國立臺北科技大學

資訊工程系碩士班

碩士學位論文

Use Efficiency-based Genetic Programming to Create Loss Function for Image Classification Tasks

研究生：周子榆

指導教授：陳香君博士

中華民國一百一十四年五月

國立臺北科技大學

資訊工程系碩士班

碩士學位論文

Use Efficiency-based Genetic Programming to Create Loss Function for Image Classification Tasks

研究生：周子榆

指導教授：陳香君博士

中華民國一百一十四年五月

「學位論文口試委員會審定書」掃描檔

審定書填寫方式以系所規定為準，但檢附在電子論文內的掃描檔須具備以下條件：

1. 含指導教授、口試委員及系所主管的完整簽名。
2. 口試委員人數正確，碩士口試委員至少 3 人、博士口試委員至少 5 人。
3. 若此頁有論文題目，題目應和書背、封面、書名頁、摘要頁的題目相符。
4. 此頁有無浮水印皆可。

審定書不用頁碼

Abstract

Title: Use Efficiency-based Genetic Programming to Create Loss Function for Image Classification Tasks

Pages: (Fill it)

School: National Taipei University of Technology

Department: Computer Science and Information Engineering

Time: May, 2025

Degree: Master of Science

Researcher: Tzu-Yu Chou

Advisor: Shiang-Jiun Chen, Ph.D.

Keyword: Image Classification, Genetic Programming, Loss Function, Deep Learning

In the last few years, with the rapid rise of deep learning, image classification models have quickly become one of the most popular and widely known models. When training such models, the steps are broadly divided into the following: preparing image datasets, dividing them into training, validation and testing sets as needed, training the model, evaluating the model, and repeating these steps until the ideal result is met or the computational resources are exhausted.

A crucial function during the process of training a model is called the loss function, which calculate the difference between the predicted values and the ground-truth values. The results of the loss function can significantly influence the effectiveness of the model's training because it simply decide the direction of the adjustment to the model. However, designing a loss function often requires the assistance of experts in the related field, leading to a resource-intensive design process. Recent research has proposed using Genetic Programming (GP) to generate loss functions to avoid the necessity of hiring numerous domain experts for assistance. Nevertheless, using GP typically results in decreased computational efficiency. This paper aims to improve the method of using GP to generate loss functions by modifying certain genetic operations and introducing the concept of tournament selection.

這邊要加結果

Acknowledgements

所有對於研究提供協助之人或機構，作者都可在誌謝中表達感謝之意。

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

After several decades of development, deep learning [1] technology has achieved significant breakthroughs in the last ten years, largely due to the remarkable improvements in GPU computational power. Consequently, the predictive and analytical capabilities of models have achieved substantial advancements. Numerous models have been launched by major companies and applied in practical scenarios, leading to significant changes in our daily lives.

When it comes to training deep learning model, we usually refer to the following steps: collecting relevant data and organizing it into datasets, partitioning the datasets into training and validation sets as needed, initializing model parameters, training the model, evaluating the performance of the model, adjusting the model parameters, and repeating the training until the target is achieved or computational resources are exhausted. During the evaluation of the model's performance, we use a function called the loss function [2]. Its purpose is to calculate the difference between the model's predicted results and the ground truth, which helps model adjust its parameter to better fit the target. Therefore, different loss functions can influence the direction of model adjustments, significantly impacting the final outcomes of the model.

However, the design of a loss function is often closely related to the propose of the model [3]. In other words, different models may require experts from specific fields to assist in designing the loss function to enhance the speed and effectiveness of model training. Nevertheless, recent research has shown that it's feasible to use genetic programming (GP) [4] to automatically generate loss functions. Due to the domain-independent nature of GP, it allows us to develop an algorithm that can automatically create the required loss function without needing specialized knowledge in that particular field. However, this method typically requires a significant amount of time and computational resources.

