GCN HYPERPARMETERS ESTIMATION USING EVOLUTIONARY ALGORITHMS

Student Name: Nguyen, Duy

UW Number: 20868631

CONTENTS

1 INTRODUCTION

Convolutional Neural Network (CNN) is a class of deep neural networks which was created especially for dealing with computer vision tasks. At the initial stage it is used for image classification only, but throughout recent years it has also gained tremendous successes in other computer vision problems, such as object detection and art generation. CNNs takes advantages of the spatial dependencies of pixels in an image: it defines a convolution operator, through that the network can learn to use all the information of pixels in a particular region to detect features (for example, an edge). By doing so, it effectively reduces the number of parameters a network must estimate, thus allow for better fitting of the image dataset and more sophisticated network structure.

However, by definition of the convolution operator, CNNs can only operate on regular grid structures (such as images), which can also be considered a special instance of graphs. There are many problems in real-life where we must deal with a general graph structure (for example: social network analysis, molecular structures). Thus, researchers want to find an extension of CNNs to graphs. Another problem to CNNs in particular and neural networks in general is that their performance greatly depends on the network architectures and other hyperparameters.

The graph convolutional network (GCN - recently proposed by Kipf et al, 2017) [1] has produced some promising results in converting the success of traditional CNNs to social network problems. In their works, the authors defined a novel convolution operator on graphs based on the graph Laplacian matrix, then later used it to introduce an efficient way to propagate information directly on graphs.

In this project, we want to address the problem of hyperparameters selection for graph convolutional networks, particularly by adapting evolutionary algorithms (EA). EA is a family of metaheuristic optimization algorithms, inspired by biological evolution mechanisms, and often used when we know little to nothing about the fitness landscape. In 2017, Xie et al. [2] introduced a way to learn the structure of CNNs using genetic algorithm. Later in 2018, Baldominos et al. [3] used genetic algorithms and grammatical evolution to address the MNIST handwritten digit classification problem. On our side, we want to further explore other kinds of evolutionary algorithms, such as Particle Swarm Optimization (PSO - [4]), or Artificial Bee Colony (ABC - [5]) and apply them to GCNs. The particular problem we want to try on is the citation networks problem. This is a semi-supervised classification problem where we are given a graph representing citation relations among papers and labels for a subset of them; the objective is to label all the remaining papers.

2 LITERATURE REVIEW

2.1 Hyperparameters estimatation for neural network

In general, a neural network in particular or any learning algorithm in general have the same objective: to find a function f that minimizes the loss $\mathcal{L}(x; f)$ with respect to given i.i.d samples x drawn from a natural distribution $\mathcal{G}_x$. A learning algorithm $\mathcal{A}$ is a functional that maps a dataset $X^{(train)}$ to a function f. Typically, the learning algorithm itself depends on the hyperparameters λ, and the actual learning algorithm is the one obtained after choosing λ, which can be denoted as $\mathcal{A}_\lambda$, and $f = \mathcal{A}_\lambda(X^{(train)})$ for a training set $X^{(train)}$.

What we need is to choose λ to minimize the expected loss: $E_{x\sim\mathcal{G}_S}[\mathcal{L}(x;\mathcal{A}_\lambda(X^{train}))]$. This is an optimization problem for which in general we do not have efficient algorithms to solve. There are two main strategies which have been widely used: grid search and random search. In grid search, for each hyperparameter λ_i we must specify a set of values to search: $(L_{i1}...L_{ik})$, so the number of trials may grow exponentially with the number of hype-parameters K. Thus, although it is easy and straightforward to implement, grid search is not considered a great choice for most practical tasks.

Meanwhile, random search is used more frequently. As the authors lay out in [6], random experiments are more efficient because not all hyperparameters are equally important to tune. Grid search experiments allocate too many trials to the exploration of dimensions that do not matter and suffer from poor coverage in dimensions that are of greater importance. Aside from that, thanks to the statistical independence of each trial, random search offers us the flexibility to stop the experiment at any time or add new trials to the experiment half-way without having to adjust the grid. However, grid search is still limited to the search space distribution. As pointed out in [7], for some problems the accuracy space may vary greatly according to the change interval of hyperparameters. Therefore, we are still in need of a strategy that at least may serve as a promising alternative in case grid search performs poorly.

