2018 IEEE 5th International Conference on Data Science and Advanced Analytics

Learning to Make Predictions on Graphs with Autoencoders

Phi Vu Tran
Strategic Innovation Group
Booz | Allen | Hamilton
San Diego, CA USA
tran_phi@bah.com

***Abstract*—We examine two fundamental tasks associated with graph representation learning: link prediction and semi-supervised node classification. We present a novel autoencoder architecture capable of learning a joint representation of both local graph structure and available node features for the multi-task learning of link prediction and node classification. Our autoencoder architecture is efficiently trained end-to-end in a single learning stage to simultaneously perform link prediction and node classification, whereas previous related methods require multiple training steps that are difficult to optimize. We provide a comprehensive empirical evaluation of our models on nine benchmark graph-structured datasets and demonstrate significant improvement over related methods for graph representation learning. Reference code and data are available at https://github.com/vuptran/graph-representation-learning.**

***Index Terms*—network embedding, link prediction, semi-supervised learning, multi-task learning**

I. Introduction

As the world is becoming increasingly interconnected, graph-structured data are also growing in ubiquity. In this work, we examine the task of learning to make predictions on graphs for a broad range of real-world applications. Specifically, we study two canonical subtasks associated with graph-structured datasets: link prediction and semi-supervised node classification (LPNC). A graph is a partially observed set of edges and nodes (or vertices), and the learning task is to predict the labels for edges and nodes. In real-world applications, the input graph is a network with nodes representing unique entities, and edges representing relationships (or links) between entities. Further, the labels of nodes and edges in a graph are often correlated, exhibiting complex relational structures that violate the general assumption of independent and identical distribution fundamental in traditional machine learning [10]. Therefore, models capable of exploiting topological structures of graphs have been shown to achieve superior predictive performances on many LPNC tasks [23].

We present a novel densely connected autoencoder architecture capable of learning a shared representation of latent node embeddings from both local graph topology and available explicit node features for LPNC. The resulting autoencoder models are useful for many applications across multiple domains, including analysis of metabolic networks for drug-target interaction [5], bibliographic networks [25], social networks such as Facebook ("People You May Know"), terrorist networks [38], communication networks [11], cybersecurity [6], recommender systems [16], and knowledge bases such as DBpedia and Wikidata [35].

There are a number of technical challenges associated with learning to make meaningful predictions on complex graphs:

- Extreme class imbalance: in link prediction, the number of known present (positive) edges is often significantly less than the number of known absent (negative) edges, making it difficult to reliably learn from rare examples;
- Learn from complex graph structures: edges may be directed or undirected, weighted or unweighted, highly sparse in occurrence, and/or consisting of multiple types. A useful model should be versatile to address a variety of graph types, including bipartite graphs;
- Incorporate side information: nodes (and maybe edges) are sometimes described by a set of features, called *side information*, that could encode information complementary to topological features of the input graph. Such explicit data on nodes and edges are not always readily available and are considered *optional*. A useful model should be able to incorporate optional side information about nodes and/or edges, whenever available, to potentially improve predictive performance;
- Efficiency and scalability: real-world graph datasets contain large numbers of nodes and/or edges. It is essential for a model to be memory and computationally efficient to achieve practical utility on real-world applications.

Our contribution in this work is a simple, yet versatile autoencoder architecture that addresses all of the above technical challenges. We demonstrate that our autoencoder models: 1) can handle extreme class imbalance common in link prediction problems; 2) can learn expressive latent features for nodes from topological structures of sparse, bipartite graphs that may have directed and/or weighted edges; 3) is flexible to incorporate explicit side features about nodes as an optional component to improve predictive performance; and 4) utilize extensive parameter sharing to reduce memory footprint and computational complexity, while leveraging available GPU-based implementations for increased scalability. Further, the autoencoder architecture has the novelty of being efficiently trained end-to-end for the joint, multi-task learning (MTL) of both link prediction and node classification tasks. To the

978-1-5386-5090-5/18/$31.00 ©2018 IEEE
DOI 10.1109/DSAA.2018.00034

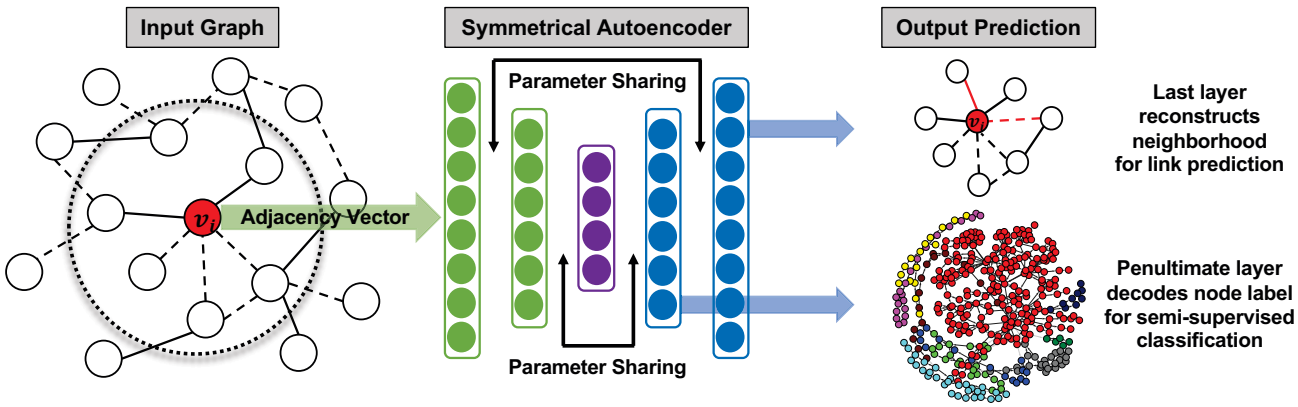

Fig. 1. Schematic depiction of the Local Neighborhood Graph Autoencoder (LoNGAE) architecture. *Left*: A partially observed input graph with known positive links (solid lines) and known negative links (dashed lines) between two nodes; pairs of nodes not yet connected have unknown status links. *Middle*: A symmetrical, densely connected autoencoder with parameter sharing (tied weights) is trained end-to-end to learn node embeddings from the adjacency vector for graph representation. *Right*: Exemplar multi-task output for link prediction and node classification.

best of our knowledge, this is the first architecture capable of performing simultaneous link prediction and node classification in a single learning stage, whereas previous related methods require multiple training stages that are difficult to optimize. Lastly, we conduct a comprehensive evaluation of the proposed autoencoder architecture on nine challenging benchmark graph-structured datasets comprising a wide range of LPNC applications. Numerical experiments validate the efficacy of our models by showing significant improvement on multiple evaluation measures over related methods designed for link prediction and/or node classification.

