DE-GCN: Differential Evolution as an optimization algorithm for Graph Convolutional Networks

Shakiba Tasharrofi
Amirkabir University of Technology
Electrical Engineering Department
Master of Science in Digital Electronics Systems
Tehran, Iran
sh.tasharrofi@aut.ac.ir

Hassan Taheri
Amirkabir University of Technology
Electrical Engineering Department
Associate Professor
Tehran, Iran
htaheri@aut.ac.ir

2021 26th International Computer Conference, Computer Society of Iran (CSICC) | 978-1-6654-1241-4/20/$31.00 ©2021 IEEE | DOI: 10.1109/CSICC52343.2021.9420542

***Abstract*—Neural networks had impressive results in recent years. Although neural networks only performed using Euclidean data in past decades, many data-sets in the real world have graph structures. This gap led researchers to implement deep learning on graphs. The graph convolutional network (GCN) is one of the graph neural networks. We propose the differential evolutional optimization method as an optimizer for GCN instead of gradient-based methods in this work. Hence the differential evolution algorithm applies for graph convolutional network's training and parameter optimization. The node classification task is a non-convex problem. Therefore DE algorithm is suitable for these kinds of complex problems. Implementing evolutionally algorithms on GCN and parameter optimization are explained and compared with traditional GCN. DE-GCN outperforms and improves the results by powerful local and global searches. It also decreases the training time.**

Keywords— Graph convolutional network, Graph node classification, Neuroevolution, Differential evolution algorithm, Anomaly Detection

I. INTRODUCTION

Graphs are utilizable in a wide range of applications, especially where we see the interplay between things and graphs. Graphs may consist of thousands or millions of nodes and edges (e.g., transportation networks, social networks and Bitcoin transactions). In recent years, learning with graphs and extracting latent information from networks is a hot research topic [1]. Graphs' data structure is more complicated than the Euclidian data structure. As a result, this leads to complicated learning process in graphs. Authors in [2] have formulated neural networks for graphs.

These graph neural network models try obtaining the parameter collections that minimize the value of the loss function of the network. There are various optimization algorithms that reduce the loss function. Many of the deep learning processes have applied gradient descent as an optimization algorithm in the learning process. Another strategy for optimization of neural networks originates from the field of neuroevolution [3, 4]. Neuroevolution combines evolutionary algorithms and artificial neural networks [5] and enables advanced search and extensive parallelization [6]. Hence these optimization approaches correspond to graph neural network structures, which are nonconvex and large-scale optimization problems. Evolutionary algorithms belong to population-based stochastic search algorithms that are promoted from ideas and principles of natural evolution [5].

In recent years, extensive research has been conducted on the optimization of artificial intelligence issues. Many algorithms can be used to optimize AI problems which in turn consider optimization problems. If we suppose any deep learning problem as a padlock and optimizer as a key, then we should obtain the proper key for our optimization problem. This choice depends on the problem conditions such as dimension, data volume, and parallel or unified approach.

In general, optimization algorithms are divided into two categories: deterministic algorithms and stochastic algorithms. Algorithms such as gradient-based algorithms fall into the category of deterministic algorithms, while algorithms such as differential evolution and genetic algorithms are called stochastic algorithms. In general, there is no appropriate research on evaluating which of these algorithms is more suitable for any specific problem. Of course, each of these algorithms has strengths and weaknesses. Deterministic algorithms, for example, have more reliability, better convergence, and require fewer iterations. One of their shortcomings is the cost of derivative computation, which may be difficult or impossible for some problems to calculate. These algorithms may be confined to the local minimum. Their convergence also depends on the initial values.

Stochastic algorithms do not depend on the initial values. Most of these algorithms do not require derivative computation. Also, these algorithms do not fall into local optimum. One of the stochastic algorithm weaknesses is the low speed of convergence. Besides, due to the random state and uncertainty of them, we may receive different answers each time of the execution. Of course, this problem can be solved by limiting the range of solutions. In non-convex or more complex problems, we may face local optima challenges; so, stochastic algorithms are preferred to use.

In this paper, we propose differential evolution algorithm as an optimization algorithm on graph convolutional networks for node classification task to extract latent information from the graph. We use Elliptic Bitcoin transaction network dataset [7] for testing this proposed algorithm. Indeed, DE-GCN uses DE algorithm to find optimum weights for GCN.

