IE643: Deep Learning : Theory and Practice **July-Dec 2021**

End-term Project Report : Trajectory Prediction using Simplicial NNs

Team Name: Sapitors *Team Members: 190100117, 19D100017*

Abstract

Simplicial neural networks are generalizations of graph neural networks that work on data in the form of simplicial complexes. These are natural multi-dimensional extensions of graphs that encode not only pairwise relationships but also higher-order interactions between vertices: allowing us to consider richer data, including vector fields and n-fold collaboration networks. The problem that we considered in our course project was of trajectory prediction, i.e, given a set of prefix nodes, we want to find the suffix node for an agent moving over a network. This is a supervised learning task, and the comparative analysis of SCoNE (the neural network under consideration) with Graph Neural Networks (precursor network) displayed better results.

1 Introduction

Path is a general term used for denoting the movement of an agent over a network. All the points (called nodes) through which the agent passes makes the path. Some examples of this are: a student surfing one website after another with some pre-defined goal (such as finding information for the IE 643 course project), a customer looking at one product after another on the Amazon app, or a truck changing roads while moving from one destination to another. Trajectory prediction is the supervised learning method of training a neural network over a set of paths, and then using this network for determining the location of an agent in the next time step, given an incomplete (prefix) path. Previous work on this task includes the usage of Graph Neural Networks, while Simplicial Neural Networks had been used for other tasks (such as labelling nodes). The paper under consideration extends these Simplicial Neural Networks for the task of trajectory prediction, and provides an admissible structure for the neural network.

This report is structured in the following manner:

We list the main contributions of our project in Section 2. Related works to the project are described in Section 3. Section 4 talks about the details of the methods and approaches used by us throughout the project. Data used by the neural networks has its details mentioned in Section 5. Experiments conducted on the neural networks mentioned in Methods and Approaches section are summarised in Section 6. Finally, the results are presented in Section 7, future work in Section 8, and the conclusions in Section 9.

2 Contributions

1. The paper [4] is a rigorous mathematical paper that skips certain mathematical preliminaries in graph theory and topology which are not common knowledge. Through an extensive literature survey, we have not only provided the necessary definitions and preliminaries, but also narrowed down only those details which are relevant for implementing SNNs for trajectory prediction.
2. We have validated the experiments conducted in paper [4], and provided a code walkthrough for ease of understanding

3. We have done a comparative analysis of the effect of different odd non-linear functions on the performance, keeping other parameters same, in order to find the optimal performing function for SCoNE, the algorithm proposed in [4]
4. Further, we have done a comparative analysis of the effect of the number of hidden units and layers on the performance, keeping the other parameters constant.

3 Related Work

This section deals with three related papers to the paper [4], the primary paper. The first is on Graph Neural Networks, while the other two are on the application of Simplicial Neural Networks to other problems.

3.1 Extrapolating Paths with Graph Neural Networks

The first paper, titled **Extrapolating Paths with Graph Neural Networks** [2] tackles the same problem that we wish to solve, but instead of using simplicial neural networks, it uses graph neural networks. As expected, the performance of this neural network architecture is poorer than Simplicial Neural Networks, but the advantage is that it provides an approximate path over several time steps (till the horizon), instead of just one subsequent time step.

The authors of the paper provide a neural network called GRETEL, which is used in a two step process to get candidate suffix paths with their probabilities, given the prefix path and the horizon. The first is to train a neural network f_θ to encode all possible information about the path prefix into a latent graph in the following manner:

$$\Phi \stackrel{\text{def}}{=} (\mathcal{V}, \mathcal{E}, w_\phi), \quad \text{with } w_\phi = f_\theta(G, \phi).$$

Note that the new graph Φ has the same vertices and edge sets as the old graph G, its edges are re-weighted so as to point towards the directions the agent is most likely to follow. This network is then applied to approximate the likelihood of any suffix s by the graph dependent model g ($\mathbf{x}_t$ captures the last known position of the agent):

$$\mathrm{P}(s \mid h, \phi, G) \simeq g(s, \mathbf{x}_t, \Phi)$$

The paper details the theory behind the training of the model, followed by determination of the graph dependent model g. Next, the authors provide experiments on three datasets: first, to check if the network can predict straight line paths (artificial data), and next on GPS Trace Extrapolation and Wikipedia User Navigation (real data). All three of these display promising results, hence validating the model.

