EVOLUTION OF GRAPH CLASSIFIERS

Miguel Dominguez, Rohan Dhamdhere†, Naga Durga Harish Kanamarlapudi*, Sunand Raghupathi‡, Raymond Ptucha**
Rochester Institute of Technology*, Oak Ridge National Laboratory†, Columbia University‡

ABSTRACT

Architecture design and hyperparameter selection for deep neural networks often involves guesswork. The parameter space is too large to try all possibilities, meaning one often settles for a suboptimal solution. Some works have proposed automatic architecture and hyperparameter search, but are constrained to image applications. We propose an evolution framework for graph data which is extensible to generic graphs. Our evolution mutates a population of neural networks to search the architecture and hyperparameter space. At each stage of the neuroevolution process, neural network layers can be added or removed, hyperparameters can be adjusted, or additional epochs of training can be applied. Probabilities of the mutation selection based on recent successes help guide the learning process for efficient and accurate learning. We achieve state-of-the-art on MUTAG protein classification from a small population of 10 networks and gain interesting insight into how to build effective network architectures incrementally.

***Index Terms*—** deep learning, evolution, convolutional neural networks, graph convolution

1. INTRODUCTION

Neural network designers are faced with a plethora of architectural decisions. With initial model selection complete, the designer must iterate hyperparameter tuning with network modifications. Further, manually chosen hyperparameters or grid/random searches can inject human biases into the hyperparameter search. This process is manual, tedious, and often more of an art than a science. An algorithm that could efficiently search through the architecture and hyperparameter space will help advance AI development. Several recent works have proposed automatic learning algorithms for image classification tasks using evolution [1], genetic algorithms [2], and reinforcement learning [3, 4]. We propose extending these ideas to various graph datasets (see Figure 1). Graphs are interesting because definitions of basic operations such as convolution and pooling are up for debate. An architecture search framework for graph data could not only learn an effective architecture for these types of datasets, but also quantify the effectiveness of different methods. We propose an evolution search with a reward mechanism that chooses mutations from a dictionary of neural network layers. The probabilities of each mutation are adjusted periodically based on whether the mutation tends to improve or worsen the neural network's performance. The probabilities of each mutation over time can be used as a measurement of their effectiveness during learning, giving insight into architecture design and efficacy.

Fig. 1: A population of graph neural networks is maintained. Random members of the population are chosen and compared on a classification task, with the winner being mutated in an attempt to improve upon the previous result.

Our contributions are as follows:

- We introduce a probabilistic approach for mutation selection that rewards successful mutations, allowing one to evaluate over the learning cycle which architectural and hyperparameter decisions are the most successful.
- To the best of our knowledge, this is the first research to apply evolution-based neural network creation methods to graph neural networks.
- We propose a novel vertex attention layer and evaluate it in evolutionary search, showing that it is useful in shallow neural networks.
- We achieve state of the art on MUTAG, a protein classification dataset, evolving from a population of only 10 networks, compared to 1000 in other evolution work.
- Our framework is open source at `https://github.com/rohand24/GraphCNN_evolution`.

2. RELATED WORK

2.1. Neural Network Evolution

Research to automate the design of neural networks has a long but sporadic history. Miller et al. [5] used genetic-search based methods to evolve Multi-Layer Perceptron

978-1-7281-4352-1/19/$31.00

(MLP) based neural networks to automate the task of network design. Initially, the evolution was only restricted to evolving weights of static architectures. Stanley et al. [6] introduced the addition of new connections between neurons and splitting existing connections by inserting new neurons. These mutations work well for shallow networks, but may not be sufficient for deep networks. Current research directions focus on reinforcement learning and evolutionary strategies. Both these directions have focused on improving Convolutional Neural Network (CNN) architectures designed manually by humans.

Reinforcement learning based approaches have been able to achieve success on real-world image classification tasks. Most reinforcement learning based approaches use indirect encoding schemes for network representation. Zoph et al. [3] used reinforcement learning on a deeper fixed-length architecture, adding one layer at a time. Their mutations included addition/removal of skip connections as well as tunable hyperparameters. Baker et al. [4] used Q-learning to decide the number of layers in the network.