978-1-6654-1241-4/21/$31.00 ©2021 IEEE

Authorized licensed use limited to: Hebei University of Technology. Downloaded on January 06,2023 at 10:06:11 UTC from IEEE Xplore. Restrictions apply.

In section 2, we will have an overview of the previous researches. Sections 3 and 4 explain the method proposed in this paper and the experiment performed, respectively. Then, we review the results and conclusions in sections 5 and 6.

II. Related Works

The most relevant set of approaches to this paper is incorporation of neural networks and stochastic optimization algorithms. In this work, we mainly concentrate on the graph convolutional network (GCN) [2] and differential evolution as an optimization algorithm [8, 9]. Hence we emphasize on two research category, namely GCN and evolutional algorithms.

GNC architecture can be static or dynamic. EvolveGCN [10] is a temporal GCN which computes separate GCN model for each time-step. Then, a recurrent neural network (RNN) captures dynamics of the graph from GCNs. In EvolveGCN, weights of the GCNs are called the system state. This architecture is suitable for dynamic graphs that change in time. The model uses a gated recurrent unit (GRU) or a long short-term memory (LSTM) to update the system every time. The input is the graph data-structure at the current time-step. In EvolveGCN, the graph information is illustrated by embedding's of the top-k influential neighbor nodes in the graph [11]. Authors in [10] compare the prediction performance between the non-temporal GCN and the temporal EvolveGCN. EvolveGCN outperforms GCN, but the improvement is not significant for some datasets, e.g. Elliptic Bitcoin dataset.

Authors in [11] proposed research on detecting Bitcoin laundering transactions on Elliptic dataset using graph features and GCN node embedding. They formed objective graphs for 49 time-steps, using the first 34 graphs for training and the rest 15 graphs for testing and obtained F1-score of over 0.7 with GCN node embedding and Random Forest. Both [10, 11] use gradient-based algorithms for GCN learning process. For graph node classification, [10, 11] used the Elliptic dataset [7] for the anomaly detection task in the Bitcoin network. Elliptic is a cryptocurrency intelligence company focused on safeguarding cryptocurrency ecosystems against criminal activity [11].

Many of novel deep learning researches concentrate on neuroevolution, in which neural network parameters are trained through evolutionary algorithms. Neuroevolution approaches apply evolutionary algorithms to optimize neural networks, motivated by the fact that physical brains themselves are the results of an evolutionary process [6]. We can define at least three kinds of tasks for the DE algorithm in artificial neural networks, i.e. obtaining optimum weights of networks, searching for optimum architectures and determining optimum learning rules [5]. For example, in psoCNN [3], the authors proposed a novel algorithm to search for deep convolutional neural networks (CNNs) architectures based on particle swarm optimization, which is a stochastic optimization algorithm.

In this work, we concentrate on getting the suitable parameters of the artificial neural network weights, exclusively GCN using the DE algorithm. Several evolutionary algorithms suggested in dynamic continuous optimization were implemented for training supervised feed-forward neural networks in classification tasks [12, 13 and 14]. Authors in [12] suggest and illustrate a workflow that pairs feed-forward DNN with evolutionary algorithms. The authors propose a hybrid approach where the DNN is used only for preselection and initialization that is more effective at optimization. Authors in [14] illustrated that learning neural network with differential evolution algorithm solves classification problems with raised accuracy. In [13], a differential evolution algorithm was implemented to train feed-forward multilayer perceptron neural networks (MLP). The authors showed that performance of differential evolution optimization algorithm is better than the gradient-based methods.

In [15], the authors concluded that training artificial neural networks using the DE algorithm, achieve better data classification at the same time duration as the gradient-based algorithms. They showed that DE has some advantages, e.g. the chance of getting the global minimum of a nonlinear function regardless of initial values of its parameters is more suitable than gradient-based algorithms. The quick convergence and the little number of parameters to set up at the start of the algorithm's operation is better than traditional gradient-based algorithms. Indeed, the experimental results determine that stochastic optimization algorithms had smaller classification error than a gradient-based algorithm.