2.2 Hyperparameters estimation using Genetic Algorithm for Convolutional Neural Networks

In general, a CNN architecture comprise of several types of layers mixed all together. Thus, when designing a network architecture, an expert has to make a lot of design choices. This drives the need for a systematic way to efficiently estimate the hyperparameters in a large search space. Genetic Algorithm (GA) is an evolutionary search algorithm used to solve optimization and modeling problems by selecting, combining, and varying parameters using biological evolution mechanisms. It simulates the process of natural selection where best individuals have higher chance of surviving to produce offsprings. To adapt GA scheme into the hyperparameter estimating problem, each individual here is represented as a vector where each element is considered a hyperparameter to optimize. The fitness of each individual is solely determined by the accuracy on the validation set of the network which uses that set of hyperparameters.

In 2017, Xie and Yuille propose a GA scheme to automatically learn CNN topology to perform visual recognition [2]. Although it produced some interesting results, the proposed scheme is only used to explore the network structure, it does not touch the training process.

Later, in 2018, Baldominos et al. [3] used two different evolutionary computation techniques: genetic algorithms and grammatical evolution to tackle the MNIST handwritten digit classification problem. They tried to optimize both the hyperameters governing the network structure and those governing the learning process at the same time. Moreover, the proposed scheme can be used both for designing a CNN from scratch and optimizing an existing one. However, partly due to that "greediness", the computational complexity is huge and in fact, the scheme was applied to a problem with a relatively small dataset (MNIST handwritten digit) only.

Recently, Han et al. [7] have brought another perspective to the optimization problem. There has been a general consensus that deeper and more complex network structure should have better performance in terms of accuracy compared to the simpler ones; however such complex structures may take a lot of time to make a prediction, so sometimes it is more preferable to choose a simple network without compromising the accuracy too much. To this end, the authors

added verification time as the second factor contributing to the fitness function, thus required the optimal network structure to be simple and have high accuracy simultaneously.

There are a wide range of research papers that attempt to adapt EAs to hyperparameter estimation problem for machine learning algorithms, as in [8], [9], [10]. There is also an article comparing strengths and weaknesses of grid search, random search and genetic algorithm in neural architecture search [11].

3 PROPOSED SOLUTION

In this project, we aim to do the following things:

- First, we try to address the hyperparameters optimization problem for GCNs using 3 different algorithms in the evolutionary family: Genetic Algorithm (GA), Particle Swarm Algorithm (PSO), and Artificial Bee Colony (ABC). There have been some researches on using GA to tune hyperparameters for CNNs. Thus, this may be considred as an *innovation by model transfer*. Meanwhile, PSO and ABC are two metaheuristic algorithms which have been developed by the community of optimization scientists; here we attempt to adapt them to the neural network hyperparameter estimation problem. Thus, this may be considered as an *innovation by reduction*.
- In their implementation for GCN at [12], the authors run the training algorithms through a fixed number of epochs. The process can be made more efficient by checking for early stopping condition. This little improvement may be considered as an *innovation by numerics*.
- Next, we try to address a problem: training a neural network may take a huge amount of time. To this end, we propose a modified cost function (fitness function) which comprises of validation accuracy and training time. There was an attempt to incorporate verification time (time taken to make one prediction) in [7]. However, due to the different nature of graphs (as compared to images), we believe in this task it is more suitable to use training time instead. This may be considered as an *innovation by optimization*.

3.1 Hyperparameters tuning for GCNs with evolutionary algorithms

3.1.1 *Genetic Algorithm (GA)*

- **Individual encoding** We encode the hyperparameters that need to be estimated by a vector of size K, where K is the number of hyperparameters. In the source code, we use a Python dict object to represent a vector. For example, the following is the vector representation of the hyperparameter configuration of a network which will be trained for 50 epochs, hidden layer consists of 16 units, drop out rate $= 0.5$, learning rate $= 0.01$ and weight decay $= 5 \times 10^{-4}$.

  ```
  {"epochs": 50, "n_hidden": 16, "dropout": 0.5, "lr": np.log10(0.01),
  "weight_decay": -4 + np.log10(5)}
  ```

  Note that learning rate and weight decay above are log10 of the actual values since we want to search for those two parameters on log-scale.

- **Fitness function** We define fitness function for each individual as the accuracy of the corresponding neural network on validation set.
- **Genetic operations** The selection process is performed at the end of every generation. We constantly generate new individuals for the next generation until the number reaches population size. The process of generating new individuals involve crossover and mutation. Crossover involves first picking two individuals as parents, then generate the new offspring which inherit traits from both. The algorithm will be more likely to pick individuals with better fitness values. Mutation involves randomly perturb some positions in the vector encoding of one individual. The point here is to try to escape from local minimum if it exists.