3.2 Simplicial Neural Networks

This was the pioneering work that first introduced the notion of Simplicial Neural Networks, which operated on data on simplicial complexes. In the paper **Simplicial Neural Networks** [3], the authors first

define an appropriate concept of convolution over simplices, followed by the application of this network to find missing data in co-authorship complexes.

We first take a look at the concept of convolution, which, once defined, would directly lead us to the simplicial neural networks by its repeated application. The convolution is of the form $\psi \circ (f * \varphi_W)$. Here, φ_W is a function with small support parametrized by learnable weights W, ψ is some non linearity and bias. This method is motivated by the Hodge-de Rham theory (the details of which can be found in the book: Madsen, I. H., Tornehave, J. (1997). From calculus to cohomology: De Rham cohomology and characteristic classes. Cambridge: Cambridge University Press).

The convolution layer is then represented as:

$$\psi \circ \left(\mathcal{F}_p^{-1}(\varphi_W) *_p c\right)$$

Where $\mathcal{F}^{-1}$ refers to the inverse Fourier transform. The Fourier transform is defined on simplicial complexes as:

$$\mathcal{F}_p : C^p(K) \to \mathbb{R}^{|K_p|}$$
$$\mathcal{F}_p(c) = \left(\langle c, e_1\rangle_p, \langle c, e_2\rangle_p, \ldots, \langle c, e_{|K_p|}\rangle_p\right)$$

Where e_{is} are the Eigencochains of the Laplacians of the complex ordered by the respective Eigenvalues. The kth Laplacian of a complex S is:

$$\mathbf{L}_k = \mathbf{B}_k^* \mathbf{B}_k + \mathbf{B}_{k+1} \mathbf{B}_{k+1}^* .$$

Further, the convolution is defined as follows:

$$c *_p c' = \mathcal{F}_p^{-1} \left(\mathcal{F}_p(c)\mathcal{F}_p(c')\right)$$

Once the convolution operation has been formally defined, its repeated application at each hidden unit in every layer gives us a convolutional neural network equivalent for simplicial complexes.

The authors of this paper implemented this neural network architecture on co-authorship data, to predict missing labels (regression task). The architecture displayed promising results. The results can be found in the paper [3].

3.3 Simplicial 2-Complex Convolutional Neural Networks

This paper, **Simplicial 2-Complex Convolutional Neural Networks** [1] provides a modification to the simplicial neural network proposed in the paper [3], and applies it to MNIST data for the purpose of image classification, by first converting image data to simplicial complexes. The method is as follows:

First we revisit the concept of Laplacians of a complex. The kth Laplacian of a complex S is-

$$\mathbf{L}_k = \mathbf{B}_k^* \mathbf{B}_k + \mathbf{B}_{k+1} \mathbf{B}_{k+1}^* .$$

Here, **B**s denote the matrix representation of the boundary operator.