In this work, we deal with the differential evolution algorithm proposed in [8, 9] to search for GCN [2] weights, instead of gradient-based optimization algorithms. Hence, we form GCN; then differential evolution (DE) optimum node embeddings will be achieved. Therefore, similar to [11], the nodes will be classified using dataset features and GCN node embeddings by some traditional machine learning methods, e.g. SVM, Random Forest and Logistic Regression. In other words, we offer vanilla differential evolution algorithm as an optimizer for learning supervised feed-forward graph convolutional networks in node classification task.

III. The Proposed Method

A. Differential Evolution Algorithm

Differential evolution algorithm is a new evolutionary algorithm first proposed by Storn and Price in 1997 [8, 9]. The algorithm is a heuristic one for global optimization over continuous spaces. An evolutionary optimization algorithm is in a vast collection of algorithms motivated by biology [12]. The principal concept of this algorithm is to generate an initial random population solution set and then rebuild the population evolutionary. This algorithm has an excellent ability to solve a variety of numerical optimization problems, better global convergence, and robustness [16]. Selection, crossover, and mutation are the basic operations of this algorithm.

The population of the generation G contains N, D-dimensional parameter vectors called individuals. The individual i of generation G is:

$$X_{iG} = (x_{iG}^1, x_{iG}^2, \ldots, x_{iG}^D),\ i = 1,2,\ldots N \quad (1)$$

The initial population set covers the entire search space widely by uniform random individuals.

1) Mutation Operation

An individual could be generated by equation (2):

$$V_{n_1,G+1} = X_{n_1,G} + F * (X_{n_2,G} - X_{n_3,G}) \quad (2)$$

Authorized licensed use limited to: Hebei University of Technology. Downloaded on January 06,2023 at 10:06:11 UTC from IEEE Xplore. Restrictions apply.

In (2) n_1, n_2, and n_3 are random numbers produced inside the interval [1, *N*], and variation factor *F* is a real number in the range [0, 2]. F checks the differential variable's amplification degree for $X_{n2,G}$ - $X_{n3,G}$.

2) Crossover Operation.

Crossover operation is implemented to the Xi,G that is the target individual and its corresponding mutant vector, which is calculated in equation (2), to create a test vector:

$$u_{i,G+1}^{j} = \begin{cases} V_{i,G+1}^{j} & If\ rand(j) \le Cr \\ x_{i,G}^{j} & Otherwise \end{cases}, j = 1,2,\dots,D \quad (3)$$

Where rand(*j*) is a uniform distribution value in the interval [0, 1], Cr is a crossover probability in the range [0, 1] and $u_{i,G+1}$ is the test vector.

3) Selection Operation

In the selection operation, the candidate individual that is constructed from mutation and crossover operation conflict with target individual:

$$x_{i,G+1} = \begin{cases} u_{i,G+1} & If\ f(u_{i,G+1}) \le f(x_{i,G}) \\ x_{i,G} & Otherwise \end{cases} \quad (4)$$

In (4) f is the objective function. Suppose that the test vector has less or equal objective function value than the corresponding target vector. In that case, the test vector will replace the target vector and enters the next generation's population. Otherwise, the target vector will remain in the population for the next generation.

Eventually, this procedure continues until the new generation of N individuals will be created. This process is then repeated in the same way to obtain the stopping criteria.

B. Graph Convolutional Networks

Learning on graph data-structured is a novel subject [17, 18]. A GCN contains [2] several graph convolution layers. GCN is similar to MLP but it aggregates the neighborhood embedding using spectral convolution.

Consider the graph G = (N,E), where N is the nodes' set and E is the edges' set. The input of GCN l-th layer uses the adjacency matrix A and the node embedding matrix $H^{(l)}$ and a weight matrix $W^{(l)}$. Then, it creates the node embedding matrix called $H^{(l+1)}$ as output. Totally, we have L layers in GCN. We can write:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-0.5}\tilde{A}\tilde{D}^{-0.5}H^{(l)}W^{(l)}\right),$$

$$\tilde{A} = A + I_N,\ \ \tilde{D}_{ii} = \sum_j \tilde{A}_{ij} \quad (5)$$

Here σ is the activation function. It's usually ReLU except for the output layer (sigmoid or softmax). The first layer embedding matrix is the node features, i.e., $H^{(0)} = X$. For the node classification problem, $H^{(L)}$ is the probabilities of predictions.

