Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 51-2 (2018) 433–438

Learning Graph Dynamics using Deep Neural Networks

Apurva Narayan*. Peter H. O'N Roe**

* Electrical and Computer Engineering, University of Waterloo, Waterloo, ON Canada N2L 3G1 (e-mail: apurva.narayan@uwaterloo.ca).

** Systems Design Engineering Department, University of Waterloo, ON Canada N2L 3G1 (e-mail: phroe@uwaterloo.ca)

Abstract: A large number of real-world problems have high dimensional data. The data obtained from these problems is highly structured and usually in the form of graphs. Graphs represent spatial information about the system in the form of vertices and edges. Often graphs evolve with time and the underlying system exhibits dynamic behavior. Hence, these graphs contain both spatial and temporal information about the system. Understanding, visualizing, and learning large graphs is of key importance for understanding the underlying system and is a challenging task due to the data deluge problem.

Our work here utilizes both spatial and temporal information from structured graphs. We learn spatial and temporal information using a specific type of neural network model. Our model is robust to the kind of graphs and their dynamics of evolution. Our approach is scalable to not only the size of the graph (number of vertices and edges) but also the number of attributes (features) of the data. We show that our approach is simple, generic, parallelizable, and performs at-par with the state-of-the-art techniques. We also compare the results of our model against other existing techniques.

Keywords: Graph Theory, Learning Graphs, Deep Learning.

1. INTRODUCTION

Many real-world problems take the form of graphs. Therefore, learning useful graph representations is critical to the success of many machine learning algorithms. The application of graphs spans a broad spectrum of research domains such as classification (Sen et al., 2008; Vishwanathan et al., 2010), link prediction (Liben-Nowell and Kleinberg, 2007), dynamic network analysis (Carrington et al., 2005), and so on.

The availability of structured datasets in various domains such as social networks, transportation networks, financial networks, or even brain networks is highly pervasive. There are only a few algorithms that can handle large quantities of such structured data and make useful interpretations from it. Current machine learning research or *Deep learning* research focuses on using unstructured datasets. We are also able to interpret results easily that are in the form of images, audio clips, and text. Our ability to interpret and draw useful and significant conclusions from the structured datasets (most of them are in the form of graphs) is limited. Our objective here is to bridge this gap by developing a neural network model, upon some of the existing work for analyzing, predicting, and classifying graphs. We propose a novel approach combining convolutional neural networks and recurrent neural networks (Long Short-Term Memory Networks) to capture both the structural and temporal dynamics of the graphs.

2. RELATED WORKS

It is quite important to note that a large variety of structured datasets are in the form of graphs. Some of the most commonly occurring datasets are knowledge graphs, social networks, protein-interaction networks, and the world wide web. To handle structured data such as graphs and achieve good results in even simple machine learning tasks such as classification, regression, prediction, etc is challenging. In (Zhu et al., 2003), the authors propose a semi-supervised learning algorithm based on Gaussian random field model. The learning problem is articulated as belief propagation problem based on the Gaussian random field. Here relationships are factor nodes and vertices are variable nodes. An efficient and scalable algorithm for feature learning in networks optimizes a novel network aware neighborhood preserving the objective function using stochastic gradient descent (Grover and Leskovec, 2016, p. 2). DeepWalk (Perozzi et al., 2014) uses truncated random walks to learn representations of vertices in graphs efficiently.

Unfortunately, none of the techniques mentioned above can deal with dynamically changing graphs. There is only a fraction of models that have been specifically designed to classify nodes in dynamic networks (Li et al., 2013; Yao and Holder, 2014). The technique proposed by (Li et al., 2013) learns latent feature representation and captures the dynamic behavior. Whereas, in (Yao and Holder, 2014) authors present a support vector machine (SVM) based approach that combines the support vectors from previous time instant with the current time instant to exploit temporal relationships. The approach of (Pei et al., 2016) uses dynamic factor graph model for classification in dynamic social networks. This method organizes the data from these dynamic graphs to the sequence of graphs. They utilize three different kinds of factors, node factor, correlation factor, and dynamic factor to

10.1016/j.ifacol.2018.03.074

capture the global and local properties of graph structures while the dynamic factor exploits the temporal information.

