Applying Graph Neural Networks on Heterogeneous Nodes and Edge Features

Frederik Diehl
fortiss
Research Institute of the Free State of Bavaria
associated with Technical University of Munich
Munich, Germany
f.diehl@tum.de

Abstract

While there have been many works on Graph Neural Networks (GNNs), most of these approaches only consider homogeneous-node graphs without edge features. However, many interesting problems feature different types of nodes and edge features. Based on the example of a power grid, we show that GNNs can learn to model heterogeneous nodes with little additional information given, but that special-purpose network architectures can improve upon that. We compare five different approaches which form a base for any practitioner to start experiments for their specific problem.

1 Introduction

Modelling problems as graphs promises both data-efficiency and the introduction of useful prior knowledge into the problem structure. This has been shown to work on problems from different domains: Modelling physical interaction between sets of objects (Battaglia et al., 2016), operating on citation networks (for example Kipf & Welling (2016)), on co-purchasing graphs (Shchur et al., 2018), or on source code (Allamanis et al., 2018). Most papers benchmark their approach on citation networks, a small protein dataset, or on online interaction datasets.

However, these datasets all share one characteristic: Nodes are homogeneous, sharing the same features and type (for example, all nodes are academic papers). Edges usually do not have any features; if they do, they also are homogeneous (for example citations between papers). Accordingly, there is a lack of approaches working on heterogeneous node and edge types. This poses an obstacle to wide-spread adoption of GNNs to more interesting problems, which often feature diverse types of nodes and edge features.

In this work, we present a single case study of such a problem, showcase different ways of modelling it as a graph, and finally conduct benchmarking to present the reader with an estimate on the different methods' performances. We do not provide a definite answer to the problem of how to model heterogeneous graphs, but instead aim to inspire the reader and provide a starting point for their own exploration.

2 Problem formulation and modelling

The problem considered in this paper is modelling Optimal Power Flow (OPF) in a power grid. A power grid consists of physical interconnections (busses) connected by branches, which may be transmission lines, transformers, or cables. Each bus may be connected to one or several components:

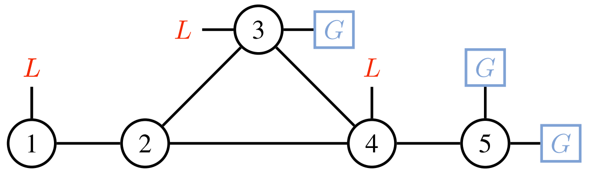

Figure 1: Exemplary five-bus power grid. Busses are depicted as (numbered) circles, loads and generators are depicted with an L and a G respectively. As can be seen, busses can have multiple generators associated with them (bus 5), can have both generators and loads (bus 3) or might act only as an interconnection (bus 2). Busses, generators, loads, shunts, and branches all have features associated with them, while bus-to-component connections do not.

generators (producing power), loads (requesting power), and shunt capacitors (for stabilizing the load). An example power grid is shown in Fig. 1.

OPF refers to the problem of optimizing the generator outputs such that both physical limits for every component are respected and the total power generation cost is minimized. Since solving that problem optimally is NP-hard (Bienstock & Verma, 2015), we are interested in training a GNN to predict it. For this, we predict several continuous output values for busses, generators, and branches. We use a synthetic example with 2000 busses and hourly load data for a year (Birchfield et al., 2017). This power grid represents the transmission grid of Texas, and we split it by days.

Accordingly, our problem both features heterogeneous nodes, edge features, and predictions for multiple node types.

3 Modelling the power grid

Most GNN methods consider only node features and adjacencies, ignoring edge features. Battaglia et al. (2018) proposed a general framework for GNNs which can operate on node, edge, and global features. We adapt this framework, ignoring global features. Accordingly, each update step first computes the new edge features based on source and target

$$e^n = f_e^n \left(v_s^{n-1} \| e^{n-1} \| v_t^{n-1} \right), \tag{1}$$

where ^{n} is the layer, e are the edge attributes, f is a neural network, v_s and v_t are source and target nodes, and $\|$ is concatenation. Afterwards, the new node features are computed with

$$v^n = f_v^n \left(v^{n-1} \| \sum_{v,e \in N} f_{ve}^n \left(v^{n-1} \| e^n \right) \right), \tag{2}$$

where N is shorthand for the neighbours.

