ARTICLE **Communicated by Olaf Sporns**

Learning the Structural Vocabulary of a Network

Saket Navlakha
navlakha@salk.edu
The Salk Institute for Biological Studies, Integrative Biology Laboratory, La Jolla, CA 92037 U.S.A.

Networks have become instrumental in deciphering how information is processed and transferred within systems in almost every scientific field today. Nearly all network analyses, however, have relied on humans to devise structural features of networks believed to be most discriminative for an application. We present a framework for comparing and classifying networks without human-crafted features using deep learning. After training, autoencoders contain hidden units that encode a robust structural vocabulary for succinctly describing graphs. We use this feature vocabulary to tackle several network mining problems and find improved predictive performance versus many popular features used today. These problems include uncovering growth mechanisms driving the evolution of networks, predicting protein network fragility, and identifying environmental niches for metabolic networks. Deep learning offers a principled approach for mining complex networks and tackling graph-theoretic problems.

1 Introduction

The structural analysis of complex networks has become essential for numerous problems in systems biology, including predicting protein interactions and protein function (Sharan, Ulitsky, & Shamir, 2007), identifying robust and fragile modules (Song & Singh, 2013; Navlakha, He, Faloutsos, & Bar-Joseph, 2014), and understanding links between cellular perturbations and disease outcomes (Barabasi, Gulbahce, & Loscalzo, 2011). The success of these applications has hinged on human reasoning to craft network features to compare and classify graphs. These human-designed features range from local measures, including modularity (Newman, 2006), motifs (Milo et al., 2002), and graphlets (Przulj, 2007), to global measures, such as path lengths, degree distributions (Barabasi & Albert, 1999), and graph spectra (Patro & Kingsford, 2012). While enormously useful, these features have collectively required decades of empirical data analysis to identify and have demanded the careful design of algorithms to extract each individual feature. Furthermore, network science continues to yield important advances, suggesting that important features have yet to be discovered.

Neural Computation **29, 287–312** (2017)
doi:10.1162/NECO_a_00924
© 2017 Massachusetts Institute of Technology

There are several reasons to believe that deep learning (DL) can be used as an unsupervised technique to automatically discover discriminative features of a network. First, by viewing the adjacency matrix of a graph as a two-dimensional image, DL can naturally process any raw input graph without transformation. Second, robust feature detection in graphs requires recognition of topological invariances, which parallels similar requirements in image analysis, where images can be rotated, translated, or scaled in different ways (Goodfellow, Lee, Le, Saxe, & Ng, 2009). Third, DL has recently achieved tremendous success in discovering discriminative features in imaging (Le et al., 2012), speech (Hinton et al., 2012), genomic (Alipanahi, Delong, Weirauch, & Frey, 2015; Zhou & Troyanskaya, 2015), and gene expression data (Tan, Hammond, Hogan, & Greene, 2016; Tan, Ung, Cheng, & Greene, 2015) that have improved on manually designed features (LeCun, Bengio, & Hinton, 2015).

Here, we show how to build simple yet effective autoencoders trained using thousands to millions of random graphs. We then use features derived from hidden unit activity to compress, classify, and compare real-world networks. To demonstrate the utility of this approach, we consider four questions:

1. Can autoencoders more accurately compress and find features in graphs compared to other popular dimensionality-reduction methods?
2. Can DL features forecast the growth mechanisms driving the evolution of a network?
3. Can DL features better predict the effect of node deletion (knockout) on the robustness of molecular interaction networks?
4. Can DL features better predict how challenges from different ecologies sculpt the topology of bacterial metabolic networks?

Our goal here is not to develop new deep learning methods but rather to show how existing techniques can be adapted and applied for network analysis. The improved performance of the deep learning feature vocabulary versus many human-designed features commonly used today for these problems suggests that our approach may be useful across many domains.

1.1 Related Work. Many topological features have been developed to analyze network structure (Chakrabarti & Faloutsos, 2006; Newman, 2010). These features, however, have largely been developed by human curation and visual data analysis, a process that we try to circumvent here. Prior work on graph compression and summarization seeks to find a low-cost representation for an input graph using information-theoretic or reference encoding schemes (Adler & Mitzenmacher, 2001; Navlakha, Rastogi, & Shrivastava, 2008). These works compress nodes with similar neighborhoods

into supernodes connected by superedges, yet these methods do not offer a general technique to find features in graphs beyond approximate bi-cliques. Genetic programming has been used to evolve generative models for graphs but still require a target topological feature to evaluate fitness (Patro et al., 2012), whereas DL is based on reconstructing the data. Graph transformer networks take a segmentation graph as input and find a high-likelihood Viterbi path through the segmentation graph using backpropagation (Bottou, Bengio, & Le Cun, 1997). While successful for many image processing tasks, this algorithm also was not designed to find general structural features in graphs.

Recently, DL was used to cluster graphs by first computing a similarity measure between nodes in the graph and then using deep autoencoders to find a low-dimensional embedding of the similarity matrix (Tian, Gao, Cui, Chen, & Liu, 2014); DL has also been used for learning social representations of vertices (Perozzi, Al-Rfou, & Skiena, 2014) and to predict missing edges in time-evolving graphs using conditional temporal restricted Boltzmann machines (Li et al., 2014). Wavelets have been used to compress signals coming from an underlying graphical structure by using the lifting scheme to sparsely represent an entire class of signals (Rustamov & Guibas, 2013). While the success of deep learning for these problems is impressive, these works do not attempt to learn topological features for comparing and classifying graphs, which is our main focus.

2 A Deep Learning Framework for Analyzing Networks

First, we cast graph compression as an unsupervised learning problem using autoencoders. The autoencoder should learn graph-theoretic features that highlight dominant and discriminative topologies present in the input training graphs. Formally:

Problem 1: **Given** a set of graphs $G_1, G_2, \ldots, G_r$.

$$\textbf{Find } R^* = \operatorname*{argmin}_{R \in \mathcal{R}} \sum_{i=1}^{r} h(G_i, R(G_i)).$$

The representation R is an autoencoder with one or more hidden layers and with features latent within the hidden units. The distance function h computes the overlap between edges in the input graph G_i and edges in the predicted output graph $R(G_i)$ following encoding and decoding. The aim is to find a representation, defined by the autoencoder weights, that can accurately reconstruct the input graphs.

Our second goal is to classify graphs using features learned without supervision. Given a function M that maps each input graph to one of j

class labels, we want to learn a deep classifier Y with j output nodes that can predict the class label of a given graph. Formally:

Problem 2: **Given** a set of graphs $G_1, G_2, \ldots, G_r$.

$$\textbf{Find } Y^* = \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} \sum_{i=1}^{r} \delta(M(G_i), Y(G_i)).$$

The δ function returns 1 if the true label of the graph $M(G_i)$ and the predicted label from the classifier $Y(G_i)$ are the same and 0 otherwise.

For both problems, the r input graphs used for training are generated using random graph models, as described in the next section.

3 Probabilistic Models for Generating Random Graphs

To generate a broad spectrum of input training graphs for learning robust features, we use long-studied probabilistic random graph models (Chakrabarti & Faloutsos, 2006). Building robust autoencoders and classifiers requires large training sets, and unlike image processing applications where millions of training examples are readily available (Le et al., 2012), databases for real-world networks are much smaller. Random graph models help overcome this limitation by providing an iterative growth procedure to generate random graphs with topological features similar to those observed in real-world networks. These features include power-law degree distributions, small diameter, and community structure, among others (Chakrabarti & Faloutsos, 2006). These models have served as the foundation for many downstream graph mining tasks, including anomaly detection, network design and clustering, and growth forecasting (Chakrabarti & Faloutsos, 2006). While many generative models exist, the five we selected are well studied, parsimonious (requiring two or fewer parameters), fast (generating graphs in linear time with respect to the number of edges), and realistic (having topological properties of many real-world networks). Although these models are designed by humans and may contain some biases in the features they generate, by using many different models and model parameters, we can generate a large and diverse training set of graphs sampled from the space of possible graphs (Goldenberg, Zheng, Fienberg, & Airoldi, 2010).