From a perspective, the multiplication with adjacency and degree matrices in the graph convolutional network is an aggregation of the transformed embeddings of the nearby nodes. In the second attitude, the graph convolution layer is a feedforward layer, except for the multiplication with these matrices. The learning parameters of the GCN are the weight matrices $W^{(l)}$, in each layer. Therefore, the learning process is similar to a neural network and uses gradient descent to update weights and finally, finds the optimum weights in each layer.

C. DE-GCN

In this paper, we propose a new approach so-called DE-GCN, which uses a differential evolution algorithm as optimizer (instead of gradient-based optimizers) in the learning process of graph convolution network. In this section we indicate how our algorithm works and differs from traditional GCN.

DE-GCN designed for node classification task in graphs. In this method, the goal is to find the optimal weights for each GCN layer. Therefore, the solution dimension corresponds to the weights' dimension of each layer. First, the initial population is created according to (6). The size of the weight matrix of each layer is: l*(l+1). For individual *i* of generation we define:

$$W_{iG} = (W_{iG}^{(0)}, W_{iG}^{(1)}, \dots, W_{iG}^{(L)}),\ i = 1,2,\dots,N \quad (6)$$

Where for weight matrix of each layer we randomize N initial matrices. G indicates the number of generation.

In the first step, we use random weights from (6) and stack GCN layers. Hence, the weights are multiplied by the previous layer node embeddings, and the adjacency matrix is according to Equation (5). The last layer of GCN is the probabilities of prediction in the node classification task. Therefore, the output of GCN and true values feed into loss function. The loss function's role is equivalent to the objective function of DE algorithm. Hence, we use loss function (i.e. a weighted cross-entropy) as DE objective function to evaluate individuals.

At each stage of the algorithm, the weights are updated using DE algorithm according to Section A.

After applying DE algorithm steps, the selected weight matrices proceed to the next generation. Then, the chosen weights are feed to the GCN according to (5), and the loss function or equivalently the DE objective function is calculated. In each iteration, this algorithm seeks to select optimal weights that minimize the loss function. This process will continue until touching the stopping criteria. We have various approaches for stopping criteria, e.g. define a maximum iteration number or minimum loss value as stopping criteria. Another method is the epsilon constraint, which is focused on difference of parameters of the solution in two sequential iterations. If this difference is less than epsilon level, the stopping criteria are touched.

In the epsilon constraint method, the constraints are controlled by the epsilon level ε, which can boost to preserve the diversity of the employed population in the state when most answers are infeasible. Specifically, if the overall

Authorized licensed use limited to: Hebei University of Technology. Downloaded on January 06,2023 at 10:06:11 UTC from IEEE Xplore. Restrictions apply.

limitation violation of a solution is smaller than ε, this solution is considered achievable. In this work, we combined maximum iteration and ε-constraint as stopping criteria. If ε-constraint didn't touch at all, we stop execution after a maximum number of iteration and choose the best answer due to loss function. Hence the L-th layer of GCN indicate the probabilities of prediction; we use the output of L-1 layer, namely $H^{(L-1)}$ as GCN node embeddings.

The node classification task, based on GCN, is a nonconvex and complex problem with a large number of parameters. The local optima problem may occur. Graphs are dynamic and data volume increases in time. In such issues, we may require to distribute the learning process. For distributing computation, the non-gradient-based algorithms are preferred because parameter sharing in gradient-based optimizers compose overhead in the distributed execution.

IV. Experiment

A. Dataset

The Elliptic dataset shows a network of bitcoin transactions. Each edge indicates a payment flow, and the nodes describe transactions. Approximately 20% of the transactions have been recorded to real entities belonging to licit categories (exchanges, wallet providers, miners, legal services, etc.) and versus illicit ones (scams, malware, terrorist organizations, ransomware, Ponzi schemes, etc.) [11]. The goal is node classification of unlabeled transactions.

1) Nodes and Edges.

There are 203,769 node and 234,355 directed edges. The full Bitcoin network has approximately 438M nodes and 1.1B edges [11]. In the Elliptic Dataset, two percent (4,545) of nodes are in the illicit category. Twenty-one percent (42,019) are labelled licit. The remaining transactions are not labelled, but have other features as licit and illicit transactions.

