Save Your Priors for Another Day: Discovering Data Dependent Invariances and Equivariances

Nikhil Agarwal* **Deepsikha Das*** **Sharmistha S. Gupta*** **Dhruvesh Patel**
College of Information and Computer Sciences
University of Massachusetts Amherst

Balasubramaniam Srinivasan
Department of Computer Science
Purdue University

Manzil Zaheer
Research Scientist
Google Deepmind

Abstract

A plethora of real world problems involve making predictions on data which might have implicit symmetry. For instance, in an outlier detection task, the outlier remains unchanged even if the given set of objects is permuted. State of the art research in this field hardwires properties under symmetry group transformations in the network architecture. However, recent research has shown that symmetry discovery is possible without enforcing such inductive biases into the networks. In this work, we propose a generic architectural design using multi-layer perceptrons that recovers invariance/equivariance properties from data as a part of the learning process, without explicitly building them into the network architecture or design.

1 Introduction

1.1 Preliminaries

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ be a dataset. A machine learning task, in its most general form, consists of learning a function $f : X \to Y$, where X, Y denote the input and output spaces respectively.

Definition 1 (**Group**). A group is a set G with an operator (*) that has four properties: closure, identity, inverse and associativity.

Definition 2 (**Symmetry Group**). A symmetry group is a group G where the group elements are transformations or functions, and the operator is a function composition.

Definition 3 (**Group Action**). G acts on X if there is a map $* : G \text{ x } X \to X$, so that if $g \in G$ and $x \in X$, then $g * x \in X$, such that:

1. For every $g, h \in G, x \in X$, we have $(g * h) * x = g * (h * x)$.
2. For every $x \in X, e * x = x$, where $e \in G$ is the identity.

Definition 4 (**Invariant Function**). A function $f : X \to Y$ is said to be invariant w.r.t the action of a group G if for all $g \in G$, $f(g(x)) = f(x)$.

Definition 5 (**Equivariant Function**). A function $f : X \to Y$ is said to be equivariant w.r.t the action of a group G, which acts on both X and Y, if for all $g \in G$, $f(g(x)) = g(f(x))$.

1.2 Task Description

Specific architectures have been largely successful in leveraging invariance/equivariance properties present in data. However, the design of these neural networks requires prior knowledge of the domain and symmetry groups present in the data as well encoding some inductive bias into the networks. We explore the possibilities where we do not have to impose data specific biases into the networks. We look into designing generic invariant/equivariant architectures without leveraging human-engineered architectural biases for symmetry discovery. In this work, we propose an approach that leverages parameter sharing schemes and sparsification of model parameters for learning symmetry properties. An extension of this approach would be to further distill knowledge from the final learnt model to a smaller MLP, but this is beyond the scope of our work (Section 4). We formulate three tasks for experimenting our approach targeting permutation invariance in a set of real numbers, translation invariance in an image, and permutation equivariance in a graph respectively (Section 3).

1.3 Motivation and Limitations of Existing Work

The domain of invariant and equivariant machine learning has parallels with data representations like sets, point clouds, images and a combination

* indicates equal contribution

of these data types. The DeepSets (Zaheer et al., 2017) architecture gives state of the art performance on certain tasks involving sets. However, the matrix used for performing transformations on the input is explicitly designed to be permutation invariant. For images, convolution neural networks are known to give state of the art performance on various tasks like image classification which directly leverage the translation invariance of the image based input. Most of the commonly used models like DenseNet (Huang et al., 2017) and ResNet (He et al., 2015) use convolution neural network architectures. Similarly, other works like VectorNeurons (Deng et al., 2021), Equivariant Graph Neural Networks (Satorras et al., 2021) make use of prior knowledge of symmetry groups on the data for network design.

These works primarily design an invariant layer (or equivariant layer) and then stack different variations of such layers to form a complete neural network architecture. The obvious benefit of such networks is parameter sharing which makes it possible to overcome overfitting while training and in turn, achieve better test time performance. However, introducing such heavy inductive bias restricts the exploration of more generic functions which can potentially perform better.

In this work, we aim at designing neural networks which do not explicitly leverage invariance/equivariance layers. We hypothesize that such neural networks can provide potentially comparable results as the existing counterparts under ambient training settings.

1.4 Proposed Approaches

1.4.1 Promoting Parameter Sharing

We initially proposed a modified loss function to learn parameter sharing schemes in a MLP for a given symmetry. The formulation was as follows:

$$Loss = L + \sum_{j,k} \lambda_i |\theta_j - \theta_k|$$

where θ_j and θ_k are all pairs of weights in one layer and λ_i is the hyperparameter that controls the influence of their difference on the total loss. The L term in the above loss is the task-specific loss such as cross-entropy for classification and L1 or L2 loss for regression. One can also use squared differences instead of absolute differences in the second part of the loss.

The set of hyperparameters essentially determines the parameter-sharing scheme. Note that the number of hyperparameters in this loss function is $\binom{n}{2}$ where n is the number of parameters in the neural network. It is infeasible to search so many hyperparameters which poses a problem. The simplest approach is to have just one hyperparameter. However, this is too simplistic and does not promote parameter sharing in a way that can benefit learning. For the image datasets, we have tried sharing parameters in a block of the weights in the matrices. However, this does not mimic a CNN kernel sharing as CNN kernel in general have different values within it. As a result, the method of block sharing of parameters did not work in the setting we used. Some alternative approaches that can benefit are 1) A min-max approach. This implies that only a few of the λ_is would be allowed to be non zero. 2) Initialize the weights from a multi-modal normal distribution to promote soft sharing 3) A reinforcement learning approach to explore how to make parameter pairs have the same value.

But all the above approaches are very sensitive to the choice of hyper-parameter modes and demands rigorous tuning which is not a feasible approach for scalable architectural designs without an efficient technique.

Instead, we have designed data-specific parameter sharing schemes for multi-layer perceptrons. Note that this method in contrast to the current state of the art architectures does not impose network design constraints, i.e we still use a generic design for all the symmetries we have experimented on.

1.4.2 Promoting Parameter Sparsification

An extension of the approach mentioned above is sparsification of the final weight matrix (i.e addition of $\alpha_i|\theta_i|$ terms to the loss function) which would then make the loss function as follows:

$$Loss = L + L_{parameterSharing} + \sum_i \alpha_i |\theta_i|$$

where α_i are hyperparameters. In our experiments, we have found that sometimes simple L1 sparsification of the weights works much better than the parameter sharing loss with just one hyperparameter chosen instead of $\binom{n}{2}$. Also, in other cases, sparsification does not help at all and ruins the sharing scheme. In future work, we will explore efficient ways of adding sparsification for decreasing model sizes without hampering the sharing pattern.