They then define some matrices that are required in the algorithm as follows:

$$\widetilde{\mathbf{D}}_2 = \max(\mathrm{diag}(|\mathbf{B}_1|\,\mathbb{1}), \mathbf{I}) \qquad \widetilde{\mathbf{D}}_3 = \mathbf{I}$$

$$\mathbf{D}_1 = 2\,\mathrm{diag}(|\mathbf{B}_1|\,\mathbf{D}_2\,\mathbb{1}) \qquad \mathbf{D}_2 = \max(\mathrm{diag}(|\mathbf{B}_2|\,\mathbb{1}), \mathbf{I}) \qquad \mathbf{D}_3 = \frac{1}{3}\mathbf{I}$$

$$\mathbf{D}_4 = \mathbf{I} \qquad \mathbf{D}_5 = \mathrm{diag}(|\mathbf{B}_2|\,\mathbb{1})$$

$$\widetilde{\mathbf{L}}_0 = \mathbf{B}_1\,\widetilde{\mathbf{D}}_3\,\mathbf{B}_1^*\,\widetilde{\mathbf{D}}_2^{-1}$$

$$\widetilde{\mathbf{L}}_1 = \mathbf{D}_2\,\mathbf{B}_1^*\,\mathbf{D}_1^{-1}\,\mathbf{B}_1 + \mathbf{B}_2\,\mathbf{D}_3\,\mathbf{B}_2^*\,\mathbf{D}_2^{-1}$$

$$\widetilde{\mathbf{L}}_2 = \mathbf{D}_4\,\mathbf{B}_2^*\,\mathbf{D}_5^{-1}\,\mathbf{B}_2$$

$$\mathbf{A}_0^u = \widetilde{\mathbf{D}}_2 - \mathbf{B}_1\,\widetilde{\mathbf{D}}_3\,\mathbf{B}_1^* \qquad \mathbf{A}_1^u = \mathbf{D}_2 - \mathbf{B}_2\,\mathbf{D}_3\,\mathbf{B}_2^*$$

$$\mathbf{A}_1^d = \mathbf{D}_2^{-1} - \mathbf{B}_1^*\,\mathbf{D}_1^{-1}\,\mathbf{B}_1 \qquad \mathbf{A}_2^d = \mathbf{D}_4^{-1} - \mathbf{B}_2^*\,\mathbf{D}_5^{-1}\,\mathbf{B}_2 .$$

Finally, the S-2CCNN convolution layer is defined as follows:

$$\widetilde{\mathbf{A}}_0^u = (\mathbf{A}_0^u + \mathbf{I})(\widetilde{\mathbf{D}}_2 + \mathbf{I})^{-1} \qquad \widetilde{\mathbf{A}}_1^u = (\mathbf{A}_1^u + \mathbf{I})(\mathbf{D}_2 + \mathbf{I})^{-1}$$

$$\widetilde{\mathbf{A}}_1^d = (\mathbf{D}_2 + \mathbf{I})(\mathbf{A}_1^d + \mathbf{I}) \qquad \widetilde{\mathbf{A}}_2^d = (\mathbf{D}_4 + \mathbf{I})(\mathbf{A}_2^d + \mathbf{I}).$$

This layer is used to make a neural network architecture, which is applied on MNIST data after converting to simplicial complexes. The image classification results are compared between three architectures: First with a 1-D Conv layer followed by a FC layer. Second with a GConv added before the 1-D conv, while the third with a SCConv applied before the 1-D Conv. The results displayed superior performance of the SCConv by quite a margin.

4 Methods and Approaches

This section details all the methods and approaches followed throughout the project, in order to get a complete view.

4.1 Pre Mid-Term Review

We first began with the mathematical preliminaries required for this project, which are listed below:

- **Graph:** A mathematical structure consisting of (V,E,ϕ) V: Vertices (Nodes), E: Edges, ϕ: map that assigns an edge to a pair of vertices. It is useful for entities that have pairwise relations between themselves. Each vertex and edge has a vector corresponding to it, which consists of the features.

E.g.: A graph that details a friend group can have the personal details such as name, age, school at the nodes, while the edges will have pairwise details like the number of classes shared together, number of messages shared between each other, etc.

