram uma média MRR de $0,57 \pm 0,02$ e [Iyer et al. 2016] obtiveram $0,44 \pm 0,01$, porém numa amostra de pares de questões e códigos-fontes em SQL disponibilizada por [Iyer et al. 2016]. O próximo passo é avaliarmos a nossa arquitetura utilizando os mesmos dados e procedimentos proposto por [Yao et al. 2018].

Uma frente ainda a ser explorada no problema do *code retrieval* é a disponibilização de dados para avaliação dos modelos. A área de reconhecimento de imagem tem o *ImageNet* [Deng et al. 2009] um vasto banco de dados com mais de 14 milhões de imagens anotados manualmente. A área de inferência em linguagem natural para determinar se uma hipótese é verdadeira, falsa ou neutra contém mais de 570 mil sentenças em inglês anotadas manualmente [Bowman et al. 2015].

O próprio problema de *answer selection* utiliza Trec-QA [Wang et al. 2007] e InsuranceQA [Feng et al. 2015]. O trabalho de [Yao et al. 2018] anotou 4.884 pares de questões e trechos de código-fonte manualmente. Isto é um passo importante e conforme mais dados curados e organizados forem disponibilizados, mais a área de *machine learning* aplicado a código-fonte e engenharia de software tende a ganhar.

Referências

Allamanis, M., Barr, E. T., Devanbu, P., and Sutton, C. (2018). A survey of machine learning for big code and naturalness. *ACM Comput. Surv.*, 51(4):81:1–81:37.

Allamanis, M., Tarlow, D., Gordon, A. D., and Wei, Y. (2015). Bimodal modelling of source code and natural language. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2123–2132. JMLR.org.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Confe-*

rence on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Feng, M., Xiang, B., Glass, M. R., Wang, L., and Zhou, B. (2015). Applying deep learning to answer selection: A study and an open task. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 813–820.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Iyer, S., Konstas, I., Cheung, A., and Zettlemoyer, L. (2016). Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083, Berlin, Germany. Association for Computational Linguistics.

Lai, T. M., Bui, T., and Li, S. (2018). A review on deep learning techniques applied to answer selection. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2132–2144, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Tan, M., dos Santos, C., Xiang, B., and Zhou, B. (2015). Lstm-based deep learning models for non-factoid answer selection. *CoRR*, abs/1511.04108.

Wang, M., Smith, N. A., and Mitamura, T. (2007). What is the Jeopardy model? a quasi-synchronous grammar for QA. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 22–32, Prague, Czech Republic. Association for Computational Linguistics.

Yao, Z., Weld, D. S., Chen, W.-P., and Sun, H. (2018). Staqc: A systematically mined question-code dataset from stack overflow. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 1693–1703, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Zhang, S., Yao, L., Sun, A., and Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.*, 52(1):5:1–5:38.