Currently, not much emphasis has been put on extending the power of deep neural architectures to more structured datasets. Recently, there has been an increase of research articles published where they apply neural network architectures to graphs (Bruna et al., 2013; Defferrard et al., 2016; Duvenaud et al., 2015; Kipf and Welling, 2016; Li et al., 2015). It is promising to see increasing research interest in this domain which was previously missing. Neural networks seem to have become an indispensable part of machine learning research. In (Scarselli et al., 2009), developed a novel neural network architecture called the *Graph Neural Network (GNN)*. GNNs allow one to process data in the form of the graph through a neural network architecture. GNNs are applicable in different kinds of graphs such as acyclic, cyclic, directed, and/or undirected. They map the graph and its architecture onto a Euclidean space that one uses in tasks such as forecasting and classification. The model from (Scarselli et al., 2009) was extended by (Li et al., 2015) where they introduced the Gated Recurrent Unit to improve the efficiency. In (Bruna et al., 2013), authors proposed two variants of GNN, one based on the hierarchical clustering and the second based on the spectrum of the Laplacian graph. A similar approach based on the spectral graph appears in (Defferrard et al., 2016). The most significant feature of their work is that the complexity of the algorithms remains unaffected. In (Kipf and Welling, 2016), they present an approach for semi-supervised learning on graph-structured data using Convolutional Neural Network (CNN). They use the local first-order approximation of the spectral graph convolutional network based on (Hammond et al., 2011).

Most works cited above do not handle or cannot handle temporal information efficiently. Our work tries to bridge the gap in the research community by providing a framework that can handle both spatial and temporal information of structured data such as graphs. Our approach is simple, efficient, and learns temporal correlations efficiently.

3. BACKGROUND

In this section, we provide a brief introduction to the necessary background for developing deep neural architecture for learning graphs. There are three main components for our framework 1) Convolutional Neural Networks (CNN), 2) Recurrent Neural Networks (RNN), and 3) Graph theory.

1.1. Convolutional Neural Networks on Graphs

CNNs are similar to the typical feed-forward neural networks. They are a well-known architecture of neural networks in deep learning. CNNs derive their inspiration from the natural visual perception mechanism of living beings. Hubel and Wiesel discovered (Hubel and Wiesel, 1959) that cells in the visual cortex of a living being were responsible for detecting light. Later, Kunihiko Fukushima proposed the model of the Neocongnitron that is the predecessor of the current day CNNs (Fukushima, 1980). The modern-day CNN framework was proposed by LeCun et al. (LeCun et al., 1990). They developed a multi-layer network called LeNet-5. There are numerous well-known models of CNN such as AlexNet (Krizhevsky et al., 2012), VGGNet (Simonyan and Zisserman, 2014), and many others. The evolution of CNN reveals increasing depth (number of layers) and performance of these networks.

Recently, interest in developing algorithms to model the spatio-temporal relationship of graphs has increased. There are two classes of work that exist. Firstly, a generalization of the spatial definition of convolution is proposed by (Masci et al., 2015; Niepert et al., 2016). The second approach involves taking product in the graph Fourier domain using convolution as outlined in (Bruna et al., 2013; Defferrard et al., 2016). The work of (Niepert et al., 2016) has a four-step approach to learning the spatial structure of graphs. Our work extends the approach of (Niepert et al., 2016) to incorporate sequence learning in graphs. Sequence learning approaches are common in tasks such as time series forecasting, natural language processing, language translation, etc. In context of graphs, sequence learning means to predict a graph in future based on the learnt relationships from historical graphs. We use an intuitive architecture by combining CNNs for graphs as feature extractors and another subclass of neural networks for learning the time dynamics of these graphs.

1.2. Recurrent Neural Networks

RNNs are networks with loops that allow for information to persist from previous time intervals. A generic model of a RNN is shown in Figure 1(a). Here, x_t denotes the input vector, h_t denotes the output vector, and **A** depicts a standard feedforward neural network. The loop on the network **A** allows for information to be passed from one step of the network to the next. These loops provide RNNs the strength to remember information from previous time instances. A RNN can be considered as multiple copies of the same network, each passing a message to a successor as shown in Figure 1(b).