3.1.2 Particle Swarm Optimization (PSO)

We use the same individual encoding and fitness function as in GA. In PSO, a population consists of a fixed number of particles; at a particular time every particle's position and velocity are recorded. These particles are moved around the search space. The algorithm always keeps track of the best known position in the entire swarm (gbest), and each particle itself always keep track of its best known position ever (pbest). The movement of each particle is guided by the gbest and its pbest.

For our problem, an individual encoding is stored in the form of a particle's position.

3.1.3 Artificial Bee Colony (ABC)

We use the same individual encoding and fitness function as in GA. In the ABC model, the colony consists of three groups of bees: employed bees, onlookers and scouts. Each employed bee is assigned one food source and they constantly exploring around their sources until the source gets exhausted. The onlooker bees supervise the employed bee, evaluate the nectar information (fitness value) of each source then each onlooker bee in its turn choose a promising source to go to exploit, until it gets exhausted. The exhausted sources will be replaced by new sources discovered by scout bees.

In our optimization problem, each hyperparameter configuration is represented as a food source. A promising source will get exploited by the algorithm for a number of times before it gets exhausted, which mean no improvements can be made after a certain amount of time. In our implementation, we make a simplification by removing the scout bees: each employed bee or onlooker bee will become a scout bee, seeking for new food source once it has finished exploiting the old source.

3.2 Optimizing hyperparameters with respect to accuracy and training time

Training a neural network may take a lot of time. And, especially for the GCN model: both the adjacency matrix and Laplacian matrix are of size $N \times N$ where N is the number of nodes, or the sample size. Adding one more node would modify these above matrices so we must retrain the whole network from scratch. Therefore, it becomes imperative for us to come up with an algorithm capable of picking a network structure that not only has a good accuracy, but also being simple enough to train in a short amount of time.

To this end, we propose an alternative fitness function, which incorporates training time as a judging factor. The idea is, we can set a tolerance level by a time-out number. When training time is under tolerance level, it has no effect on the fitness value; but when it is over the tolerance level, fitness value is a decreasing function of it:

$$f = f_0 - \max\left\{\alpha\left(\frac{t}{t_o} - 1, 0\right)\right\} \tag{1}$$

where f_0 is the validation accuracy, t is training time and t_o is the timeout value.

Block diagram goes here

In the original paper, the authors do not mention whether or how they optimize their hyperparameters. Therefore, my proposed solution should be thought of as more of an extension to the original work. All of my experiments will be implemented on top of the authors' codebase at [12].

4 EXPERIMENTAL RESULTS

The following experiments were conducted to assess the effectiveness of our proposed solution.

4.1 Experimental set-up

We use the model proposed by Kipf et al. in [1] as baseline. We also trained and tested all of our models on the same dataset splits as in the paper. For all the parts relating to defining, training and testing a GCN, we reuse the code provided by the authors at [12] and only implement the EAs ourselves to optimize for hyperparameters.

4.2 Experiment 1 - Genetic Algorithm

In this experiment we use genetic algorithm to select the best hyperparameter configuration with respect to accuracy. Results of running the algorithm on 3 different datasets are summarized in Table 1 and Table 4 below.

GA parameters: population = 15, generations = 10, mutation rate = 0.1. The optimal configuration found by the algorithm is:

Hyperparameteres	Citeseer	Cora	Pubmed
Epochs	49	128	128
Number of hidden layers	69	62	66
Dropout	0.634	0.830	0.243
Weight decay (log-scale)	-3.174	-3.323	-2.780
Learning rate (log-scale)	-1.649	-2.364	-2.364

Table 1: Optimal solution found by GA

4.3 Experiment 2 - Particle Swarm Optimization

In this experiment we use PSO algorithm to select the best hyperparameter configuration with respect to accuracy. Results of running the algorithm on 3 different datasets are summarized in Table 2 and Table 4 below. PSO parameters: iterations = 10, number of particles = 15, $W = 0.5, C_1 = 0.8, C_2 = 0.9$.

Hyperparameteres	Citeseer	Cora	Pubmed
Epochs	109	250	109
Number of hidden layers	77	81	76
Dropout	0.731	0.750	0.733
Weight decay (log-scale)	-2.809	-3.684	-2.703
Learning rate (log-scale)	-2.206	-2.524	-2.189

Table 2: Optimal solution found by PSO

4.4 Experiment 3 - Artificial Bee Colony

In this experiment we use ABC algorithm to select the best hyperparameter configuration with respect to accuracy. Results of running the algorithm on 3 different datasets are summarized in Table 3 and Table 4 below. ABC parameters: iterations = 10, colony size = 14.