The five models we use are:

1. The duplication (DMC) model inspired by biology (Vazquez, Flammini, & Maritan, Vespignani, 2003)
2. The forest fire model (FF) inspired by social networks (Leskovec, Kleinberg, & Faloutsos, 2007)
3. The preferential attachment (PA) model inspired by the evolving web (Barabasi & Albert, 1999)

4. The small-world (SMW) model inspired by small-world phenomena (Watts & Strogatz, 1998)
5. Erdős-Rényi (ER) random graphs

These models typically start with a small initial seed network and then apply an iterative procedure so the network grows over time. In each iteration, a new node joins the network and connects to some existing nodes (existing edges may also be deleted in this step). This growth process continues until each network has exactly n nodes. The models differ in the mechanisms by which the network evolves and the model parameters they use.

3.1 The Duplication-Mutation with Complementarity Model (DMC). The DMC model is inspired by the duplication-divergence principle in biology, in which a gene duplication event initially produces two topologically equivalent genes, which is followed by some divergence of their shared interactions (Vazquez et al., 2003). This model can reproduce many features of real protein interaction networks (Middendorf, Ziv, & Wiggins, 2005; Navlakha & Kingsford, 2011). The model begins with two nodes connected by an edge and proceeds as follows in each iteration:

1. Node v joins the network by duplicating from a random existing node u. Initially v is connected to each neighbor of u.
2. For each shared neighbor w of u and v, with probability q_{mod}, delete edge (v, w) or its complement (u, w) with equal probability. With probability $1 - q_{\text{mod}}$, retain both edges.
3. Add edge (u, v) with probability q_{con}.

Large q_{mod} and small q_{con} can lead to very sparse networks with little community structure and no power-law degree distribution. Small q_{mod} and large q_{con} can produce dense, highly connected graphs with topological symmetry between duplicate nodes. In between, a wide spectrum of topologies can be generated.

3.2 The Forest Fire Model (FF). The FF model is inspired by the growth of online social networks (Leskovec et al., 2007), in which an existing user u invites a new user v to join the network; node v links to u, as well as some of u's friends, some of their friends, and so on, mimicking the probabilistic spread of a fire. The model begins with two nodes connected by an edge and in each iteration proceeds as follows:

1. Node v joins the network and links to a random existing node u.
2. Node v links to x random neighbors of u, where x is an integer chosen from a geometric distribution with mean $q_{\text{fire}}/(1 - q_{\text{fire}})$. These nodes are added to a queue of nodes to be visited. If u has fewer than x neighbors, all neighbors are linked to.
3. Pop a new (previously unvisited) node u from the queue and recursively apply step 2.

4. Stop when the queue becomes empty.

This model produces strong community structure (modularity) with higher density as q_{fire} increases. It also produces heavy-tailed degree distributions and small diameters (Leskovec et al., 2007).

3.3 The Preferential Attachment Model (PA). The PA model is inspired by the growth of the web and uses a rich-get-richer principle: popular web pages are more likely to be linked to by new web pages, leading to large hubs (Barabasi & Albert, 1999). The model begins with one node connected to q_{pa} other nodes and in each iteration proceeds as follows:

1. Create a probability distribution where each node u is assigned probability $d_u/(2m)$, where d_u is the current degree of node u and m is the total number of edges in the current graph.
2. Choose q_{pa} random nodes with probabilities according to the distribution.
3. Node v joins the network and connects to the q_{pa} nodes chosen in step 2.

This model produces power-law degree distributions, albeit no community structure. Different values for q_{pa} can shift the degree distribution left and right.

3.4 The Small-World Model (SMW). The SMW model is inspired by the observation that most pairs of nodes in real-world networks are a short distance apart despite most connections being local (Kleinberg, 2000; Watts & Strogatz, 1998). The model begins with n isolated nodes arranged in a ring topology.

1. Connect each node to its q_{ring} nearest neighbors in the ring.
2. For each edge (u, v), with probability q_{rewire}, replace the end point (v) of the edge with a randomly chosen node in the graph that u is not already connected to.

This model produces community structure and short path lengths between nodes, indicative of the small-world phenomenon. Different values of q_{ring} can modulate the strength of the community structure, and different values of q_{rewire} can modulate the randomness of the graph.

3.5 The Erdős-Rényi Model (ER). Finally, in the ER random graph model, each of the $\binom{n}{2}$ possible edges exists independently with probability q_{er}. This process produces a binomial degree distribution.

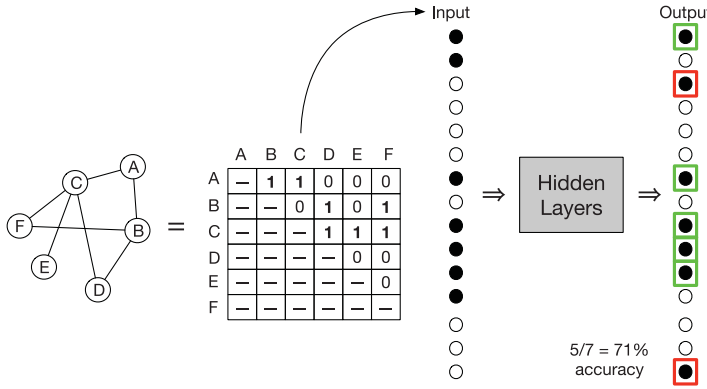

Figure 1: Overview and analysis of graphs using deep learning. The adjacency matrix of a graph is vectorized and input into the input layer of an autoencoder. Each input unit represents a potential edge in the graph (filled circles denote edge presence). The DL architecture consists of some number of hidden layers (1–5 are used in this letter). Reconstruction accuracy compares the actual versus predicted edges following encoding and decoding.

4 Neural Network Design and Optimization

Next, we describe how to use deep learning methods to process graphs. The autoencoder takes r d-dimensional vectors as the input training data set, $X \in \mathbb{Z}^{r \times d}$, where r is the number of training graphs and $d = \binom{n}{2}$ with one dimension for each possible undirected edge (n is the number of nodes; self-loops are ignored). We assume unweighted graphs; therefore, the value of each dimension is 0 or 1 indicating the presence or absence of an edge. Each d-dimensional vector represents the upper triangle of the adjacency matrix of the input graph. The rows (nodes) of the adjacency matrix are sorted from highest to lowest degree (to solve the node identifiability problem and make the input labeling independent). The matrix is then flattened into a vector and input into the autoencoder (see Figure 1).

The autoencoder architecture consists of a single input layer with $\binom{n}{2}$ inputs, followed by some number of hidden layers and a single output layer with $\binom{n}{2}$ outputs. The autoencoder uses tied weights, which equates decoding weights to the transpose of the encoding weights in corresponding layers. For example, for $n = 100$, the layer sizes could be $4950 \to 150 \to 25 \to 150 \to 4950$. For all autoencoders, we used a rectilinear activation function for the output units ($g(z) = \max(0, z)$) and a normalization activation function for the hidden units ($g(z) = z - \text{std}(z)$). These were selected after performing a grid search across 15 activation functions (hyperbolic tangent, linear, logistic, rectilinear, normalization, sigmoid—and several of their variants).

For the autoencoder, each output unit predicts the presence or absence of an edge in the input graph and therefore should have the value 0 or 1. Such a binary output can be forced by defining special output activation functions (e.g., a threshold function that maps summed inputs above a threshold to 1, and 0 otherwise). This approach, however, can lead to slower convergence (more parameters to learn), and it does not guarantee that the output graph has the same number of edges as the input graph. To circumvent these problems, we allow real-valued outputs during training, but during testing, we binarize outputs by setting the top m most active output nodes to 1 and the rest to 0 (m is the number of edges in the input graph, which is known). This procedure guarantees that the reconstructed graph has the same number of edges as the input graph. This procedure is akin to taking the linear relaxation of an integer-linear program during training, followed by a rounding procedure for each test example. In our case, the rounding procedure ensures that exactly m output nodes have value 1, effectively creating an m-winner-take-all network.