1.4.3 Generalizing Architecture Design

As mentioned in the problem definition, our sole purpose is to understand the ability of neural networks to learn various symmetry properties about the input data distribution without enforcing any

architectural bias. For this, all our experiments are modelled around multi-layered perceptrons irrespective of the type of the input data. This essentially removes the constraint of having an inductive bias and the learning property of the networks is dependent only on metadata like data dimensions, the number of classes (for classification), and the parameter-sharing scheme used for training. Although the sharing scheme is conditioned on the task description, this is not equivalent to an architectural bias as the architectural design remains the same for all data distributions.

The design follows a generic pattern:

- Designing a suitable learnable weight matrix conditioned on the metadata and the type of properties we want MLPs to learn. For classification tasks, we have used C different weights where C is the number of classes in the data. We hypothesize that this method allows for better model training and visualizations which plays a major role in understanding the extent of the networks in symmetry discovery.
- Using appropriate non-linearities. This is important for simultaneous interpretable visualizations and faster convergence in training.
- Specific feature aggregation schemes.
- Designing an appropriate parameter-sharing scheme conditioned on the data symmetry. The sharing is enforced by adding a corresponding regularization loss to the task-specific loss.

2 Related Work

Specialized Architectures: Deep Sets (Zaheer et al., 2017) and Vector Neurons (Deng et al., 2021) give state of the art performance on tasks that include data with specific symmetry groups. However, they use invariant/equivariant layers stacked together to give a biased neural network architecture. Another work on constructing equivariant MLPs for matrix groups (Finzi et al., 2021) discusses building MLPs with equivariant layers for an arbitrary symmetry group. Although it is an expansion of DeepSets and Vector Neurons, it is generic with respect to arbitrary symmetry groups once they are known. Unlike these approaches, we propose to use unbiased architectures that do not use the knowledge of symmetries present.

Constrained Loss Functions: A paper on simplifying neural networks by soft weight-sharing (Nowlan and Hinton, 2018) uses a mixture of Gaussian distributions to select weights of the neural network. It then designs a loss function which penalizes weights proportional to their distance from the nearest mode. This drives the model to select weights which are close to one of the modes of the Gaussian mixture. Similarly, another paper on entropy-SGD (Chaudhari et al., 2017) proposes a new objective function which biases the model towards well generalizable wide valley local minima instead of sharp valleys which are not well generalizable. By doing this, it forces the function to learn something simpler instead of learning something very complex. As opposed to these models, the loss function we use promotes parameter sharing in a much more straightforward way. It focuses directly on pairs of weight parameters.

Network Modification: A model on frame averaging for designing invariant/equivariant networks (Puny et al., 2021) focuses on designing architectures without compromising on their expressivity. It defines the novel frame averaging operation and incorporates it within already existing neural networks. This differs from our approach as we are using MLPs and promoting parameter sharing through the loss function.

Learned Parameter Sharing: (Yeh et al., 2022) This paper considers a model's parameter sharing scheme as an optimization problem. They consider the variables controlling the model's sharing scheme as learnable parameters instead of hyperparameters. This can potentially solve the problem of searching high number of good hyperparameters. They follow a dual training approach in which the available data is split into two sets, a validation dataset on which the learned parameter sharing is learned. This sharing is then fixed and the model weights are trained on the task at hand.

3 Experiments

We are moving from specialized architectures that capture known symmetries towards a generalized architecture that is able to discover the symmetries from data without incorporating any inductive bias. Thus, a robust evaluation of our proposed approach would be to use the same architectural design to discover a wide range of symmetries from data that are known to exhibit them. As the class of architectures we will be using (Fully-connected Neural Networks) does not differ for different cases, it can be inferred that we are not imposing any engineered bias.

3.1 Permutation Invariance in a Set

In this experiment, our aim is to verify whether our proposed approach is able to recover permutation invariance in sets. The task is to calculate the sum of numbers in a given set and show that the output of the network remains invariant to any permutation of the set elements.

Dataset: For simplicity, we handled sets of fixed sizes. We used a synthetic dataset which consisted of inputs that are sets of ten randomly sampled real numbers in $[0, 1]$ and the output is the sum of these numbers. We used 8k training samples, 1k validation samples, and 1k testing samples.

Baseline: We implemented the **DeepSets** (Zaheer et al., 2017) architecture as our baseline model, which uses an identity matrix as the weight matrix and yields state of the art performance on set-permutation invariance tasks. The input is a $N \times 10$ matrix where N is the batch size. The output is a $(N \times 1)$ real-valued prediction. We used mean L1 loss as the training loss function. The test loss was **0.015**.

Implemented Architecture: The weight matrix for this experiment is a 10×10 matrix. The architecture is a simple linear transformation of the input $(N \times 10)$ with the weight designed. The aggregation function used is sum-pooling over the computed output. For better comparision, we used mean L1 loss as our training loss. As a parameter-sharing scheme we used the following regularization loss:

$$L_{reg}(\Theta) = 0.01 * \sum_{i=1}^{|\Theta|} \frac{1}{1+e^{-100*\theta_i}}$$

We add this regularization to the loss to sparsify the model weights. Here the constant 0.01 is chosen from a list of hyperparameters ([100, 10, 1, 1e-2, 1e-3, 1e-4, 1e-5]) after hyperparameter tuning. The learning rate is fixed to be 0.1. We evaluate this model on the test set using mean L1 loss which is found to be **0.032**. We expected to recover any column permutation of the 10x10 identity matrix. On observing our results we realised that all weight matrices where each of the columns sum up to 1 indicate permutation invariance. In other words, if the absolute difference of a weight matrix summed along the columns and an all 1s row vector is close to 0, permutation invariance is indicated. Figure 2 shows the heat map of the final weight matrix.

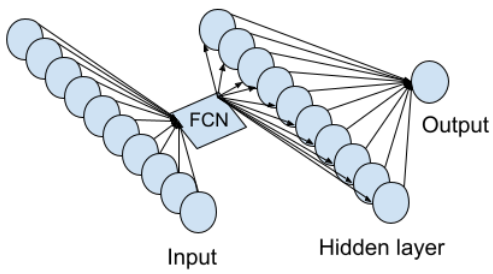

Figure 1: Architecture

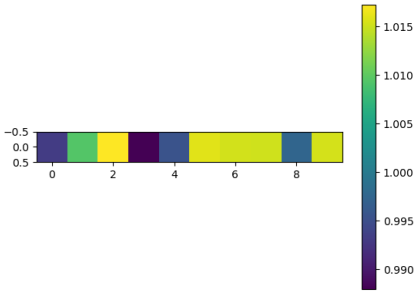

Figure 2: Final Weight Matrix: Permutation Invariance in a Set

Results: The absolute difference difference between this weight matrix and n all 1s column vector of the same size is **0.1145** which shows that our hypothesis is valid.

