Improving Graph Networks through Selection-based Convolution
Supplemental Material

1. Code

We provide code to demonstrate how to incorporate selection into graph convolution layers. Specifically, we provide our code for SelGraphConv in Fig. 1. The code is implemented in Python and incorporates PyTorch Geometric. The code builds nicely on the existing framework. Selection can be added to most graph networks in three simple steps:

1. Define selection weights during the initialization of the network.
2. Pass selection values through the forward function.
3. Multiply corresponding selection weights during the message-passing step.

All other aspects of the code can be inherited from the original graph network. These steps are illustrated in our provided code. Though we give a single example, these same steps can be used to add selection to any existing graph convolution layer.

2. Training Configurations

We provide details of the training configurations used for our results to improve their reproducibility. The specific configuration for each task is provided in the following subsections. We will additionally provide code at the time of publication.

2.1. Spatial Datasets

For both of the presented spatial tasks, the networks trained for 60 epochs with a learning rate of 0.001 and 0.1 decay after every 20 epochs. We trained with a batch size of 32 for MNIST and 16 for CoMA. The graph networks contained 3 layers with a hidden size of 64, followed by a linear output layer.

For both the MNIST Superpixels and CoMA datasets, we used the train and test sets provided from PyTorch Geometric. We then split the training data randomly to generate validation data. For MNIST Superpixels, this gives 48K training graphs, 12K validation graphs, and 10K testing graphs. For CoMA, this gives appoximately 15K training graphs, 3.5K validation graphs, and 1.5K testing graphs.

2.2. Traffic Prediction Datasets

For the METR-LA dataset, the networks trained for 30 epochs with a learning rate of 0.001 and 0.1 decay after every 10 epochs. We trained with a batch size of 32. We used a hidden size of 64 for our spatial GCNs.

In our evaluation, we used the masked metrics found in TwoResNet [Li *et al*. 2022] and other works. For both datasets, we used temporal splitting to generate the graphs. We used a train/validation/test split of 72% / 8% / 20%.

2.3. QM9 Dataset

The configuration we used for DimeNet and SelDimeNet matched the configuration from the original paper, specifically:

- Hidden Channel Size = 128
- Number of Blocks = 6
- Number of Bilinear Filters = 8
- Number of Spherical Filters = 7
- Number of Radial Filters = 6
- Max Distance Cutoff = 5.0
- Envelope Exponent = 5
- Number of Layers Before Skip Connections = 1
- Number of Layers After Skip Connections = 2
- Number of Output Layers = 3

For each individual target, the networks trained for 170000 steps (approximately 200 epochs). The learning rate had 3000 steps of warm up, then the learning rate exponentially decayed from 0.001 to 0.00001 over the training. We trained with a batch size of 64.

We used the same train/validation/test split as the original DimeNet implementation, including the same random seed. Specifically, we trained on 110,000 molecules, validated on 10,000, and tested on 10,831.

3. Selection Ablations

In the paper, we presented results using a particular set of selection functions. We present further findings with other possible selection functions for each task.

```
class SelGraphConv(GraphConv):
   def __init__(self, in_channels, out_channels, selection_count):
      self.selection_count = selection_count
      self.weight = Parameter(torch.randn(selection_count,in_channels,in_channels))
      super().__init__(in_channels, out_channels)

   def reset_parameters(self):
      torch.nn.init.xavier_uniform_(self.weight)
      super().reset_parameters()

   def forward(self, x, edge_index, selections, edge_weight = None):
      self.cur_selections = selections
      return super().forward(x,edge_index,edge_weight)

   def message(self, x_j, edge_weight):
      for s in range(self.selection_count):
         nodes = torch.where(self.cur_selections == s)[0]
         x_j[...,nodes,:] = torch.matmul(x_j[...,nodes,:], self.weight[s])
      return super().message(x_j,edge_weight)
```

Figure 1. Example code demonstrating how to add selection to an existing graph convolution layer in PyTorch Geometric. Note: self.weight may need to use out_channels instead of in_channels if aggregation happens after a linear layer in the network.

3.1. MNIST Superpixels

We conduct an ablation over possible selection functions on the MNIST Superpixel dataset. This includes varying the number of directions in the selection function, as well as experiments with distance selection. This ablation study can be found in Table 1. These results show that changing the number of directions does not have a significant impact on the performance. Additionally, distance-based selection does not perform as well as directional selection, but still improves performance over the baseline.