The operation mode of GP can be simply divided into the following steps: Representing the solution we hope to find (which may be a value or a function) in an encoded form, then forming these solutions into a population. After evaluating the whole population, we perform genetic operations (e.g., mutation or crossover) on the population's solutions, reevaluate them, and repeat the above steps until the target is achieved or computational resources are exhausted.

Although the computational power of GPUs today is far superior to that of the past, both training models and using GP to find solutions still require substantial computational resources and time to achieve a decent result. In this paper, an efficiency-oriented GP algorithm is proposed to reduce the computational resources and the time to train the model required for GP to search for the optimal algorithm while maintaining the same level of effectiveness. We aim to improve the genetic operation part of the existing GP framework by referencing the concept that offspring in a better living environment can usually receive better care. We modify the GP's genetic operation such that only a certain number of high-scoring populations can execute it. Here's what we want to do: Select a certain proportion of high-scoring offspring from the population, then randomly select a fixed number from these offspring to perform genetic operations. The randomized selection can avoid always using the same population for genetic operations, giving more excellent offspring the opportunity to improve.

這邊要加結果

The structure of this paper is as follows: Chapter 2 is related work. This chapter provides a detailed illustration to the development and history of GP, the detailed description of the loss function and how they collaborate with deep learning model, and how loss functions are randomly generated via GP. Chapter 3 is proposed algorithm and main methodology. In this chapter, we will explain the architecture of the algorithm implemented in this paper. After that, we will then describe the experimental environment setup and the initial parameter values. Finally, we will present the methods and procedures used in this paper. Chapter 4 is results analysis. This section will show the detail parameter settings among all compared peer-algorithm. After that, we will present the analysis of the results by organizing the experimental results into charts and figures. Finally, we will explain the outcomes and analyze why our algorithm can achieve the similar result while decreasing the usage of computational resources. Chapter 5 is conclusion and future work. In this chapter, we will provide the conclusion of this paper by outlining the contribution we have made so far. After that, we would like to discuss potential directions for future research or possible way to apply this algorithm in practical.

Chapter 2 Related Work

In the following section, we will introduce the concept of metaheuristics [5, 6] . Then, we will discuss the importance of loss functions and their role in deep learning [1] models. Finally, we will explore image classification [7] and its real-life applications.

2.1 Metaheuristic

Metaheuristic [8] was first proposed by Glover. It can be regarded as a high-level procedure used to find solutions to optimization problems or other issues that conventional algorithms cannot solve. Within metaheuristics, an important concept is that these procedures must find a potentially optimal solution under reasonable computational costs or insufficient information. In this paper, we will primarily focus on nature-inspired metaheuristics [5]. In these methods, a common approach is to use a population-based strategy to find the optimal solution. Within a population, each individual typically represents a potential solution. We perform various operations on the individuals within the population to achieve our goal of approximating the optimal solution. A subset of specialized algorithms falls under population-based methods [9], commonly referred to as evolving algorithms (EAs) [10]. Commonly used methods within EAs include genetic algorithms (GA) [11], genetic programming (GP) [4] , evolutionary programming (EP) [12], and differential evolution (DE) [13]. We will briefly discuss EAs and GAs, and then we will focus primarily on GPs in the following paragraph.

EAs can be described as algorithms inspired by the concept of ”survival of the fittest [14].” These algorithms often take cues from natural evolutionary mechanisms to create algorithms that mimic animal behavior in the search for optimal solutions. Common examples include Ant Colony Optimization (ACO) [15], GA, and Particle Swarm Optimization (PSO) [16]. In this type of algorithm, the typical approach is to initialize a population, which consists of several individuals. Each individual represents a potential optimal solution. After initializing the population, we usually set a number to represent how many generations we want this population to evolve. Before the start of each generation, a special function is used to calculate the fitness value of each individ-

ual, determining their level of excellence. Next, we perform selection, mutation, and crossover on the population. Selection involves choosing individuals based on their fitness to reproduce the next generation. Mutation generally refers to making random changes to an individual, while crossover combines the characteristics of two individuals to create the next generation. These steps are repeated until the predefined number of generations is reached, or the individuals fail to achieve the expected results, leading to a forced stop. Ultimately, the best individual in the population is obtained as the solution.

