ARL-TR-9988 • Sep 2024

Functional Graph-to-Graph Transformation Layers

by Berend Christopher Rinderspacher

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-9988 ● SEP 2024

Functional Graph-to-Graph Transformation Layers

by Berend Christopher Rinderspacher
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
September 2024	Technical Report	10/01/2023	09/30/2024

4. TITLE AND SUBTITLE

Functional Graph-to-Graph Transformation Layers

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)

Berend Christopher Rinderspacher

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

US Army Research Laboratory
ATTN: FCDD-RLA-MC
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER

ARL-TR-9988

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

primary author's email: <berend.c.rinderspacher.civ@army.mil>.

14. ABSTRACT

We derive three types of layers for use in graph-to-graph machine learning models on labeled graphs. Three layers are introduced: a vertex preserving layer; a graph contraction layer, which reduces the number of vertices; and a graph expansion layer, which increases the number of vertices. All three layers are constructed using the theory of intersection graphs. The framework is tested against the Modified National Institute of Standards and Technology image classification task, for which a very small model of just over 2000 parameters achieves over 95% accuracy in both training and test sets, compared with the state-of-the-art performance of ProjectionNet with over 77,000 parameters or the parametric matrix model with just under 5000 parameters.

15. SUBJECT TERMS

machine learning; graph transformations; computer vision; Network, Cyber, and Computational Sciences; image classification; artificial neural networks

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	17

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
B Christopher Rinderspacher	301-580-8974

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

Contents

1. Introduction

Labeled graphs are pervasive in science and engineering.[1] They are used to encode relationships of all kinds, such as networks of cities, electrical grids, chemical compounds, or reaction networks. In computer science, programs and functions are represented as labeled graphs for execution. A large component of science is the approximation of reality through functions and thereby finding graphs to compute them.

Graph machine learning encompasses the gamut of problems addressed with graphs, such as representation learning,[2] function approximation,[3] or optimization.[4] Graph neural networks (GNNs)[5] are usually graph-preserving graph convolutional networks[6] or graph transformers.[7]

In this report, the fact that every graph can be constructed from intersections of sets is used to expand graph-to-graph computations.[8]

2. Background and Methods

A graph G is defined by its vertex set V and its edge set $E \subset V \times V$. An intersection graph is a graph with a vertex (multi-)set comprised of sets $v \subset X$ with corresponding edge set $E = \{(v, v') | v \cap v' \neq \emptyset\}$. The vertices of a labeled graph $G = (V, E, L)$ are associated with labels $\ell : V \to L, v \mapsto l$, where ℓ is an injection mapping vertices to labels. In many applications, these labels are vectors from a vector space.

The set of all finite graphs with labels in L is $\Gamma = \{(V, E, L) | E \subset V \times V, V \subset W\}$, where W is countable. The problem at hand is the modeling of functions $f : \Gamma \to \Gamma$. Artificial neural networks (ANNs) approximate functions via function composition of primitive functions called "layers,"

$$f(x) \approx (l_{1,\theta_1} \circ l_{2,\theta_2} \circ \cdots \circ l_{n,\theta_n})(x), \tag{1}$$

where $l_{i,\theta}$ is a function parameterized by θ. When graphs are the domain of an ANN, the ANN is referred to as a GNN. GNNs have four layer types:

1. Keeping both edges and vertices constant (graph-preserving)
2. Keeping only the number of vertices constant

3. Reducing the number of vertices

4. Increasing the number of vertices

The majority of GNNs (e.g., graph convolutional networks[9]) focus only on the first option. In the following, we explore the remaining three options. The graph in the proposed approaches is defined via set intersections computed from the labels.

2.1 Graph Generation Layer

The first option computes new edges and labels for a set of vertices. For an array of objects, a position generator P computes positions $p_k = P(o_k) \in \mathbb{R}^n$ and a radius generator R computes radii $r_k = R(o_k) \in (0, \infty)^n$ from each object. Then for each tuple (o_k, p_k, r_k), the set $N_k = \{l | p_l \in B(p_k, r_k)\}$ can be computed, where $B(x, b) = \{y | \|y_i - x_i\| < b_i \forall i \in 1 \dots n\}$. Then, a directed graph G can be constructed by setting edges from node k to node l if $l \in N_k$. It is noteworthy that the sets N_k are the nearest neighbors of each node k. Furthermore, the graph is scaling and translationally invariant; i.e., $N_k = \{l | s(p_l - t) \in B(s(p_k - t), sr_k)\}$ remains the same for all $s \in \mathbb{R}^+$ and $t \in \mathbb{R}^n$.