Evolutionary approaches [1,2,7] have been used to evolve architectures and tune hyperparameters simultaneously. More recent approaches [7] added weight inheritance for architectures. Suganuma et al. [2] used a direct encoding approach using Cartesian Genetic Programming (CGP) [8].

Our work uses evolutionary approaches similar to Real et al. [1] and the Graph CNN defined by Petroski Such et al. [9] to introduce an evolutionary search algorithm for Graph CNNs. We extend these methods by introducing more mutations and making mutation selection adaptive.

2.2. Graph Convolutional Neural Networks

Graphs are a tuple $(\boldsymbol{V}, \boldsymbol{A})$ where $\boldsymbol{V} \in \mathbb{R}^{N \times F}$ is a vertex matrix with N vertices and F features per vertex and $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is an adjacency matrix where entry a_{ij} is nonzero if there is a connection between vertices i and j. Learning convolutions on this data is challenging because the data is less structured than array-based data (such as audio or images).

Several early Graph CNNs were based on spectral graph convolutions [10–12]. The basic principle is that the eigenvectors of the graph Laplacian form an analogue to the Discrete Fourier Transform matrix. Convolution in the graph domain is the same as multiplication in this spectral domain. Filter taps scale the graph frequencies. This approach is useful for datasets where the adjacency matrix of the graph is the same for every sample. However, if the adjacency matrix varies, the DFT matrix for each sample is different and the weights do not generalize.

Spatial networks [9, 13–15] attempt to define filters that directly operate on the graph structure. Many of them are polynomial filters where a scalar filter tap is learned on each polynomial term of the graph structure as in (1). The h terms are the learnable filter taps and $\boldsymbol{A}$ is the graph structure adjacency matrix. $\boldsymbol{A}$ can be replaced with the Laplacian [14] or Chebyshev polynomials of the eigenvalues of the Laplacian [15].

$$\boldsymbol{H} = h_0\boldsymbol{I} + h_1\boldsymbol{A} + h_2\boldsymbol{A}^2 + h_3\boldsymbol{A}^3 \ldots h_k\boldsymbol{A}^k \qquad (1)$$

Each $\boldsymbol{A}^k$ term can be interpreted as a collection of edges from the current vertex to a vertex k hops away. The advantage of these networks is that they can generalize to different graph structures.

3. METHOD

We introduce an evolutionary game that attempts to learn the best neural network model by building and training neural networks, one layer at a time. Real et al. [1] trained competitive networks on the CIFAR10 and CIFAR100 [16] datasets by starting with a population of weak (single layer MLP) classifiers. Over time, these networks mutate to produce stronger networks. Two random networks in the population are chosen and their validation accuracies are compared. The winner is first duplicated, then mutated (e.g. add a convolution layer, change the learning rate, etc.). The loser is deleted. Each layer has a ReLU activation applied.

We propose a similar process on graph datasets. We repeatedly compare two networks and mutate the winner. Our set of mutations is designed for neural networks that operate on graph datasets, which is a superset of image datasets.

3.1. Mutations

1. Add/Remove 1×1 convolution with random number of filters (COO)
2. Add/Remove graph convolution with random number of filters (GC)
3. Add/Remove a fully connected layer with a random number of hidden neurons (FC)
4. Add/Remove a Graph pooling layer with random pooling ratio (GP)
5. Add/Remove a graph attention layer (ATT)
6. Add/Remove a skip connection (SKP)
7. Change number of filters of any existing convolution type layer.
8. Change learning rate
9. Change Regularization (ℓ_2 or ℓ_1) λ parameter

Learning rate, regularization, skip connections, fully connected layers, and 1×1 convolutions are the same as in standard neural networks. The remaining operations are defined below. We learn over a population of 10 models, compared to a population of 1000 in the image classifier evolution work of Real et al. [1]

3.1.1. Graph Convolution

We use the graph convolution method in Petroski Such, et al. [9] and Sandryhaila, et al. [13], described in (2).

$$\boldsymbol{V}' = \boldsymbol{V}\boldsymbol{H_0} + \boldsymbol{A}\boldsymbol{V}\boldsymbol{H_1} \tag{2}$$

The first term in (2) is a 1×1 convolutional filtering of each individual vertex, representing the "self-connection" term in (1). The second term aggregates all vertices in a neighborhood ($\boldsymbol{AV}$) and then filters the result $\boldsymbol{H_1}$. This is the second term of (1). The filter matrices $\boldsymbol{H_0}, \boldsymbol{H_1} \in \mathbb{R}^{F_1 \times F_2}$ represent F_2 filters learning linear combinations of F_1 input features.

