Neural Network As Neural Network Input

A Technical Report

Fan-Yun Sun, Bo-Cheng Chu, Winston Hsu
National Taiwan University
{b04902045,b04902044, whsu}@ntu.edu.tw

Introduction

Both supervised and unsupervised learning on graph structured data has made incredible progress in recent years. However, benchmark graph datasets are limited to social networks, citation networks, or bioinformatic datasets such as protein-protein interaction. In this paper, we extend the realm of graph benchmark dataset to computation graphs of neural networks based on (Bennani-Smires et al. 2018). Analyzing computation graph has the potential to aid the automation of neural network architecture design. Other than that, There are many other potential useful cases in other domains. For example, in RL, different style of implementation can often lead to significantly different results. In that case, directly analyzing computation graphs can help identify the underlying reason and ease the process of debugging. This we left as future work for now.

Dataset

Inspired by (Bennani-Smires et al. 2018), we scraped tensorflow model files using github API and further tag them with associated meta data. However, in (Bennani-Smires et al. 2018), they tagged those graphs by matching keywords in either the title or the description of the repository[1]. In this paper, we further scraped the READMEs of those repositories as a corpus for keyword matching in order to increase the number of labelled model files. Duplicate graphs from the same repository are removed.

We can summarize the special properties of this dataset as below:

- **Rich meta data**: These datasets not only contain basic graph structure, node labels, and node attributes, they also contains rich graph-level meta data such as the README associated with every repository.
- **Hierarchical representation**: The node attribute NAME gives graph instances a hierarchical representation(as shown in Tensorboard).
- **Direct Acyclic Graph**: Existing graph classification benchmark datasets[2] are collected as undirected graph. Nevertheless, computational graphs are inherently directed acyclic graphs.

Gitgraph Dataset

We introduced a benchmark dataset for graph classification, *Gitgraph*. For every graph instances in both datasets, every node is associated with a label OP and an string iden-

[1]confirmed with the author

[2]`https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets`

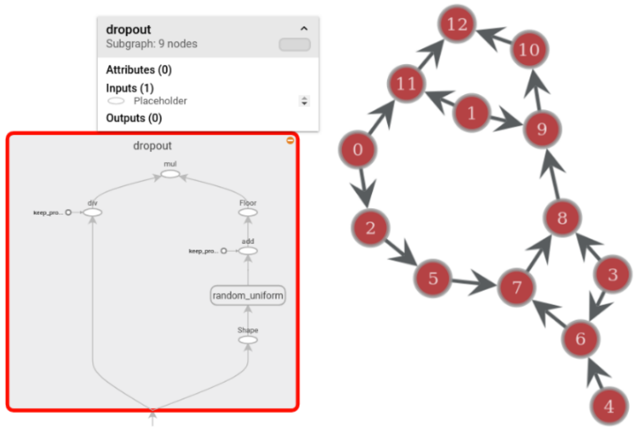

Figure 1: The left figure is the Tensorboard visualization of the graph on the right.

	NAME	OP
0	Placeholder	Placeholder
1	dropout/keep_prob	Const
2	dropout/Shape	Shape
3	dropout/random_uniform/min	Const
4	dropout/random_uniform/max	Const
5	dropout/random_uniform/RandomUniform	RandomUniform
6	dropout/random_uniform/sub	Sub
7	dropout/random_uniform/mul	Mul
8	dropout/random_uniform	Add
9	dropout/add	Add
10	dropout/Floor	Floor
11	dropout/div	RealDiv
12	dropout/mul	Mul

Table 1: NAME and OP for graph nodes shown in Figure 1.