3.2 Translation Invariance in an Image

For this experiment, we conducted a **classification task** where the network has to predict the label of an image. Our objective was to recover a structure similar to CNNs or rather a cyclic pattern in the final weight matrix as they are known to be translation invariant. The focus of this experiment was to see whether our implemented architecture can achieve a similar performance as equivalent CNNs and be invariant to translations in the input data.

For experimentation, we first apply our generic architecture to one and two dimensional synthetic image datasets and then move on to MNIST (Deng, 2012) to show that this formulation works in general for any image dataset.

Methodology: Let a dataset consist of images of size $d \times d'$ and have C different classes. The baseline is a single-layer convolutional neural network implementation with a kernel of size $k \times k$ and stride s and s'. Then our implemented architecture will be parameterized with a weight matrix of size $C \times \frac{d}{s} \times \frac{d'}{s'} \times d \times d'$. There will be C different weights (one for each class) and each class-weight will have a size of $\frac{d}{s} \times \frac{d'}{s'} \times d \times d'$. We hypothesize that this allows better learning of the intra-class

features and will give clear visualizations of the patterns. Each class-weight is so designed as to capture all possible translations in both the directions (vertically and horizontally) with a respective stride of s and s'. Intuitively, in each direction there will be $\frac{D}{S}$ possible strides where D is the dimension along that direction and S is the corresponding stride. Thus, a total of $\frac{d}{s} \times \frac{d'}{s'}$ translations can be captured for the above mentioned image dimensions and stride assumptions.
The architecture computes a dot product between the images and each class-weight and then sums up the outputs along the last 2 dimensions (or 1 dimension in case of 1-dimensional images) giving matrices of size $\frac{d}{s} \times \frac{d'}{s'}$ which can be interpreted as outputs of all possible translations for each class. This is followed by some non-linearity. Finally, average is computed over each output matrix. Following this, we get C different class scores on which softmax is applied to convert them into a probability vector. Figure 3 shows the architecture for computing a single class score for an image of size 28×28 and a stride of 7 giving $4 \times 4 = 16$ translations.

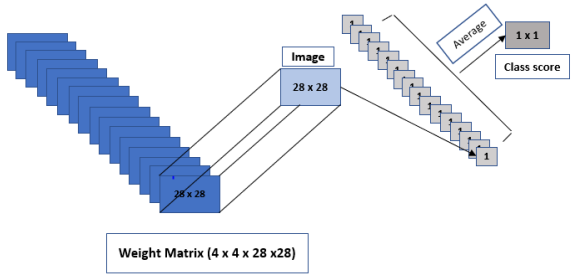

Figure 3: Implemented Architecture for 1 class-score

For training the model, we have used a customized loss function. The loss consists of the classification loss (cross-entropy) and a parameter-sharing induced loss term. For parameter-sharing, we randomly initialize 2 offsets, b and b' for each class-weight w_k where $k \in [1, C]$ and then applied a L1 penalty term which calculates the absolute element-wise difference between $w_{k,i,j,m,n}$ and $w_{k,i+b,j+b,m+b',n+b'}$ for all k. The loss function is formularized as follows:

$$L = L_{CE} + \lambda \Sigma_k |w_{k,i,j,m,n} - w_{k,i+b,j+b,m+b',n+b'}|$$

where λ, b, b' are hyperparameters and k ranges from 1 to C.
Any architecture that abides by translation symmetry will show some kind of cyclic pattern in the learned weights. As shown later in the visualizations of the trained weights of our implemented architectures, there are distinct cyclic patterns in the weights which proves that the architectures have learnt some translation invariant characteristics.

One Dimensional Synthetic Dataset: We developed a synthetic 1D image dataset to discover translation invariance in a one dimensional setting. It consists of 50,000 samples of 10 x 1 images. There are five different classes:
Class 0 - One bright pixel
Class 1 - Two consecutive bright pixels
Class 2 - Three scattered bright pixels
Class 3 - Four consecutive bright pixels
Class 5 - Five scattered bright pixels
The training, validation and the test dataset consisted of 35k, 10k and 5k samples respectively. The examples from classes can be seen in Figure 11.

Baseline: We implemented a single-layer CNN as our baseline. The CNN consists of a 1D convolutional layer where the number of input channels is 1 and number of output channels is 5. The size of the kernel is 4 and the stride is of length 1. The model is trained with cross-entropy loss. The accuracy achieved is **65%**.

Implemented Architecture: Since the dataset has 1-dimensional images of size 1×10 and 5 different classes, the weight matrix is designed as $5 \times 10 \times 10$ where the stride is 1. We have implemented 3 different variants which consists of different activation functions. All the variants follow the above mentioned generic architecture (3.2). The **test accuracy** of all the three implementations are also reported below:

- Base Variant: Sigmoid activation, **57.26%**
- Zero-centered Activations: Replaced sigmoid with tanh and ELU, tanh: **63.6%**, ELU: **78.3%**
- Promoting Sparsity: Performed $L1$ sparsification of all the weight parameters, **81%**

All the models were trained with learning rate 10^{-2}, parameter-sharing hyperparameter 10^{-4} etc fixed. Figure 4 shows the plots of the weights of class 5 of the three variants in order (sigmoid, tanh, ELU, ELU with sparsification). The weight matrices for the rest of the classes can be found in figures 12, 13, 14 and 15 respectively.

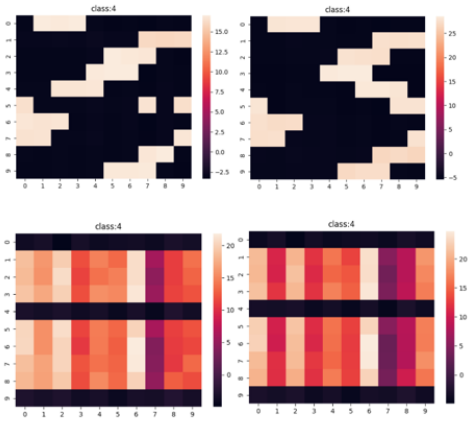

Figure 4: Weight matrices of class 5

Two Dimensional Synthetic Dataset: We developed a synthetic 2D image dataset to discover translation invariance in a 2D setting. We created 30,000 samples of 10 x 10 images.The dataset consists of the following three classes:
Class 1 - Line Segment
Class 2 - Triangle
Class 3 - Square
The training, validation and the test dataset consisted of 21k, 6k and 3k samples respectively. The examples from classes can be seen in Figure 16.