For classification, the same input structure and activation functions are used as the autoencoder, with a softmax operator applied to the output. The output layer contains exactly j nodes, where j is the number of class labels.

For both real-world biological networks (protein interaction network and metabolic networks), we used the same architecture: two hidden layers with 150 and 25 hidden units.

4.1 Training the Network. For the autoencoder, the objective loss function $\mathcal{L}$ computes the mean squared error between the input and output of the autoencoder R_θ (with parameters θ) following encoding and decoding,

$$\mathcal{L}(X, \theta) = \frac{1}{r} \sum_{i=1}^{r} \left\| R_\theta(x_i) - x_i \right\|_2^2 + \lambda(\theta), \tag{4.1}$$

where $x_i \in X$ is a d-dimensional vector corresponding to the input graph G_i and λ is a regularizer.

For classification, the loss function is the cross-entropy between $Y_\theta(x_i)_j$ (the softmax output from the deep classifier for predicting label j for input x_i) and the true output $M(x_i)_j$ (which equals 1 if x_i has label j, and 0 otherwise), summed over all class labels:

$$\mathcal{L}(X, M, \theta) = -\frac{1}{r} \sum_{i=1}^{r} \sum_{j} M(x_i)_j \log Y_\theta(x_i)_j + \lambda(\theta). \tag{4.2}$$

For optimization, we use simple and well-established techniques often used in the deep learning community: stochastic gradient descent (learning rate $= 1e^{-5}$), Nesterov momentum (momentum $= 0.99$), dropout

regularization (dropout = 0.25; Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014), and GPU-accelerated computation for efficiency.

4.2 Testing and Performance Evaluation. Generative model parameters were selected such that all models produced graphs with a similar range of edges to challenge the autoencoder to discover discriminative features beyond those based on edge density. These parameters were selected uniformly within a range: for DMC, $q_{\text{mod}} \in [0.35 - 0.7]$, $q_{\text{con}} \in [0.8, 0.9]$ (average of 371 edges); for FF, $q_{\text{fire}} \in [0.4, 0.5]$ (average of 375 edges); for PA, $q_{\text{pa}} \in [2, 7]$ (average of 432 edges); for SMW, $q_{\text{ring}} \in [5, 11]$, $q_{\text{rewire}} \in [0.05, 0.25]$ (average of 370 edges); and for ER, $q_{\text{er}} \in [0.035, 0.130]$ (average of 401 edges). Selecting all parameters uniformly within the range $[0, 1]$ improved reconstruction accuracy (see the appendix), but we discuss the more difficult of the two tests here. We generated 50,000 training and 50,000 test examples (graphs) with $n = 100$ nodes unless otherwise noted.

For the autoencoder, reconstruction accuracy is the percentage overlap in true versus predicted edges on the test examples. We compare four methods:

- *PCA + SVD:* PCA using singular value decomposition (SVD) to keep the most significant singular vectors. We use the same number of components (k) as hidden units in the smallest hidden layer in the autoencoder.
- *ICA* (Hyvarinen & Oja, 2000): We use the FastICA algorithm, with the same number of mutually independent components (k) as hidden units in the smallest hidden layer in the autoencoder.
- *Nonnegative matrix factorization* (NMF; Lin, 2007): We use the same number of components (k) as hidden units in the smallest hidden layer in the autoencoder.
- *Dictionary learning* (DictLearn; Mairal, Bach, Ponce, & Sapiro, 2009): We find a dictionary (set of atoms) that can best represent the input data using a sparse code. We use the same number of dictionary elements (k) as hidden units in the smallest hidden layer in the autoencoder.

For classification, accuracy is the average number of test examples labeled correctly. We compare to a K-nearest-neighbors classifier ($K = 5$, Minkowski distance), decision trees (Gini criterion), AdaBoost (with decision tree base estimators), and SVMs (RBF kernels; Pedregosa et al., 2011).

4.2.1 Application to Real-World Networks. We applied the autoencoder to real-world networks by iteratively processing 100-node local subnetworks of the real-world network. Such subnetworking is done for two reasons. First, it enhances scalability (the size of the adjacency matrix is $\mathcal{O}(n^2)$, which may otherwise be a prohibitively large size for the input layer when n is

large). Second, it highlights features that are prevalent, independent of network size. The downside is that some global-level features may not be captured.

To create local subnetworks (of the real-world network) of size $n = 100$, we performed a breadth-first walk starting from each node u, stopped when exactly n nodes were visited, and then input the induced subgraph of all n visited nodes into the autoenoder. For the prediction tasks (yeast essentiality, metabolic niche breadth), we averaged the hidden unit activity over all 100-node subnetworks, applied PCA to create a single feature value, and then correlated this value with essentiality or niche breadth. For the kernel subgraphs feature, we computed occurrence counts for each of the 11 four-node subgraph kernels in the graph and then similarly applied PCA to create a single feature value for the network. For both, the single summary feature accounted for 90% or more of the variance, indicating minimal loss of information due to PCA. In all cases, we start with the largest connected component of the network.

5 Results

First, we determine how well autoencoders can compress random networks. Second, we investigate the topological features learned in hidden layers and compare them to human-crafted features. Third, we use these features to predict growth mechanisms of evolving networks, network robustness of protein interaction networks, and environmental niches for bacterial metabolic networks.

5.1 Accurate Reconstruction of Graphs Using Autoencoders. Can autoencoders compress graphs with high reconstruction accuracy using few hidden units? To learn discriminative topological patterns, autoencoders must encode features in hidden units that allow for accurate reconstruction of the input graphs. We trained an autoencoder using 50,000 graphs ($n = 50$ nodes per graph) generated by the five random graph models (see section 3) in equal proportions. The autoencoder achieved better compression than other dimensionality-reduction methods, reducing the input size to 40% of its original (from 1225 input units to 500 hidden units in a single hidden layer) while still reconstructing 93.1% of the original edges (see Figure 2A). With further reduction to 20% of the input dimension (250 hidden units), the autoencoder achieved a 73.4% reconstruction accuracy, also outperforming other methods, including PCA + SVD, ICA (Hyvarinen & Oja, 2000), nonnegative matrix factorization (Lin, 2007), and dictionary learning (Mairal et al., 2009). Prior work has established a correspondence between PCA and autoencoders with linear hidden units (Baldi & Lu, 2012). Our improvement over this baseline is thus largely attributed to the nonlinearity of the autoencoder units.

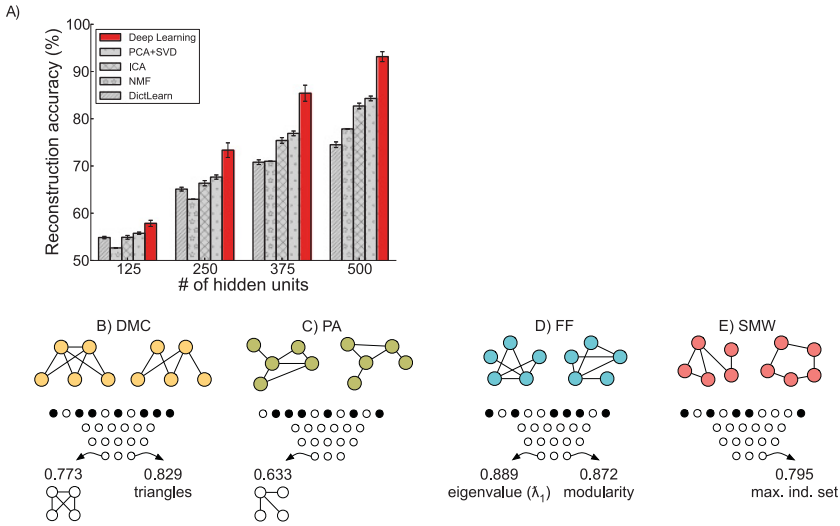

Figure 2: Reconstruction accuracy and learned features. (A) Reconstruction accuracy of five dimensionality-reduction methods using graphs sampled from all five random graph models in equal proportions. The five methods we compare are deep learning, PCA with SVD, ICA, nonnegative matrix factorization (NMF), and dictionary learning (DictLearn). The DL architecture consists of only a single hidden layer with the shown number of hidden units. (B–E) Example features learned by four autoencoders, each trained using graphs from a different random graph model. For each model, we show two schematic input graphs and an autoencoder with three hidden layers. The actual DL architecture consists of an input layer with 4950 units ($n = 100$), followed by three hidden layers with 1000, 250, and 50 units, respectively. Nodes in the third hidden layer learn to recognize model-specific topological features. For example, for DMC, one hidden unit learned to recognize the shown subgraph (correlation of 0.889 between the hidden unit's activity and the number of instances of the shown subgraph in the input graph). For FF, the activity of one hidden unit was highly correlated (0.872) with the modularity of the input graph.