- **Path:** A sequence $[i_1, i_2, i_3, ..., i_m]$ of vertices of a graph G
- **Simplex:** It is the generalisation of a triangle or tetrahedron to arbitrary dimensions. It can be thought of as a mathematical entity having faces. If d is the dimension of the simplex, then, d=0 is a point, 1, a segment, 2, a triangle, and 3 a tetrahedron. More formally, a k-simplex is a k-dimensional polytope which is the convex hull of its k+1 vertices. Suppose the k+1 points $u_0, \ldots, u_k \in \mathbb{R}^k$ are affinely independent, which means $u_1 - u_0, \ldots, u_k - u_0$ are linearly independent. Then, the affine combinations of $u_0, \ldots, u_k$ determine the set of points of the simplex.
- **Simplicial Complex:** A simplicial complex $\mathcal{K}$ is a set of simplices that satisfies the following conditions:
  1. Every face of a simplex from $\mathcal{K}$ is also in $\mathcal{K}$.
  2. The non-empty intersection of any two simplices $\sigma_1, \sigma_2 \in \mathcal{K}$ is a face of both σ_1 and σ_2.
- **Boundary Operator:** set of linear maps between chains (analogous to matrices). For an oriented k-simplex $\sigma = [i_0, i_1, ..., i_k]$, the boundary operator $\partial_k : C_k \to C_{k-1}$ is defined as:

$$\partial_k \sigma = \sum_{j=0}^{k} (-1)^j [i_0, \ldots, i_{j-1}, i_{j+1}, \ldots, i_k].$$

After acquiring the necessary background, we moved onto the problem statement:

The problem we consider entails estimating the likelihood P(s | p, G) of a suffix path s = $(v_{t+1}, ..., v_{t+h})$ given a prefix path p = $(v_1, ..., v_t)$ on graph G. In simple words, we assume an agent is moving from one node to another. Given the location history of our agent, we wish to predict the locations at which our agent will be in the future.

To get started, we looked at some preliminary works on trajectory prediction. These involved the use of Graph Neural Networks. A broad overview of all types of Graph Neural Networks was found in the paper [5]. The application of Graph Neural Networks to trajectory prediction was found in the paper [2] (the details of which can be found in Section 3)

We finally moved onto the SCoNe algorithm as suggested in the paper. The algorithm is as follows:

Algorithm 1 SCoNe for Trajectory Prediction

Input: partial trajectory $[i_0, i_1, \ldots, i_{m-1}]$
Parameters:
boundary operators $\{\partial_k\}_{k=0}^{2}$
number of layers L
hidden dimensions $\{F_\ell\}_{\ell=0}^{L+1}, F_0 = F_{L+1} = 1$
weight matrices $\{\{\mathbf{W}_k^\ell \in \mathbb{R}^{F_\ell \times F_{\ell+1}}\}_{\ell=0}^{L}\}_{k=0}^{2}$
activation function ϕ
Initialize: $\mathbf{c}_1^0 \in \mathcal{C}_1, \mathbf{c}_1^0 = 0.$
for $j = 0$ **to** $m - 2$ **do**
    $\mathbf{c}_1^0 \leftarrow \mathbf{c}_1^0 + [i_j, i_{j+1}]$
end for
for $\ell = 0$ **to** $L - 1$ **do**
    $\mathbf{c}_1^{\ell+1} \leftarrow \phi(\partial_2 \partial_2^\top \mathbf{c}_1^\ell \mathbf{W}_2^\ell + \mathbf{c}_1^\ell \mathbf{W}_1^\ell + \partial_1^\top \partial_1 \mathbf{c}_1^\ell \mathbf{W}_0^\ell)$
end for
$\mathbf{c}_0^{L+1} \leftarrow \partial_1 \mathbf{c}_1^L \mathbf{W}_0^L$
$\mathbf{z} \leftarrow \text{softmax}(\{[\mathbf{c}_0^{L+1}]_j : j \in \mathcal{N}(i_{m-1})\})$
Return: $\widehat{i}_m \leftarrow \arg\max_j z_j$

The algorithm can be understood in two parts. The first part being- Representation of Trajectories as 1-Chains. The partial trajectory is converted into a 1-chain at the end of the first for loop (a 1-chain is a vector with two elements). Although it seems as though there is a large loss of information in this conversion, it is not the case because trajectories consist of sequences of adjacent nodes, and the sequential information mostly gets captured by this representation.