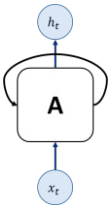

Figure 1(a): A recurrent neural network with loops

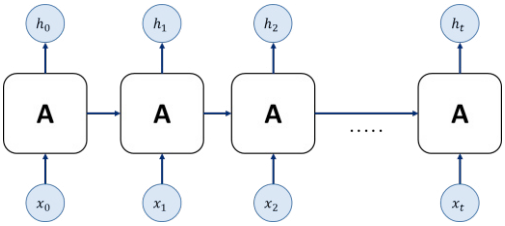

Figure 1(b): An unrolled recurrent neural network

In standard RNNs, the repeating node implements a simplified feed-forward neural network with activation function such as a sigmoid, hyperbolic tangent, etc. Long Short-Term Memory (LSTM) networks are a special kind of

RNNs which improve upon the standard RNNs. LSTMs are a special kind of RNN that prevent the problem of vanishing and exploding gradient as introduced by Hochreiter et al. (Hochreiter and Schmidhuber, 1997). Long Short-Term Memory networks have been quite popular recently for modeling long-term dependencies in various sequence modeling task. We adopt the LSTM model explained in Graves, 2013.

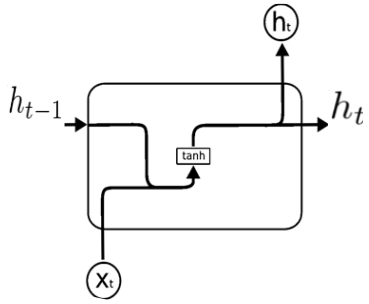

Figure 2(a): Internal architecture of a standard RNN

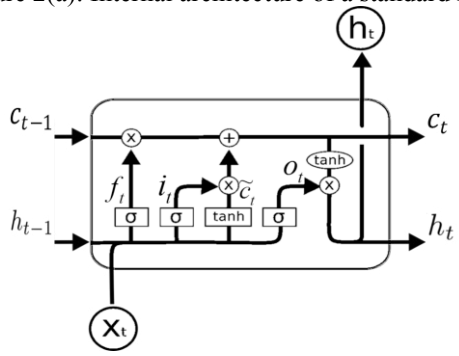

Figure 2(b): Internal architecture of LSTM RNN

The LSTM formulation as described in (Graves, 2013) is shown by Equations 1-6. In Figure 2(a) and Figure 2(b), we show the internal architecture of a standard RNNs and the internal architecture of the LSTM RNN respectively.

In Figures 2(a) and 2(b) each line carries an entire vector, from the output of one node to the inputs of others. the rectangular boxes inside the recurring unit represent layers of a neural network. In Figure 2(a), it is represented by a single layer of neurons with the $\tanh$ activation function. Whereas in Figure 2(b) there are three different layers of neural networks with varying activation functions such as σ and $\tanh$. The operator $\otimes$ represents the pointwise multiplication operation and $\oplus$ is the pointwise addition operation.

The central idea behind the LSTM network is the cell state, i.e. the horizontal line running at the top on Figure 2(b) denoted by c_t. The cell state runs down the entire unit with minimal changes or linear interactions. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through". A LSTM has three of these gates, to protect and control the cell state. The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a $\tanh$ layer creates a vector of new candidate values, $\tilde{c}_t$. that could be added to the state. In the next step, we'll combine these two to create an update to the state. It's now time to update the old cell state, c_{t-1} into the new cell state c_t. The previous steps already decided what to do, we just need to actually do it. We multiply the old state by f_t forgetting the things we decided to forget earlier. Then we add $i_t \oplus \tilde{c}_t$. This is the new candidate value, scaled by how much we decided to update each state value. Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through $\tanh$ (to push the values to be between −1 and −1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

The overall model of LSTM is given by Eq 1-6 where $\sigma(.)$ denotes the sigmoid function, i_t, f_t, o_t denote the input forget and output gates. W_i, W_f, W_c, W_o and b_i, b_f, b_o, b_c are the weight matrices and biases for each of the layers of networks respectively.