II. Autoencoder Architecture for Link Prediction and Node Classification

We now characterize our proposed autoencoder architecture, schematically depicted in Figure 1, for LPNC and formalize the notation used in this paper. The input to the autoencoder is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of $N = |\mathcal{V}|$ nodes. Graph $\mathcal{G}$ is represented by its adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. For a partially observed graph, $\mathbf{A} \in \{1, 0, \text{UNK}\}^{N \times N}$, where 1 denotes a known present positive edge, 0 denotes a known absent negative edge, and UNK denotes an unknown status (missing or unobserved) edge. In general, the input to the autoencoder can be directed or undirected, weighted or unweighted, and/or bipartite graphs. However, for the remainder of this paper and throughout the numerical experiments, we assume undirected and symmetric graphs with binary edges to maintain parity with previous related work.

Optionally, we are given a matrix of available explicit node features, i.e. side information $\mathbf{X} \in \mathbb{R}^{N \times F}$. The aim of the autoencoder model $h(\mathbf{A}, \mathbf{X})$ is to learn a set of low-dimensional latent variables for the nodes $\mathbf{Z} \in \mathbb{R}^{N \times D}$ that can produce an approximate reconstruction output $\hat{\mathbf{A}}$ such that the error between $\mathbf{A}$ and $\hat{\mathbf{A}}$ is minimized, thereby preserving the global graph structure. In this paper, we use capital variables (e.g., $\mathbf{A}$) to denote matrices and lower-case variables (e.g., $\mathbf{a}$) to denote row vectors. For example, we use $\mathbf{a}_i$ to mean the ith row of the matrix $\mathbf{A}$.

A. Link Prediction

Research on link prediction attempts to answer the principal question: given two entities, should there be a connection between them? We focus on the structural link prediction problem, where the task is to compute the likelihood that an unobserved or missing edge exists between two nodes in a partially observed graph. For a comprehensive survey on link prediction, to include *structural* and *temporal* link prediction using unsupervised and supervised models, see [33].

Link Prediction from Graph Topology Let $\mathbf{a}_i \in \mathbb{R}^N$ be an *adjacency vector* of $\mathbf{A}$ that contains the local neighborhood of the ith node. Our proposed autoencoder architecture comprises a set of non-linear transformations on $\mathbf{a}_i$ summarized in two component parts: encoder $g(\mathbf{a}_i)\colon \mathbb{R}^N \to \mathbb{R}^D$, and decoder $f(\mathbf{z}_i)\colon \mathbb{R}^D \to \mathbb{R}^N$. We stack two layers of the encoder part to derive D-dimensional latent feature representation of the ith node $\mathbf{z}_i \in \mathbb{R}^D$, and then stack two layers of the decoder part to obtain an approximate reconstruction output $\hat{\mathbf{a}}_i \in \mathbb{R}^N$, resulting in a four-layer autoencoder architecture. Note that $\mathbf{a}_i$ is highly sparse, with up to 90 percent of the edges missing at random in some of our experiments, and the dense reconstructed output $\hat{\mathbf{a}}_i$ contains the predictions for the missing edges. The hidden representations for the encoder and decoder parts are computed as follows:

$$\mathbf{z}_i = g(\mathbf{a}_i) = \text{ReLU}\left(\mathbf{W} \cdot \text{ReLU}\left(\mathbf{V}\mathbf{a}_i + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}\right) \quad \text{Encoder Part,}$$

$$\hat{\mathbf{a}}_i = f(\mathbf{z}_i) = \mathbf{V}^{\mathrm{T}} \cdot \text{ReLU}\left(\mathbf{W}^{\mathrm{T}}\mathbf{z}_i + \mathbf{b}^{(3)}\right) + \mathbf{b}^{(4)} \quad \text{Decoder Part,}$$

$$\hat{\mathbf{a}}_i = h(\mathbf{a}_i) = f(g(\mathbf{a}_i)) \quad \text{Autoencoder.}$$

The choice of non-linear, element-wise activation function is the rectified linear unit $\text{ReLU}(\mathbf{x}) = \max(0, \mathbf{x})$ [21]. The last decoder layer computes a linear transformation to score the missing links as part of the reconstruction. We constrain the autoencoder to be symmetrical with shared parameters for $\{\mathbf{W}, \mathbf{V}\}$ between the encoder and decoder parts, resulting in almost $2\times$ fewer parameters than an unconstrained architecture. Parameter sharing is a powerful form of regularization that helps improve learning and generalization, and is also the main motivation for MTL, first explored in [2] and most recently in [37]. Notice the bias units $\mathbf{b}$ do not share parameters, and $\{\mathbf{W}^{\mathrm{T}}, \mathbf{V}^{\mathrm{T}}\}$ are transposed copies of $\{\mathbf{W}, \mathbf{V}\}$. For brevity of notation, we summarize the parameters to be learned in $\theta = \{\mathbf{W}, \mathbf{V}, \mathbf{b}^{(k)}\}, k = 1, ..., 4$. Since our autoencoder learns node embeddings from local neighborhood structures of the graph, we refer to it as LoNGAE for Local Neighborhood Graph Autoencoder.