The second part is the forward pass for one layer. The second step of the algorithm can be understood by making analogies with the conventional neural networks. The boundary operators are equivalent to known, fixed matrices, the weights W are the parameters to be learnt, ϕ is an odd non-linear activation function. This is recursively done as many times as the number of layers, and the output of every layer has these operations performed on it.

Finally, a softmax layer present at the output layer gives the probabilities of the agent being present at every node in the neighborhood of the previous time step. Taking the maximum gives us the predicted trajectory of our agent.

After understanding the algorithm, the preliminary experiment we carried out was the implementation of the associated code and validation of the accuracy obtained by the authors. The details of the experiment are found in the Section 6, and the associated results can be found in Section 7.

This concludes the work done before the mid term review.

4.2 Post Mid-Term Review

We decided to focus on different implementations and experiments after the mid-term review. However, to get a more complete understanding (as suggested in the mid-term comments), we decided to consult papers **Simplicial 2-Complex Convolutional Neural Networks** [1] and **Simplicial Neural Networks** [3]. The details of these are mentioned in Section 3.

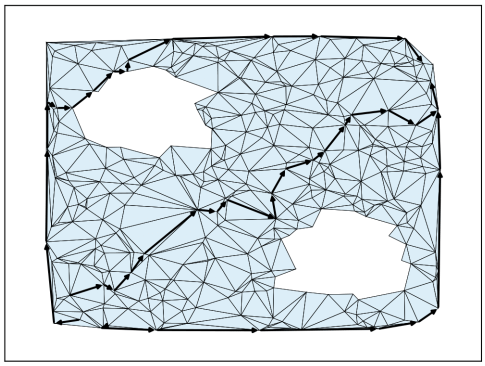
Now with respect to the implementation, we studied the effect of the change in activation functions on the performance. The functions considered were: tanh, sigmoid, ReLU, identity, sinh, and leaky ReLU. The implementation details are available in Section 6, and the comparative study is found in Section 7. Next, we did the same study firstly for number of nodes in each hidden layer, and next, for number of hidden layers. The respective details are in the same sections as mentioned above.

5 Data set Details

Type of data: Simplicial complex.

We generate a simplicial complex by drawing 400 points uniformly at random in the unit square, and then applying a Delaunay triangulation to obtain a mesh, after which we remove all nodes and edges in two regions.

To generate a set of trajectories, we choose a random starting point in the lower-left corner, connect it via shortest path to a random point in the upper-left,center, or lower-right region, which we then connect via shortest path to a random point in the upper-right corner (shown below).



The exact implementation details have been discussed in code walkthrough.

Real datasets: We consider the trajectory prediction problem for the Global Drifter Program dataset, localized around Madagascar.5 This dataset consists of buoys whose coordinates are logged every 12 hours. We tile the ocean around Madagascar with hexagons and consider the trajectory of buoys based on their presence in hexagonal tiles at each logged moment in time. Treating each tile as a node, drawing an edge between adjacent tiles, and filling in all such planar triangles yields a natural simplicial structure. The homology of the complex shows a large hole, corresponding to the island of Madagascar.

Additionally, we consider a map of a section of Berlin, where each point on a grid is designated impassible or passible based on the presence of an obstacle. For a set of trajectories, we consider a set of 1000 shortest paths between random pairs of points in the largest connected components, divided into an 80/20 train/test split.

6 Experiments

The training algorithm has been discussed in detail in the 'Pre-Mid-Term Review' section.