Baseline: We implemented a single-layer CNN as the baseline. The CNN consists of a 2D convolution layer where the convolutional kernel is of size $1 \times 3 \times 5 \times 5$ and the stride is 1. There are 3 different outputs for 3 different classes.

Implemented Architecture: The input images are of size 10×10. The model consists of $3 \times 10 \times 10 \times 10 \times 10$ weight matrix (for stride=1) as its only trainable parameter where each class weight is of size $10 \times 10 \times 10 \times 10$. The non-linearity used for the network is ELU. The weights of the model can be seen in Figure 5. The **test accuracy** achieved is 100%. Note that with so many parameters, there is a chance of overfitting but the model is performing well on the test data under the enforced parameter-sharing scheme which shows that the model is leveraging symmetry discovery despite its large size.

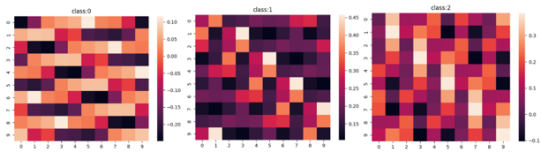

Figure 5: Weight matrices of the model

MNIST Dataset: MNIST consists of 10 classes where images are 28×28 in size and most of the digits are positioned around the center. This is not desirable for our experiments as we want the digits to be positioned randomly to capture sufficient translation in our dataset. Thus, to introduce the symmetry we padded the original MNIST images to create 56×56 images where the padding on all sides are random. This created more randomized positions. Some samples from the dataset can be seen in Figure 6. The training, validation and the test dataset consisted of 48k, 12k and 10k samples respectively.

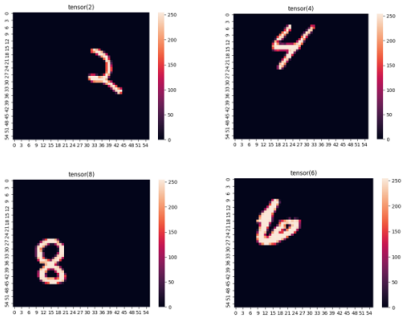

Figure 6: Dataset

Baseline: A single layer CNN was trained where the CNN kernel was of size $1 \times 10 \times 28 \times 28$ and the stride was 2. The output channels in the kernel is designed as 10 since there are 10 different classes, and hence, the convolutional layer is not followed by a fully-connected layer. Both the baseline and the implemented architecture are designed as single-layer implementations for better comparative analysis. The baseline achieved an accuracy of **88.04%**.

Implemented Architecture: The input images are each of 56×56. The model is parameterized with a weight matrix of dimension of $10 \times 28 \times 28 \times 56 \times 56$, i.e there are 10 different weights for 10 different classes. Hence by the above mentioned formulation, for stride = 2 there will be 28 $(\frac{58}{2})$ possible strides in each direction and thus, a total of 28×28 translations can be captured. The non-linearity used is ELU.

Results: Following are the results with a fixed λ (from the loss function defined in 3.2) value of 10^{-4}:

From table 1, it can be seen that with no parameter-sharing ($b = 0, b' = 0$), the accuracy is quite low, but with some sharing (positive values for b, b') the accuracy increases significantly.

b	b'	**Accuracy**
0	0	66%
16	4	**83.68%**
8	4	82.9%
16	2	82.16%

Table 1: Results on MNIST

Thus, this endorses the requirement for parameter-sharing for symmetry discovery. Moreover, the average accuracy achieved with parameter-sharing is close to the accuracy achieved by the baseline CNN. Furthermore, the table also shows that the accuracy doesn't vary a lot with the choice of b and b' which removes the constraint of stringent hyperparameter tuning for the same.

Figure 7 shows the learned weights for clasess 0,3 and 5. The rest of the weight visualizations can be seen in figures 17 and 18.

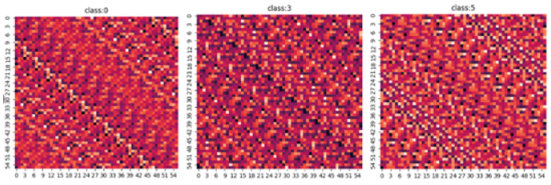

Figure 7: Learned weights

The heatmaps for all the weights of the models trained on the above datasets show clear cyclic patterns with simultaneous competitive results as the baselines and hence, we claim that this formulation is able to discover properties of translation symmetry.

3.3 Permutation Equivariance in a Graph

Dataset: We developed a synthetic graph dataset in the form of adjacency matrices. We created a total of 60000 undirected graphs with 10 nodes each. Out of these, 10000 graphs were generated by randomly inserting edges among the 10 nodes. For each of these 10k graphs, we created 5 more graphs as mere random permutations on the order of nodes of these graphs making the total dataset size to be 60000. Out of these 48000 are used for training and 12000 are used for testing the baseline and the implemented architecture. Note that there are no self edges

Task: The task is to predict the degree of each node of the input graph. We chose this task because it is permutation equivariant with respect to the order of the nodes. The degree predictions are shuffled by the same permutation the input nodes are shuffled by.

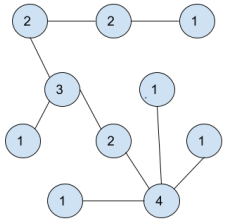

Figure 8: Degree prediction of a graph

Evaluation Metric: We have used accuracy as the evaluation metric where accuracy is the total fraction of nodes across all graphs for which correct degree predictions are made.

Regression vs Classification: We can formulate this problem as a classification task where 0 to 9 are 10 degree classes that the model needs to predict. However, since there is an order in these classes (class 6 and 7 are "closer" compared to class 1 and 7), we have formulated this problem using regression. We clamp the model output in (-0.5, 9.5) and then round it off to the nearest integer.

Baseline: We have used a Graph Neural Network using two layers of Graph Convolution (Kipf and Welling, 2016) operation with (in-channels, out-channels) as (1, 10) and (10, 1). We use mean L1 loss to train the model. We observe an accuracy of 57.69% on the test dataset.