$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, x_t\right] + b_i\right) \quad (1)$$

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t\right] + b_f\right) \quad (2)$$

$$\tilde{c}_t = \tanh\left(W_c \cdot \left[h_{t-1}, x_t\right] + b_c\right) \quad (3)$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tilde{c}_t \quad (4)$$

$$o_t = \sigma\left(W_o \cdot \left[h_{t-1}, x_t\right] + b_o\right) \quad (5)$$

$$h_t = o_t \otimes \tanh\left(c_t\right) \quad (6)$$

1.3. Graph Theory

A graph G is a pair (V, E) with $V = \{v_1, v_2, \ldots, v_n\}$ the set of vertices and $E \subseteq V \times V$ the set of edges. Let us consider that the graph has n vertices and m edges. Therefore, each graph can be represented by its adjacency matrix A of dimensions $n \times n$. An edge from vertex v_i to v_j is represented by 1 in the adjacency matrix i.e. $A_{ij} = 1$ otherwise $A_{ij} = 0$. The positioning of a vertex is its index i or j in A. For the purpose of understanding we can stick to index i being the position of a vertex in the graph. Now, we can say that if $A_{ij} = 1$ then v_i and v_j are adjacent vertices. Similarly, a walk on a graph is a sequence of consecutive vertices on a graph that are connected by edges. A path is a walk with distinct nodes. The distance between any two vertices u and v in a graph is denoted by $d(u, v)$, that is the shortest path between vertex u and v. All vertices

adjacent to a vertex v form the neighborhood of a vertex called 1-neighborhood and is denoted by $N_1(u,v)$.

Labeling and Node Partitions

Graph labeling is used to impose an order on the nodes. A graph labeling l is a function $l: V \to S$ from the set of vertices V to an ordered set S. A ranking function $r: V \to \{1,\ldots,|V|\}$ is used to rank all the labeled nodes of a graph. The ranking and labeling are related in a way that $r(u) < r(v)$ if and only if $l(u) > l(v)$. If the labeling l of the graph G is injective, it determines a total order of $G's$ vertices and a unique adjacency matrix $A^l(G)$ of G where vertex v has position $r(v)$ in $A^l(G)$. Moreover, every graph labeling induces a partition $\{V_1,\ldots V_n\}$ on V with $u,v \in V_i$, if and only if $l(u) = l(v)$.

Various examples of graph labeling procedures are vertex degree and other measures of centrality commonly used in the network analysis. For example, the *betweenness centrality* of a vertex v computes the fractions of shortest paths that pass through v. We use the tool PATCHY-SAN that implements these labeling procedures and a few others such as degree, page-rank, eigenvector centrality etc.

4. LEARNING DYNAMIC GRAPHS USING CNN

CNNs have been quite popular in the domain of computer vision. Most common application of CNNs on images is where a filter or the so-called receptive field moves across the image with a specified stride length. The filter is said to convolve with the image. Smaller images are created from the original after the convolution operation. Images have an inherent spatial order that allows one to move the filter across the images in a structured pattern (ex. from left to right). The spatial correlation in the images determines the way the nodes transform to a vector space. It allows the pixels with a similar structural role to have a single representation within the receptive field.

In our work, we expand upon the work of (Niepert et al., 2016) to learn the spatial structure as well as the time dynamics of the underlying graphs.

The approach for learning the spatial structure of graph is similar to CNN for images, where locally connected neighborhood is constructed from the input graphs. These neighborhoods serve as receptive fields of a convolutional architecture that allows the framework to learn effective graph representations.

An image can be represented as a square grid whose vertices represent pixels. Now, a CNN can be seen as traversing a vertex sequence and generating fixed size neighborhood graphs for each of the vertices. The neighborhood graphs serve as the receptive fields to read feature values from the pixel vertices. Given the implicit spatial order of the pixels, the sequence of vertices for which neighborhood graphs are created, from left to right and top to bottom, is uniquely determined. However, for numerous graph collections a problem specific ordering (spatial, temporal, or otherwise) is missing and the vertices of the graphs are not in correspondence. We need to solve two problems in this case to perform CNN operations on graphs: 1) Determine the vertex sequences for which neighborhood graphs are created and 2) compute a normalization of neighborhood graphs, that is, a unique mapping from a graph representation into a vector space representation. The work by (Niepert et al., 2016) solves both of the above problems. For each of the input graph, a neighborhood consisting of k (a system parameter) vertices are extracted and normalized, that is, it is uniquely mapped to a space with a fixed linear order. The normalized neighborhood serves as the receptive field for a vertex under consideration. Subsequently, the feature learning components such as convolution and dense layers are combined with the normalized neighborhood graphs as the CNN's receptive field in (Niepert et al., 2016). The features extracted by applying CNN model of (Niepert et al., 2016) is input to our LSTM RNN.

