OUTCOME CORRELATION IN GRAPH NEURAL NETWORK REGRESSION

Junteng Jia and Austin R. Benson

SIAM Workshop on Network Science 2020
July 9–10 · Toronto

Summary

A typical graph neural network (GNN) pipeline assumes that node labels are conditionally independent given their neighborhoods. However, this assumption is far from true in many real-world datasets. We address this problem with an interpretable and efficient framework that can improve any graph neural network architecture simply by exploiting correlation structure in the regression residuals.

Correlated Graph Neural Networks

In many graphs, nodes have attributes; for example, an online social network may have information on a person's location, gender, and interests. Semi-supervised prediction problems on graphs combine the graph topology and node attributes with some labels on a subset of nodes to make predictions for nodes where such labels are missing. For example, in online social networks, we may have the age of some users from registration or survey data and want to infer the age of other users for better targeted advertising.

Graph neural networks (GNNs) are a successful class of methods for such tasks [1]. The basic idea of GNNs is to first encode the local environment of each node i into a vector representation $\mathbf{h}_i$ by aggregating its own features along with the features of its neighbors. The outcome of each node is predicted independently as a function of the vector representation. More formally,

$$\mathbf{h}_i = f\left(\mathbf{x}_i, \{\mathbf{x}_j : j \in N_K(i)\}, \theta\right); \quad \hat{y}_i = g(\mathbf{h}_i, \theta), \qquad (1)$$

where $\mathbf{x}_i$ is a feature vector for node i and $N_K(i)$ is the K-hop neighborhood of i ($K = 2$ in practice). In regression, the training error is usually measured with a squared-error loss $\sum_{i \in L}(\hat{y}_i - y_i)^2$ on the set L of labeled nodes.

However, a fundamental limitation of GNNs is that they predict each outcome independently given the representations and ignore outcome correlation of neighboring nodes. Figure 1 shows an example illustrating why this is problematic using a graph with topological and feature symmetry but monotonically varying node labels. The GNN fails to distinguish nodes v_2 and v_4 (Fig. 1(b)) and therefore cannot predict them both correctly. On the other hand, traditional graph-based semi-supervised learning algorithms such as those based on label propagation (LP) [3], work very well in this case as the labels vary smoothly over the graph.

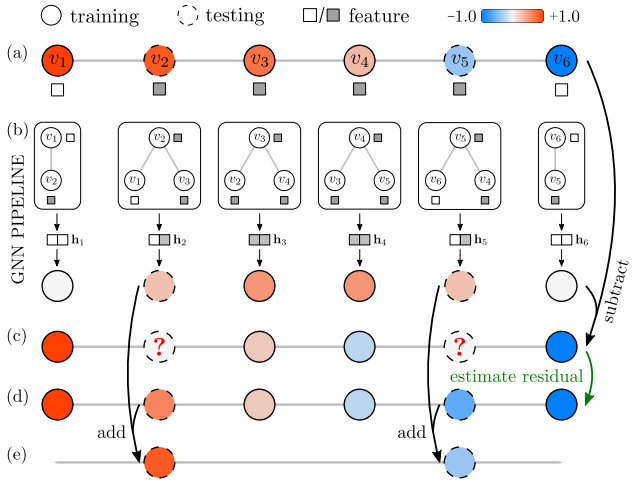

Figure 1: **Limitations of GNN regression and our proposed fix. The node labels decrease from v_1 (+1.0) to v_6 (−1.0), and most interior nodes have positive-valued labels. (a) Each node's degree is used as its feature. Nodes are colored based on their labels. The training nodes are v_1, v_3, v_4, v_6. (b) The GNN encodes nodes by vectors $\mathbf{h}_i$, which are used independently for label prediction.st The GNN captures the positive trend for interior nodes but fails to distinguish v_1, v_2, v_3 from v_6, v_5, v_4 due to symmetry. (c) GNN regression residuals for the training nodes. (d) Our *Correlated GNN* method estimates the residuals on testing nodes v_2, v_5. (e) The estimated residuals are added to GNN outputs as our final predictions, yielding good estimates.**

In Figure 1, node features are somewhat — but not overwhelmingly — predictive. We propose *Correlated Graph Neural Networks* (C-GNNs) to take advantage of outcome correlation to improve prediction performance. A C-GNN uses a GNN as a base regressor to capture the (possibly mild) outcome dependency on node features and then further models the regression residuals on all nodes.