In the appendix, we discuss additional technical results, including reconstruction accuracy for each random graph model individually, model parameter variation, network size variation, experiments with directed graphs, and algorithm run time.

5.2 Graph Features Learned by the Autoencoder. What network features were learned in hidden layers that enabled accurate reconstruction? We trained one autoencoder using graphs from each random graph model separately to validate whether the activity of hidden units strongly correlated with features known to be produced by the model (e.g., hidden

units that preferentially fired for graphs with high modularity for FF). Indeed, we found that hidden unit activity correlated with several popular human-designed graph features (see Figures 2B–2E). The activity of units in the first hidden layer for all autoencoders was highly correlated with simple features related to the number of edges in the graph. For example, in autoencoders trained using the FF/PA models, a 0.795/0.847 correlation was observed between average graph degree and hidden unit activity of several units. In the third hidden layer, model-specific features began to emerge. For the DMC autoencoder, triangles and a four-node subgraph with pairs of topologically redundant nodes were two top recognized features (correlation of 0.773 and 0.829, respectively; see Figure 2B). Both of these features are expected to emerge from the duplication process, where two paralogous genes often share common interaction partners (Vazquez et al., 2003). For the PA autoencoder, a four-node hub subgraph that exemplifies the preferential attachment mechanism emerged as a latent feature (see Figure 2C). For FF, community structure (modularity; see Figure 2D) was learned, which is also a feature that explicitly inspired the design of this model (Leskovec et al., 2007). Interestingly, there were also hidden units in each autoencoder whose activity did not correlate with any of the human features tested. (See the appendix for a list of features tested and further analysis of hidden unit correlations.)

While interpreting hidden units by correlating their activity to known features provides some evidence of what hidden units may be encoding, truly understanding what has been learned in hidden units remains a challenge when using deep learning approaches (see section 6). One alternative approach is to present the autoencoder with many input graphs and record, for each hidden unit, the graph that maximally activated the unit (Le et al., 2012). This analysis, presented in the appendix, shows that there is indeed a range of topologies that activate different units.

Overall, these results suggest that autoencoders can compress graphs with higher accuracy than several popular dimensionality-reduction methods, that DL features can recapitulate some popular network features used today (correlatively), and that some DL features may be novel and useful for graph mining tasks, as we show next.

5.3 Inferring Parameters of Evolution Using Deep Classifiers. Given a network generated using the DMC model, can we predict the exact DMC model parameters (q_{mod} and q_{con}) used in the growth process? Such an inference can determine the relative importance of divergence between paralogs in a species or how wiring has evolved differently across multiple species (Makino, Suzuki, & Gojobori, 2006; Fraser, Hirsh, Steinmetz, Scharfe, & Feldman, 2002).

To pose this as a classification problem, we grew 100-node random DMC graphs using 25 combinations of $q_{\text{mod}}, q_{\text{con}} \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, used for training. We tried to predict the exact parameters (one parameter

pair = one class) using a DL classifier with an architecture with two hidden layers and 25 output units. We compared with two previous studies that used a decision tree (DT) classifier and handcrafted features for a similar graph classification problem (Middendorf et al., 2005; Navlakha, Faloutsos, & Bar-Joseph, 2015). The first (DT3) is trained with three features: the largest eigenvalue of the adjacency matrix, the number of connected components, and the fraction of degree 1 nodes. The second (DT11) is trained using 11 features: occurrence counts for each of the 11 four-node subgraph kernels in the graph (Wernicke & Rasche, 2006). We also compared versus SVMs and AdaBoost (with decision tree base estimators) trained using the same 11 subgraph features and versus a *K*-nearest-neighbors (KNN) classifier using the number of edges in the graph as the only feature.

The highest accuracy was achieved by the DL classifier (68.8%), improving over AdaBoost (66.0%), DT11 (62.6%), SVM11 (59.5%), DT3 (51.1%), KNN1 (32.7%), and random guessing (1/25 = 4%). We also tested the performance of the DL classifier when the input was the 11 subgraph occurrence counts, as opposed to the raw vectorized adjacency matrices. In other words, the autoencoder contained 11 input units with two hidden layers and 25 output units. Here, the DL classifier performed slightly worse than when provided the raw input (67.4% versus 68.8%) but still slightly better than AdaBoost (66.0%), DT11 (62.6%), and SVM11 (59.5%). We also tested the performance of the standard classifiers when provided the raw vectorized adjacency matrices as the input. Here, the standard classifiers performed much worse than the DL classifier: 11.6% for KNN, 17.8% for AdaBoost, 22.4% for the decision tree, and 53.7% for the SVM. Thus, if the standard classifiers are trained on the raw data, they do much worse than DL, which highlights the ability of deep learning to extract informative features from raw data. On the other hand, if DL is trained using the 11 subgraph features, it does marginally better than the human-crafted features and marginally worse than the raw DL features.

While the improved performance of DL versus these machine learning classifiers is consistent with previous observations (LeCun et al., 2015), the novelty here is in demonstrating that this improvement also extends to network analysis. Next, we show that these latent features are also useful for analyzing real-world biological networks.

5.4 Predicting Fragility in Molecular Interaction Networks. Can DL features improve prediction of cellular robustness compared to existing features? Network topology can provide important clues about how protein-protein interactions are perturbed due to node failures or environmental noise (Kitano, 2004; Albert, Jeong, & Barabasi, 2000; Barabasi & Oltvai, 2004; Alon, 2003; Zotenko, Mestre, O'Leary, & Przytycka, 2008), and how these perturbations can lead to disease (Barabasi et al., 2011). Prior work has hypothesized several, often contradictory, topological structures that the cell uses to enhance fault tolerance to noise and failures, including

redundancy (Song & Singh, 2013), sparsity (Navlakha et al., 2014), and specific motifs (Milo et al., 2002).

We focused on a protein network fragility task, where the goal is to determine whether network topology can be used to predict whether the cell survives a perturbation on the network. We collected 20 protein-protein interaction networks, each containing a set of *S. cerevisiae* proteins involved in a unique biological process, along with their induced interactions (Chatr-Aryamontri et al., 2015; MacIsaac et al., 2006). These biological processes include proteins involved in, for example, nuclear transport, DNA replication, and cytoskeleton organization, and each contains at least 100 nodes. These biological processes are all required for cell survival and growth under normal growth conditions (Ashburner et al., 2000). Summed over all 20 networks, there were 5796 proteins and 79,988 edges (physical interactions between proteins). We associated each network with a fragility score equal to the proportion of genes in the network that were experimentally determined to be essential (i.e., individual knock-out of these genes results in cell death; Giaever et al., 2002). Of the 5796 proteins, 1122 (19.4%) were essential. Over the 20 networks, fragility scores ranged from 9.5% to 66.3%. The goal was to determine if there were topological differences between these networks that were predictive of their fragility.