Hyperparameteres	Citeseer	Cora	Pubmed
Epochs	83	234	86
Number of hidden layers	63	75	100
Dropout	0.616	0.695	0.234
Weight decay (log-scale)	-3.031	-4.247	-2.649
Learning rate (log-scale)	-1.232	-2.379	-0.022

Table 3: Optimal solution found by ABC

A summary of performance of all the methods, compared to the result reported in the original paper is in Table 4 below. The wall-clock training time in seconds of the optimal solutions are also reported (in brackets). To alleviate the effect of randomness in recording training times, each optimal network was run 10 times; results are then averaged. All the networks are trained on the same machine (with GPU) under the same condition.

Method	Citeseer	Cora	Pubmed
GCN	70.5	81.5	79.0
GCN + GA	**72.5** (1.09)	**82.5** (2.78)	**79.9** (3.90)
GCN + PSO	**72.0** (2.37)	**82.2** (4.52)	78.6 (3.30)
GCN + ABC	**70.8** (1.55)	81.1 (4.21)	77.2 (2.38)

Table 4: Summary of results in terms of classification accuracy (in percent). Numbers in bold indicate an improvement over the original GCN.

.

4.5 Experiment 4 - GA with new fitness function

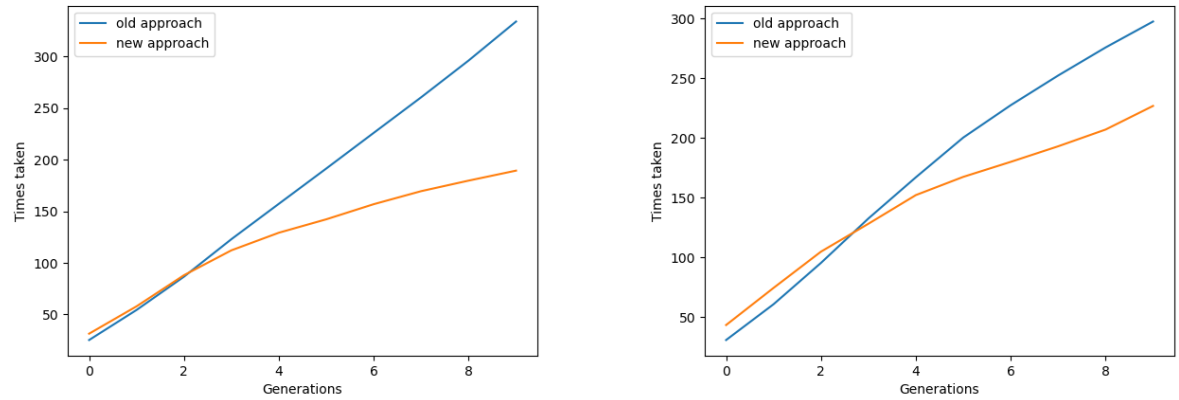
In this experiment we use genetic algorithm to select the best hyperparameter with respect to the alternative fitness function defined in (1).

Here we fix $\alpha = 0.1$ (that means for a network with training time k times bigger than timeout the validation accuracy will be subtracted by $0.1 * k$). Results of running the algorithm on different datasets are reported in the table 5 below. These results are not reproducible, since it is impossible to train a network in an exact amount of time, times to times. We can see clearly there is a trade-off between accuracy and training time when comparing these new solutions with old solutions.

Dataset	timeout	Accuracy	Training time	Optimal solution
Cora	1	81.4	0.92	[43, 67, 0.243, -2.928, -1.013]
	0.5	81.3	0.49	[28, 66, 0.243, -2.928, -0.771]
Citeseer	1	72.5	0.68	[28, 68, 0.654, -2.928, -1.013]
	0.5	68.6	0.52	[28, 8, 0.346, -2.824, -0.646]
Pubmed	1	78.9	0.96	[28, 62, 0.326, -2.928, -1.814]
	0.5	78.7	0.84	[28, 8, 0.243, -3.188, -1.013]

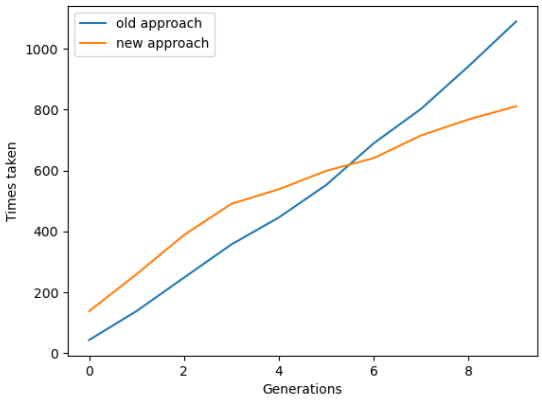
Table 5: Optimal solution found by GA, with respect to new fitness function

Using new fitness function also results in shorter running time for EAs. That is because in later stages all the slow-running networks are more likely to be ruled out of the population. In Figure 1 we compare the cumulative running time of the algorithm when we use the new fitness function, versus the old one. We can see that for the old fitness function, total training time increases linearly with iteration number; meanwhile for our new fitness function, time taken for each generation becomes shorter over time.