3.1.2. Graph Pooling

We use the graph pooling method described by Petroski Such et al. [9]. This pooling operation learns a graph convolution as in (2), but then treats the output as an embedding matrix. The graph vertices and adjacency matrix are reduced as in (3).

$$\begin{aligned} \boldsymbol{V}_{pool} &= \boldsymbol{V}'^{T}\boldsymbol{V} \\ \boldsymbol{A}_{pool} &= \boldsymbol{V}'^{T}\boldsymbol{A}\boldsymbol{V}' \end{aligned} \tag{3}$$

Each output vertex is a combination of every input vertex.

3.1.3. Vertex Attention

We propose a vertex attention layer to emphasize certain vertices over others. The attention weights are formed by a 1×1 convolution with a softmax applied, as in (4).

$$\begin{aligned} \boldsymbol{\beta} &= \boldsymbol{V}\boldsymbol{H}, \boldsymbol{V} \in \mathbb{R}^{N \times F_1}, \boldsymbol{H} \in \mathbb{R}^{F_1 \times F_2} \\ \alpha_n &= \frac{e^{\max_{f_2} \boldsymbol{\beta}_{nf_2}}}{\sum_j^N e^{\max_{f_2} \boldsymbol{\beta}_{jf_2}}} \\ \boldsymbol{V}'_n &= (\alpha_n + 1)\boldsymbol{V}_n \end{aligned} \tag{4}$$

First, we filter the vertices with a 1×1 convolution to produce $\boldsymbol{\beta}$. Then, the filtered features are flattened along the f_2 dimension with a max function to produce a set of logits. The logits have a softmax function applied to produce a set of weights $\alpha_n, 1, 2, \ldots N$ that scale each vertex's features. We multiply each vertex by $(\alpha_n + 1)$ so that the features are not attenuated, but amplified according to the learned importance.

3.2. Adaptive Mutation Strategy

Initially, mutations are chosen with equal probability. We adaptively update mutation probabilities based on recent performance. For a series of C cycles, a score is kept that measures the effectiveness of each mutation, m. After each cycle c, mutation m is given a reward based on whether it improved, leading to a score s_m as defined in (5).

$$s_{m(c+1)} = \begin{cases} p_m & c \bmod C = 0 \\ s_{m(c)}(1+r) & m \text{ improves performance} \\ s_{m(c)}(1-r) & m \text{ worsens performance} \\ s_{m(c)} & m \text{ not chosen in cycle } c \end{cases} \tag{5}$$

After C cycles, the scores are converted into the probabilities for the next C cycles by normalizing, as in (6).

$$p_m = \frac{s_{mC}}{\sum_{n=1}^{M} s_{nC}} \tag{6}$$

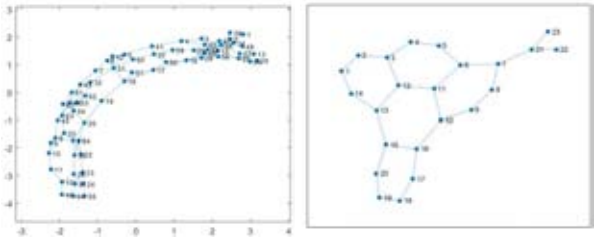

Fig. 2: Two example protein graphs in the ENZYMES (left) and MUTAG (right) datasets.

4. RESULTS

We evaluate our method on two protein classification datasets, ENZYMES and MUTAG [17]. These are both chemical compound classification datasets where the vertices and edges encode the chemical structure of the sample. Two example graphs are in Figure 2. Each vertex has a single feature and the edges are binary: 1 if a connection exists between two vertices and 0 if it does not. The task of ENZYMES is to classify enzymes into one of six broad classes, and the task of MUTAG is to identify whether the compound would have a mutagenic effect on *Salmonella typhimurium*. Table 1 compares our method to recent literature.