Each network was input into the trained autoencoder and was associated with a single feature value based on the vector of hidden unit activity in the deepest hidden layer. The correlation between DL features and network fragility was at least 30% greater than that of five popular human-designed graph features previously used for this task: 0.612 for DL versus 0.472 for average degree, 0.438 for largest eigenvalue of the adjacency matrix, 0.379 for modularity, 0.344 for average shortest-path length, and 0.313 for kernel subgraph motifs (see Figure 3A).

By relating hidden unit activity to existing features, we found that more highly connected networks (i.e. those with a higher average degree and eigenvalue) were more fragile (less robust) than sparsely connected networks. This is somewhat unintuitive because many prior works have suggested that higher connectivity enhances robustness, as measured by the change in network feature (node or edge connectivity, the eigenvalue of the adjacency matrix (Chan, Akoglu, & Tong, 2014), the diameter (Albert et al., 2000), and the number of connected components (Schneider, Moreira, Andrade, Havlin, & Herrmann, 2011)) following node removal. Our finding, however, agrees with more recent observations that modules within the cell adapt their topologies to the cellular environment in which it lies (Song & Singh, 2013): for biological processes that occur external to the cell (e.g., cell wall maintenance, endosomal transport), sparser topologies are preferred to mitigate the spread and effect of cascading environmental noise. High connectivity is used in modules that occur internal to the cell, where such environmental perturbations are less prevalent and information can be transmitted more efficiently. This hybrid strategy that trades off efficiency and robustness is better captured by DL features than

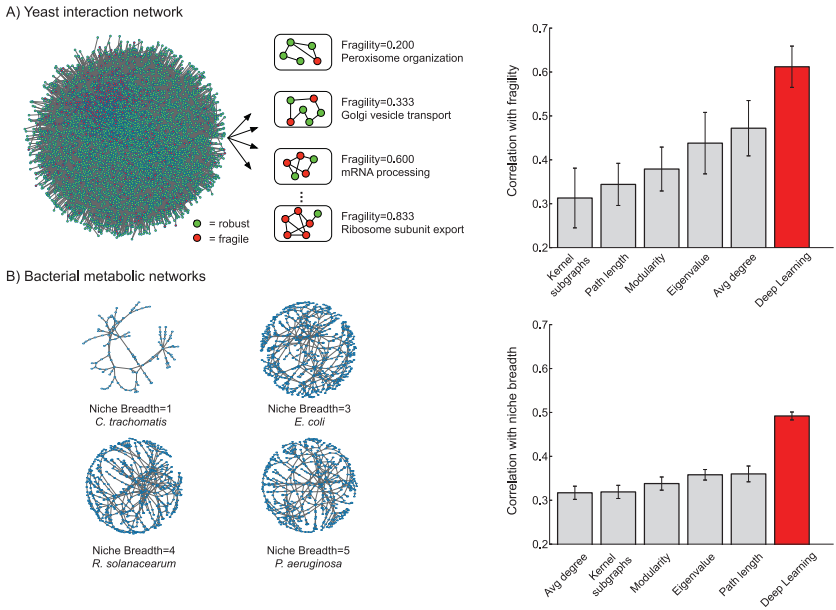

Figure 3: Predicting protein network fragility and bacterial environmental niche from topology. (A) Twenty protein interaction networks for *S. cerevisiae*, each corresponding to a unique biological process. The fragility of each network is the percentage of essential genes (colored red) in the subnetwork. DL features yielded the highest correlation with network fragility, outperforming features based on kernel subgraph counts (motifs), the average shortest-path length, network modularity, network eigenvalue, and average degree. (B) Metabolic networks for 67 bacterial species were each annotated with a niche breadth score ranging from 1 to 5. DL features again yielded the highest correlation with niche breadth. We also projected all five human-crafted features into one dimension and computed their correlation with fragility and niche breadth. The correlations increased slightly compared to each individual feature but were still outperformed by the DL features (fragility: 0.494 for the combination of features versus 0.612 for DL; niche breadth: 0.384 for the combination of features versus 0.492 for DL). Error bars indicate standard deviation of the correlation coefficient computed using leave-one-out cross-validation. In each fold, we removed one data point (network) and computed the correlation coefficient using the remaining networks. All comparisons of DL versus the other features are significantly different ($p < 0.01$, two-sample t-test).

existing features and helps explain the range of topologies found in previous work.

5.5 Predicting Environmental Niches of Bacterial Metabolic Networks. Understanding how network topology is shaped by the environment in which it operates is also an important problem in network

design and evolution (Haldane & May, 2011; Buldyrev, Parshani, Paul, Stanley, & Havlin, 2010; Moore, Shannon, Voelker, & Savage, 2003). In bacteria, prior work has suggested that challenges from different ecologies play an important role in sculpting strategies for efficient and adaptive metabolism (Kreimer, Borenstein, Gophna, & Ruppin, 2008; Kim, Zorraquino, & Tagkopoulos, 2015). To test this effect at the network level, we collected bacterial metabolic networks for each of 67 unique species (Ma & Zeng, 2003), each containing at least 100 nodes. Nodes in these networks correspond to metabolites and an edge implies an enzymatic reaction transforming one metabolite to another. Each network/species was associated with a niche breadth score ranging from 1 to 5, indicating the diversity and fluctuation of metabolites in the environment under which the organism must function (1 = narrow and stable environment, 5 = highly complex and dynamic environment; Morine, Gu, Myers, & Bielawski, 2009). The goal was to determine if there were topological differences between these networks that were predictive of their niche breadth.

As above, each network was input to the autoencoder and associated with a single feature value. The correlation between DL features and niche breadths was at least 37% greater than other standard features: 0.492 for DL versus 0.360 for average shortest-path length, 0.358 for network eigenvalue, 0.338 for modularity, 0.319 for subgraph kernels, and 0.317 for average node degree (see Figure 3B).

The results of these two sections demonstrate the potential promise of DL features for linking network topology to fragility and environment compared to common features used today. While domain-specific knowledge may refine these results further, we emphasize that these improvements use the same feature vocabulary for both problems.

6 Discussion

Networks pervade almost every scientific field today, yet most network analyses to date have relied on human-designed features and custom graph-theoretic algorithms for extracting such features. We developed a framework for using unsupervised deep learning (DL) to analyze networks and found that features latent within hidden units represent a structural feature vocabulary capable of compressing and classifying graphs with improved performance versus several population dimensionality-reduction methods and classifiers. We also used this feature vocabulary to predict protein network fragility and the effect of environmental pressures on the topology of bacterial metabolic networks with better accuracy than many popular hand-crafted network features. Our primary contribution is not in developing new deep learning methods but rather in showing how existing, well-established algorithms used in the machine learning community can be used in an important application area. As the field continues to mature, we hope the results presented here can be further improved.

Our results demonstrate initial promise for using deep learning for network analysis; however, there are many avenues of future research. One potential shortcoming of deep learning is the size limitation of the input layer. In our representation, every potential edge was represented by an input unit. Thus, for large graphs, the input layer could easily contain millions of units, which makes training computationally very expensive and limits the scale of features that can be learned. One potential solution could be to represent input graphs using sparse matrix representations, which have recently been successfully used in other applications without significant loss in accuracy (Liu, Wang, Foroosh, Tappen, & Penksy, 2015). Another option could be to use graph summarization methods to present a compressed version of the network to the input layer (Boldi & Vigna, 2004; Navlakha et al., 2008). This compressed version, consisting of supernodes (sets of original nodes) and superedges (representing a collection of original edges between two supernodes), can significantly reduce the size of real-world graphs yet can still be inverted to recover most of the original edges. A third option could be to analyze small subgraphs of the input graph, which enables scability with some loss in the ability to detect global features.