3.2. Traffic Prediction

We conduct an ablation over possible selection functions on the METR-LA dataset. This includes varying the number of distance bins in our selection function. Additionally, we experiment with linearly separated bins by canceling the exponential distance used in the dataset for edge computation. Lastly, we experiment with directional selection based on the GPS coordinates of the sensors in the dataset. Again, we find that the specific selection function used has minimal impact on the final results, but that all selection functions have significantly better performance than the baseline network. Results of this ablation are shown in Table 2.

We also provide parameter and training time information in Table 2. Since the traffic graph is static in the METR-LA dataset (the sensors don't move over time), the selection function only needs to be processed once. This means only the repetition of the convolution and backprogation for each selection weight is increasing the training time. Addi-

Table 1. An ablation of selection functions using the SelGraphConv network. F1 scores are reported on the test set for MNIST Superpixels.

Selection	F1 Score
-	0.625
2 Directions	0.877
3 Directions	0.949
4 Directions	0.955
5 Directions	0.960
6 Directions	0.953
7 Directions	0.958
8 Directions	0.961
9 Directions	0.959
10 Directions	0.963
11 Directions	0.965
12 Directions	0.963
12 Directions	0.964
13 Directions	**0.966**
14 Directions	0.965
15 Directions	0.955
16 Directions	**0.966**
3 Distances	0.677
4 Distances	**0.681**
5 Distances	0.665
4 Directions + 4 Distances	0.957
8 Directions + 4 Distances	**0.968**

Table 2. The prediction of traffic speed (in mph) for a 3-layer SelGCN network using different selection functions. The mean absolute error after different amounts of time is reported for the METR-LA dataset. Parameter and training information is also included.

Selection	15 Min	30 Min	60 Min	# params	training time
-	6.225	6.331	6.521	51.1 K	1.4 h
3 Log Distances	2.846	3.277	3.826	161 K	7.2 h
4 Log Distances	2.814	3.211	3.707	198 K	8.3 h
5 Log Distances	2.821	3.218	3.727	235 K	9.4 h
6 Log Distances	2.801	3.194	3.676	272 K	10.5 h
7 Log Distances	2.817	3.204	3.673	309 K	11.6 h
3 Linear Distances	2.840	3.262	3.798	161 K	7.3 h
4 Linear Distances	2.823	3.217	3.709	198 K	8.4 h
5 Linear Distances	2.831	3.214	3.703	235 K	9.3 h
6 Linear Distances	2.803	3.194	3.681	272 K	10.5 h
7 Linear Distances	2.829	3.204	3.677	309 K	11.6 h
4 Directions	2.838	3.219	3.719	198K	8.4 h
8 Directions	2.805	3.176	3.617	346 K	12.7 h
4 Directions + 4 Distances	3.091	3.348	3.751	530 K	19.2 h

Table 3. The mean absolute error of SelDimeNet with various selection functions on multiple QM9 molecular targets.

Target	μ	α	ϵ_{HOMO}	ϵ_{LUMO}	$\langle R^2 \rangle$	ZPVE	U_0	U	H	G	c_v
Selection \Units	D	${a_0}^3$	meV	meV	${a_0}^2$	meV	meV	meV	meV	meV	$\frac{\text{cal}}{\text{mol K}}$
5 distances	0.030	0.046	23.6	18.3	0.458	1.35	9.43	8.83	8.89	9.28	0.024
8 distances	0.030	0.046	24.2	18.9	0.512	1.35	8.92	9.89	10.04	9.66	0.024
10 distances	0.029	0.047	**22.8**	18.8	0.487	1.33	9.72	9.57	9.34	9.92	0.025
3 angles	**0.028**	**0.044**	23.1	**18.0**	**0.451**	**1.25**	**7.88**	**7.87**	8.71	9.11	**0.023**
6 angles	**0.028**	0.046	24.4	18.3	0.455	1.27	8.00	8.32	**8.25**	**8.59**	0.024
9 angles	0.029	0.046	23.2	18.8	0.492	1.29	8.15	8.52	8.36	8.79	0.024
12 angles	0.031	0.049	25.0	19.7	0.531	1.32	8.74	8.45	9.05	9.56	0.024
5 dist & 3 ang	**0.028**	0.046	23.5	18.4	0.464	1.28	9.38	9.16	9.01	9.76	0.024

tionally, our traffic networks were trained on a single GPU rather than a multi-GPU setup. Because of these factors, these experiments are particularly well suited for demonstrating how the number of parameters and training time scale with the number of selections. The increase in time for each added selection illustrates that the smallest possible selection number that gives the desired level of task performance should be used.

3.3. QM9

For the QM9 dataset, we perform additional ablation over the number of distance and angle bins used in the selection function. Interestingly, we find that no one number of distance or angle bins performs consistenly better than other selection functions. Those results are shown in Table 3.