This formulation gives us a network architecture which supports edge features. Several strategies are possible for acting on heterogeneous nodes. These are also depicted in Fig. 2.

Independence assumption The simplest model assumes independence of components, and learns one feed-forward network for each component type. This represents nothing more than a baseline, as it does not take graph topology into account.

Heterogeneous nodes We can build a graph considering both busses and all components as nodes. Edges are both the original branches and component-to-bus connections. To deal with different node feature sizes, we learn separate linear embeddings into the same space for busses and components. Since component-to-bus connections do not have any features, we learn fixed representations for each type of component-to-bus connection.

Summarizing features Classically, we could also concatenate component features to their corresponding bus. Since one bus might have multiple components of the same type, we rely on existing

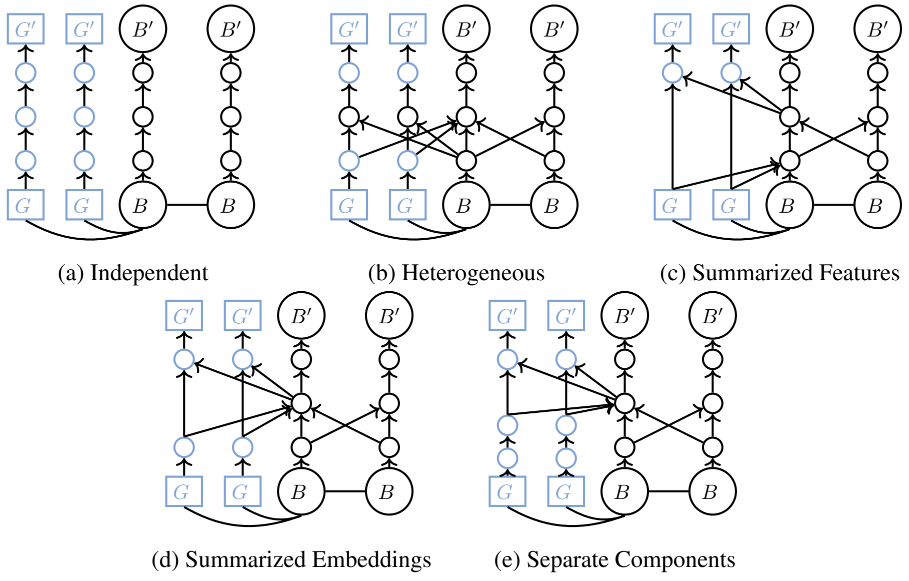

Figure 2: Graphical overview over the different model types operating on a very small graph and the resulting dataflow. All networks have one encoding, one GNN layer (excepting the Independent model), and one classification layer. The two generators are assumed to be connected to one bus. We do not show the influence of edge features onto computation. Colour-coding is used to denote parameter sharing for that layer, organized by components.

knowledge on how to combine them. For example, we can sum up maximum and minimum power generation capabilities for generators. To enable different predictions for generators assigned to the same bus, the output layer concatenates both the bus' final representation and the original generator features.

Summarizing embeddings Instead of working on raw features, we can also first embed each component. Afterwards, we apply an associative and commutative function (we use the sum) to summarize them. By doing this, we allow the output layer to act on a representation of the component instead of the original features. This is similar to the approach taken by Zaheer et al. (2017) for representing object sets.

