Grafos do Código Fonte como Representação de Programas

Maria Fernanda Almeida Oliveira (PIC/CNPq/FA/Uem), Anderson Faustino da Silva (Orientador), e-mail: ra118597@uem.br.

Universidade Estadual de Maringá / Centro de Tecnologia / Maringá, PR.

Ciência da Computação / Sistema de Computação

Palavras-chave: programa, aprendizagem de máquina, grafos.

Resumo

Geração de código requer que o compilador extraia informações do código fonte de forma a determinar as ações que devem ser consideradas durante tal processo. Atualmente, os sistemas inteligentes de geração de código, empregam estratégias de aprendizagem de máquina de forma a adicionar conhecimento ao sistema e assim melhorar a tomada de decisão. Tais sistemas requerem que o conhecimento seja representado de alguma maneira, neste contexto, a questão fundamental é como representar programas. Este projeto visa analisar diferentes representações de programa, baseadas em grafo. Como consequência, identificar qual representação baseada em grafo fornece a maior acurácia ao sistema.

Introdução

A geração de código é um processo realizado por compiladores (Cooper e Torczon, 2016). Neste contexto, sistemas inteligentes podem ser usados para selecionar transformações e, consequentemente, aplicá-las ao programa sendo compilado.

Sistemas que utilizam aprendizagem de máquina criam um modelo de previsão, na fase de treinamento, baseado no comportamento dos programas de treino, e posteriormente, utiliza tal modelo de previsão, na fase de teste, para encontrar o melhor conjunto de transformações para um programa ainda não visto pelo sistema (Lima e Outros, 2013; Long e O'Boyle, 2004; Agakov e Outros, 2006). Tais sistemas utilizam diferentes representações para caracterizar o conhecimento, em outras palavras, para caracterizar programas.

Vale destacar que antes de estabelecer um sistema de geração de código, baseado em aprendizagem de máquina, é necessário estabelecer como o conhecimento será representado. Neste contexto, este projeto visa analisar diferentes representações baseadas nas relações existentes entre objetos, mais especificamente, em grafo; os quais representam as relações entre as instruções do código fonte do programa. Portanto, o interesse deste projeto é avaliar representações baseadas em grafo, sendo este extraído do código fonte do programa.

Materiais e Métodos

O processo de extração de grafos, presente neste projeto, tem por objetivo representar as relações entre as instruções de programas escritos em linguagem C. Tais grafos podem ser utilizados durante a fase de treinamento por uma rede neural, como uma forma de modelar o conhecimento. Este trabalho avalia três diferentes grafos: (1) árvore sintática abstrata (AST); (2) árvore sintática abstrata anota com dados (AST_DATA); (3) e grafo de fluxo de controle e dados (CDFG). A extração é realizada utilizando o framework Compy-Learn[1] (CL). Este, por sua vez, possui diferentes funcionalidades que descrevem a geração de uma rede neural a partir de um grafo.

Inicialmente, o módulo ClangDriver, o qual aceita um conjunto de parâmetros que descrevem como o programa deve ser compilado, compila o programa e gera as informações necessárias para a criação de um determinado grafo. Após, o módulo GraphBuilder monta, a partir das informações extraídas por ClangDriver, o grafo especificado. Por fim, CL fornece um modelo de redes neurais, baseado em grafo, que pode ser utilizado em diferentes problemas.

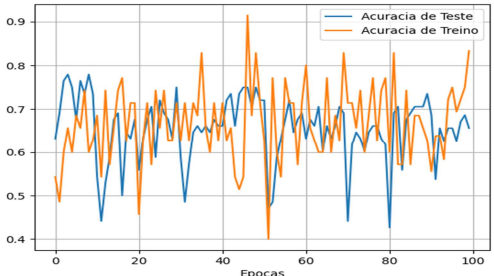
Como mencionado anteriormente, um dos primeiros passos na criação de sistemas de geração de código baseados em aprendizagem de máquina, requer a especificação de algum formalismo para representar programas. Portanto, embora nosso alvo final seja um sistema de geração de código, neste projeto cujo objetivo é determinar uma boa representação de código, o problema utilizado é determinar se um programa deve ser executado na CPU e na GPU. A justificativa para não avaliar representações em um sistemas de geração de código está no fato de tais sistemas envolverem diversas variáveis. O modelo de rede neural, disponibilizado em CL, é o especificado por Mou e Outros (2016).

Resultados e Discussão

Os experimentos utilizam validação StratifiedKFold, realizando a divisão dos dados em treino e teste em 10 agrupamentos, os quais são utilizados pela rede neural durante 100 épocas. Em adição, para avaliar a eficiência da classificação, baseada em uma determinada representação, é utilizada a métrica acurácia. A base de dados possui 170 programas, extraídos de diferentes benchmarks. As Figuras 1, 2 e 3 apresentam a acurácia - ou taxa de acerto - do treino e do teste em cada época de acordo com cada grafo (representação).