2) Features.

Each node has 166 features. The first 94 features show local information about the transactions (e.g. time-step, the number of inputs/outputs, transaction fee, output volume.). The resting 72 features are aggregated features. They are obtained by aggregating transaction information of one-hop backward/forward from the target node, e.g. the maximum, minimum, standard deviation and correlation coefficients of the neighbor transactions for the local information (number of inputs/outputs, transaction fee, etc.) [11].

3) Temporal Information.

Each node has a timestamp, representing an estimation of the time that the Bitcoin network validates the transaction. There are 49 different time steps, evenly spaced with an interval of about two weeks. Neither of the edges are connecting other time steps. The node distribution over time is uniform. There are 1,000 to 8,000 nodes in each time step [11]. Figure 1 presents the dataset distribution in time.

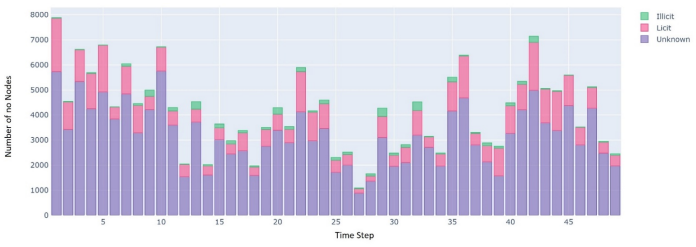
B. Implementing DE-GCN

First, we should implement GCN. In [19] using sub-graph is recommended. The authors expressed that learning process on graph suffers from either a high computational cost or a large space requirement for keeping the entire graph and the embedding of each node in memory. Clustering GCN in this work is suitable too. We Cluster our graph to 49 sub-graphs in time. By clustering the graph, the learning task of each sub-graph can be implemented by a processor. Hence the time has been saved, and the network can spread deeper. So, in this work we formed 49 objective graphs for 49 time-steps.

To implement the DE-GCN algorithm, we first randomly initialize the population weights of the various GCN layers. We define two layers for GCN. So weight matrices, i.e. $W^{(1)}$, $W^{(2)}$ should be initialized in the first step. In this implementation, weights are randomized uniformly. Then using equation (5), we get $H^{(l+1)}$ for each output layer. In the last layer, we have the probability of classification as the predicted value, which we give this value with the true value (for the nodes that are labelled) to the weighted cross-entropy loss function, where the weights of the illicit class is higher than the licit class. For stopping criteria, the combination of ε-constraint and maximum iteration approach, which are described in sec 3.3, is used. In our implementation, the maximum iteration number is 300, and the epsilon is 1e-08. So, the algorithm terminates if the most significant absolute difference between the coordinates of the population members is lesser this epsilon, until maximum iterations.

After applying the DE algorithm, the optimal weights are obtained. In the next step, with optimal weights, adjacency matrix and equation 5, node embeddings of each node achieved (i.e. the output of the L-1 layer of GCN). These node embeddings, along with the dataset features, are given to a classifier (e.g. Random Forest) to classify nodes and label unlabeled nodes. In this work, we choose the best parameters by hyperparameter tuning: crossover probability is 0.5, the scaling factor (F in equation 7) is 1, and the number of initial population is 50.

For experimentation, we used the first 34 time steps for the training phase and 15 remaining time steps for the test phase. We implement a GCN with two layers, which has 50 and 1 neurons in each layer. For this structure, first we examine gradient-based optimizer, we used in Adam optimizer. The number of epochs is 1000, and the learning rate is 0.001. For DE implementation, we use 50 initial population, and the number of maximum iteration is 300 times. We denote the number of iteration as stopping criteria when the epsilon constraint does not reach. We furthermore implement SVM, MLP (with one layer, which has 50 neurons, 60 epochs and Adam optimizer with 0.001 learning rate implemented in PyTorch), Logistic Regression and Random Forest (with 60 estimators implemented in scikit-learn) as classifiers for Elliptic dataset with 166 features. The Random Forest is the most desirable classifier for this problem. Therefore we couple GCN with Random Forest to gain more efficient outcomes. In all steps after hyperparameter tuning, we set all of the values listed above, e.g. number of layers, number of neurons, crossover probability and learning rate.