New labels are computed from the graph via

$$o_i = \sum_{j \in N_k} o_j \tau(p_i, p_j, r_i), \tag{2}$$

where τ is a weighting function such that $\tau(x, y, b) = 0 \forall y \notin B(x, b)$. Due to the scaling invariance for generation of graph G, it may be useful for τ to be scale-invariant as well. Since all graphs can be computed in this manner, the worst case computational complexity is $O(\#V^2)$.

2.2 Decimation Layer

A decimation layer uses a parameter r to establish which nodes are to be consolidated. This parameter can be fixed or learned. As before, a position generator computes a position for each object, but the single parameter r is used to establish the sets $B(p_k, r)$. If r is fixed and used in conjunction with the graph generation layer, then r effectively forces a minimum distance between nodes. With this collection

(multiset) $V = \{N_k\}$ of sets in hand, a minimum cover of all sets can be computed:

$$\begin{aligned} W^* = \arg\min_{W \subset V} \quad & \#W \\ \text{subject to} \quad & \bigcup_{w \in W} w = \bigcup_{v \in V} v. \end{aligned} \tag{3}$$

Each set $w \in W^*$ corresponds to a node. As in the generation layer, new objects o_i are computed from the sets $w_i \in W$ via Eq. 2.

It is noteworthy that the resultant graph is equivalent to a graph for which the dropped vertices have radii and labels of zero. Thus, the graph generation layer is capable of simulating reduced-order graphs, albeit without potential memory or computational savings.

2.3 Expansion Layer

As with the decimation layer, the expansion layer relies on a parameter ξ in addition to the position generator. From the positions, an n-dimensional tessellation is computed with the positions at the vertices. For each n-dimensional face with a diameter larger than ξ, a new node is introduced at the centroid of the face according to Eq. 2 with $r = d/2$, where d is the diameter of the face.

As with the decimation layer, the graph generation layer can emulate an expansion layer. While the decimation layer is constrained by the graph with a single node, expansion layers are limited to graphs with $d + \binom{d}{n-1}$ vertices, where d is the order of the incoming graph. Hence, $\binom{d}{n-1}$ vertices with radius and labels equal to zero have to be available to the graph generation layer to emulate a graph expansion layer.

2.4 Image Classification Architecture

The test-bed of the aforementioned graph generation layer will be image classification into 10 possible classes. Images are labeled lattice graphs. For simplicity, positions and radii are included explicitly in the labels of each node. Hence, the label of a pixel consists of its color c, its integer position in the plane, and the constant radius of one. So the input layer takes an array of tuples $(c_{x,y}, p_{x,y}, (1, 1))$ for each pixel at (x, y). These are first expanded by a linear layer with $\tanh$ activation to (o_k, p_k, r_k), where the dimension of o_k is 11 to accommodate the final number of

classes. For all intermediate activations, we use the rectified linear unit (ReLU) because of its approximation properties.[10] The model architecture was implemented in the PyTorch machine learning framework.[11]

Listing 1: Model for constructing the graph representation.

```
def setup():
    return Cat(torch.nn.Identity(), image_pos),
    SingleInputMultiOutput(#1
        torch.nn.Sequential(
            torch.nn.Linear(3,11),
            torch.nn.Tanh()
        ),
        torch.nn.Sequential(
            Narrow(2, 1, 2),
            Center(1)
        )
    ),
    SingleInputMultiOutput(
        Get(0), # Values
        Get(1), # Positions
        torch.nn.Sequential(
            Get(1),
            FuncModule(image_bounds),
            Permute((0,2,1))
        )
    )
```

In Listing 1, `Cat` concatenates two arrays in the last dimension, `SingleInputMultiOutput` is a module that takes a single input and computes the output for each module given in its constructor arguments as a tuple of outputs, and `Get(n)` returns the nth item in a tuple. `FuncModule` is a wrapper class that applies a static function to inputs.

In Listing 2, `TensorRelator` computes the graph from (o_k, p_k, r_k) and then uses `TensorReductor` to compute the new node labels, (i.e., it corresponds to τ in Eq. 2).