Link Prediction with Node Features Optionally, if a matrix of explicit node features $\mathbf{X} \in \mathbb{R}^{N \times F}$ is available, then we concatenate $(\mathbf{A}, \mathbf{X})$ to obtain an *augmented* adjacency matrix $\bar{\mathbf{A}} \in \mathbb{R}^{N \times (N+F)}$ and perform the above encoder-decoder transformations on $\bar{\mathbf{a}}_i$ for link prediction. We refer to this variant as αLoNGAE. Notice the augmented adjacency matrix is no longer square and symmetric. The intuition behind the concatenation of node features is to enable a shared representation of both graph and node features throughout the autoencoding transformations by way of the tied parameters $\{\mathbf{W}, \mathbf{V}\}$. This idea draws inspiration from recent work by Vukotić et al. [32], where they successfully applied symmetrical autoencoders with parameter sharing for multi-modal and cross-modal representation learning of textual and visual features.

The training complexity of αLoNGAE is $\mathcal{O}((N + F)DI)$, where N is the number of nodes, F is the dimensionality of node features, D is the size of the hidden layer, and I is the number of iterations. In practice, F, $D \ll N$, and I are independent of N. Thus, the overall complexity of the autoencoder is $\mathcal{O}(N)$, linear in the number of nodes.

Inference and Learning During the forward pass, or inference, the model takes as input an adjacency vector $\mathbf{a}_i$ and computes its reconstructed output $\hat{\mathbf{a}}_i = h(\mathbf{a}_i)$ for link prediction. The parameters θ are learned via backpropagation. During the backward pass, we estimate θ by minimizing the Masked Balanced Cross-Entropy (MBCE) loss, which only allows for the contributions of those parameters associated with observed edges, as in [24]. Moreover, $\mathbf{a}_i$ can exhibit extreme class imbalance between known present and absent links, as is common in link prediction problems. We handle class imbalance by defining a weighting factor $\zeta \in [0, 1]$ to be used as a multiplier for the positive class in the cross-entropy loss formulation. This approach is referred to as balanced cross-entropy. Other approaches to class imbalance include optimizing for a ranking loss [20] and the recent work on focal loss by Lin et al. [18]. For a single example $\mathbf{a}_i$ and its reconstructed output $\hat{\mathbf{a}}_i$, we compute the MBCE loss as follows:

$$\mathcal{L}_{\text{BCE}} = -\mathbf{a}_i \log\left(\sigma\left(\hat{\mathbf{a}}_i\right)\right) \cdot \zeta - \left(1 - \mathbf{a}_i\right) \log\left(1 - \sigma\left(\hat{\mathbf{a}}_i\right)\right),$$

$$\mathcal{L}_{\text{MBCE}} = \frac{\mathbf{m}_i \odot \mathcal{L}_{\text{BCE}}}{\sum \mathbf{m}_i}.$$

Here, $\mathcal{L}_{\text{BCE}}$ is the balanced cross-entropy loss with weighting factor $\zeta = 1 - \frac{\text{\# present links}}{\text{\# absent links}}$, $\sigma(\cdot)$ is the sigmoid function, $\odot$ is the Hadamard (element-wise) product, and $\mathbf{m}_i$ is the boolean function: $\mathbf{m}_i = 1$ if $\mathbf{a}_i \neq \text{UNK}$, else $\mathbf{m}_i = 0$.

The same autoencoder architecture can be applied to a row vector $\bar{\mathbf{a}}_i \in \mathbb{R}^{N+F}$ in the augmented adjacency matrix $\bar{\mathbf{A}}$. However, at the final decoder layer, we slice the reconstruction $h(\bar{\mathbf{a}}_i)$ into two outputs: $\hat{\mathbf{a}}_i \in \mathbb{R}^N$ corresponding to the reconstructed example in the original adjacency matrix, and $\hat{\mathbf{x}}_i \in \mathbb{R}^F$ corresponding to the reconstructed example in the matrix of node features. During learning, we optimize θ on the concatenation of graph topology and side node features $(\mathbf{a}_i, \mathbf{x}_i)$, but compute the losses for the reconstructed outputs $(\hat{\mathbf{a}}_i, \hat{\mathbf{x}}_i)$ separately with different loss functions. The motivation behind this design is to maintain flexibility to handle different input formats; the input $\mathbf{a}_i$ is usually binary, but the input $\mathbf{x}_i$ can be binary, real-valued, or both. In this work, we enforce both inputs $(\mathbf{a}_i, \mathbf{x}_i)$ to be in the range $[0, 1]$ for simplicity and improved performance, and compute the augmented αMBCE loss as follows:

$$\mathcal{L}_{\alpha\text{MBCE}} = \mathcal{L}_{\text{MBCE}(\mathbf{a}_i, \hat{\mathbf{a}}_i)} + \mathcal{L}_{\text{CE}(\mathbf{x}_i, \hat{\mathbf{x}}_i)},$$

where $\mathcal{L}_{\text{CE}(\mathbf{x}_i, \hat{\mathbf{x}}_i)} = -\mathbf{x}_i \log\left(\sigma\left(\hat{\mathbf{x}}_i\right)\right) - \left(1 - \mathbf{x}_i\right) \log\left(1 - \sigma\left(\hat{\mathbf{x}}_i\right)\right)$ is the standard cross-entropy loss with sigmoid function $\sigma(\cdot)$. At inference time, we use the reconstructed output $\hat{\mathbf{a}}_i$ for link prediction and disregard the output $\hat{\mathbf{x}}_i$.

B. Semi-Supervised Node Classification

The αLoNGAE model can also be used to perform efficient information propagation on graphs for the task of semi-supervised node classification. Node classification is the task of predicting the labels or types of entities in a graph, such as the types of molecules in a metabolic network or document categories in a citation network.

For a given augmented adjacency vector $\bar{\mathbf{a}}_i$, the autoencoder learns the corresponding node embeddings $\mathbf{z}_i$ to obtain an optimal reconstruction. Intuitively, $\mathbf{z}_i$ encodes a vector of latent features derived from the concatenation of both graph and node features, and can be used to predict the label of the ith node. For multi-class classification, we can decode $\mathbf{z}_i$ using the softmax activation function to learn a probability distribution over node labels. More precisely, we predict node labels via the following transformation: $\hat{\mathbf{y}}_i = \text{softmax}(\tilde{\mathbf{z}}_i) = \frac{1}{\mathcal{Z}} \exp(\tilde{\mathbf{z}}_i)$, where $\mathcal{Z} = \sum \exp(\tilde{\mathbf{z}}_i)$ and $\tilde{\mathbf{z}}_i = \mathbf{U} \cdot \text{ReLU}\left(\mathbf{W}^{\mathrm{T}} \mathbf{z}_i + \mathbf{b}^{(3)}\right) + \mathbf{b}^{(5)}$.

