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 $h_i$ 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,

$$h_i = f(x_i, \{x_j : j \in N_K(i)\}, \theta); \quad \hat{y}_i = g(h_i, \theta),$$ (1)

where $x_i$ 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.

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:

\[ p(y \mid G) = \prod_{i \in V} p(y_i \mid h_i); \quad y_i \mid h_i \sim \mathcal{N}(\hat{y}_i, \sigma^2) \quad (2) \]

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:

\[ y \sim \mathcal{N}(\hat{y}, \Gamma^{-1}) \iff r \equiv y - \hat{y} \sim \mathcal{N}(0, \Gamma^{-1}) , \quad (3) \]

where \( \Gamma = \Sigma^{-1} \) is the inverse covariance (or precision) matrix, and \( r \) 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: \( \Gamma = \beta(I - \alpha S) \), where \( S \) is the normalized adjacency matrix. Here, \( \beta \) controls the overall magnitude of the residual and \( \alpha \) reflects the strength and direction of the correlation. To be valid, \( \Gamma \) 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 \( \alpha \to 1 \), \( \Gamma \) 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 \( \alpha \) and \( \beta \) as well as the GNN weights \( \theta \) 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 cubically 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,

\[
\begin{bmatrix} y_L \\ y_U \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} y_L \\ y_U \end{bmatrix}, \Gamma_{LL}^{-1} \Gamma_{LU} \Gamma_{UL}^{-1} \Gamma_{UU} \right) . \quad (4)
\]

Conditioning on the labeled nodes \( L \), the outcome distri-