Listing 2: GNN for Modified National Institute of Standards and Technology (MNIST) classification.

```
torch.nn.Sequential(
    setup(),
    SingleInputMultiOutput(
        # Takes 3 inputs returns 1 tensor
```

```
        GL.TensorRelator(TensorReductor),
        Get(1) # old positions
    ),
    consolidate_values_and_positions(),
    # Compute new positions
    SingleInputMultiOutput(
        Get(0), # old values
        update_positions(),
        new_bounds()
    ),
    SingleInputMultiOutput(
        GL.TensorRelator(TensorReductor),
        Get(1) # old positions
    ),
    consolidate_values_and_positions(),
    # Compute new positions
    SingleInputMultiOutput(
        Get(0), # old values
        update_positions(),
        new_bounds(),
    ),
    GL.TensorRelator(TensorReductor),
    torch.nn.Linear(11, 11),
    torch.nn.ReLU(),
    torch.nn.Linear(11,11),
    torch.nn.ReLU(),
    torch.nn.Linear(11,11),
    MeanPool(1),
    torch.nn.Linear(11,11),
    torch.nn.ReLU(),
    torch.nn.Linear(11,11),
    torch.nn.ReLU(),
    torch.nn.Linear(11,11),
    torch.nn.ReLU(),
    torch.nn.Linear(11,10),
    torch.nn.Softmax(dim=1)
)
```

In this case, $\tau(x, y, r) = \prod_i ReLU(1 - ((y_i - x_i)/r_i)^6)^2$. This choice satisfies the requirement that $y \notin B(x, r) \Rightarrow \tau(x, y, r) = 0$ as well as being continuously differentiable and varying slowly around x. Overall, the model includes three graph generation layers followed by three linear layers activated by ReLU. Finally,

the information is collected by averaging over all nodes and fed through another three-layer fully connected ANN activated by ReLU and the softmax activation for modeling the classification probabilities.

After the graph generation layer has computed new labels, these new labels are processed further by joining the position information. Listing 3 gives the ANN implementation. Again, the ReLU is used for activation in three layers. Three layers are chosen to comply universal approximation guarantees.[10]

Listing 3: Model to incorporate position information into values.

```
def consolidate_values_and_positions():
    return SingleInputMultiOutput(
        # This incorporates the position into value generation
        torch.nn.Sequential(
            AllCat(),
            torch.nn.Linear(13,11),
            torch.nn.ReLU(),
            torch.nn.Linear(11,11),
            torch.nn.ReLU(),
            torch.nn.Linear(11,11),
            torch.nn.ReLU()
        ),
        Get(1)
    )
```

Listing 4 provides the model for the position generator. This model merely updates the old positions and then centers the positions because τ, as defined above, is translationally invariant.

Listing 4: Model for position generator.

```
def update_positions():
  return Add( # skip connection
                Get(1), # old positions
                torch.nn.Sequential(
                    Get(0),
                    torch.nn.Linear(11,3),
                    torch.nn.ReLU(),
                    torch.nn.Linear(3,3),
                    torch.nn.ReLU(),
                    torch.nn.Linear(3,3),
                    torch.nn.ReLU(),
```

```
                torch.nn.Linear(3,2, bias=False),
                Center(1, scale=True)
            ) # modification
        )
```

`Center(1, scale=True)` not only centers but also rescales positions by the standard deviation to preserve interpretability and optimization smoothness. The embedding dimension for positions is two. Hence, only intersection graphs embeddable in the plane are possible in this implementation.

Listing 5 covers the computation of the radii.

Listing 5: Model for radii computation.

```
def new_bounds():
    return torch.nn.Sequential(
                Get(0),
                torch.nn.Linear(11,3),
                torch.nn.ReLU(),
                torch.nn.Linear(3,3),
                torch.nn.ReLU(),
                torch.nn.Linear(3,3),
                torch.nn.ReLU(),
                torch.nn.Linear(3,2),
                torch.nn.Sigmoid()
            )
```

Again, three ReLU layers are computed and finally piped through the sigmoid function. The final activation ensures that the radii are between 0 and 1, which (in conjunction with the rescaling in Listing 4) defends against trivial complete graphs. Complete graphs lead to labels that are approximately the same for all nodes.

3. Results

The MNIST dataset[12] contains 60,000 images with dimensions 28×28 for training and 10,000 images for testing. The training dataset was split further into a training and validation set with a 70/30 ratio, leaving 42,000 images for training. The training set was randomly split further into batches of 32 instances for each epoch. The Adam optimization method[13] was applied with a learning rate of 5e-4. L2 regularization* was applied to all parameters with a penalty multiplier of 1e-5. The model

*The objective is penalized by $\|\theta\|_2^2$, where θ is the vector of parameters.