4.1 RgCNN-Long Short-Term Memory on Graph CNN

The most obvious and straightforward extension of the existing approach is to stack a graph CNN (gCNN) model with a recurrent neural network (RNN), abbreviated as RgCNN. To extend the understanding of CNNs on images to graphs one can consider graphs having channels c_v equivalent to the number of vertex attributes and c_e edge attributes or channels in the language of CNNs. On applying the receptive field of size k using the tool PATCHY-SAN (Niepert et al., 2016) to the input graph G we obtain the vertex and edge attributes as tensors of size (k, c_v) and (k, k, c_e) for vertices and edges respectively. The second tensor can be combined to form a tensor of size (k^2, c_e). We apply 1-dimensional convolution to these tensors to obtain features that are supplied to our LSTM framework. The architecture of our CNN network is fixed for all our dataset as it has shown to be robust for varying graph datasets. If there are n attributes for each vertex in the input tensor and field size k_1, with M filters in the first layer convolving with a stride length k_1 and N filters in the second layer with stride length 1 of field size k_2. Then the last flattened dense layer is a tensor in one dimension of length MN that represents the feature of the given graph. For details of the CNN architecture one may refer to (Niepert et al., 2016).

The one-dimensional tensor of length MN is obtained for a dynamically changing graph G at each time instant t and fed to our LSTM network. The architecture of RgCNN is kept constant to see its robustness against varying datasets. The overview of the workflow of our model is shown in Figure 3.

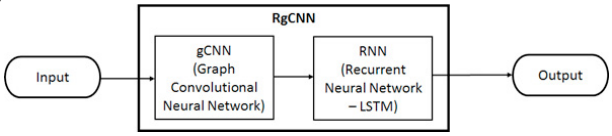

Figure 3: Overview of the RgCNN framework

The input layer of the LSTM network has MN input neurons as that is the output from the CNN fed to it. The output from the RgCNN framework is subjected to a fully connected dense layer for the purpose of classification and prediction.

The Equations 7 – 12 represent the model for LSTM for learning graph dynamics. Here the only difference from the standard LSTM is that the input vector is now depicted by x_t^{gCNN} which is the output from the gCNN.

$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, x_t^{gCNN}\right] + b_i\right) \quad (7)$$

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t^{gCNN}\right] + b_f\right) \quad (8)$$

$$\widetilde{c}_t = \tanh\left(W_c \cdot \left[h_{t-1}, x_t^{gCNN}\right] + b_c\right) \quad (9)$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \widetilde{c}_t \quad (10)$$

$$o_t = \sigma\left(W_o \cdot \left[h_{t-1}, x_t^{gCNN}\right] + b_o\right) \quad (11)$$

$$h_t = o_t \otimes \tanh\left(c_t\right) \quad (12)$$

During the training phase we first used the feature vectors from the gCNN for all time steps for the graphs and trained the LSTM RNN using the standard backpropagation through time algorithm from (Werbos, 1990). The input fed to the LSTM network is the feature vectors obtained from gCNN.

5. EXPERIMENTS

In this section, we describe the experimental setup, the used dataset and the results obtained by our approach.

Dataset

The dataset is a subset of the DBLP dataset as described in (Pei et al., 2016). This dataset contains details of academic conferences across six major research domains including artificial intelligence and machine learning, algorithms and theory, database, data mining, computer vision, and information retrieval. The dataset considers co-author relationships for a decade from 2001 to 2010. Each year's data is organized systematically in a graph form, where each node of the graph represents an author and an edge between two nodes represents the presence of collaboration in a specific year. As explained in (Manessi et al., 2017) the nodal features are extracted using DeepWalk (Perozzi et al., 2014) and consist of 64 values. The main dataset contains 25,215 authors across ten years. Every year 4,786 authors publish on an average, and approximately 114 authors publish almost every year. For the sake of comparison, we matched our dataset with that used in (Manessi et al., 2017). They considered 500 authors who had the highest number of connections during the period of analysis. Therefore, final dataset emerges as 10 adjacency matrices for each of size 500×500.