Como podemos observar nas figuras acima, pode-se considerar que a representação com o grafo de análise sintática abstrata é a mais eficiente. A cada época, a rede vai reduzindo a taxa de erro para que sejam reconhecidas especificações e a mesma chegue ao resultado esperado com mais rapidez. Entre as épocas 40 e 50 (metade), a acurácia máxima foi de 91% para a AST, na fase de treino, enquanto as outras ficaram entre 82% e 85%.

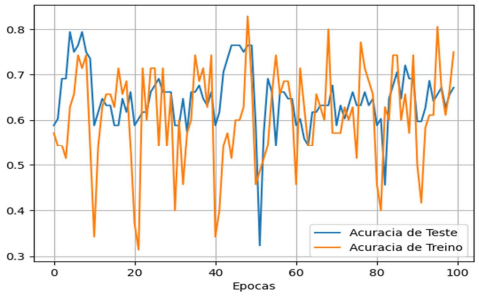
1 https://github.com/tud-ccc/compy-learn



	Acurácia			CE
	Max	Med	Min	
TE	0,78	0,66	0,42	85
TR	0,91	0,66	0,4	85

TE = teste TR = treino
CE = Ciclo de estagnação

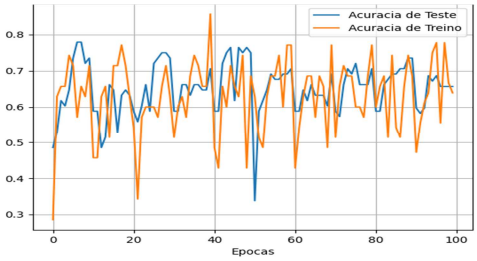
Figura 1: Acurácia obtida pela representação AST



	Acurácia			CE
	Max	Med	Min	
TE	0,79	0.64	0,32	50
TR	0,82	0.6	0,31	70

TE = teste TR = treino
CE = Ciclo de estagnação

Figura 2: Acurácia obtida pela representação AST_DATA



	Acurácia			CE
	Max	Med	Min	
TE	0,77	0.62	0,35	78
TR	0,85	0.65	0,28	21

TE = teste TR = treino
CE = Ciclo de estagnação

Figura 3: Acurácia obtida pela representação CDFG

A taxa de acerto do AST ficou entre 40% e 91%. Já a AST_DATA ficou entre 31% e 82% e a CFDG ficou entre 28% e 85%. Ainda podemos perceber que a variação média entre os três se mostrou bem próxima, entre 60% e 66%.
Um fato interessante é que as representações apresentam uma queda significativa da acurácia na época 50. Já para as outras épocas, o gráfico nos mostra que a primeira mantém mais vezes a taxa de acerto em crescimento em relação às outras.

Por fim, por volta das épocas 70 e 80, os três gráficos nos mostraram um período de estagnação na fase de treino. Por outro lado, a mesma parece ter estagnado na fase de teste por volta de 50 e 85.

Conclusões

Este trabalho apresenta uma rede neural baseada em grafos, que classifica programas escritos em linguagem C em diferentes classes e nos mostra diferentes resultados de acordo com as representações. Com isso, conclui-se que dependendo da forma como representamos um determinado programa, a rede pode ter ou não mais flexibilidade ao aprender. Sendo assim, é importante que análises sejam feitas para que tenhamos o resultado esperado e o objetivo de demonstrar a caracterização dos programas seja alcançado.

Agradecimentos

Agradecemos ao Programa Institucional de Iniciação Científica.

Referências

Cooper K, Torczon L. Engineering a Compiler (2nd edition). Morgan Kaufmann, USA, 2011.

Lili Mou, Ge Li, Lu Zhang, Tao Wang, Zhi Jin. Convolutional Neural Networks over Tree Structures for Programming Language Processing. In AAAI, 2016.

Lima E D, Souza Xavier T C, Silva A F, Ruiz L B. Compiling for performance and power efficiency. In Proc. the 23rd International Workshop on Power and Timing Modeling, Optimization and Simulation., Sept. 2013, pp.142-149.

Long S, O'Boyle M. Adaptive Java optimisation using instance-based learning. In Proc.the 18th International Conference on Supercomputing, June 26-July 1, 2004, pp.237-246.

Agakov F, Bonilla E, Cavazos J, Franke B, Fursin G, O'Boyle M F P, Thomson J, Toussaint M, Williams C K I. Using machine learning to focus iterative optimization. In Proc. the International Symposium on Code Generation and Optimization, March 2006, pp.295-305.