The best architecture that our evolution algorithm found for ENZYMES was COO256, COO128, GC256, GC512, FC214, FC6. The best for MUTAG is FC328, FC284, FC2. The results of this experiment compare against the literature in Table 1. We achieve a perfect score on MUTAG. It is surprising that we are able to achieve a perfect score with just carefully-tuned fully-connected layers on the vertices. This

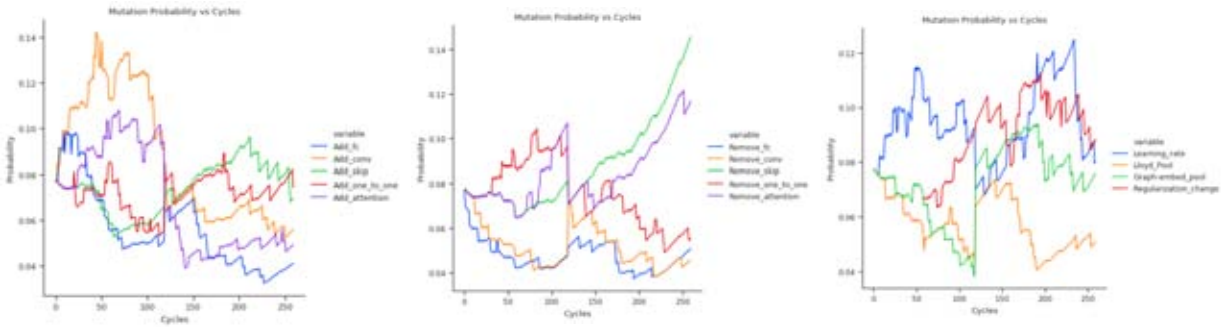

Fig. 3: Probability of different mutations over time in the ENZYMES experiment. Split into three figures for clarity.

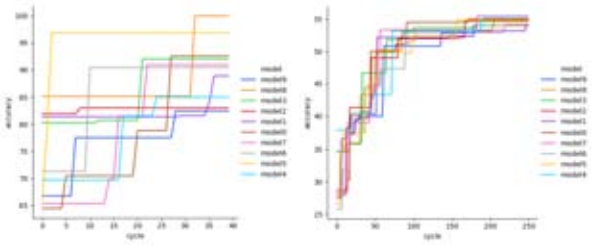

Fig. 4: Classification accuracy of each member of the population over the course of the experiment. **Left**: MUTAG experiment. **Right**: ENZYMES experiment.

may be all this dataset requires, and more complex networks only make it more difficult to learn and generalize. Although we do not achieve state of the art on ENZYMES, from the learning curve in Fig. 4, it appears that with enough time the network could improve. One inhibiting factor is that the small population size (10 networks) can lead to cases where the population all have similar features, and improvements primarily come from adjusting the learning rate and regularization.

Plotting the probabilities of each mutation over time gives us insight into what mutations are most effective, and when. In Fig. 3 there appears to be two stages in our training: an early and a late stage. The probabilities of most operations significantly shift halfway through the experiment. For example, in the beginning (when the networks are only a couple of fully-connected layers), adding more convolution layers help, but pooling layers do not. However, as the networks get deeper, pooling layers that increase receptive field become important. Additional weighted layers such as graph convolutions or fully connected layers are no longer helpful. Adjusting the learning rate is useful consistently throughout the experiment, which is intuitive as each cycle is run for a fixed number of iterations. Our graph attention layer seems to be useful in a shallow networks, but not deeper networks. 1×1 convolutions are useful throughout to adjust individual vertex features to fit the data.

Table 1: Protein Classification Accuracy Using Evolution.