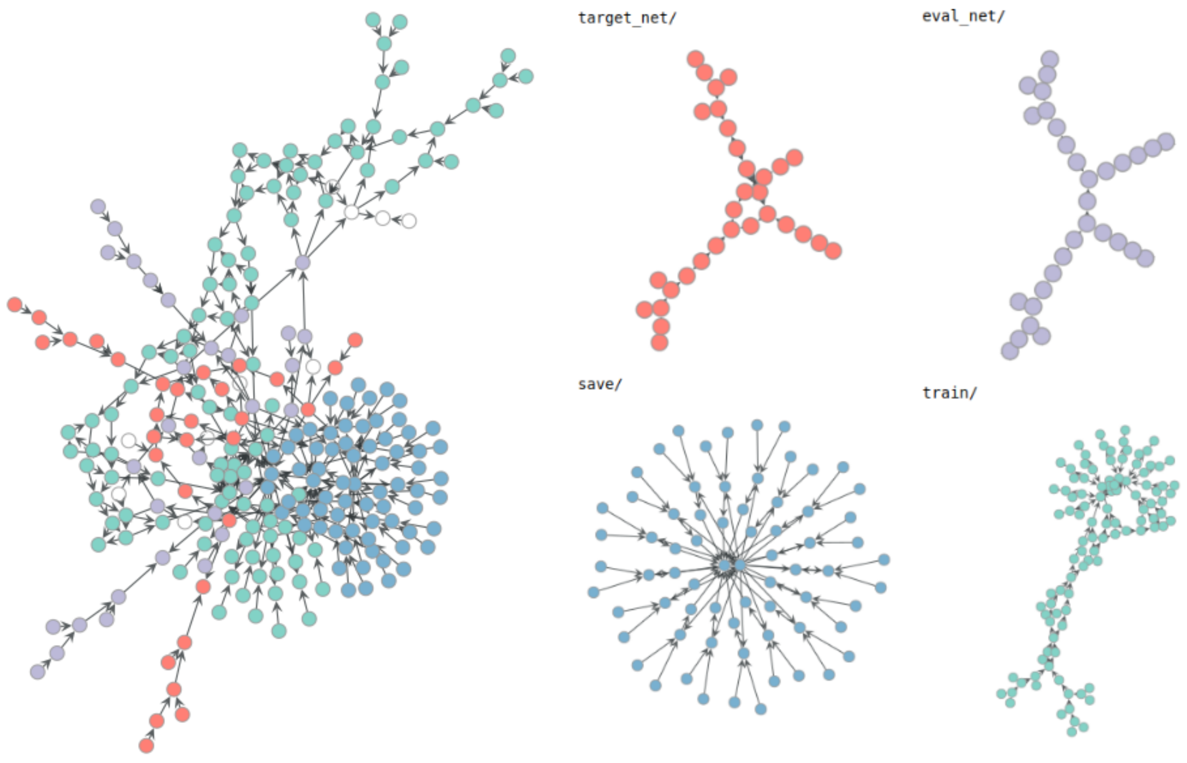

Figure 2: A random computational graph and four subgraphs visualized. This is a computational graph for reinforcement learning(separate target networks). Note that subgraphs are disjoint in this dataset.

Listing 1: dropout code snippet

```
import tensorflow as tf
x = tf.placeholder("float", shape=[None, 1])
tf.nn.dropout(x, .5, name=None)
```

Table 2: Dataset statistics

Dataset	Gitgraph
# of graphs	306
# of classes	3
# of nodes (avg.)	510.19
# edges (avg.)	758.98
# distinct node labels	246

tifier NAME; OP denotes the operation for the node and NAME is the unique identifier for the node. Usually, we can infer what modules that node belong to by inspecting its NAME. For example, consider the following code snippet[3] in Listing 1. The last line of code creates nodes with a prefix of "dropout". From table 1, we can infer that node 3-8 forms a sub-module under the module "dropout" with a common prefix "random_uniform". This is also how Tensorboard (Wongsuphasawat et al. 2017) group modules, as shown in the left diagram of Table 1. The statistics of the the dataset is summarized in Table 2 and we provide a visualization of a random sample in Figure 2.

This dataset contains 306 tensorflow model files(computational graphs), 88 of them are tagged *Reinforcement*, 119 are tagged *Image*, and 79 are tagged *Text*. The keywords used to match those labels are *reinforcement*,

[3]Tensorflow v1.4, Python 3.6

nlp translation sentiment ner, and *image* respectively.

Gitgraph-stars Dataset

We introduced another benchmark dataset for regression on graphs. For this dataset, we further web scraped the stars[4] of every github repository[5]. Ideally, to compare the stars of different repositories, we should consider the established time of the repositories since repositories that exist longer may accumulate more views. Nevertheless, most repositories get the majority of their stars in a rather short period of time (shortly after publication or public exposure) so we chose not to further adjust it. This is inherently a regression problem but since the label may be noisy, we cn also discretize them and formulate it into a graph classification task. If there are multiple graph instances in a single repository, we picked one at random.