In many applications, only a small fraction of the nodes are labeled. For semi-supervised learning, it is advantageous to utilize unlabeled examples in conjunction with labeled instances to better capture the underlying data patterns for improved learning and generalization. We achieve this by jointly

training the autoencoder with a masked softmax classifier to collectively learn node labels from minimizing their combined losses:

$$\mathcal{L}_{\text{MULTI-TASK}} = -\text{MASK}_i \sum_{c \in C} \mathbf{y}_{ic} \log(\hat{\mathbf{y}}_{ic}) + \mathcal{L}_{\text{MBCE}},$$

where C is the set of node labels, $\mathbf{y}_{ic} = 1$ if node i belongs to class c, $\hat{\mathbf{y}}_{ic}$ is the softmax probability that node i belongs to class c, $\mathcal{L}_{\text{MBCE}}$ is the loss defined for the autoencoder, and the boolean function $\text{MASK}_i = 1$ if node i has a label, otherwise $\text{MASK}_i = 0$. Notice in this configuration, we can perform multi-task learning for both link prediction and semi-supervised node classification, *simultaneously*.

III. Related Work

The field of graph representation learning is seeing a resurgence of research interest in recent years, driven in part by the latest advances in deep learning. The aim is to learn a mapping that encodes the input graph into low-dimensional feature embeddings while preserving its original global structure. Hamilton et al. [9] succinctly articulate the diverse set of previously proposed approaches for graph representation learning, or graph embedding, as belonging within a unified encoder-decoder framework. In this section, we summarize three classes of encoder-decoder models most related to our work: matrix factorization (MF), autoencoders, and graph convolutional networks (GCNs).

MF has its roots in dimensionality reduction and gained popularity with extensive applications in collaborative filtering (CF) and recommender systems [16]. MF models take an input matrix $\mathbf{M}$, learn a shared linear latent representation for rows ($\mathbf{r}_i$) and columns ($\mathbf{c}_j$) during an encoder step, and then use a bilinear (pairwise) decoder based on the inner product $\mathbf{r}_i \mathbf{c}_j$ to produce a reconstructed matrix $\hat{\mathbf{M}}$. CF is mathematically similar to link prediction, where the goal is essentially matrix completion. Menon and Elkan [20] proposed an MF model capable of incorporating side information about nodes and/or edges to demonstrate strong link prediction results on several challenging network datasets. Other recent approaches similar to MF that learn node embeddings via some encoder transformation and then use a bilinear decoder for the reconstruction include DeepWalk [22] and its variants LINE [30] and node2vec [8]. DeepWalk, LINE, and node2vec do not support external node/edge features.

Our work is inspired by recent successful applications of autoencoder architectures for collaborative filtering that outperform popular matrix factorization methods [24], [28], [17], and is related to Structural Deep Network Embedding (SDNE) [34] for link prediction. Similar to SDNE, our models rely on the autoencoder to learn non-linear node embeddings from local graph neighborhoods. However, our models have several important distinctions: 1) we leverage extensive parameter sharing between the encoder and decoder parts to enhance representation learning; 2) our αLoNGAE model can optionally concatenate side node features to the adjacency matrix for improved link prediction performance; and 3) the αLoNGAE model can be trained end-to-end in a single stage for multi-task learning of link prediction and semi-supervised node classification. On the other hand, training SDNE requires multiple steps that are difficult to jointly optimize: i) pre-training via a deep belief network; and ii) utilizing a separate downstream classifier on top of node embeddings for LPNC.

Lastly, GCNs [14] are a recent class of algorithms based on convolutional encoders for learning node embeddings. The GCN model is motivated by a localized first-order approximation of spectral convolutions for layer-wise information propagation on graphs. Similar to our αLoNGAE model, the GCN model can learn hidden layer representations that encode both local graph structure and features of nodes. The choice of the decoder depends on the task. For link prediction, the bilinear inner product is used in the context of the variational graph autoencoder (VGAE) [15]. For semi-supervised node classification, the softmax activation function is employed. The GCN model provides an end-to-end learning framework that scales linearly in the number of graph edges and has been shown to achieve strong LPNC results on a number of graph-structured datasets. However, the GCN model has a drawback of being memory intensive because it is trained on the full dataset using batch gradient descent for every training iteration. We show that our models outperform GCN-based models for LPNC while consuming a constant memory budget by way of mini-batch training.

IV. Experimental Design

In this section, we expound our protocol for the empirical evaluation of our models' capability for learning and generalization on the tasks of link prediction and semi-supervised node classification. Secondarily, we also present results of the models' representation capacity on the task of network reconstruction.

A. Datasets and Baselines

We evaluate our proposed autoencoder models on nine graph-structured datasets, spanning multiple application domains, from which previous graph embedding methods have achieved strong results for LPNC. The datasets are summarized in Table I and include networks for `Protein` interactions, `Metabolic` pathways, military `Conflict` between countries, the U.S. `PowerGrid`, collaboration between users on the `BlogCatalog` social website, and publication citations from the `Cora`, `Citeseer`, `Pubmed`, `Arxiv-GRQC` databases. {`Protein`, `Metabolic`, `Conflict`, `PowerGrid`} are reported in [20]. {`Cora`, `Citeseer`, `Pubmed`} are from [25] and reported in [14], [15]. And {`Arxiv-GRQC`, `BlogCatalog`} are reported in [34].