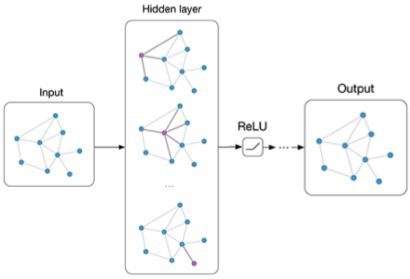

Figure 9: Graph Neural Network

Implemented Architecture: We use a one layer MLP with a 10 x 10 size followed by a ReLU activation and a row sum. We use mean L1 loss to train the model. We observe an accuracy of 100% on the test dataset.

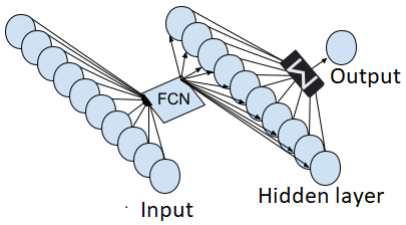

Figure 10: Implemented Architecture

Results:

Model	Accuracy
GNN (Baseline)	57.69%
MLP	100%

Table 2: Evaluation results

Table 2 shows the accuracy achieved. Note that despite the reported accuracy, the Graph Neural Network is expected to give better results. We have tried a few variations in number of layers and channels, activations like ReLU, sigmoid, tanh, but did not get better results. Different operators like Spline-Convolution (Fey et al., 2017), Edge-Convolution (Wang et al., 2018), etc. are also worth trying. For the models that we have used, the learned weight matrices have been added in the appendix. 19, 20, 21

3.4 Challenges and Mitigations

- The **large number of hyperparameters** in the initial proposed approach made training expensive. We have explored different ways as discussed in 1.4. But finally, we had to do away with learning the sharing schemes and focus more on removing network design constraints.
- **Visualizing** the final set of parameters proved to be difficult. We used Python visualization libraries for interpretation. We also had to apply suitable transformation on high-dimensional weight matrices for plotting purposes.
- Coming up with appropriate **comparison metric** of the learned weights of our architectures with those of the specialized architectures. This proved to be difficult as the weight dimensionalities don't match and hence, visual comparison is not possible. Instead, we looked for specific patterns (numeric and visual) that are expected from the learned weights.
- The **large parameter space of MLPs** was most difficult to overcome. MLPs have a lot more parameters than the state of the art architectures. We are still looking into appropriate sparsification and distillation techniques to handle this.

4 Future Work

4.1 Learning parameter-sharing schemes using Dual Training

We can use the technique mentioned in (Yeh et al., 2022) to learn parameter sharing scheme instead of doing a manual search in the hyperparameter space, and thus avoiding the problems associated with a large hyperparameter search space. To implement this method, the data is split into two groups, the learning of the parameter sharing hyperparameters is carried on the first group followed by a training of the model weights on the second group for the task at hand after fixing the parameter sharing scheme.

4.2 Efficient Model Size Reduction Techniques

We have observed that L1 sparsification can weaken the effect of parameter sharing. More sparsification techniques in addition to L1 regularization should be explored that achieve a balance between generalizability and capacity of the model. Knowledge distillation can also be explored to transfer the knowledge of a large model to a smaller one.

4.3 Experiments on additional datasets and symmetry groups

Additional experiments can be ran on datasets involving point clouds. Point cloud classification is a rotation, translation, and reflection invariant task. Point cloud segmentation on the other hand is a permutation equivariant task. These experiments could give more insights about the power of Fully Connected architectures for this kind of data.

Conclusion

After training and analysing the fully connected architectures for solving tasks like image classification, regression on numbers, and graph node classification, we have observed that the networks learn appropriate parameter sharing to some extent and yield results comparable to baselines under suitable parameter sharing schemes. In addition to this, we have introduced a generic procedure to design single layer fully connected architectures which were tested against three different tasks. We also came up with different metrics to compare FCN weight matrices to biased weight matrices (ex. CNN filters) like correlation and mean absolute difference, and appropriate visualizations and interpretations of the learned weight matrix patterns. The domain of generic unbiased architectures is a complex problem to analyse but worth exploring to answer some fundamental questions about model optimization, generalizability, and capacity. In addition to answering theoretical questions, the practical applications are critical to fields like biomedical images and natural language where the symmetries present can be unknown or complex for specialized architectures.

References

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. 2017. Entropy-sgd: Biasing gradient descent into wide valleys.

Congyue Deng, Or Litany, Yueqi Duan, Adrien Poulenard, Andrea Tagliasacchi, and Leonidas Guibas. 2021. Vector neurons: A general framework for so(3)-equivariant networks.

Li Deng. 2012. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142.

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. 2017. Splinecnn: Fast geometric deep learning with continuous b-spline kernels.

Marc Finzi, Max Welling, and Andrew Gordon Wilson. 2021. A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks.

Steven J Nowlan and Geoffrey E Hinton. 2018. Simplifying neural networks by soft weight sharing. In *The Mathematics of Generalization*, pages 373–394. CRC Press.

Omri Puny, Matan Atzmon, Heli Ben-Hamu, Edward J. Smith, Ishan Misra, Aditya Grover, and Yaron Lipman. 2021. Frame averaging for invariant and equivariant network design.

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. 2021. E(n) equivariant graph neural networks. *CoRR*, abs/2102.09844.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2018. Dynamic graph cnn for learning on point clouds.

Raymond A. Yeh, Yuan-Ting Hu, Mark Hasegawa-Johnson, and Alexander G. Schwing. 2022. Equivariance discovery by learned parameter-sharing.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

5 Appendix

We include different flow charts to explain model architectures, model weight heat maps, and other related visualizations in this section.