Model. Following standard statistical arguments common for ordinary least squares, the typical loss for a GNN

in the regression setting is equivalent to maximizing the likelihood of a fully factorized joint distribution of labels, where each label distribution conditioned on the node representation is a univariate Gaussian:

$$p(\mathbf{y} \mid G) = \textstyle\prod_{i \in V} p(y_i \mid \mathbf{h}_i); \quad y_i \mid \mathbf{h}_i \sim \mathcal{N}(\hat{y}_i, \sigma^2) \qquad (2)$$

Consequently, the residuals $r_i = \hat{y}_i - y_i$ are implicitly assumed to be independent with mean zero. There's no a priori reason to assume independence, and we observe that errors tend to be correlated in real-world data. We choose a simple multivariate Gaussian to model the correlation:

$$\mathbf{y} \sim \mathcal{N}\left(\hat{\mathbf{y}}, \Gamma^{-1}\right) \iff \mathbf{r} \equiv \mathbf{y} - \hat{\mathbf{y}} \sim \mathcal{N}\left(0, \Gamma^{-1}\right), \qquad (3)$$

where $\Gamma = \Sigma^{-1}$ is the inverse covariance (or precision) matrix, and $\mathbf{r}$ is the residual of GNN regression. We parameterize the precision matrix in a simple way that (i) uses the graph topology and (ii) turns out to be computationally tractable: $\Gamma = \beta(\mathbf{I} - \alpha\mathbf{S})$, where $\mathbf{S}$ is the normalized adjacency matrix. Here, β controls the overall magnitude of the residual and α reflects the strength and direction of the correlation. To be valid, Γ must be positive definite, which is true for $-1 < \alpha < 1$ and $\beta > 0$. When $\alpha = 0$, the model reduces to the standard GNN regression. When $\alpha \to 1$, Γ is the normalized Laplacian, and the noise is assumed to be smooth over the entire graph, which is the standard assumption in classical methods [3].

Given observed outcomes y_L on labeled nodes L, the precision matrix parameters α and β as well as the GNN weights θ are learned by maximizing the marginal likelihood. Computational cost is a major concern with this approach. Standard factorization-based algorithms for computing the matrix inverse and log determinant in the likelihood function and its derivatives scale cubicly in the number of nodes, which is prohibitive for graphs beyond a few thousand nodes. We show how to reduce these computations to linear in the number of edges, using recent tricks in stochastic trace estimation [2].

At inference time, our model predicts the outcomes on the unlabeled nodes U by maximizing their probability conditioned on the labeled nodes L. If we partition Eq. 3 into the labeled and unlabeled blocks,

$$\begin{bmatrix} \mathbf{y}_L \\ \mathbf{y}_U \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \hat{\mathbf{y}}_L \\ \hat{\mathbf{y}}_U \end{bmatrix}, \begin{bmatrix} \Gamma_{LL} & \Gamma_{LU} \\ \Gamma_{UL} & \Gamma_{UU} \end{bmatrix}^{-1} \right). \qquad (4)$$

Conditioning on the labeled nodes L, the outcome distribution on U is also a multivariate Gaussian:

$$\mathbf{y}_U \mid \mathbf{y}_L \sim \mathcal{N}\left(\hat{\mathbf{y}}_U - \Gamma_{UU}^{-1}\Gamma_{UL}\mathbf{r}_L, \Gamma_{UU}^{-1}\right). \qquad (5)$$

Our model uses the expectation of this conditional distribution as the final prediction: $\mathbf{y}_U^{\text{C-GNN}} = \hat{\mathbf{y}}_U - \Gamma_{UU}^{-1}\Gamma_{UL}\mathbf{r}_L$.

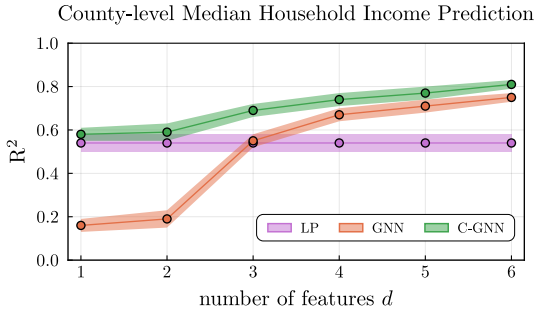

Figure 2: **Prediction accuracy for county-level median household incomes in an election map network as a function of the number of included features.**

Importantly, our approach makes no assumption on the GNN architecture, as the error modeling "sits on top" of the base regressor. One does not even have to use a GNN — under a linear model, our framework is performing generalized least squares. However, we find that GNNs indeed work well as base regressors for graph-based data.

Preview of results. We test our model on real-world datasets, including to predict demographics in U.S. election maps, traffic in road networks, and view counts in an online game streaming social network. Our model provides substantial improvements: a mean 14% improvement in R^2 over base GNNs across 10 datasets. One finding is that accounting for outcome correlation is most useful when features are only mildly predictive (Figure 2).

We also consider *inductive learning*: a model is trained on one graph where labels are available and deployed on another where labels are difficult to obtain. With a small fraction of labels on the new graph, the accuracy of a C-GNN can be even better than the transductive accuracy of a GNN. For example, we train a C-GNN to predict county-level unemployment rates with 60% labeled nodes using 2012 data. Using 10% of labels from 2016 data, the C-GNN has 0.65 test R^2, which is even more accurate than a GNN trained directly on 60% of 2016 labels ($R^2 = 0.53$).

References

[1] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[2] S. Ubaru, J. Chen, and Y. Saad. Fast estimation of tr(f(A)) via stochastic Lanczos quadrature. *SIAMX*, 2017.

[3] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *NeurIPS*, 2004.