We empirically compare our autoencoder models against four strong baselines summarized in Table II, which were designed specifically for link prediction and/or node classification. We begin our empirical evaluation with the SDNE [34] baseline, where we compare the representation capacity of our models on the network reconstruction task using the

TABLE I
SUMMARY STATISTICS OF DATASETS USED IN EMPIRICAL EVALUATION. THE NOTATION $|O^+|:|O^-|$ DENOTES THE RATIO OF OBSERVED PRESENT (POSITIVE) EDGES TO ABSENT (NEGATIVE) EDGES AND IS A MEASURE OF CLASS IMBALANCE. LABEL RATE IS DEFINED AS THE NUMBER OF NODES LABELED FOR TRAINING DIVIDED BY THE TOTAL NUMBER OF NODES.

Dataset	Nodes	Average Degree	$\lvert O^+\rvert:\lvert O^-\rvert$ Ratio	Node Features	Node Classes	Label Rate
Pubmed	19,717	4.5	1 : 4384	500	3	0.003
Citeseer	3,327	2.8	1 : 1198	3,703	6	0.036
Cora	2,708	3.9	1 : 694	1,433	7	0.052
Protein	2,617	9.1	1 : 300	76	–	–
Metabolic	668	8.3	1 : 80	325	–	–
Conflict	130	2.5	1 : 52	3	–	–
PowerGrid	4,941	2.7	1 : 1850	–	–	–
Arxiv-GRQC	5,242	5.5	1 : 947	–	–	–
BlogCatalog	10,312	64.8	1 : 158	–	–	–

TABLE II
SUMMARY OF BASELINES USED IN EMPIRICAL EVALUATION. ACRONYMS: AUC – AREA UNDER ROC CURVE; AP – AVERAGE PRECISION.

Baseline	Task	Metric
SDNE [34]	Reconstruction	Precision@k
MF [20]	Link Prediction	AUC
VGAE [15]	Link Prediction	AUC, AP
GCN [14]	Node Classification	Accuracy

`Arxiv-GRQC` and `BlogCatalog` datasets. For the MF baseline, we closely follow the experimental protocol in [20], where we randomly sample 10 percent of the observed links for training and evaluate link prediction performance on the other disjoint 90 percent for the {`Protein, Metabolic, Conflict`} datasets. For `PowerGrid`, we use 90 percent of observed links for training and evaluate on the remaining 10 percent. And for the VGAE and GCN baselines, we use the same train/validation/test segments described in [15] and [14] for link prediction and node classification, respectively, on the {`Cora, Citeseer, Pubmed`} citation networks.

B. Implementation Details

We implement the autoencoder architecture using Keras [4] on top of the GPU-enabled TensorFlow [1] backend, along with several additional details. The diagonal elements of the adjacency matrix are set to 1 with the interpretation that every node is connected to itself. We impute missing or UNK elements in the adjacency matrix with 0. Note that imputed edges are not observed elements in the adjacency matrix and hence do not contribute to the masked loss computations during training. We are free to impute any values for the missing edges, but through cross-validation we found that the uniform value of 0 produces the best results.

Hyper-parameter tuning is performed via cross-validation or on the available validation set. Key hyper-parameters include mini-batch size, dimensionality of the hidden layers, and the percentage of dropout regularization. In general, we strive to keep a similar set of hyper-parameters across datasets to highlight the consistency of our models. In all experiments, the dimensionality of the hidden layers in the autoencoder architecture is fixed at N-256-128-256-N. For reconstruction and link prediction, we train for 50 epochs using mini-batch size of 8 samples. For node classification, we train for 100 epochs using mini-batch size of 64 samples. We utilize early stopping as a form of regularization in time when the model shows signs of overfitting on the validation set.

We apply mean-variance normalization (MVN) after each ReLU activation layer to help improve link prediction performance, where it compensates for noise between train and test instances by normalizing the activations to have zero mean and unit variance. MVN enables efficient learning and has been shown effective in cardiac semantic segmentation [31] and speech recognition [12].

During training, we apply dropout regularization [27] throughout the architecture to mitigate overfitting, depending on the sparsity of the input graph. For link prediction, dropout is also applied at the input layer to produce an effect similar to using a *denoising* autoencoder. This denoising technique was previously employed for link prediction in [3]. We initialize weights according to the Xavier scheme described in [7]. We do not apply weight decay regularization.

We employ the Adam algorithm [13] for gradient descent optimization with a fixed learning rate of 0.001. As part of our experimental design, we also performed experiments without parameter sharing between the encoder and decoder parts of the architecture and found severely degraded predictive performance. This observation is consistent with prior findings that parameter sharing helps improve generalization by providing additional regularization to mitigate the adverse effects of overfitting and enhance representation learning [32], [37].

C. Results and Analysis

Reconstruction Results of the reconstruction task for the `Arxiv-GRQC` and `BlogCatalog` network datasets are illustrated in Figure 2. In this experiment, we compare the results obtained by our LoNGAE model to those obtained by the related autoencoder-based SDNE model [34]. The evaluation metric is precision@k, which is a rank-based measure used in information retrieval and is defined as the proportion of retrieved edges/links in the top-k set that are relevant. We use precision@k to evaluate the model's ability to retrieve edges known to be present (positive edges) as part of the reconstruction.

In comparison to SDNE, we show that our LoNGAE model achieves better precision@k performance for all k values, up to $k = 10,000$ for `Arxiv-GRQC` and $k = 100,000$ for `BlogCatalog`, when trained on the complete datasets. We also systematically test the capacity of the LoNGAE model to reconstruct the original networks when up to 80 percent of the edges are randomly removed, akin to the link prediction task. We show that the LoNGAE model only gets worse precision@k performance than SDNE on the `Arxiv-GRQC` dataset when more than 40 percent of the edges are missing at random. On the `BlogCatalog` dataset, the LoNGAE