GA is considered a type of EA, utilizing the representation of each solution in the form of chromosomes to perform selection, mutation, and crossover. GP is regarded as a special case of GAs, but due to its versatility and practicality, it is seen as a reliable solution. Unlike GAs, GP typically uses a more robust encoding method to represent chromosomes. For instance, GAs often use strings to represent chromosomes, which can present potential issues that need to be addressed, such as the priority of operations for each symbol or the validity of the equation itself. In contrast, GP usually represents chromosomes as trees, a method that can employ predefined traversal techniques to avoid these issues. It is worth noting that in GAs and GP, we can leverage their inherent ability to evolve automatically to find a reasonable optimal solution without having an in-depth understanding of the knowledge domain related to the problem we aim to solve. In the following paragraphs, we will introduce related research.

In [17], Co-Reyes et al. proposed a meta-learning reinforcement learning algorithm. They used computational graphs to represent algorithms. By doing so, algorithms could be identified, calculated, and optimized through Reinforcement Learning (RL) [18]. Notably, in this study, algorithms were represented as directed acyclic graphs (DAG) [19] of nodes. Within the DAG , all nodes were classified into three categories: input nodes, parameter nodes, and operation nodes. Once the algorithms were represented as DAGs, they could be placed into RL for training and evaluation. In this proposed algorithm, they employed the concept of regularized evolution [20] to evolve a population formed by several randomized and known algorithms. The method was as follows: initialize the population with algorithms, evaluate each algorithm in the population and record their performance, then in a loop, repeatedly use a sample tournament [21] to select algorithms, perform mutations on the algorithms by the mutator they designed, and evaluate them

again until the loop ends. During the evaluation process, they trained and assessed the algorithms using RL, continuously testing the performance of each algorithm in different training environments. They also utilized normalized training performance to avoid numerical biases caused by varying environments. Through this approach, the study successfully created two algorithms, named DQNClipped and DQNReg, which outperformed classical control tasks.

In [22], Akhmedova and Körber proposed a genetic programming-based method to find a better function for the image classification training task. They encoded the loss functions into trees, with each tree considered an individual. All individuals were aggregated into a population. In this study, the population was evolved across multiple generations. Before the start of each generation, all individuals were evaluated. Each individual had a probability of undergoing crossover with an individual from a special external archive to create a new individual; otherwise, it would perform the crossover with another individual. Following this, two probabilistic decisions were made. If successful, the new individual could perform subtree mutation or one-point mutation, respectively. At the end of each generation, the fitness value of all individuals was re-evaluated. A certain number of low-scoring individuals were eliminated to the special external archive mentioned earlier, maintaining the stability of the population's size. After these steps, a new generation began, continuing until a predefined number of generations was reached. By employing this method, this study successfully evolved an outstanding individual within the population, creating a function that could train an image classification model more effectively compared to Cross Entropy (CE) [23].

2.2 Loss Function

2.2.1 Deep Learning Model

In recent years, due to the significant increase in the computational speed of GPUs, training deep learning models to solve a wide variety of problems has become widely regarded as a feasible and popular solution. The process of training a deep learning model is often divided into several steps: collecting the data needed to train the model, and adding ground truth values to the

