Group Mahalakshmi Sabanayagam

Christiansen Lars

August 10, 2024

1 Reproducability Report - Lars Christiansen

1.1 Code Repository:

The paper provides a code basis to run all experiments, which is largely based on the existing GraphGPS repo. The authors changes are minor dataset adjustments by providing virtual nodes on the datasets.

1.2 Challenges:

Following the authors' instructions, I encountered several challenges. One involved dependency issues between PyTorch and PyTorch-Geometric. This hurdle was overcome by installing all available packages using the conda package solver instead of pip.

There were also some code-related issues that might not be present in older library versions. Due to limited documentation for PyTorch-Geometric, pinpointing the exact cause of these errors proved difficult. One example involved a duplicate optimizer configuration, which seemed to be specific to the used PyTorch-Geometric version (similar to this reported Issue](https://github.com/rampasek/GraphGPS/issues/50)). Fortunately, most of these problems were resolved through minor adjustments to the code, guided by the error messages themselves.

1.3 Install packages:

Its recommended to create a new jupyter kernel as described in the Readme.md

[ ]:
```
%conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c pytorch -c␣
↪nvidia
%conda install pyg -c pyg
%conda install lightning -c conda-forge
%conda install yacs ogb pandas scikit-learn performer-pytorch wandb
%conda install openbabel fsspec rdkit -c conda-forge
```

[2]:
```
import torch
print(torch.__version__)
print(torch.version.cuda)
```

```
2.2.2
12.1
```

Install additional packages and replace with your cuda and torch version:

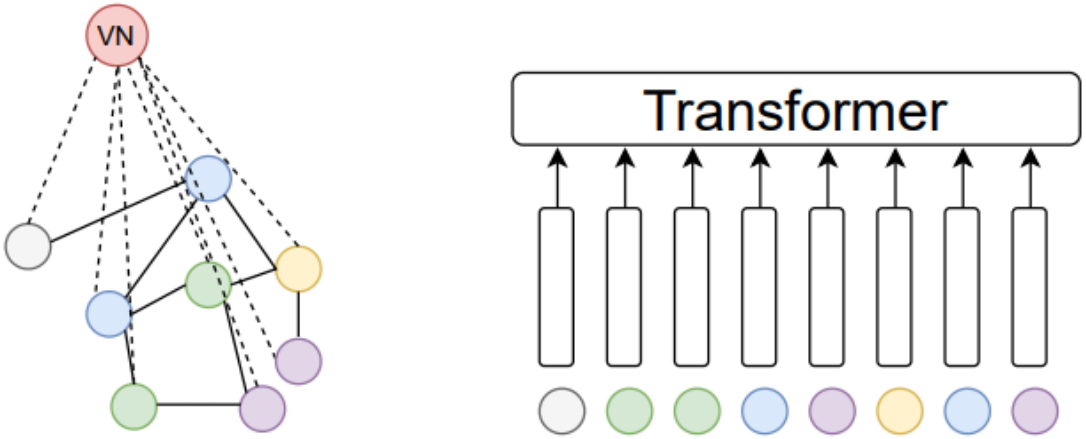
```
[ ]: %pip install pyg_lib torch_scatter torch_sparse torch_cluster torch_spline_conv␣
      ↪-f https://data.pyg.org/whl/torch-2.2.2+cu121.html
     %conda install -c dglteam/label/th22_cu121 dgl
```

1.4 Main Results of the paper

Recent advances in Graph Transformers (GTs) are explored in this paper, comparing them to the well-established Message Passing Neural Networks (MPNNs). GTs have demonstrated the ability to both outperform and approximate MPNNs. A key challenge in Graph Neural Networks (GNNs) is oversmoothing, where features across nodes become increasingly similar with deeper layers. This paper proposes a novel approach using Virtual Nodes (VNs) to mitigate this issue. VNs function like regular nodes but connect to all others, acting as a central hub for global information. Theoretically, the paper proves that VNs can simulate specific forms of Graph Transformers. Experiments further suggest that VNs may be a promising way to enhance overall model performance.

The following graph shows the first experiment of the paper. Covering the performance gains by adding a VN for graph classification tasks (peptides-func) and graph regression (peptides-struct). This shows the overall performance in long range interaction modeling tasks.

Model	# Params.	Peptides-func		Peptides-struct	
		Test AP before VN	Test AP after VN ↑	Test MAE before VN	Test MAE after VN ↓
GCN	508k	0.5930±0.0023	0.6623±0.0038	0.3496±0.0013	**0.2488±0.0021**
GINE	476k	0.5498±0.0079	0.6346±0.0071	0.3547±0.0045	0.2584±0.0011
GatedGCN	509k	0.5864±0.0077	0.6635±0.0024	0.3420±0.0013	0.2523±0.0016
GatedGCN+RWSE	506k	0.6069±0.0035	**0.6685±0.0062**	0.3357±0.0006	0.2529±0.0009
Transformer+LapPE	488k	0.6326±0.0126	-	0.2529±0.0016	-
SAN+LapPE	493k	0.6384±0.0121	-	0.2683±0.0043	-
SAN+RWSE	500k	0.6439±0.0075	-	0.2545±0.0012	-



Virtual Node connected to all other nodes in the graph vs the simplified multihead attention mechanism in transformers.