Figure 11: Samples from Blob 1D classes

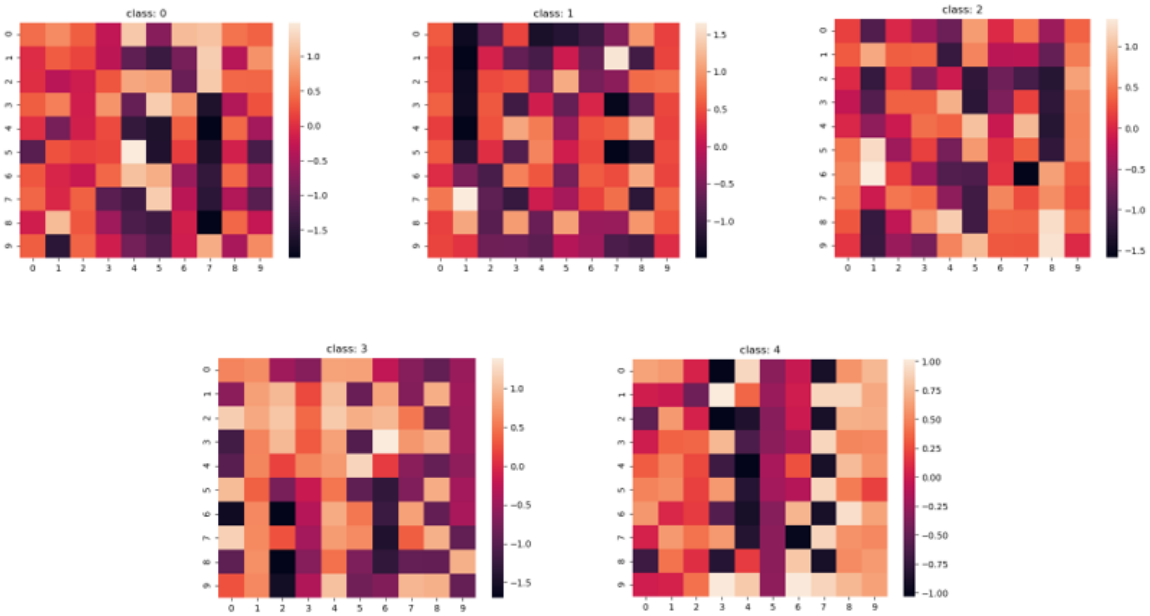

Figure 12: Weights of the 1D Base Variant

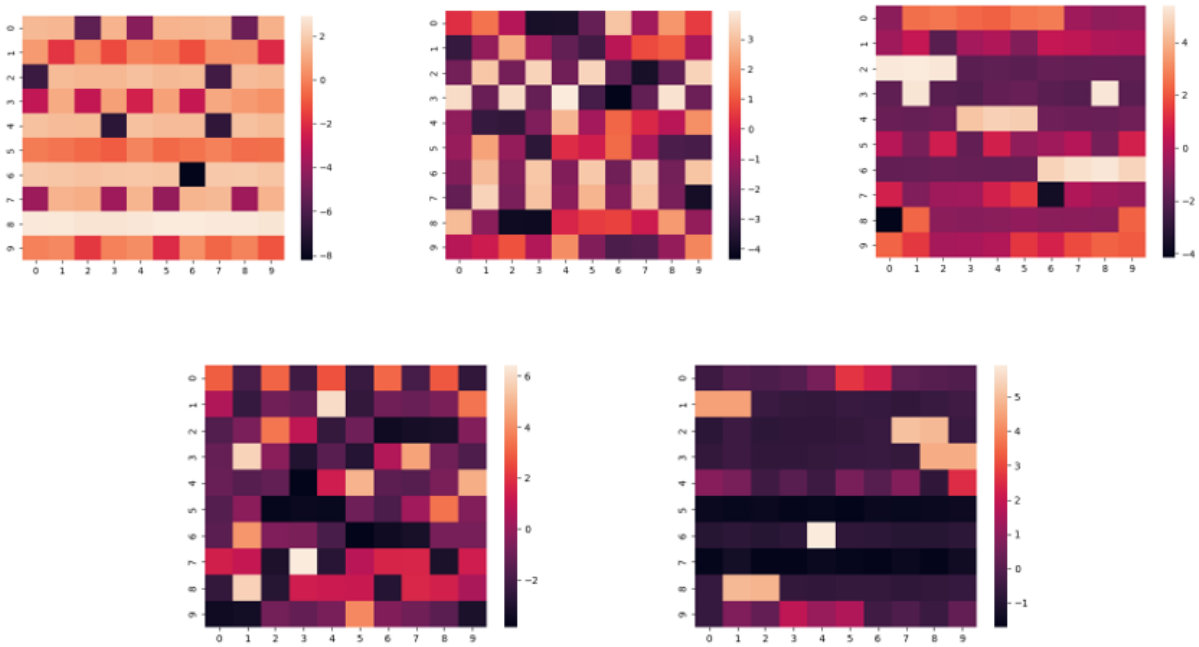

Figure 13: Weights of the 1D Tanh Variant

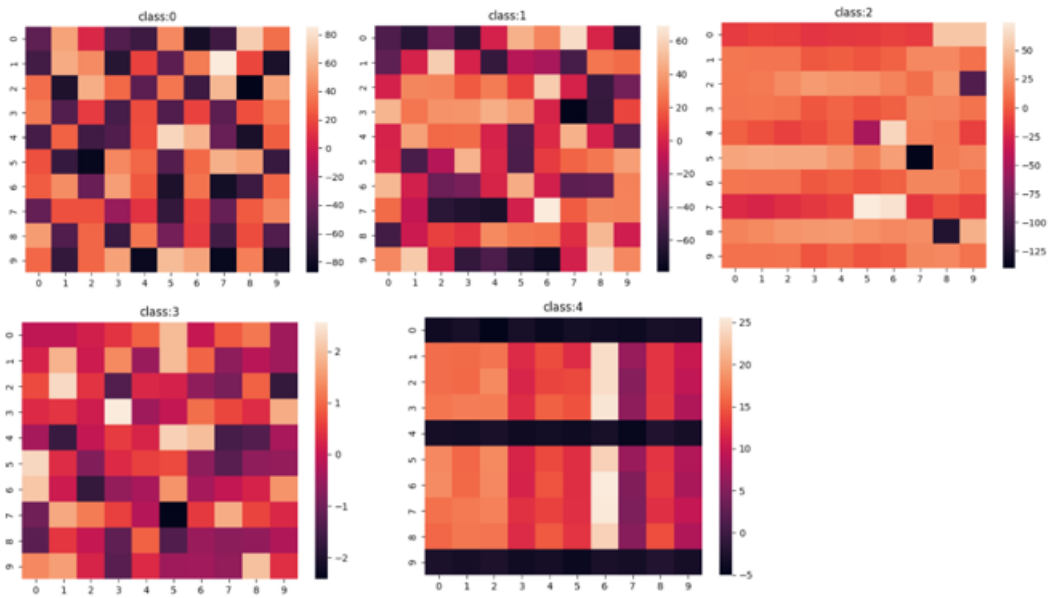

Figure 14: Weights of the 1D ELU Variant

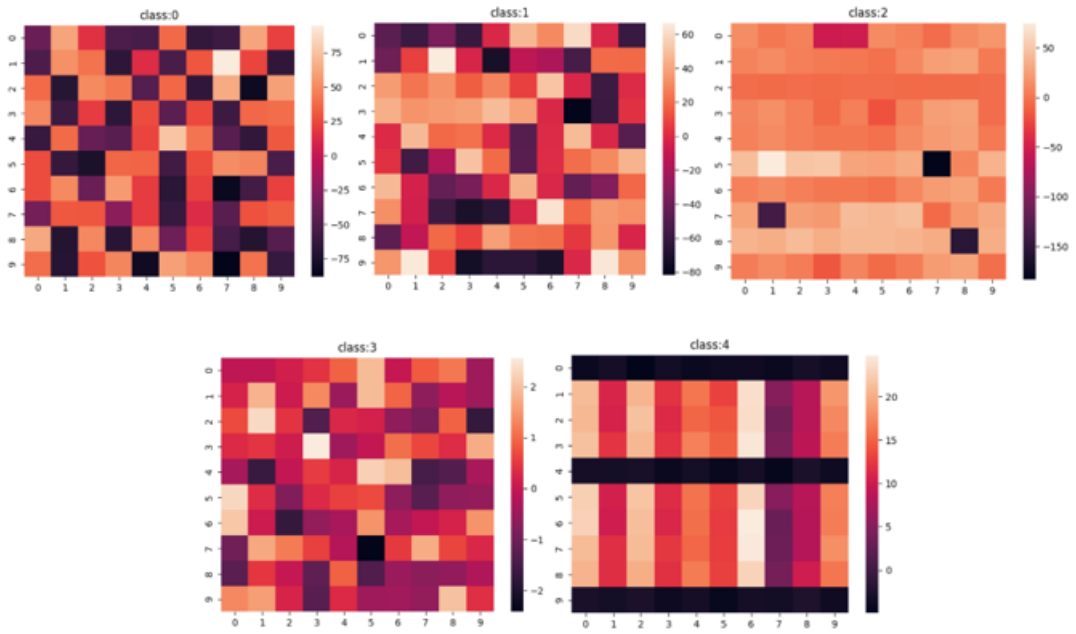