Method	MUTAG	ENZYMES
PATCHY-SAN[18]	92.63%	-%
Deep WL[19]	87.44%	53.43%
Donini et al.[20]	93.00%	-%
structure2vec[21]	88.28%	61.10%
WL[17]	83.78%	59.05%
WL-OA[22]	84.5%	59.90%
Morris et. al.[23]	87.2%	61.80%
ECC [24]	89.44%	53.50%
Cangea et. al. [25]	-%	64.17%
DiffPool [26]	-%	**64.23**%
Ours	**100.00**%	55.67%

5. CONCLUSION

We introduced an evolutionary framework for graph neural network learning. Networks in the population are learned efficiently as mutations are encouraged or penalized guided by prior success. This framework is useful as an alternative to manual search strategies in achieving results comparable to state-of-the-art, and can additionally function as an alternative means of evaluating various graph neural network layers. Whereas traditionally a neural network layer is evaluated by testing whether it improves a limited number of baselines, this method conducts a search where mutations are chosen based on past performance. Our method achieves state-of-the-art performance with no manual intervention by humans on MUTAG from a population of only 10 networks, showing that this method can be economical on small datasets.

6. REFERENCES

[1] Esteban Real et al., "Large-Scale Evolution of Image Classifiers," in *International Conference on Machine*

Learning, 2017, pp. 2902–2911.

[2] Masanori Suganuma et al., "A Genetic Programming Approach to Designing Convolutional Neural Network Architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, NY, USA, 2017, GECCO '17, pp. 497–504, ACM.

[3] Barret Zoph and Quoc V Le, "Neural Architecture Search with Reinforcement Learning," 2017.

[4] Bowen Baker et al., "Designing neural network architectures using reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[5] Geoffrey F Miller et al., "Designing Neural Networks using Genetic Algorithms.," in *ICGA*, 1989, vol. 89, pp. 379–384.

[6] Kenneth O. Stanley and Risto Miikkulainen, "Evolving Neural Networks Through Augmenting Topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, June 2002.

[7] Chrisantha Fernando et al., "Convolution by evolution: Differentiable pattern producing networks," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. 2016, pp. 109–116, ACM.

[8] "Cartesian Genetic Programming - CGP-Library," .

[9] F. P. Such et al., "Robust Spatial Filtering With Graph Convolutional Neural Networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 6, pp. 884–896, Sept. 2017.

[10] Joan Bruna et al., "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR)*, 2014.

[11] Mikael Henaff et al., "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[12] Michael Edwards and Xianghua Xie, "Graph convolutional neural network," in *British Machine Vision Conference*, Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith, Eds. BMVA Press.

[13] Aliaksei Sandryhaila and José MF Moura, "Discrete signal processing on graphs," *Signal Processing, IEEE Transactions on*, vol. 61, no. 7, pp. 1644–1656, 2013.

[14] Thomas N. Kipf and Max Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of ICLR 2017*, vol. 1609.

[15] Michal Defferrard et al., "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. pp. 3844–3852, Curran Associates, Inc.

[16] Alex Krizhevsky and Geoffrey Hinton, "Learning multiple layers of features from tiny images," 2009.

[17] Nino Shervashidze et al., "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[18] Mathias Niepert et al., "Learning convolutional neural networks for graphs," in *Proceeding of the 33rd International Conference on Machine Learning*, 2016, pp. 2014–2023.

[19] Pinar Yanardag and SVN Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1365–1374.

[20] Michele Donini et al., "Fast hyperparameter selection for graph kernels via subsampling and multiple kernel learning," 2017.

[21] Leonardo F.R. Ribeiro et al., "Struc2vec: Learning Node Representations from Structural Identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2017, KDD '17, pp. 385–394, ACM.

[22] Nils M Kriege et al., "On valid optimal assignment kernels and applications to graph classification," in *Advances in Neural Information Processing Systems*, 2016, pp. 1623–1631.

[23] C. Morris et al., "Glocalized Weisfeiler-Lehman Graph Kernels: Global-Local Feature Maps of Graphs," in *2017 IEEE International Conference on Data Mining (ICDM)*, Nov. 2017, pp. 327–336.

[24] Martin Simonovsky and Nikos Komodakis, "Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*, 2017.

[25] Cătălina Cangea et al., "Towards sparse hierarchical graph classifiers," in *NIPS 2018 Relational Representation Learning Workshop*, 2018.

[26] Rex Ying et al., "Hierarchical graph representation learning with differentiable pooling," in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, USA, 2018, NIPS'18, pp. 4805–4815, Curran Associates Inc.