(a) cora

(b) citeseer



(c) pubmed

Figure 1: Comparison of GA cumulative running time (in seconds) when optimizing with respect to the old fitness function (blue line) and the new one (red line), on 3 different datasets

4.6 Source code

Our source code for all the experiments is available at: [13]

5 DISCUSSIONS AND CONCLUSIONS

In this term paper, we have examined how evolutionary algorithms can be used to optimize hyperparameters for a graph convolutional neural network. Three algorithms were adapted and they have successfully made improvements to the results reported in the original paper. We also propose a novel fitness function to optimize for both accuracy and training time. Training time is a crucial factor in training CNNs, because unlike the traditional GCN, adding more samples involves rebuilding graphs and retraining the network from scratch. Optimizing hyperparameters

with respect to this new fitness function results in much faster-to-train optimal structures and shorter running time of genetic algorithm without compromising the accuracy too much.

One obvious drawback of this approach is the massive amount of time required to run the algorithms. Two things may contribute to this: One, verifying an instance requires training a complete network from scratch; Two, evolutionary algorithms are known to be slow since they make no assumptions of the optimization problem being solved. While it seems hard to evade the former one, we believe some improvements can be made regarding the latter. For example, the nature of EAs may allow us to do parallel computing to shorten the running time. Another concern is that EAs do not guarantee to converge to the global optimum. Nevertheless, this can be mitigated if we run the algorithm multiple times.

Further improvements could be made in various aspects.

- The choice of algorithms should not be limited to just GA/PSO/ABC, one could try other optimization algorithms.
- The parameters for EAs themselves were not optimized, one could find a systematic way to select those parameters.
- The alternative fitness function in (1) is still relatively simple, one could come up with a more sophisticated function which can better serve a specific purpose.
- It has been observed that most of the time using grid search and random search in combination achieve better results than using each of them individually. In a similar manner, one could think of a creative way to combine GA search with other methods.

REFERENCES

[1] Thomas N. Kipf, Max Welling *Semi-Supervised Classification with Graph Convolutional Networks* arXiv:1609.02907

[2] Lingxi Xie, Alan Yuille *Genetic CNN* arXiv:1703.01513

[3] Alejandro Baldominos, Yago Saez, Pedro Isasi *Evolutionary Convolutional Neural Networks: an Application to Handwriting Recognition*. Neurocomputing (2017)

[4] J. Kennedy and R. Eberhart, *Particle swarm optimization* Proceedings of ICNN'95 - International Conference on Neural Networks, Perth, WA, Australia, 1995, pp. 1942-1948 vol.4.

[5] Karaboga, D., Basturk, B. *A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Glob Optim 39, 459–471 (2007)* https://doi.org/10.1007/s10898-007-9149-x

[6] James Bergstra, Yoshua Bengio *Random Search for Hyper-Parameter Optimization* Journal of Machine Learning Research 13 (2012) 281-305

[7] Han, J., Choi, D., Park, S. et al. *Hyperparameter Optimization Using a Genetic Algorithm Considering Verifcation Time in a Convolutional Neural Network* J. Electr. Eng. Technol. 15, 721–726 (2020). https://doi.org/10.1007/s42835-020-00343-7

[8] Shih-Wei Lin, Kuo-Ching Ying, Shih-Chieh Chenc, Zne-Jung Leea *Particle swarm optimization for parameter determination and feature selection of support vector machines* Expert Systems with Applications Volume 35, Issue 4, November 2008, Pages 1817-1824

[9] Oliveira, A. L. I., Braga, P. L., Lima, R. M. F., & Cornélio, M. L. *GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation* Information and Software Technology, 52(11), 1155–1166. doi:10.1016/j.infsof.2010.05.009

[10] Zhu, X.; Li, N.; Pan, Y. *Optimization Performance Comparison of Three Different Group Intelligence Algorithms on a SVM for Hyperspectral Imagery Classification* Remote Sens. 2019, 11, 734.

[11] Petro Liashchynskyi, Pavlo Liashchynskyi *Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS* arXiv:1912.06059

[12] T.Kipf *Github Repository - Pytorch Implementation of Graph Convolutional Network* https://github.com/tkipf/pygcn

[13] https://github.com/d222nguy/project_602