Authorized licensed use limited to: Hebei University of Technology. Downloaded on January 06,2023 at 10:06:11 UTC from IEEE Xplore. Restrictions apply.

Fig. 1. Node distribution (ilicit, licit and unknown transaction) in each time step.

V. Results and Analysis

In Table 1, the licit and illicit classification results are expressed. In the table, development results of the traditional machine learning (ML) classifiers are observed. Also, the result of the combination of GCN and traditional ML methods are examined. Due to [19] attaining more practical effects, with cluster structure, was expected; so we implemented two-layer GCN for each 49 time-steps. We compare gradient-based, and DE results in Table 1. After hyperparameter tuning for DE-GCN algorithm, we run the learning process several times. The results are fixed in all executions.

This problem is non-convex, and we faced the local optima problem. Therefore, evolutionary algorithms act more beneficial in this kind of situations. Also, this dataset is unbalance, and the F1-score is a more meaningful benchmark. As [11] mentioned, the Random Forest is the best classifier for this problem. Our results approved it too. Our proposed method could improve the recall and F1-score for MLP, Logistic Regression and Random Forest. Hence, with respect to the robust local search of the differential evolution method, we have some improved results in proportion to gradient-based optimization methods.

Training time for DE-GCN method is remarkably shorter than GCN with Adam optimizer due to searching with the population and without gradient computation. DE-GCN needs nearly half of the training time of GCN with Adam optimizer. It also needs fewer iterations to converge. Based on several executions, we can conclude that our method converges because the result has not changed.

As a result, DE-GCN can improve the graph node classification outcomes with a considerable time-saving. Our method is suitable for distributed learning with smaller overhead due to not calculating gradient, and it can overcome local optima issue. We also derive that GCN features could improve the traditional ML classification results, and DE-GCN could extract more wealthy features.

TABLE I. Experimental Results

Method	Precision	Recall	F1-Score	MicroAVG F1
Logistic Regression	0.403	0.604	0.484	0.919
Logistic Regression + 2L GCN	0.404	0.616	0.488	0.919
Logistic Regression + 2L DE-GCN	0.411	0.615	0.492	0.920
MLP	0.788	0.483	0.599	0.959
MLP+ 2L GCN	0.778	0.561	0.652	0.962
MLP+ 2L DE-GCN	0.847	0.553	0.669	0.965
SVM	0.848	0.547	0.665	0.965
SVM+ 2L GCN	0.846	0.550	0.667	0.965
SVM+ 2L DE-GCN	0.848	0.547	0.665	0.965
Random Forest	0.978	0.660	0.788	0.977
Random Forest + 2L GCN	0.985	0.663	0.792	0.978
Random Forest + 2L DE-GCN	0.985	0.663	0.792	0.978

VI. Conclution

If we suppose the deep learning problem as an optimization problem, determining the fittest optimizer is essential. By adopting the more efficient optimizer, we can guarantee that we will relax from local optima, exploding problem, vanishing problem, etc. Optimizer selection depends on the problem conditions such as the dimension, data volume, online or offline approach. In this activity, we apply the vanilla differential evolution algorithm as an optimizer in graph neural networks. This method is not always the most desirable one, but it can improve the model performance for the complicated and non-convex problems such as graph node classification. This paper proposed a novel method to search for graph convolutional networks (GCNs) weights based on differential evolution optimization. This algorithm needs a significantly shorter time for training.

Additionally, it can improve the classification results by finding more efficient node embeddings using powerful local search in networks. Distributed implementation for this optimizer, which is heuristic and does not compute the gradient, is more suitable than the gradient-based algorithms. Averagely DE-GCN could enhance the results of node classification couple with traditional machine learning methods.

References

[1] Gong L., Cheng Q. “Exploiting edge features for graph neural networks.” IEEE/CVF Conference on Computer Vision and Pattern Recognition.2019. https://doi.org/10.1109/CVPR.2019.00943.

[2] Kipf, T. N., Welling M. “Semi-supervised classification with graph convolutional networks.” International Conference on Learning Representations (ICLR). 2017.