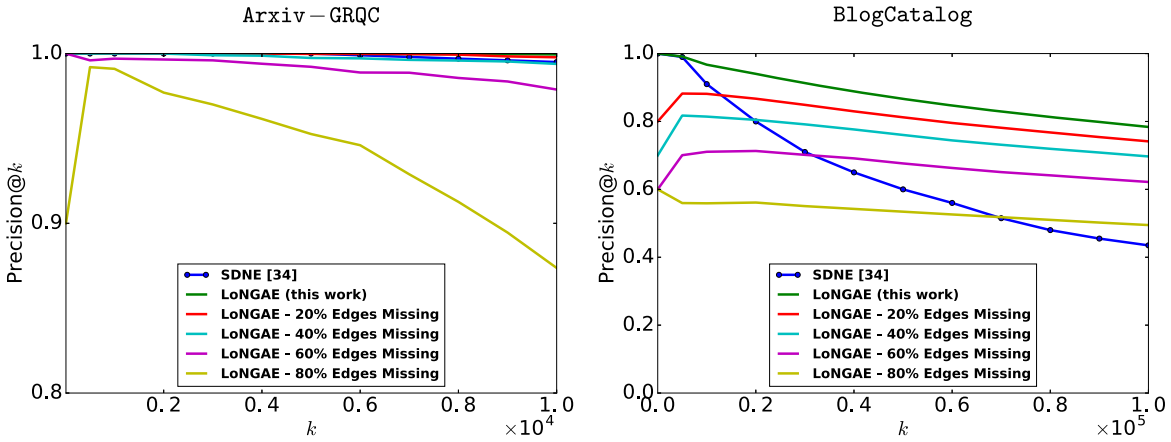

Fig. 2. Comparison of precision@k performance between our LoNGAE model and the related autoencoder-based SDNE model for the reconstruction task on the `Arxiv-GRQC` and `BlogCatalog` network datasets. The parameter k indicates the total number of retrieved edges.

TABLE III
COMPARISON OF AUC PERFORMANCE BETWEEN OUR AUTOENCODER MODELS AND THE BEST PREVIOUS MATRIX FACTORIZATION MODEL FOR LINK PREDICTION. NUMBER FORMAT: MEAN VALUE (STANDARD DEVIATION). *THESE RESULTS INCORPORATED ADDITIONAL *edge features* FOR LINK PREDICTION, WHICH WE LEAVE FOR FUTURE WORK.

Method	Node Features	Protein	Metabolic	Conflict	PowerGrid
LoNGAE (this work)	No	**0.798 (0.004)**	**0.703 (0.009)**	**0.698 (0.025)**	**0.781 (0.007)**
Matrix Factorization [20]	No	0.795 (0.005)	0.696 (0.001)	0.692 (0.040)	0.754 (0.014)
αLoNGAE (this work)	Yes	**0.861 (0.003)**	0.750 (0.011)	0.699 (0.021)	–
Matrix Factorization [20]	Yes	0.813 (0.002)	***0.763 (0.006)**	***0.890 (0.017)**	–

model achieves better precision@k performance than SDNE for large k values even when 80 percent of the edges are missing at random. This experiment demonstrates the superior representation capacity of our LoNGAE model compared to SDNE.

Link Prediction Table III shows the comparison between our autoencoder models and the matrix factorization (MF) model proposed in [20] for link prediction with and without node features. Recall that our goal is to recover the statuses of the missing or unknown links in the input graph. As part of the experimental design, we pretend that a randomly selected set of elements in the adjacency matrix are missing and collect their indices to be used as a validation set. Our task is to train the autoencoder to produce a set of predictions, a list of ones and zeros, on those missing indices and see how well the model performs when compared to the ground-truth. The evaluation metric is the area under the ROC curve (AUC). Results are reported as mean AUC and standard deviation over 10-fold cross-validation. The datasets under consideration for link prediction exhibit varying degrees of class imbalance.

For featureless link prediction, our LoNGAE model marginally outperforms MF on {`Protein`, `Metabolic`, `Conflict`} and is significantly better than MF on `PowerGrid`. Consistent with MF results, we observe that incorporating external node features provides a boost in link prediction accuracy, especially for the `Protein` dataset where we achieve a 6 percent increase in performance. `Metabolic` and `Conflict` also come with external *edge features*, which were exploited by the MF model for further performance gains. We leave the task of combining edge features for future work. Each node in `Conflict` only has three features, which are unable to significantly boost link prediction accuracy. `PowerGrid` does not have node features so there are no results for the respective rows.

Table IV summarizes the performances between our autoencoder models and related graph embedding methods for link prediction with and without node features. Following the protocol described in [15], we report AUC and average precision (AP) scores for each model on the held-out test set containing 10 percent of randomly sampled positive links and the same number of negative links. We show mean AUC and AP with standard error over 10 runs with random weight initializations on fixed data splits. Results for the baseline methods are taken from Kipf and Welling [15], where we pick the best performing models for comparison. Similar to the MF model, the graph embedding methods that can combine side node features *always* produce a boost in link prediction accuracy. In this comparison, we significantly outperform the best graph embedding methods by as much as 10 percent, with and without node features. Our αLoNGAE model achieves competitive link prediction performance when compared against the best model presented in [15] on the `Pubmed` dataset.

TABLE IV
COMPARISON OF AUC AND AP PERFORMANCES BETWEEN OUR AUTOENCODER MODELS AND RELATED GRAPH EMBEDDING METHODS FOR LINK PREDICTION. NUMBER FORMAT: MEAN VALUE (STANDARD DEVIATION). §DENOTES THE BEST PERFORMING MODEL PRESENTED IN [15].