Treating components separately Lastly, we can include existing knowledge: We know that components are only connected to busses, and that they lack features. We therefore define updates to the components as

$$c^n = f_c^n\left(b^{n-1} \| c^{n-1}\right), \qquad o^n = f_o^n\left(b^{n-1} \| \sum_{s \in N} s^n \| \sum_{g \in N} g^n \| \sum_{l \in N} l^n\right), \tag{3}$$

where c is a type of component, s, b, g, and l are the shunt, bus, generator, and load features, and N are shorthand for the corresponding neighbours. Branch features e are then computed according to their adjacent nodes,

$$e^n = f_e^n\left(b_s^n \| e^{n-1} \| b_t^n\right), \tag{4}$$

where b_s and b_t are source and target nodes, followed by final bus features

$$b^n = f_b^n\left(o^n \| \sum_{o,e \in N} f_{be}^n\left(o^n \| e^n\right)\right). \tag{5}$$

This model architecture allows us to include existing knowledge as a bias: Each component is treated differently but is connected to one (or several) nodes, edge features are included, and component output depends on the connected bus.

Method	MAE
Independent	0.2509
Heterogeneous	0.0203
Summarized Features	0.0186
Summarized Embeddings	0.0171
Separate Components	0.0218

Table 1: Comparison of final model performance.

4 Results and discussion

We train all models identically using the Adam optimizer at default parameters, and train for 500 epochs using an MSE loss. We scale all models to the same number of parameters (about 200 k). All models use 8 layers, residual connections, and batch normalization.

We implement the models using pytorch (Paszke et al., 2017) and the pytorch-geometric package (Fey & Lenssen, 2019).

We report mean average error in Table 1, and clearly see that the graph structure is important for this problem. At the same time, simply treating all components as nodes in the graph performs well, and even better than at least one problem-specific architecture. The Independent model only produces legal solutions for about a quarter of our test cases. Except for the Separate Component model, which fails on two cases, all other models produce a legal solution for every single one of our 2208 test samples without any further enforcement of physical knowledge.

On the other hand, the right architecture using prior knowledge surpasses such a general-purpose architecture.

In summary, the results clearly shows that introducing prior knowledge through network architecture is necessary to achieve the best performance on more complex datasets. Given the heterogeneity of such datasets, no single recommendation can be made on how to achieve it; however, we have shown several potential strategies which readers can use as a base for experiments.

Yet, even a simple general-purpose representation can already deliver good results, leaving the inclusion of prior knowledge for hyper-parameter tuning.

5 Conclusion

Since typical GNNs are applied on simple problems of homogeneous nodes and lacking edge weights, there is a lack of discussion on dealing with more interesting problems featuring nodes of different types and edge weights. In this work, we presented a case study of applying GNNs on a power grid task, showcasing different strategies to construct an architecture. By comparing five different approaches, we have shown that incorporating more problem-specific knowledge than just a graph structure into the model results in better performance. At the same time, our experiments indicate that the simplest approach can already give good results. We therefore give practitioners a valuable starting point to their own experiments to apply GNNs to an ever larger class of problems.

Acknowledgments

This research was funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy, project Dependable AI.

References

Allamanis, M., Brockschmidt, M., and Khademi, M. Learning to represent programs with graphs. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BJOFETxR-`.

Battaglia, P., Pascanu, R., Lai, M., and Rezende, D. J. Interaction Networks for Learning about Objects, Relations and Physics. 2016.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261 [cs, stat]*, June 2018.

Bienstock, D. and Verma, A. Strong NP-hardness of AC power flows feasibility. 2015. URL `http://arxiv.org/abs/1512.07315`.

Birchfield, A. B., Xu, T., Gegner, K. M., Shetye, K. S., and Overbye, T. J. Grid structural characteristics as validation criteria for synthetic networks. 32(4):3258–3265, 2017. ISSN 0885-8950. doi: 10.1109/TPWRS.2016.2616385.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR 2017*, 2016.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of Graph Neural Network Evaluation. *arXiv:1811.05868 [cs, stat]*, November 2018.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep Sets. In *Advances in Neural Information Processing Systems*, pp. 3391–3401, 2017.