Figure 15: Weights of the 1D ELU-Spatsity Variant

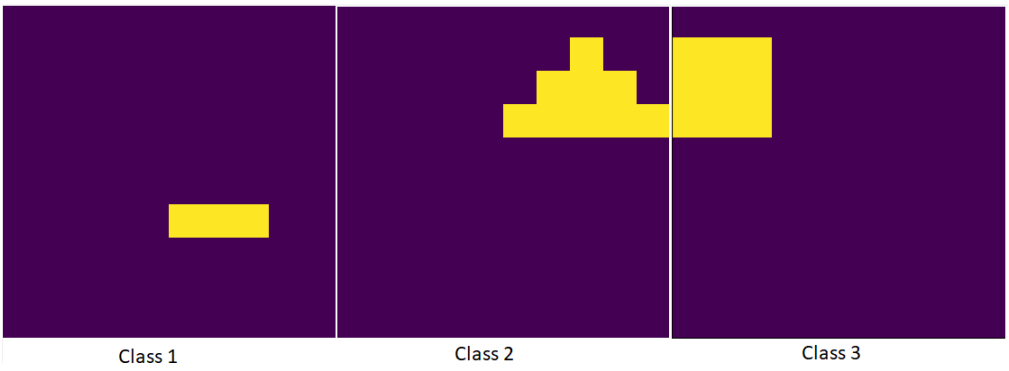

Figure 16: Samples from Blob 2D classes

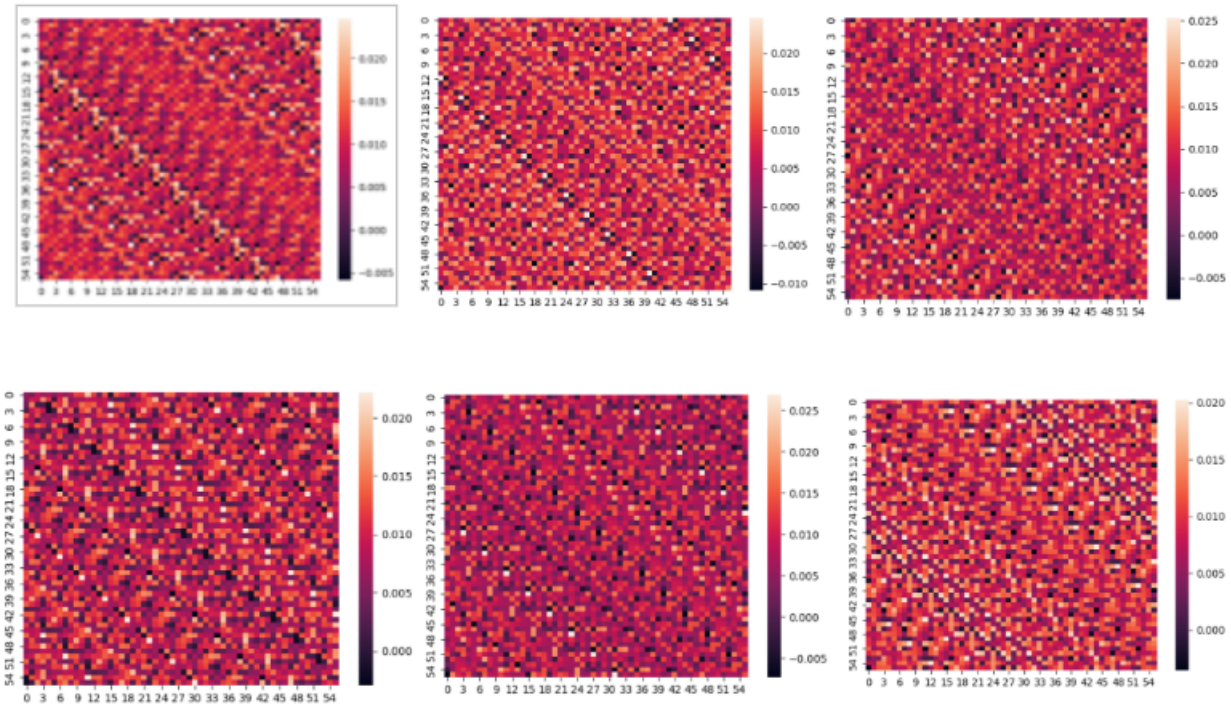

Figure 17: Learned weights of MNIST classes 0-5

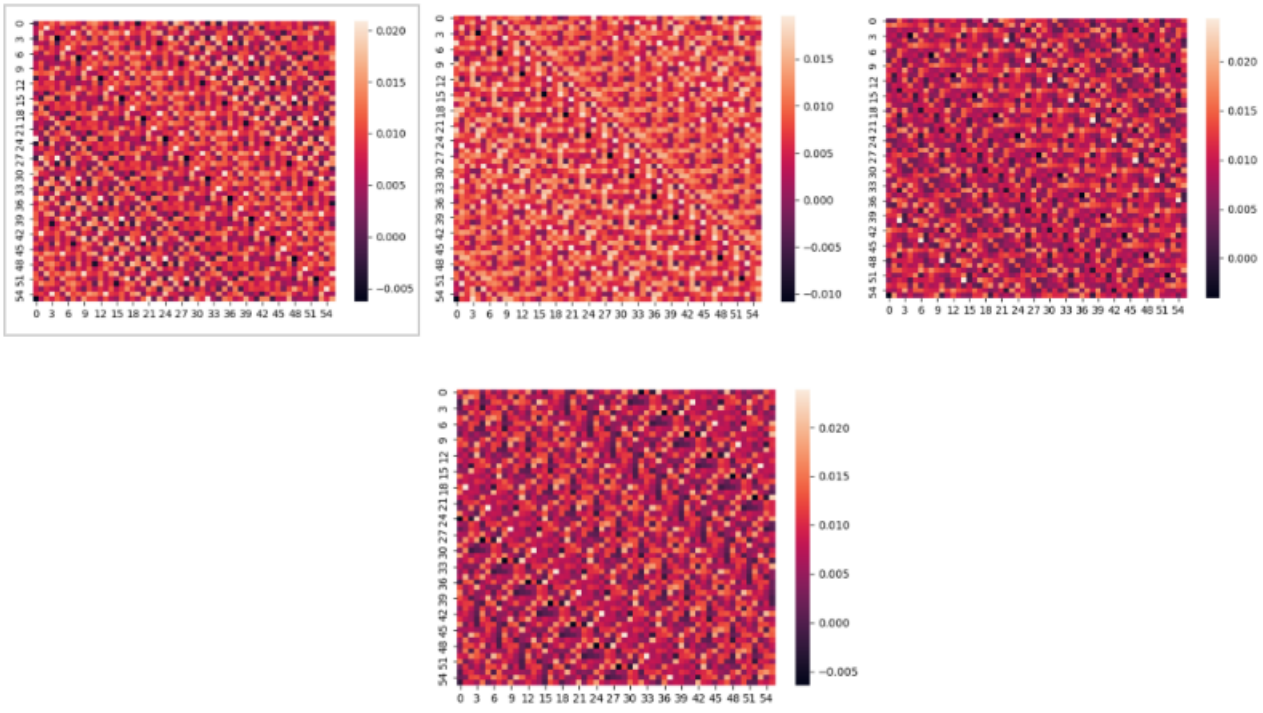

Figure 18: Learned weights of MNIST classes 6-9

