جامعة ٦ أكتوبر
October 6 University

علوم الحاسب ونظم المعلومات
Info Sys & Comp Science

Machine Learning in Compiler Optimization

By

Name: Mazen Barakat Al-Yamani Mowafi

ID: 202010752

Supervised By

DR. Nariman Abdo

Introduction:

- Some people have questioned why machine learning should be used to build compilers. Compilers are used to translate programming languages into executable code for computers and have been studied for many years with an emphasis on correctness. Machine learning, on the other hand, is a field of artificial intelligence focused on detecting and predicting patterns in diverse areas. machine learning can also be applied to compiler design, as it can improve accuracy, handle complex languages, automate tedious tasks, and adapt to changing code.

Compilers are vital in translating programs into binary and optimizing them for efficiency, which can be challenging and require developing heuristics. Machine learning can automate tedious aspects of compilation by predicting outcomes based on prior data. This enables compiler writers to use machine learning to determine which optimization to apply in order to execute programs faster. The learning stage involves systematically representing programs using quantifiable properties called features, which may require feature engineering to find high-quality features. In the deployment stage, a learning algorithm derives a model from training data to predict the optimal optimization option for new programs based on their features. The quality of the learned model depends on the features and training data chosen, so feature engineering and training data generation may need to be repeated. Machine learning brings new opportunities for innovation and evidence-based science in compilation.

machine learning models used for compiler optimization, with two major techniques: supervised and unsupervised learning. Supervised learning trains a model on performance data and program properties to predict optimal optimizations for new programs. This allows compilers to learn from large amounts of code and identify patterns to optimize code. For instance, recognizing commonly occurring code

patterns in a language or application can lead to more efficient code generation. Supervised learning can also predict performance issues and generate optimizations based on learned patterns, improving overall compiler design.

In compiler design, decision trees are commonly used in the process of optimizing code generation. The decision tree algorithm is a supervised learning algorithm that can be used for both classification and regression problems.

In the context of compiler design, decision trees are used to generate code that is optimized for a given target architecture. This is done by analyzing the code and constructing a decision tree that represents the optimal sequence of instructions to generate for a given input program. The decision tree is constructed using a set of rules that are derived from the analysis of the input program.

The decision tree algorithm works by recursively partitioning the input data set into smaller subsets based on the values of the input features. At each node of the tree, the algorithm selects the feature that best splits the data set and creates a new node for each possible value of the feature. This process continues until the tree is fully constructed.

In the context of code generation, each node in the decision tree represents a decision point in the code generation process. The algorithm uses the rules derived from the input program to determine the optimal instruction sequence at each decision point. The output of the algorithm is a sequence of instructions that is optimized for the target architecture.

One advantage of using decision trees for code generation is that they can be easily visualized and understood. This makes it easier for developers to understand and debug the generated code. Additionally, decision trees can be trained on large data sets, allowing them to learn complex patterns in the input data and generate highly optimized code.

Unsupervised learning is a machine learning technique that discovers patterns in input data without relying on labeled output data. It groups input data items into subsets using clustering, which is used in compiler design to identify patterns and structure in program behavior and source code. For example, k-means clustering can group program execution into phase groups, allowing for a few samples to represent entire program phases for optimization strategies. Principal Component Analysis (PCA) can reduce feature dimension and discover common patterns in datasets for clustering exercises. Using unsupervised learning techniques in compiler design can improve program optimization strategy.

Resources

- " Machine Learning in Compiler Optimization Zheng Wang and Michael O’Boy