The default hyperparameters used were: 'epochs': 1000, 'learning_rate': 0.001, 'weight_decay': 0.00005, 'batch_size': 100, 'hidden_layers': [(3, 16), (3, 16), (3, 16)]. The number '3' represents the number of shift matrices and is a a constant for the SCoNe model. As part of our experiments, we noted the effect of changing the number of hidden layers on the accuracy of the results. We also studied the effect of using various other activation functions on the performance of the algorithm.

7 Results

7.1 Pre Mid-Term

	Markov	RNN	SCoNe (tanh)	SCoNe (tanh, no tri.)	SCNN	S2CCNN
STD.	0.69	**0.74**	0.67	0.57	0.66	0.61
REV.	0.24	0.02	**0.60**	0.51	0.44	0.45
TRA.	-	-	**0.63**	0.60	0.41	0.59

(a) The performance of different methods on the standard train/test split

	tanh	Id.
STD.	**0.66**	0.30
REV.	**0.63**	0.32

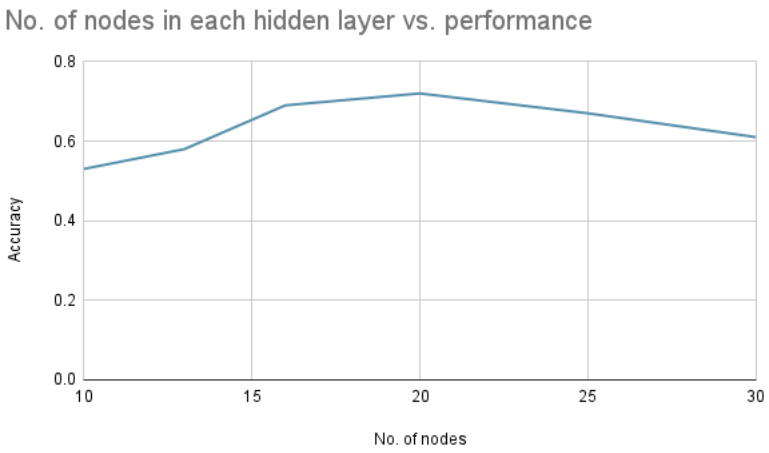
(b) Comparison of an odd, nonlinear activation with a (odd) linear activation

	Markov	RNN	SCoNe
OCEAN	0.47	0.44	**0.50**
BERLIN	0.74	0.81	**0.92**

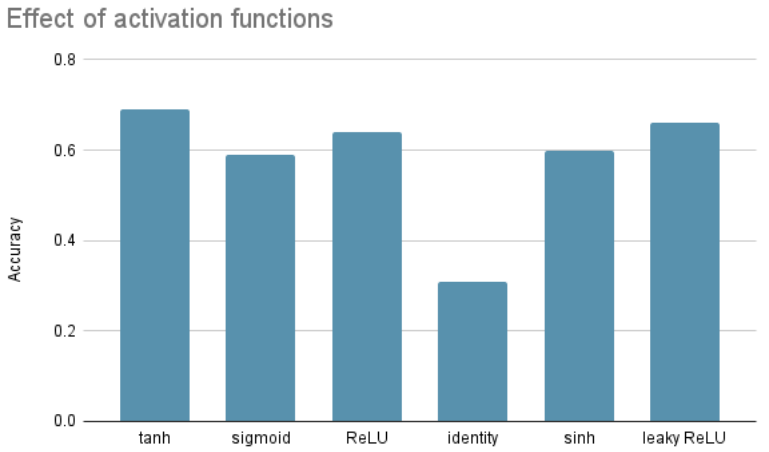
(c) Results on the real test datasets

A simple implementation of the code provided by the authors validated their results on all the datasets used. The best performing model is highlighted in bold.