Future work also needs to better interpret the complex topological features latent within hidden units and the hierarchical relationships of features across multiple hidden layers (Yosinski, Clune, Nguyen, Fuchs, & Lipson, 2015). Our work demonstrated that DL features correlated with some existing human-designed features, while others appeared to be novel features not yet discovered. For the real-world network tasks, we extracted hidden unit features using PCA, which represents a linear approach to codify features, even though hidden unit activities are likely better related using nonlinear transformations. Learning what has been learned is an important challenge in many DL contexts, especially for network analysis compared to image analysis, where the optimal image that activates a hidden unit can be visualized (Zeiler & Fergus, 2014). Visual inspection of graph layouts may provide some intuition of what is being learned by hidden units (see Figure 4); however, more future work is needed to open up this black box, especially when features are combinatorially coupled across multiple hidden units, or hierarchically across multiple layers.

Future work can also leverage long short-term memory algorithms, which, coupled with our framework, may identify spatiotemporal dependencies that improve prediction of how networks change or develop over time (Hochreiter & Schmidhuber, 1997; Greff, Srivastava, Koutník, Steunebrink, & Schmidhuber, 2015). Our work also opens the door to using DL to study combinatorial optimization problems that can be encoded as graphs (e.g., SAT or graph isomorphism; see Figure 5). Critical for training deep networks are probabilistic random graph models that enable sampling of a broad range of topologies for learning features. While using simulated networks to analyze real-world networks has strong precedence, there are

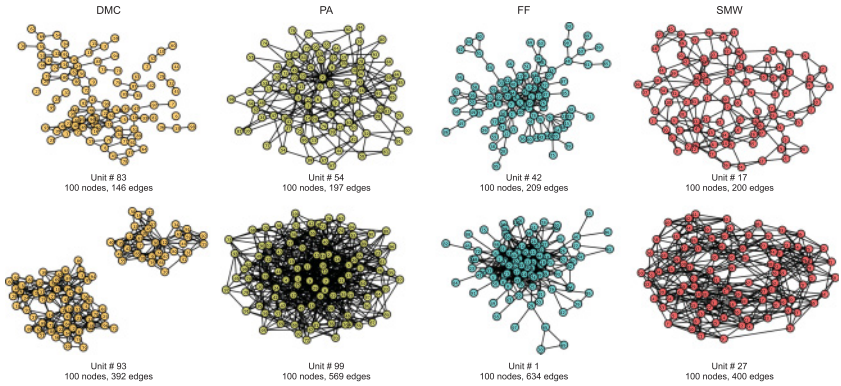

Figure 4: Maximally recognized graphs. For each random graph model, we show two hidden units and the two random graphs that maximally activated the unit. Graphs were displayed using a standard spring embedded layout algorithm.

likely biases in the features that these human-designed models generate, which can limit analyses of some real-world networks. Our work warrants further development of these models (Patro et al., 2012) and presents a novel application for their use.

Finally, in conjunction with deep learning, human experience-driven analysis still plays two key roles in network analysis. First, humans are still needed to interpret hidden unit features, including their relevance and meaning for a particular application. For example, certain subgraphs, like triangles, may be overrepresented in a network but these may be inevitable, not discriminative, if the network is spatially embedded. Moreover, many topological properties can be correlated with one another, though some likely provide more mechanistic insight than others depending on the application. Second, humans are still needed to derive and select appropriate generative models for training.

Appendix

A.1 Additional Reconstruction Results, Parameter Variation, and Directed Graphs. When training and testing one autoencoder per random graph model individually, we found that graphs generated by the SMW and FF models were the easiest to reconstruct (89.8% for SMW, 84.3% for FF, 74.4% for PA, 72.8% for DMC using 1100 hidden units with $n = 100$). ER graphs showed the least compressibility (63.7%), perhaps because of the inherent randomness of edge presence, though some inference was possible because the degree distribution is binomial.

A.1.1 Parameter Variation and Directed Graphs. We achieved similar results when varying the size of the input graph. We varied the graph size between $n = 50, 100, 150$. Autoencoders with a single hidden layer and 20% the number of hidden units as input units had reconstruction accuracies of 0.766, 0.777, and 0.737, respectively (similar gains versus other methods, including using 10% hidden units). We also varied FF, DMC, and ER model parameters to lie in [0,1]. When 10%/20% hidden units ($n = 100$) were used, reconstruction accuracy increased by 11%/5% compared to the parameter range discussed in the main text. We also experimented with directed networks for each model and found similar ordering of performance for all methods (omitted here for brevity).

A.1.2 Run Time. Training an autoencoder using 50,000 graphs ($n = 100$) on a single machine took approximately 128 minutes.

A.2 Further Analysis of Graph Features Learned by the Autoencoders. To complement the results discussed in the main text, we trained a deeper autoencoder with five hidden layers ($4950 \rightarrow 1500 \rightarrow 750 \rightarrow 400 \rightarrow 100 \rightarrow 25$) using 1 million graphs from all five models collectively. We then generated 5000 test graphs using all models and for each graph computed several popular hand-curated features.[1] We then inputted each graph to the autoencoder and recorded the activation of each hidden layer unit. Our goal was to determine if there were hidden units whose activity was strongly correlated (Pearson > 0.6) with the feature value (e.g., nodes that mostly fired for high-modularity graphs).

Table 1 shows the features recognized by units in the fourth and fifth hidden layers. Interestingly, in the fifth layer, several units learned to recognize the normalized eigenvalue of the graph, but none of them recognized other measures that often correlate with the eigenvalue, including the average degree and density of the graph. Thus, while some features are themselves correlated (which can lead to some redundancies), this suggests that higher-order features are learned that are not simply related to the number of edges in the graph. Overall, 19 of the 28 features tested were recognized by some hidden layer 4 or 5 unit (see Table 1). There were also features that were recognized by multiple hidden units, suggesting a representation that is distributed over many units (Hinton, McClelland, & Rumelhart, 1986).

To provide intuition of what is encoded by individual hidden units that does not rely on correlating activity with human-derived features,

[1] Features: average degree, first eigenvalue of the adjacency matrix, number of edges, number of components, size of the maximal independent set, number of triangles, clustering coefficient, k-core size, number of articulation points, fraction of degree 1 nodes, cover time of the graph, betweenness centrality, closeness centrality, network density, average all-pairs shortest path length, modularity, number of modules, and counts for each of the 11 four-node nonisomorphic undirected kernel subgraphs.

Table 1: Features Learned by an Autoencoder with Five Hidden Layers.

Hidden Layer 4			Hidden Layer 5		
Feature	Number of Units	Correlation	Feature	Number of Units	Correlation
Clustering coefficient	2	0.621	Subgraph-9 counts	4	0.620
Subgraph-6 counts	5	0.625	|Maximal independent set|	4	0.624
Modularity	14	0.625	Number of triangles	1	0.627
Eigenvalue	16	0.643	Effective cover time	14	0.633
Number of modules	4	0.661	Number of articulation points	13	0.639
|Maximal independent set|	28	0.671	Subgraph-7 counts	4	0.639
Betweenness centrality	7	0.701	Fraction of degree 1 nodes	17	0.653
Closeness centrality	10	0.704	Eigenvalue	21	0.707
Average SP length	8	0.718			
|k-core|	9	0.725			
Number of edges	7	0.772			
Average degree	7	0.775			
Density	7	0.775			

Note: For each feature, we show the number of hidden units whose activity significantly correlated with the value of the feature and the average correlation between feature value and hidden unit activity.

we performed the following test. We presented the autoencoder described above with 5000 random graphs selected from four models (DMC, PA, FF, SMW) in equal proportions. For each of the 100 hidden units in the fourth hidden layer, we recorded the graph (and the model that generated it) that maximally activated the unit. In other words, for each hidden unit j, we found

$$G^{*,j} = \underset{G_i}{\operatorname{argmax}} |a(j, G_i)|, \tag{A.1}$$

where $i = 1 : 5000$ corresponding to the input graphs, $a(j, G_i)$ is the activation of unit j when presented input graph G_i, and $G^{*,j}$ is the graph that unit j maximally responds to. In Figure 4, we show eight hidden units, along with a visualization of the graph (and the graph's generating model) that maximally activated the unit. We show two units per model; one unit that responded to a relatively dense graph and one that responded to a relatively sparse graph. For example, hidden unit 42 was maximally responsive to an FF-generated graph that contained many hub-and-spoke subgraphs. Unit 83 was maximally responsive to a DMC graph with many chain-like structures. While it is still difficult to translate these visualizations to precise human understanding, it does provide some intuition of the range of features that may be encoded across hidden units.