[3] Fernandes J. F. E., Yen G. G. “Particle swarm optimization of deep neural networks architectures for image classification.” Swarm and Evolutionary Computation Journal, 49, 62-74. 2019. https://doi.org/10.1016/j.swevo.2019.05.010.

[4] Otte S., Butz M. V., Koryakin D., Becker F., Liwicki M., Zell A. “Optimizing recurrent reservoirs with neuro-evolution”. Neurocomputing Journal, 192, 128-138. 2016. https://doi.org/10.1016/j.neucom.2016.01.088.

[5] Volna E. “Neuroevolutionary optimization”. International Journal of Computer Science Issues (IJCSI), 7. 2010.

[6] Stanley K. O., Clune J., Lehman J., Miikkulainen R. “Designing neural networks through neuroevolution”. Nature Machine Intelligence Journal, 1, 24-35. 2019. https://doi.org/10.1038/s42256-018-0006-z.

[7] [Dataset] Elliptic Research (2019). The Elliptic Data Set maps Bitcoin transactions to real entities belonging to licit categories versus illicit ones, v1. https://www.kaggle.com/ellipticco/elliptic-data-set.

[8] Storn R., Price K. “Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces.” Journal of Global Optimization, 11, 341- 359. 1997. https://doi.org/10.1023/A:1008202821328.

[9] Storn R., Price K. “Differential evolution for multi-objective optimization.” Proceedings of the Congress on Evolutionary Computation (CEC), 4, 2696 – 2703. 2004. https://doi.org/10.1109/CEC.2003.1299429.

[10] Pareja A., Domeniconi G., Chen J., Ma T., Suzumura T., Kanezashi H., Kaler T., Schardl T. B., Leiserson, C. E. “EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs”. Proceedings of the AAAI Conference on Artificial Intelligence, 34, 5363-5370. 2020. https://doi.org/10.1609/aaai.v34i04.5984.

[11] Weber M., Domeniconi G., Chen J., Weidele D. K. I., Bellei C., Robinson T., Leiserson C. E. “Anti-Money Laundering in Bitcoin:

Authorized licensed use limited to: Hebei University of Technology. Downloaded on January 06,2023 at 10:06:11 UTC from IEEE Xplore. Restrictions apply.

Experimenting with Graph Convolutional Networks for Financial Forensics". KDD Workshop on Anomaly Detection in Finance. 2019.

[12] Hegde R. S ."Photonics Inverse Design: Pairing Deep Neural Networks With Evolutionary Algorithms". IEEE Journal of Selected Topics in Quantum Electronics, 26, 1-8. 2019. https://doi.org/10.1109/jstqe.2019.2933796.

[13] Ilonen J., Kamarainen J. K., Lampinen J. "Differential evolution training algorithm for feed-forward neural networks". Neural Processing Letters, 17, 93-105. 2003. https://doi.org/10.1023/A:1022995128597.

[14] Chauhan N., Ravi V., Chandra K. D. "Differential evolution trained wavelet neural networks: Application to bankruptcy prediction in banks". Expert Systems with Applications Journal, 36, 7659-7665. 2009. https://doi.org/10.1016/j.eswa.2008.09.019.

[15] Slowik A., Bialko M. "Training of artificial neural networks using differential evolution algorithm". Conference on Human System Interaction (HIS). 2008. https://doi.org/10.1109/HSI.2008.4581409.

[16] Huang Z., Chen Y. "An improved differential evolution algorithm based on statistical log-linear model". Journal of Control Science and Engineering, 2013. https://doi.org/10.1155/2013/462706.

[17] Hamilton W., Ying R., Leskovec J. "Inductive Representation Learning on Large Graphs". 31st Conference on Neural Information Processing Systems (NIPS), Long Beach, CA, USA. 2017.

[18] Li Y., Tarlow D., Brockschmidt M., Zemel R. "Gated graph sequence neural networks". In International Conference on Learning Representations (ICLR). 2016.

[19] Chiang W. L., Xuanqing L., Si S., Li Y., Bengio S., Hsieh C. J. "Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks". in Proc. of KDD, ACM. 2019. https://doi.org/10.1145/3292500.3330925.

Authorized licensed use limited to: Hebei University of Technology. Downloaded on January 06,2023 at 10:06:11 UTC from IEEE Xplore. Restrictions apply.