1.5 Setup:

1.5.1 Data:

The data originates from the Long Range Graph Benchmark (LRGB). This benchmark provides two particularly relevant datasets for our study: Peptides-func and Peptides-struct. Both datasets focus on chemical compounds and their structures, represented as graphs. Importantly, achieving strong performance on these tasks necessitates long-range reasoning capabilities within the model.

1.5.2 Models:

This section explores the performance of various graph neural network architectures on tasks requiring long-range reasoning. The analysis utilizes several established models, including GCN, GINE, GatedGCN variants, Transformer with RWSE, and Spectral Attention Network (SAN) with different pre-processing techniques (LapPE and RWSE). The results (presented in a separate table) indicate that Transformer models generally achieve superior performance on both graph regression and graph classification compared to models without virtual nodes. Notably, all models exhibit a relatively small and comparable parameter footprint.

1.6 Reproduce Results:

To run all models as an interactive bash job run following commands. Note that this takes roughly 24 hours to train all models.

[1]:
```
%cd experiment1
```

```
/home/schwollie/Desktop/paper-20/experiment1
```

```
/home/schwollie/anaconda3/envs/paper/lib/python3.12/site-
packages/IPython/core/magics/osm.py:417: UserWarning: using dhist requires you
to install the `pickleshare` library.
  self.shell.db['dhist'] = compress_dhist(dhist)[-100:]
```

[ ]:
```
!cat runAll.sh
!chmod +x runAll.sh
!bash -i ./runAll.sh
```

Prepare results for generating plot: Use path “experiment1/results_pre_trained/” for pretrained variant

[3]:
```
%cd ..
```

```
/home/schwollie/Desktop/paper-20
```

[3]:
```
from experiment1.results_processing import *

results_folder = "experiment1/results_pre_trained/"

json_results = get_stats_json_paths(results_folder)
print("model count: " + str(len(json_results)))
```

```
# Now find the best stat for each model.
best_stats = find_best_stat(json_results)

# generate tables:
table_func = generate_table("peptides-func", best_stats, False)
table_struct = generate_table("peptides-struct", best_stats, False)
```

```
model count: 46
```

create the plot:

[4]:
```
# Create a figure and an axis

fig, ax = plt.subplots(2, 1, figsize=(15, 6))

def plot_table(pos, data, name):
    cellText = data.values.tolist()
    colLabels = data.columns.tolist()

    ax[pos].table(cellText=cellText, colLabels=colLabels, loc='center')
    ax[pos].set_title(name, pad=2)
    ax[pos].axis('off')

plot_table(0, table_func, "peptides-func")
plot_table(1, table_struct, "peptides-struct")
plt.tight_layout()
plt.show()
```

peptides-func

Model & Parameters	ap↑ (before VN)	ap↑ (after VN)
GCN #508K	0.58323	0.67096
GINE #476K	0.56104	0.62751
GatedGCN #510K	0.58673	0.66052
GatedGCN+RWSE #506K	0.61699	0.66643
SAN #493K	0.58229	nan
SAN+RWSE #500K	0.60335	nan
Transformer+LapPE #489K	0.64112	nan

peptides-struct

Model & Parameters	mae↓ (before VN)	mae↓ (after VN)
GCN #509K	0.34666	0.24763
GINE #476K	0.35825	0.25677
GatedGCN #510K	0.34239	0.25238
GatedGCN+RWSE #507K	0.33593	0.2533
SAN #493K	0.26923	nan
SAN+RWSE #500K	0.26634	nan
Transformer+LapPE #489K	0.25631	nan

Comparing my results to the paper's tables reveals a high degree of similarity. While some of my models appear to have performed slightly worse (probably due to too few epochs in trainging time) the key finding remains consistent: the inclusion of virtual nodes demonstrably improves the

performance of all models.

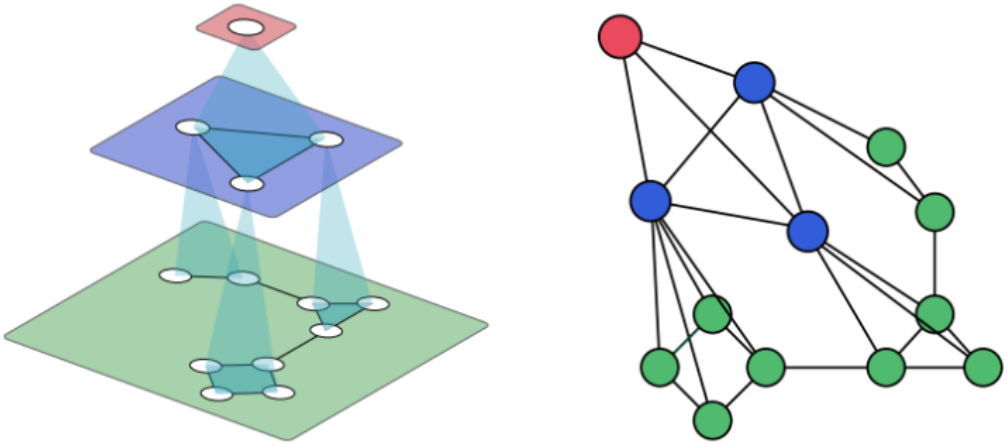
1.7 Extension:

The primary concept for the extension is to introduce additional virtual nodes. For large graphs containing millions of nodes, a single virtual node may not suffice to store all pertinent information. Various strategies exist to leverage the addition of multiple virtual nodes. The initial strategy is to simply incorporate more than one virtual node. A more sophisticated approach involves utilizing the graph structure to connect specific nodes to distinct virtual nodes. Overall, this method promises to efficiently capture more long-range dependencies between nodes, as multiple virtual nodes can store more information than a single one.

1.7.1 Multiple Virtual Nodes:

In this approach, we introduce multiple virtual nodes to all existing nodes. For v virtual nodes and n nodes in the graph, we add $(v \times n \times 2)$ additional edges connecting to all virtual nodes. The critical question is whether more virtual nodes can genuinely enhance learning. For instance, in smaller graphs, the graph structure might become dominated by the virtual nodes, potentially impeding learning. Conversely, there is potential to capture more information that a single virtual node might miss. To ensure that different virtual nodes encapsulate distinct information, we introduce 2 new edge types for each virtual node (in/out), ensuring each virtual node is treated uniquely within the graph.

1.7.2 Hierarchical Support Graph:



The concept behind this extension is to encode structural data through recursive graph clustering. Instead of connecting all graph nodes to virtual nodes, this method connects only a subset of nodes at each layer. Consequently, a hierarchical graph structure is formed by clustering nodes, linking them to a virtual node, and then recursively connecting these virtual nodes. With each layer, the number of nodes is reduced. This process continues until a single node remains. Thus, the graph is augmented with a support graph rather than individual nodes. The Metis algorithm is often employed for graph partitioning. However, this approach has the drawback that the shortest path between node pairs is no longer guaranteed to be 2; in the worst case, it could be twice the number of layers. It is important to note that the current implementation and the results only use clustering

but the clustered nodes have no intermediate connection.

1.7.3 Results:

[6]:
```
table_func = generate_table("peptides-func", best_stats, True)
table_struct = generate_table("peptides-struct", best_stats, True)
fig, ax = plt.subplots(2, 1, figsize=(15, 6))
plot_table(0, table_func, "peptides-func")
plot_table(1, table_struct, "peptides-struct")
plt.tight_layout()
plt.show()
```

peptides-func

Model & Parameters	ap↑ (before VN)	ap↑ (after VN)	ap↑ (after 5 VN)	ap↑ (after 10 VN)	ap↑ (after HSG)
GCN #508K	0.58323	0.67096	0.66839	0.66062	0.62158
GINE #476K	0.56104	0.62751	0.61175	0.61764	0.59893
GatedGCN #510K	0.58673	0.66052	0.67176	0.6647	0.6188
GatedGCN+RWSE #506K	0.61699	0.66643	0.67888	0.67577	0.61541
SAN #493K	0.58229	nan	nan	nan	nan
SAN+RWSE #500K	0.60335	nan	nan	nan	nan
Transformer+LapPE #489K	0.64112	nan	nan	nan	nan

peptides-struct

Model & Parameters	mae↓ (before VN)	mae↓ (after VN)	mae↓ (after 5 VN)	mae↓ (after 10 VN)	mae↓ (after HSG)
GCN #509K	0.34666	0.24763	0.24648	0.24856	0.26168
GINE #476K	0.35825	0.25677	0.25936	0.25381	0.26328
GatedGCN #510K	0.34239	0.25238	0.24854	0.25779	0.26192
GatedGCN+RWSE #507K	0.33593	0.2533	0.25393	0.25416	0.25888
SAN #493K	0.26923	nan	nan	nan	nan
SAN+RWSE #500K	0.26634	nan	nan	nan	nan
Transformer+LapPE #489K	0.25631	nan	nan	nan	nan

The experimental results visualized in the plot indicate a discernible performance enhancement when employing multiple virtual nodes. This improvement is particularly pronounced in the GCN models, with the optimal configuration observed at five virtual nodes. Notably, this outperforms the reported performance of benchmark models such as GatedGCN. Conversely, the Hierarchical support graph extension did not yield the anticipated results. While demonstrating superior performance compared to the absence of virtual nodes, its effectiveness is surpassed by the simpler approach of adding a single virtual node. This suggests some problems within the training or the data manipulation and there seems to be more potential than this results suggest.