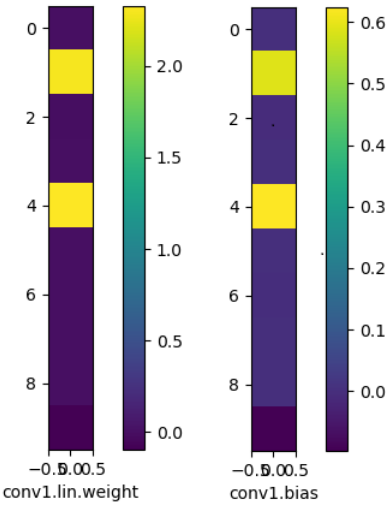

Figure 19: Weight and biases of the first layer of the GNN baseline

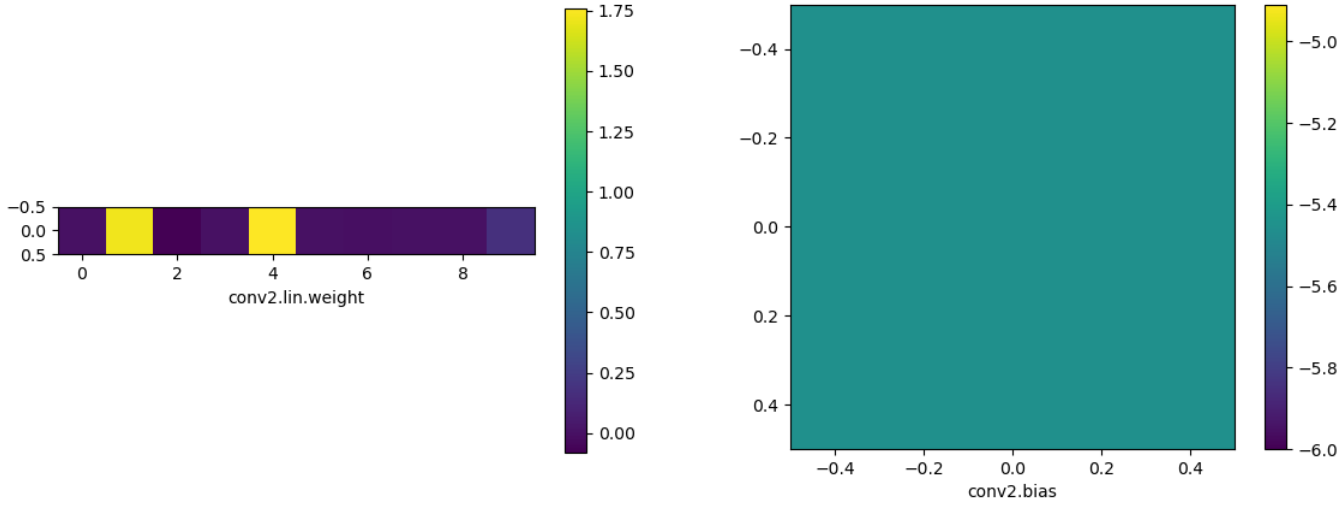

Figure 20: Weight and biases of the second layer of the GNN baseline

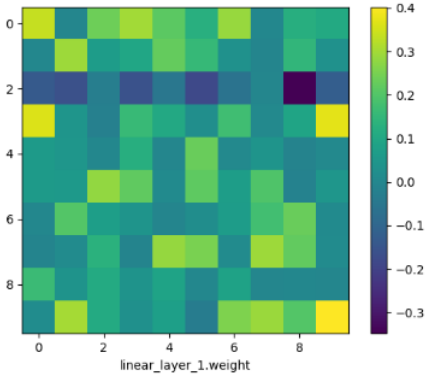

Figure 21: Weights of the MLP for degree prediction in a graph