A.3 Graph Isomorphism: Detecting Invariances in Graphs. Detecting invariances is a critical component of feature extraction across many domains (Goodfellow et al., 2009), including graphs. To explicitly test whether DL features are robust to topological invariances, we challenged it to solve a notoriously difficult graph-theoretic problem: graph isomorphism. This problem is defined as follows: Given two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, are G_1 and G_2 isomorphic? That is, does there exist a mapping $f : V_1 \to V_2$ such that $u, v \in E_1$ iff $f(u), f(v) \in E_2$?

To cast this as a learning problem, we created a training data set of pairs of isomorphic and nonisomorphic graphs as follows. First, we generated a graph G_1 using a random graph model. To create an isomorphism of G_1, we copy G_1 to G_2 and then randomly swap the labels of G_2 (i.e., shuffle its rows and columns). To create a nonisomorphism of G_1, in addition to copying and label swapping, we also select two existing edges in G_2 and swap their end points: (u_1, v_1) and (u_2, v_2) become (u_1, v_2) and (u_2, v_1), ensuring that neither of the latter two edges already exists. Crucially, this procedure leaves both graphs with the same degree of distribution. In both cases, G_1 remains fixed, and we varied the number of edge swaps as a percentage of the number of edges in G_2. Training and test sets were split evenly between isomorphic and nonisomorphic pairs. The input to the DL classifier is the concatenation of the flattened adjacency matrices of both graphs.

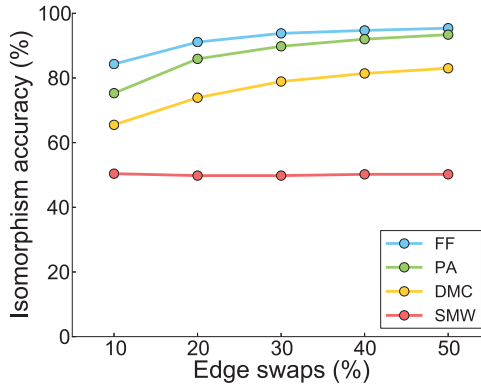

Figure 5: Graph isomorphism using deep learning. Classification accuracy of a deep classifier for predicting whether two graphs are isomorphic. Results are shown for each random graph model separately as a percentage of the number of edge swaps in the isomorphism test.

We trained one DL classifier per random graph model using pairs of 50-node graphs from the model. The classifier had 2450 input units (representing the two input graphs), one hidden layer with 500 units, and two output nodes (isomorphic or not). With many edge swaps (50%), we could predict whether two graphs are isomorphic with very high accuracy: 95.4% for FF, 93.4% for PA, and 83.0% for DMC (see Figure 5). With fewer edge swaps (20%), average accuracy across the three models was still high: 91.1% for FF graphs, 85.9% for PA, and 73.9% for DMC. Accuracy for SMW was nearly random, perhaps because of the strong symmetries imposed by this model. This suggests that DL can detect some, but not all, invariances in the topology and that DL features are independent of the number of nodes and edges in the graph and the graph's degree distribution.

Acknowledgments

Thanks to Carl Kingsford, Krishnan Padmanabhan, and Alan Saghatelian for helpful comments on the manuscript. I also acknowledge the support of the NVIDIA Corporation for donating a GPU used for this research.

References

Adler, M., & Mitzenmacher, M. (2001). Towards compressing web graphs. In *Proc. of the Data Compression Conference* (pp. 203–212). Washington, DC: IEEE Computer Society.

Albert, R., Jeong, H., & Barabasi, A. L. (2000). Error and attack tolerance of complex networks. *Nature*, *406*(6794), 378–382.

Alipanahi, B., Delong, A., Weirauch, M. T., & Frey, B. J. (2015). Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nat. Biotechnol.*, *33*(8), 831–838.

Alon, U. (2003). Biological networks: The tinkerer as an engineer. *Science*, *301*(5641), 1866–1867.

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., ... Sherlock, G. (2000). Gene ontology: Tool for the unification of biology: The Gene Ontology Consortium. *Nat. Genet.*, *25*(1), 25–29.

Baldi, P., & Lu, Z. (2012). Complex-valued autoencoders. *Neural Netw.*, *33*, 136–147.

Barabasi, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, *286*(5439), 509–512.

Barabasi, A. L., Gulbahce, N., & Loscalzo, J. (2011). Network medicine: A network-based approach to human disease. *Nat. Rev. Genet.*, *12*(1), 56–68.

Barabasi, A. L., & Oltvai, Z. N. (2004). Network biology: Understanding the cell's functional organization. *Nat. Rev. Genet.*, *5*(2), 101–113.

Boldi, P., & Vigna, S. (2004). The webgraph framework I: Compression techniques. In *Proc. of the 13th Intl. Conf. on World Wide Web* (pp. 595–602). New York: ACM.

Bottou, L., Bengio, Y., & Le Cun, Y. (1997). Global training of document processing systems using graph transformer networks. In *Proc. IEEE Intl. Conf. on Computer Vision and Pattern Recognition* (pp. 489–494). Washington, DC: IEEE Computer Society.

Buldyrev, S. V., Parshani, R., Paul, G., Stanley, H. E., & Havlin, S. (2010). Catastrophic cascade of failures in interdependent networks. *Nature*, *464*(7291), 1025–1028.

Chakrabarti, D., & Faloutsos, C. (2006). Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, *38*(1).

Chan, H., Akoglu, L., & Tong, H. (2014). Make it or break it: Manipulating robustness in large netwotks. In *SIAM Intl. Conf. on Data Mining*. Philadelphia: SIAM.

Chatr-Aryamontri, A., Breitkreutz, B. J., Oughtred, R., Boucher, L., Heinicke, S., Chen, D., ... Tyers, M. (2015). The BioGRID interaction database: 2015 update. *Nucleic Acids Res.*, *43*(database issue), D470–D478.

Fraser, H. B., Hirsh, A. E., Steinmetz, L. M., Scharfe, C., & Feldman, M. W. (2002). Evolutionary rate in the protein interaction network. *Science*, *296*(5568), 750–752.

Giaever, G., Chu, A. M., Ni, L., Connelly, C., Riles, L., Veronneau, S., ... Johnston, M. (2002). Functional profiling of the *Saccharomyces cerevisiae* genome. *Nature*, *418*(6896), 387–391.

Goldenberg, A., Zheng, A. X., Fienberg, S. E., & Airoldi, E. M. (2010). A survey of statistical network models. *Found. Trends Mach. Learn.*, *2*(2), 129–233.

Goodfellow, I., Lee, H., Le, Q. V., Saxe, A., & Ng, A. Y. (2009). Measuring invariances in deep networks. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, & A. Culotta (Eds.), *Advances in neural information processing systems, 22* (pp. 646–654). Red Hook, NY: Curran.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2015). LSTM: A search space odyssey. *CoRR*, abs/1503.04069.

Haldane, A. G., & May, R. M. (2011). Systemic risk in banking ecosystems. *Nature*, *469*(7330), 351–355.

Hinton, G. E., Deng, L., Yu, D., Dahl, G. E., Mohamed, A., ... Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.*, *29*(6), 82–97.

Hinton, G. E., McClelland, J. L., & Rumelhart, D. E. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1. Cambridge MA: MIT Press.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, *9*(8), 1735–1780.

Hyvarinen, A., & Oja, E. (2000). Independent component analysis: Algorithms and applications. *Neural Netw.*, *13*(4–5), 411–430.