This dataset contains 200 tensorflow models files(computational graphs), and every single graph comes from a different repository. 69 of them have below 5 stars, 61 of them have 6 to 20 stars, and 70 of them have more than 20 stars.

Experiment and Results

We benchmark Gitgraph using existing graph-level representation learning methods and supervised graph classification methods.

Experiment Settings

10-fold cross validation is reported to make a fair comparison. For unsupervised method, parameters of downstream classifiers are independently tuned using training folds data and best average classification accuracies are reported for each method. Experiments are repeated 5 times. To ensure fair comparison, the embedding dimension of our method and all unsupervised methods are set to 512. For node embedding methods, graph representation is obtained by average all node embeddings. The classifiation accuracies is computed using LIBSVM (Chang and Lin 2011), and the C parameter was selected from $\{10^{-3}, 10^{-2}, \dots, 10^{2}, 10^{3}\}$.

Baselines

We adopted these following baselines:

- Random guess
- Shortest Path Kernel (SP) (Borgwardt and Kriegel 2005)
- Weisfeiler-lehman Sub-tree Kernel (WL) (Shervashidze et al. 2011)
- Graphlet Kernel (GK) (Shervashidze et al. 2009)
- graph2vec (Narayanan et al. 2017)
- subgraph2vec (Narayanan et al. 2016)
- graphSAGE (Hamilton, Ying, and Leskovec 2017)
- KCNN (Nikolentzos et al. 2017)

[4]More specifically, the field *watchers_count* in Github API.

[5]More specifically, stars accumulated before 2018-08-26 21:57:00

Results

Experiment results are shown in table 3.

Model	Accuracy
Random guess	0.42 ± 0.13
Shortest path kernel	OOM
Graphlet kernel	OOM
WL kernel	0.73 ± 0.12
graph2vec	0.59 ± 0.10
subgraph2vec	OOM
graphSAGE	0.69 ± 0.11
KCNN	0.77 ± 0.10

Table 3: Classification accuracy on Gitgraph dataset. “OMR” is out of memory error.

Future work

The current label may be noisy as we annotate each computation graph simply by matching keyword to their metadata. Furthermore, when we scraped data from github, we had no quality control. Thresholding stars of the repository can be a naive first step but we left it as future work.

References

[Bennani-Smires et al. 2018] Bennani-Smires, K.; Musat, C.; Hossmann, A.; and Baeriswyl, M. 2018. Gitgraph-architecture search space creation through frequent computational subgraph mining. *arXiv preprint arXiv:1801.05159*.

[Borgwardt and Kriegel 2005] Borgwardt, K. M., and Kriegel, H.-P. 2005. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*, 8–pp. IEEE.

[Chang and Lin 2011] Chang, C.-C., and Lin, C.-J. 2011. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)* 2(3):27.

[Hamilton, Ying, and Leskovec 2017] Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 1024–1034.

[Narayanan et al. 2016] Narayanan, A.; Chandramohan, M.; Chen, L.; Liu, Y.; and Saminathan, S. 2016. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928*.

[Narayanan et al. 2017] Narayanan, A.; Chandramohan, M.; Venkatesan, R.; Chen, L.; Liu, Y.; and Jaiswal, S. 2017. graph2vec: Learning distributed representations of graphs. *CoRR* abs/1707.05005.

[Nikolentzos et al. 2017] Nikolentzos, G.; Meladianos, P.; Tixier, A. J.; Skianis, K.; and Vazirgiannis, M. 2017. Kernel graph convolutional neural networks. *CoRR* abs/1710.10689.

[Shervashidze et al. 2009] Shervashidze, N.; Vishwanathan, S.; Petri, T.; Mehlhorn, K.; and Borgwardt, K. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, 488–495.

[Shervashidze et al. 2011] Shervashidze, N.; Schweitzer, P.; Leeuwen, E. J. v.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12(Sep):2539–2561.

[Wongsuphasawat et al. 2017] Wongsuphasawat, K.; Smilkov, D.; Wexler, J.; Wilson, J.; Mane, D.; Fritz, D.; Krishnan, D.; Viégas, F. B.; and Wattenberg, M. 2017. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE transactions on visualization and computer graphics* 24(1):1–12.