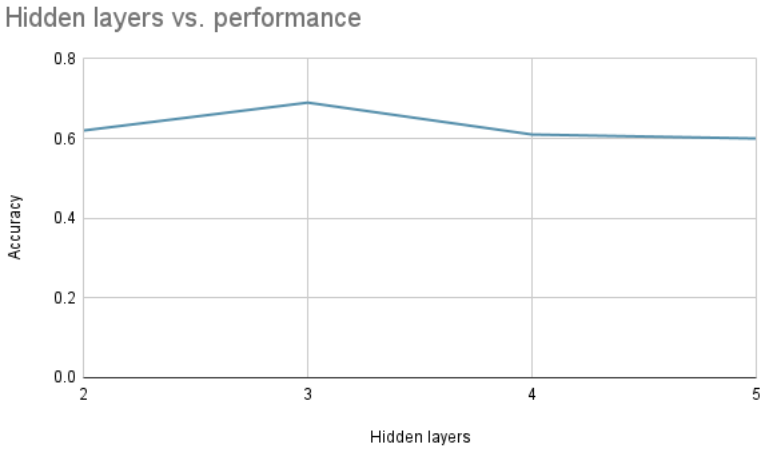
7.2 Post Mid-Term



The baseline no. of nodes is 16, as used in the paper. It can be seen that reducing the number of nodes in each hidden layer leads to decreasing performance which can be attributed to the loss of information as the nodes are decreased (this can loosely be thought of as underfitting). As the nodes are increased, we see the performance increases slightly followed by another decrease which can be due to too many training parameters relative to the no. of epochs (we could not go above a 1000 epochs as it was taking too long).



Here, the number of nodes in each hidden layer was kept constant at 16, and the number of hidden layers was kept at 3. Activation functions used were as shown in the graph. The function that displayed maximum accuracy was tanh, followed by leaky ReLU and ReLU. As expected, the identity function performed poorly. Contrary to the paper's claims (odd functions are required), ReLU and leaky ReLU (both not odd) seem to perform almost at par with tanh on this dataset.



Here, the number of nodes in each hidden layer was kept constant at 16, and the number of hidden layers was varied. tanh activation was used. Initially from 2 to 3 layers the performance increases, and then decreases and is almost same for 4 and 5 layers.

8 Future Work

1. Testing functions that are not odd, like ReLU and leaky ReLU on other datasets to check if they are performing at the same level as odd functions like tanh. Finding theoretical basis for this anomaly if they display results that are at par.
2. Using the convolution operation of SCoNE with certain FC and CNN layers in order to implement the image classification in S2-CCNN paper and compare with their neural network architecture.
3. Modification of SCoNE to other tasks (Graph Generation, Regression tasks or encoding). This would involve changes to output layer along with certain other FC or CNN layers before it.

9 Conclusion

1. A core component of graph neural networks is their equivariance to permutations of the nodes of the graph on which they act.
2. Designing architectures that respect properties such as this enables the design of systems that transfer and scale well
3. By considering additional symmetries (orientation equivariance) and truly accounting for higher-order structures (simplicial awareness), performance can be significantly improved.
4. The activation function that shows the most superior result is the tanh function
5. Surprisingly, functions that are not odd also display almost equivalent performance on this dataset, contrary to the claims in the paper. This needs to be investigated over other datasets and a theoretical reasoning needs to be found.

6. In the case of number of hidden units and hidden layers, the performance is as expected, with an initial increase due to model complexity increasing (reduction in underfit), followed by a decrease beyond a certain value (the model becomes too complex and overfitting begins).

References

[1] Eric Bunch, Qian You, Glenn Fung, and Vikas Singh. Simplicial 2-complex convolutional neural nets, 2020.

[2] Jean-Baptiste Cordonnier and Andreas Loukas. Extrapolating paths with graph neural networks, 2019.

[3] Stefania Ebli, Michaël Defferrard, and Gard Spreemann. Simplicial neural networks, 2020.

[4] Nicholas Glaze, T. Mitchell Roddenberry, and Santiago Segarra. Principled simplicial neural networks for trajectory prediction. *CoRR*, abs/2102.10058, 2021.

[5] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.