Kim, M., Zorraquino, V., & Tagkopoulos, I. (2015). Microbial forensics: Predicting phenotypic characteristics and environmental conditions from large-scale gene expression profiles. *PLoS Comput. Biol.*, *11*(3), e1004127.

Kitano, H. (2004). Biological robustness. *Nat. Rev. Genet.*, *5*(11), 826–837.

Kleinberg, J. M. (2000). Navigation in a small world. *Nature*, *406*(6798), 845.

Kreimer, A., Borenstein, E., Gophna, U., & Ruppin, E. (2008). The evolution of modularity in bacterial metabolic networks. *Proc. Natl. Acad. Sci. U.S.A.*, *105*(19), 6976–6981.

Le, Q. V., Ranzato, M., Monga, R., Devin, M., Corrado, G., Chen, K., ... Ng, A. Y. (2012). Building high-level features using large scale unsupervised learning. In *Proc. 29th Intl. Conf. on Machine Learning* (pp. 81–88). New York: Omnipress.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Leskovec, J., Kleinberg, J., & Faloutsos, C. (2007). Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, *1*(1).

Li, X., Du, N., Li, H., Li, K., Gao, J., & Zhang, A. (2014). A deep learning approach to link prediction in dynamic networks. In *Proc. SIAM Intl. Conf. on Data Mining* (pp. 289–297). Philadelphia: SIAM.

Lin, C. J. (2007). Projected gradient methods for nonnegative matrix factorization. *Neural Comput*, *19*(10), 2756–2779.

Liu, B., Wang, M., Foroosh, H., Tappen, M., & Penksy, M. (2015). Sparse convolutional neural networks. In *Proc. of the IEEE Conf. in Computer Vision and Pattern Recognition* (pp. 806–814). Washington, DC: IEEE Computer Society.

Ma, H., & Zeng, A. P. (2003). Reconstruction of metabolic networks from genome data and analysis of their global structure for various organisms. *Bioinformatics*, *19*(2), 270–277.

MacIsaac, K. D., Wang, T., Gordon, D. B., Gifford, D. K., Stormo, G. D., & Fraenkel, E. (2006). An improved map of conserved regulatory sites for *Saccharomyces cerevisiae*. *BMC Bioinformatics*, *7*, 113.

Mairal, J., Bach, F., Ponce, J., & Sapiro, G. (2009). Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 689–696). New York: ACM.

Makino, T., Suzuki, Y., & Gojobori, T. (2006). Differential evolutionary rates of duplicated genes in protein interaction network. *Gene*, *385*, 57–63.

Middendorf, M., Ziv, E., & Wiggins, C. H. (2005). Inferring network mechanisms: The *Drosophila melanogaster* protein interaction network. *Proc. Natl. Acad. Sci. U.S.A.*, *102*(9), 3192–3197.

Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., & Alon, U. (2002). Network motifs: Simple building blocks of complex networks. *Science*, *298*(5594), 824–827.

Moore, D., Shannon, C., Voelker, G., & Savage, S. (2003). Internet quarantine: Requirements for containing self-propagating code. In *Proc. of the IEEE Intl. Conf. on Computer and Communications* (vol. 3, pp. 1901–1910).

Morine, M. J., Gu, H., Myers, R. A., & Bielawski, J. P. (2009). Trade-offs between efficiency and robustness in bacterial metabolic networks are associated with niche breadth. *J. Mol. Evol.*, *68*(5), 506–515.

Navlakha, S., Faloutsos, C., & Bar-Joseph, Z. (2015). MassExodus: Modeling evolving networks in harsh environments. *Data Min. Knowl. Disc.*, *29*, 1211–1232.

Navlakha, S., He, X., Faloutsos, C., & Bar-Joseph, Z. (2014). Topological properties of robust biological and computational networks. *J.R. Soc. Interface*, *11*(96), 20140283.

Navlakha, S., & Kingsford, C. (2011). Network archaeology: Uncovering ancient networks from present-day interactions. *PLoS Comput. Biol.*, *7*(4), e1001119.

Navlakha, S., Rastogi, R., & Shrivastava, N. (2008). Graph summarization with bounded error. In *Proc. of the 2008 ACM Intl. Conf. on Management of Data* (pp. 419–432). New York: ACM.

Newman, M. E. (2006). Modularity and community structure in networks. *Proc. Natl. Acad. Sci. U.S.A.*, *103*(23), 8577–8582.

Newman, M. (2010). Networks: An introduction. New York: Oxford University Press.

Patro, R., Duggal, G., Sefer, E., Wang, H., Filippova, D., & Kingsford, C. (2012). The missing models: A data-driven approach for learning how networks grow. In *Proc. of the ACM Intl. Conf. on Knowledge Discovery and Data Mining* (pp. 42–50). New York: ACM.

Patro, R., & Kingsford, C. (2012). Global network alignment using multiscale spectral signatures. *Bioinformatics*, *28*(23), 3105–3114.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proc. 20th ACM Intl. Conf. on Knowledge Discovery and Data Mining* (pp. 701–710). New York: ACM.

Przulj, N. (2007). Biological network comparison using graphlet degree distribution. *Bioinformatics*, *23*(2), e177–e183.

Rustamov, R., & Guibas, L. (2013). Wavelets on graphs via deep learning. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Weinberger (Eds.), *Advances in neural information processing systems, 26* (pp. 998–1006). Red Hook, NY: Curran.

Schneider, C. M., Moreira, A. A., Andrade, J. S., Havlin, S., & Herrmann, H. J. (2011). Mitigation of malicious attacks on networks. *Proc. Natl. Acad. Sci. U.S.A.*, *108*(10), 3838–3841.

Sharan, R., Ulitsky, I., & Shamir, R. (2007). Network-based prediction of protein function. *Mol. Syst. Biol.*, *3*, 88.

Song, J., & Singh, M. (2013). From hub proteins to hub modules: The relationship between essentiality and centrality in the yeast interactome at different scales of organization. *PLoS Comput. Biol.*, *9*(2), e1002910.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, *15*(1), 1929–1958.

Tan, J., Hammond, J. H., Hogan, D. A., & Greene, C. S. (2016). ADAGE-based integration of publicly available *Pseudomonas aeruginosa* gene expression data with denoising autoencoders illuminates microbe-host interactions. *mSystems*, *1*(1), e00025–15.

Tan, J., Ung, M., Cheng, C., & Greene, C. S. (2015). Unsupervised feature construction and knowledge extraction from genome-wide assays of breast cancer with denoising autoencoders. *Pac. Symp. Biocomput.*, 132–143.

Tian, F., Gao, B., Cui, Q., Chen, E., & Liu, T. (2014). Learning deep representations for graph clustering. In *Proc. 28th Intl. Conf. on Artificial Intelligence* (pp. 1293–1299). Cambridge, MA: AAAI Press.

Vazquez, A., Flammini, A., & Maritan, A. Vespignani, A., (2003). Modeling of protein interaction networks. *Complexus*, *1*(1), 38–44.

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of "small-world" networks. *Nature*, *393*(6684), 440–442.

Wernicke, S., & Rasche, F. (2006). FANMOD: A tool for fast network motif detection. *Bioinformatics*, *22*(9), 1152–1153.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., & Lipson, H. (2015). Understanding neural networks through deep visualization. In *Intl. Conf. on Machine Learning Deep Learning Workshop*. arXiv: 1506.06579.

Zeiler, M., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), *Lecture Notes in Computer Science*: Vol. 8689. *Computer Vision ECCV 2014* (pp. 818–833). New York: Springer.

Zhou, J., & Troyanskaya, O. G. (2015). Predicting effects of noncoding variants with deep learning-based sequence model. *Nat. Methods*, *12*(10), 931–934.

Zotenko, E., Mestre, J., O'Leary, D. P., & Przytycka, T. M. (2008). Why do hubs in the yeast protein interaction network tend to be essential? Reexamining the connection between the network topology and essentiality. *PLoS Comput. Biol.*, *4*(8), e1000140.

Received August 4, 2016; accepted October 11, 2016.