data according to the model's requirements. Ground truth can be thought of as the ideal answers we hope to achieve after the model's computation. Once the data collection is complete, these data sets are collectively referred to as the dataset. Typically, the dataset is divided into training sets, validation sets, and testing sets. Data from the training set are used to train the model, the validation set is used to evaluate the model's effectiveness after each training loop, and the testing set is used to calculate the model's final score and determine whether the model successfully achieves its designed purpose. It is worth noting that the testing set data should not be seen during training and validation phases. After dividing the dataset, we decide on the model's architecture. Common architectures include Fully Connected Networks (FCN) [24] and Convolutional Neural Networks (CNN) [25], etc. Then the training process can begin. During training, we compare the model's output with the ground truth of the training data and use a loss function to calculate the difference between the output and the ground truth. This guides the adjustment of the model's parameters. Therefore, the loss function significantly determines the effectiveness of model training, and this aspect will be explained further later. After adjusting the model, the next round of training begins, continuing until the training is deemed ineffective or a predetermined maximum number of iterations is reached.

2.2.2 Importance of Loss Function

In [26], a loss function meet genetic programming is proposed ....

2.3 Image Classification

Chapter 3 Proposed Algorithm

This section will provide a brief overview of this chapter: First we will introduce the architecture of our algorithm and use a simple flowchart to provide a basic introduction to the proposed algorithm. Next we want to provide an in-depth explanation of our proposed algorithm, including the underlying concepts and the rationale behind our approach. Lastly we will provide a detailed description of our software and hardware specifications, as well as the setup procedures.