Method	Node Features	Cora		Citeseer		Pubmed	
		AUC	AP	AUC	AP	AUC	AP
LoNGAE (this work)	No	**0.896 (0.003)**	**0.915 (0.001)**	**0.860 (0.003)**	**0.892 (0.003)**	**0.926 (0.001)**	**0.930 (0.002)**
Spectral Clustering [29]	No	0.846 (0.01)	0.885 (0.00)	0.805 (0.01)	0.850 (0.01)	0.842 (0.01)	0.878 (0.01)
DeepWalk [22]	No	0.831 (0.01)	0.850 (0.00)	0.805 (0.02)	0.836 (0.01)	0.844 (0.00)	0.841 (0.00)
VGAE§ [15]	No	0.843 (0.02)	0.881 (0.01)	0.789 (0.03)	0.841 (0.02)	0.827 (0.01)	0.875 (0.01)
αLoNGAE (this work)	Yes	**0.943 (0.003)**	**0.952 (0.002)**	**0.956 (0.003)**	**0.964 (0.002)**	0.960 (0.003)	0.963 (0.002)
VGAE§ [15]	Yes	0.914 (0.01)	0.926 (0.01)	0.908 (0.02)	0.920 (0.02)	**0.964 (0.00)**	**0.965 (0.00)**

Node Classification Results of semi-supervised node classification for the {`Cora`, `Citeseer`, `Pubmed`} datasets are summarized in Table V. In this context of citation networks, node classification is equivalent to the task of document classification. We closely follow the experimental setup of Kipf and Welling [14], where we use their provided train/validation/test splits for evaluation. Accuracy performance is measured on the held-out test set of 1,000 examples. We tune hyper-parameters on the validation set of 500 examples. The train set only contains 20 examples per class. All methods use the complete adjacency matrix, and available node features, to learn latent embeddings for node classification. For comparison, we train and test our αLoNGAE model on the same data splits over 10 runs with random weight initializations and report mean accuracy. Kipf and Welling [14] report their mean GCN and ICA results on the same data splits over 100 runs with random weight initializations. The other baseline methods are taken from Yang et al. [36]. In this comparison, our αLoNGAE model achieves competitive performance when compared against the GCN model on the `Cora` dataset, but outperforms GCN and all other baseline methods on the `Citeseer` and `Pubmed` datasets.

TABLE V
COMPARISON OF ACCURACY PERFORMANCE BETWEEN OUR αLONGAE MODEL AND RELATED GRAPH EMBEDDING METHODS FOR SEMI-SUPERVISED NODE CLASSIFICATION.

Method	Cora	Citeseer	Pubmed
αLoNGAE (this work)	0.783	**0.716**	**0.794**
GCN [14]	**0.815**	0.703	0.790
Planetoid [36]	0.757	0.647	0.772
ICA [19]	0.751	0.691	0.739
DeepWalk [22]	0.672	0.432	0.653

Multi-task Learning Lastly, we report LPNC results obtained by our αLoNGAE model in the MTL setting over 10 runs with random weight initializations. In the MTL scenario, the αLoNGAE model takes as input an incomplete graph with 10 percent of the positive edges, and the same number of negative edges, missing at random and all available node features to simultaneously produce predictions for the missing edges and labels for the nodes. Table VI shows the efficacy of the αLoNGAE model for MTL when compared against the best performing task-specific link prediction and node classification models, which require the complete adjacency matrix as input. For link prediction, multi-task αLoNGAE achieves competitive performance against task-specific αLoNGAE, and significantly outperforms the best VGAE model from Kipf and Welling [15] on `Cora` and `Citeseer` datasets. For node classification, multi-task αLoNGAE is the best performing model across the board, only trailing behind the GCN model on the `Cora` dataset.

TABLE VI
COMPARISON OF LINK PREDICTION AND NODE CLASSIFICATION PERFORMANCES OBTAINED BY THE αLONGAE MODEL IN THE MULTI-TASK LEARNING SETTING. LINK PREDICTION PERFORMANCE IS REPORTED AS THE COMBINED AVERAGE OF AUC AND AP SCORES. ACCURACY IS USED FOR NODE CLASSIFICATION PERFORMANCE.

Method	Cora	Citeseer	Pubmed
Link Prediction			
Multi-task αLoNGAE	0.946	0.949	0.944
Task-specific αLoNGAE	**0.948**	**0.960**	0.962
Task-specific VGAE [15]	0.920	0.914	**0.965**
Node Classification			
Multi-task αLoNGAE	0.790	**0.718**	**0.804**
Task-specific αLoNGAE	0.783	0.716	0.794
Task-specific GCN [14]	**0.815**	0.703	0.790

V. DISCUSSION

In our experiments, we show that a simple autoencoder architecture with parameter sharing consistently outperforms previous related methods on a range of challenging graph-structured benchmarks for three separate tasks: reconstruction, link prediction, and semi-supervised node classification. For the reconstruction task, our LoNGAE model achieves superior precision@k performance when compared to the related SDNE model. Although both models leverage a deep autoencoder architecture for graph representation learning, the SDNE model lacks several key implementations necessary for enhanced representation capacity, namely parameter sharing between the encoder-decoder parts and end-to-end training of deep architectures.

For link prediction, we observe that combining available node features *always* produces a significant boost in predictive performance. This observation was previously reported in [20],

[15], among others. Intuitively, we expect topological graph features provide complementary information not present in the node features, and the combination of both feature sets should improve predictive power. Although explicit node features may not always be readily available, a link prediction model capable of incorporating optional side information has broader applicability.

Our αLoNGAE model also performs favorably well on the task of semi-supervised node classification. The model is capable of encoding non-linear node embeddings from both local graph structure and explicit node features, which can be decoded by a softmax activation function to yield accurate node labels. The efficacy of the proposed αLoNGAE model is evident especially on the `Pubmed` dataset, where the label rate is only 0.003. This efficacy is attributed to parameter sharing being used in the autoencoder architecture, which provides regularization to help improve representation learning and generalization.

Our autoencoder architecture naturally supports multi-task learning, where a joint representation for both link prediction and node classification is enabled via parameter sharing. MTL aims to exploit commonalities and differences across multiple tasks to find a shared representation that can result in improved performance for each task-specific metric. In this work, we show that our multi-task αLoNGAE model improves node classification accuracy by learning to predict missing edges at the same time. Our multi-task model has broad practical utility to address real-world applications where the input graphs may have both missing edges and node labels.

Finally, we address one major limitation associated with our autoencoder models having complexity scale linearly in the number of nodes. Hamilton et al. [9] express that the complexity in nodes may limit the utility of the models on massive graphs with hundreds of millions of nodes. In practice, we would implement our models to leverage data parallelism [26] across commodity CPU and/or GPU resources for effective distributed learning on massive graphs. Data parallelism is possible because our models learn node embeddings from each row vector of the adjacency matrix independently. Nevertheless, the area of improvement in future work is to take advantage of the sparsity of edges in the graphs to scale our models linearly in the number of observed edges.