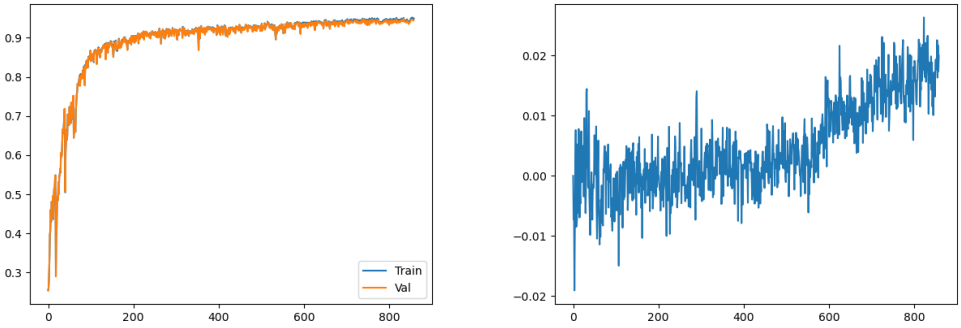

Fig. 1 Prediction accuracy on MNIST dataset for training set and validation set against training epochs (left) and loss difference between training and validation set (right)

was trained on the classification cross-entropy loss.

After 1000 epochs, the training loss was 0.092 with a classification accuracy of 95%. The corresponding validation loss and accuracy were 0.105 and 95%, respectively. The model described in Section 2.4 has 2061 parameters. Parametric matrix models[14] with just under 5000 parameters achieve an accuracy of 97.4%, whereas the ProjectionNet[15] with over 77,000 parameters achieves 95.0% accuracy.

As Fig. 1 shows, the validation losses and accuracies closely track the corresponding training losses and accuracies. The high degree of agreement between training loss and validation loss indicates a large degree of generalizability in the model.

The current implementation in Python has quadratic complexity for all graphs and cannot handle learning on graphs of varying numbers of nodes. It requires approximately 100 s over 42,000 images on a single graphics processing unit for training. Because heterogeneous numbers of nodes are not possible, neither the decimation layer, nor the expansion layer could be leveraged explicitly.

4. Conclusions

The graph generator layer allows highly efficient and performant models as demonstrated on the MNIST classification problem. Future developments will target the simultaneous training on heterogeneous input graphs, inclusion of explicit decimation and expansion layers, and speedup of training over batches by reducing computational complexity.

5. References

1. Hu W, Fey M, Zitnik M, Dong Y, Ren H, Liu B, Catasta M, Leskovec J. Open Graph Benchmark: Datasets for Machine Learning on Graphs. 2021.

2. Bengio Y, Courville A, Vincent P. Representation Learning: A Review and New Perspectives. 2014.

3. Schütt KT, Sauceda HE, Kindermans PJ, Tkatchenko A, Müller KR. SchNet – A deep learning architecture for molecules and materials. The Journal of Chemical Physics. 2018;148(24):241722.

4. Dai H, Khalil EB, Zhang Y, Dilkina B, Song L. Learning combinatorial optimization algorithms over graphs. In: Learning Combinatorial Optimization Algorithms over Graphs;.

5. Scarselli F, Yong SL, Gori M, Hagenbuchner M, Tsoi AC, Maggini M. Graph neural networks for ranking web pages. In: The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05); p. 666–672.

6. Zhang S, Tong H, Xu J, Maciejewski R. Graph convolutional networks: a comprehensive review. Computational Social Networks. 2019;6(1):11.

7. Cai D, Lam W. Graph Transformer for Graph-to-Sequence Learning. 2019.

8. Pal M. Intersection Graphs: An Introduction. 2014.

9. Coley CW, Jin W, Rogers L, Jamison TF, Jaakkola TS, Green WH, Barzilay R, Jensen KF. A graph-convolutional neural network model for the prediction of chemical reactivity. Chemical Science. 2019;10(2):370–377.

10. Kim N, Min C, Park S. Minimum width for universal approximation using ReLU networks on compact domain. 2024.

11. Paszke A et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. 2019.

12. Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998;86(11):2278-2324.

13. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. 2017.

14. Cook P, Jammooa D, Hjorth-Jensen M, Lee DD, Lee D. Parametric Matrix Models. 2024.

15. Ravi S. ProjectionNet: Learning Efficient On-Device Deep Networks Using Neural Projections. 2017.

List of Symbols, Abbreviations, and Acronyms

ANN artificial neural network

GNN graph neural network

MNIST Modified National Institute of Standards and Technology

ReLU rectified linear unit

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLB CI TECH LIB