3.1 The Architecture of Algorithm

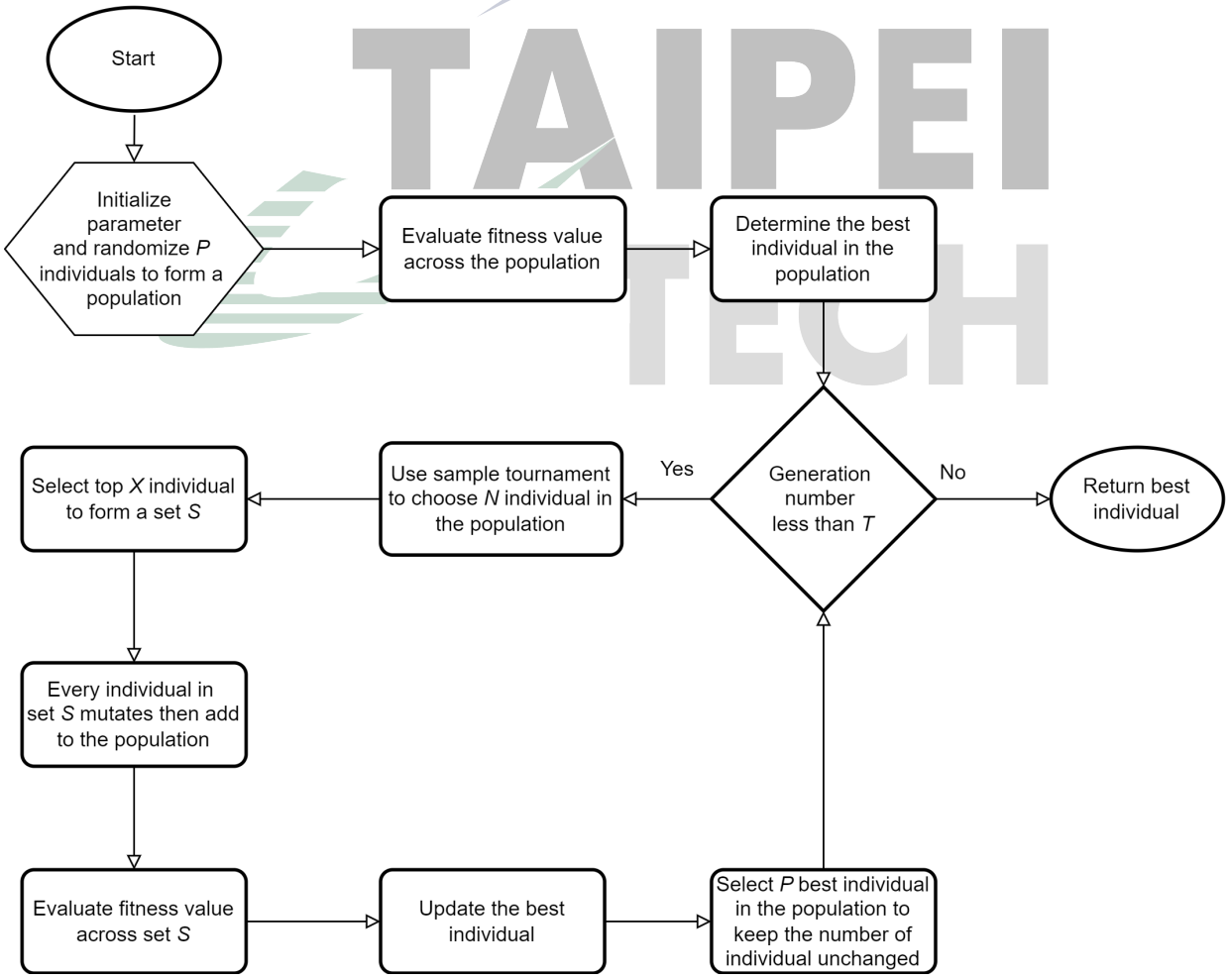

Figure 3.1 FlowChart for Efficiency-based GP

3.2 Main Method

Algorithm 1 Efficiency-based GP to generate loss function

Initialize:P =x, T =x, N =x, $M_{ST} = 0.5$
Randomly initialize population which include P trees
Evaluate GP fitness function F for each individual in the population
Determine the best individual
while generation number < T **do**
  Sample tournament G $\sim$ Uniform (P)
  Select top N trees from G to form a set S
  for Individual in S **do**
    if $rand_1 < M_{ST}$ **then**
      Apply subtree mutation to the generated child
    else
      Apply one-point mutation to the individual
    end if
    if Functional_check() is true **then**
      add individual to population
    end if
  end for
  Evaluated GP fitness function F for each generated child individual
  Update the best individual
  Select P best trees from population to form a new population containing P trees
end while

3.3 Implementation Details

3.3.1 Requirements

Table 3.1 Software setup

Package Type	Software Name	Version	Remark	License
Operating System	Ubuntu	24.04	N/A	N/A
Programming Language	Python	3.11	N/A	N/A
Programming Language	Pytorch	N/A	N/A	N/A

Table 3.1 shows our software setup. We run our code in Ubuntu 24.04. The Python version we employ is 3.8, and the Pytorch version is X. Within Python and Pytorch, we utilize the following packages: Some package.

Table 3.2 shows our hardware setup. We run our code on an AMD Ryzen 5 5600X 6-Core Processor. The system is equipped with 32GB of memory, operating at a frequency of DDR4-

Table 3.2 Hardware setup

Hardware Type	Name
CPU	AMD Ryzen 5 5600X 6-Core Processor
Memory	DDR4-3600MHz 32 GB
Graphic Card	NVIDIA GeForce RTX 3060 Ti

3600MHz. Additionally, we utilize an NVIDIA GeForce RTX 3060 Ti GPU for our computational needs.

3.3.2 Environment Setup

3.3.3 Implementation Steps

Chapter 4 Result & Analysis

Chapter 5 Conclusion & Future Work

5.1 Conclusion

5.2 Future Work

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444.

[2] Katarzyna Janocha and Wojciech Marian Czarnecki. *On Loss Functions for Deep Neural Networks in Classification*. 2017. arXiv: 1702.05659 [cs.LG]. URL: https://arxiv.org/abs/1702.05659.

[3] Lorenzo Rosasco et al. “Are Loss Functions All the Same?” In: *Neural Computation* 16.5 (2004), pp. 1063–1076. DOI: 10.1162/089976604773135104.