VI. Conclusion

In this work, we presented a new autoencoder architecture for link prediction and semi-supervised node classification, and showed that the resulting models outperform related methods in accuracy performance on a range of real-world graph-structured datasets. The success of our models is primarily attributed to extensive parameter sharing between the encoder and decoder parts of the architecture, coupled with the capability to learn expressive non-linear latent node representations from both local graph neighborhoods and explicit node features. Further, our novel architecture is capable of simultaneous multi-task learning of both link prediction and node classification in one efficient end-to-end training stage. Our work provides a useful framework to make accurate and meaningful predictions on a diverse set of complex graph structures for a wide range of real-world applications.

Acknowledgment

The author thanks Edward Raff and Jared Sylvester for insightful discussions, and gracious reviewers for constructive feedback on the paper.

References

[1] Abadi, M. et al.: TensorFlow: Large-scale machine learning on heterogeneous systems. https://github.com/tensorflow/tensorflow (2015)
[2] Caruana, R.: Multitask learning: a knowledge-based source of inductive bias. In: Proceedings of the Tenth International Conference on Machine Learning (1993)
[3] Chen, Z., Zhang, W.: A marginalized denoising method for link prediction in relational data. In: SIAM International Conference on Data Mining (2014)
[4] Chollet, F. et al.: Keras. https://github.com/keras-team/keras (2015)
[5] Fakhraei, S., Huang, B., Raschid, L., Getoor, L.: Network-based drug-target interaction prediction with Probabilistic Soft Logic. IEEE/ACM Transactions on Computational Biology and Bioinformatics 1, 775–787 (2014)
[6] Fakhraei, S., Foulds, J., Shashanka, M., Getoor, L.: Collective spammer detection in evolving multi-relational social networks. In: ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2015)
[7] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: AISTATS 9 (2010)
[8] Grover, A., Leskovec, J.: node2vec. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2016)
[9] Hamilton, W., Ying, R., Leskovec, J.: Representation learning on graphs: methods and applications. arXiv:1709.05584 (2017)
[10] Hassan, K., Bahareh, B.: A survey on statistical relational learning. In: Proceedings of the 23rd Canadian Conference on Advances in Artificial Intelligence, 256–268 (2010)
[11] Huang, Z., Lin, K.J.: The time-series link prediction problem with applications in communication surveillance. INFORMS Journal on Computing 21, 286–303 (2009)
[12] Joshi, V., Prasad, N.V., Umesh, S.: Modified mean and variance normalization: transforming to utterance-specific estimates. Circuits, Systems, and Signal Processing 35, 1593–1609 (2016)
[13] Kingma, D.P., Ba, J.L.: Adam: a method for stochastic optimization. In: International Conference on Learning Representations (ICLR) (2015)
[14] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv:1609.02907 (2016)
[15] Kipf, T.N., Welling, M.: Variational graph auto-encoders. arXiv:1611.07308 (2016)
[16] Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer 42, 30–37 (2009)
[17] Kuchaiev, O., Ginsburg, B.: Training deep autoencoders for collaborative filtering. arXiv:1708.01715 (2017)
[18] Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. arXiv:1708.02002 (2017)
[19] Lu, Q., Getoor, L.: Link-based classification. In: International Conference on Machine Learning 3, 496-503 (2003)
[20] Menon, A.K., Elkan, C.: Link prediction via matrix factorization. In: Proceedings of the 2011 European Conference on Machine Learning and Knowledge Discovery in Databases 437–452 (2011)
[21] Nair, V., Hinton, G.E.: Rectified linear units improve Restricted Boltzmann Machines. In: Proceedings of the 27th International Conference on Machine Learning (2010)
[22] Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2014)
[23] Rossi, R., McDowell, L., Aha, D., Neville, J.: Transforming graph data for statistical relational learning. Journal of Artificial Intelligence Research 45, 363–441 (2012)
[24] Sedhain, S., Menon, A.K., Sanner, S., Xie, L.: AutoRec: Autoencoders meet collaborative filtering. In: Proceedings of the 24th International Conference on World Wide Web (2015)

[25] Sen, P., Namata, G.M., Bilgic, M., Getoor, L., Gallagher, B., Eliassi-Rad, T.: Collective classification in network data. AI Magazine 29, 93–106 (2008)
[26] Shrivastava, D., Chaudhury, S., Jayadeva, D.: A data and model-parallel, distributed and scalable framework for training of deep networks in Apache Spark. arXiv:1708.05840 (2017)
[27] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 15, 1929–1958 (2014)
[28] Strub, F., Mary, J., Gaudel, R.: Hybrid collaborative filtering with autoencoders. arXiv:1603.00806 (2016)
[29] Tang, L., Liu, H.: Leveraging social media networks for classification. Data Mining and Knowledge Discovery 23, 447–478 (2011)
[30] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: large-scale information network embedding. In: Proceedings of the 24th International Conference on World Wide Web, 1067–1077 (2015)
[31] Tran, P.V.: A fully convolutional neural network for cardiac segmentation in short-axis MRI. arXiv:1604.00494 (2016)
[32] Vukotić, V., Raymond, C., Gravier, G.: Bidirectional joint representation learning with symmetrical deep neural networks for multimodal and crossmodal applications. In: Proceedings of the 2016 ACM International Conference on Multimedia Retrieval (ICMR), 343–346 (2016)
[33] Wang, P., Xu, B., Wu, Y., Zhou, X.: Link prediction in social networks: the state-of-the-art. arXiv:1411.5118 (2014)
[34] Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD) (2016)
[35] Yang, B., Yih, W., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. arXiv:1412.6575 (2015)
[36] Yang, Z., Cohen, W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings. In: International Conference on Machine Learning (ICML) (2016)
[37] Yang, Y., Hospedales, T.M.: Trace norm regularised deep multi-task learning. arXiv:1606.04038 (2017)
[38] Zhao, B., Sen, P., Getoor, L.: Entity and relationship labeling in affiliation networks. In: Proceedings of the 23rd International Conference on Machine Learning (2006)