[4] Michael O’ Neill. “Riccardo Poli, William B. Langdon, Nicholas F. McPhee: A Field Guide to Genetic Programming”. In: *Genetic Programming and Evolvable Machines* 10.2 (2009), pp. 229–230.

[5] Xin-She Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[6] Xin-She Yang. “Metaheuristic optimization”. In: *Scholarpedia* 6.8 (2011), p. 11472.

[7] Waseem Rawat and Zenghui Wang. “Deep convolutional neural networks for image classification: A comprehensive review”. In: *Neural computation* 29.9 (2017), pp. 2352–2449.

[8] Fred Glover. “Future paths for integer programming and links to artificial intelligence”. In: *Computers & operations research* 13.5 (1986), pp. 533–549.

[9] Wikipedia contributors. *Metaheuristic — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-December-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Metaheuristic&oldid=1255593755.

[10] Heinz Mühlenbein, Martina Gorges-Schleuter, and Ottmar Krämer. “Evolution algorithms in combinatorial optimization”. In: *Parallel computing* 7.1 (1988), pp. 65–85.

[11] Manoj Kumar et al. “Genetic algorithm: Review and application”. In: *Available at SSRN 3529843* (2010).

[12] Xin Yao, Yong Liu, and Guangming Lin. “Evolutionary programming made faster”. In: *IEEE Transactions on Evolutionary computation* 3.2 (1999), pp. 82–102.

[13] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. “Differential evolution: A survey of the state-of-the-art”. In: *IEEE transactions on evolutionary computation* 15.1 (2010), pp. 4–31.

[14] Diane B Paul. “The selection of the “Survival of the Fittest” ”. In: *Journal of the History of Biology* 21 (1988), pp. 411–424.

[15] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. “Ant colony optimization”. In: *IEEE computational intelligence magazine* 1.4 (2006), pp. 28–39.

[16] James Kennedy and Russell Eberhart. “Particle swarm optimization”. In: *Proceedings of ICNN’95-international conference on neural networks*. Vol. 4. ieee. 1995, pp. 1942–1948.

[17] John D. Co-Reyes et al. *Evolving Reinforcement Learning Algorithms*. 2022. arXiv: 2101.03958 [cs.LG]. URL: https://arxiv.org/abs/2101.03958.

[18] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. “Reinforcement learning: A survey”. In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[19] Veronika Thost and Jie Chen. *Directed Acyclic Graph Neural Networks*. 2021. arXiv: 2101.07965 [cs.LG]. URL: https://arxiv.org/abs/2101.07965.

[20] Esteban Real et al. “Regularized evolution for image classifier architecture search”. In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 4780–4789.

[21] David E Goldberg and Kalyanmoy Deb. “A comparative analysis of selection schemes used in genetic algorithms”. In: *Foundations of genetic algorithms*. Vol. 1. Elsevier, 1991, pp. 69–93.

[22] Shakhnaz Akhmedova and Nils Körber. *Next Generation Loss Function for Image Classification*. 2024. arXiv: 2404.12948 [cs.CV]. URL: https://arxiv.org/abs/2404.12948.

[23] Zhilu Zhang and Mert Sabuncu. “Generalized cross entropy loss for training deep neural networks with noisy labels”. In: *Advances in neural information processing systems* 31 (2018).

[24] Michael Iliadis, Leonidas Spinoulas, and Aggelos K Katsaggelos. “Deep fully-connected networks for video compressive sensing”. In: *Digital Signal Processing* 72 (2018), pp. 9–18.

[25] Jianxin Wu. “Introduction to convolutional neural networks”. In: *National Key Lab for Novel Software Technology. Nanjing University. China* 5.23 (2017), p. 495.

[26] Santiago Gonzalez and Risto Miikkulainen. *Improved Training Speed, Accuracy, and Data Utilization Through Loss Function Optimization*. 2020. arXiv: 1905.11528 [cs.LG]. URL: https://arxiv.org/abs/1905.11528.