Experimental Setup

We used Python 2.7 for the development environment. We used a single eight core machine equipped with the Intel i7-3820 CPU at 3.60 GHz and 31.4 Gb of RAM. The machine runs Ubuntu 14.04 LTS 64 bit.

The objective of our experiment was to compare the performance of our framework with other existing approaches. There are a few frameworks available that operate on structured graph data and are able to learn the sequential behavior. We compared the performance of our a) RgCNN with the b) naïve LSTM and c) a standard spectral Graph Convolutional Networks (GCN) (Bruna et al., 2013).

For the training the naïve LSTM we flattened the adjacency matrix of the network at any time instance, and fed it to the network for training. On the other hand for training and testing the performance of GCN we use the tool from (Bruna et al., 2013) directly by feeding in the network adjacency matrix to the network and the available attributes. To standardize the comparison between various architectures we employed *Accuracy* as a metric that is used as a standard in the literature (Manessi et al., 2017).

Results

The network was trained for a minimum of 200 epoch or until the errors converged. The training and test set is in the ratio of 70 % for training and 30 % for testing.

We present in Table 1, the simulation results of our architecture along with other architectures. Our proposed architecture gives promising results. It not only gives better or at-par results with the state-of-the-art architectures but also exceeds in some cases. The most promising feature of our architecture is its simplicity. The need for complex network architectures is not required and a simple stacking framework provides excellent results. The strength of PATCHY-SAN is already shown in (Niepert et al., 2016) and our results have strengthened the belief in the approach as it allows us to learn the temporal dynamics also quite efficiently. Independently GCN (gCNN or PATCHY-SAN) can learn the spatial structure of the graph efficiently whereas LSTMs can enable the network to learn temporal dynamics.

Table 1: Comparison between architectures in terms of accuracy in prediction

Network	*Accuracy*
RgCNN	71.9% ± 1.7%
LSTM-FC	60.1% ± 2.1%
spectral GCN	71.0% ± 2.1%

6. CONCLUSION AND FUTURE WORK

In this work, we introduced a neural network architecture that can deal with the classification of a sequence of vertices and graphs. The model uses a modified graph convolutional networks and long-short term memory networks. We evaluated the performance of our framework on a benchmark dataset and compared the results with other existing frameworks. We show that our framework is superior yet straightforward. It is easily parallelizable that allows for tremendous speedups. In our opinion, an interesting extension of our work would use other recurrent units such as GRUs and evaluate the performance. Moreover, the depth of the networks can be altered for developing sophisticated models.

REFERENCES

Bruna, J., Zaremba, W., Szlam, A., LeCun, Y., 2013. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203.

Carrington, P.J., Scott, J., Wasserman, S., 2005. Models and methods in social network analysis. Cambridge university press.

Cun, L., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D., 1990. Handwritten Digit Recognition with a Back-Propagation Network, in: Advances in Neural Information Processing Systems. Morgan Kaufmann, pp. 396–404.

Defferrard, M., Bresson, X., Vandergheynst, P., 2016. Convolutional neural networks on graphs with fast localized spectral filtering. Presented at the Advances in Neural Information Processing Systems, pp. 3844–3852.

Duvenaud, D.K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.P., 2015. Convolutional networks on graphs for learning molecular fingerprints. Presented at the Advances in neural information processing systems, pp. 2224–2232.

Fukushima, K., 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol. Cybernetics 36, 193–202. doi:10.1007/BF00344251

Graves, A., 2013. Generating Sequences With Recurrent Neural Networks. CoRR abs/1308.0850.

Grover, A., Leskovec, J., 2016. node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 855–864.

Hammond, D.K., Vandergheynst, P., Gribonval, R., 2011. Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis 30, 129–150.

Hochreiter, S., Schmidhuber, J., 1997. Long Short-Term Memory. Neural Comput. 9, 1735–1780. doi:10.1162/neco.1997.9.8.1735

Hubel, D.H., Wiesel, T.N., 1959. Receptive fields of single neurones in the cat's striate cortex. J Physiol 148, 574–591.

Kipf, T.N., Welling, M., 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. ImageNet Classification with Deep Convolutional Neural Networks, in: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (Eds.), Advances in Neural Information Processing Systems 25. Curran Associates, Inc., pp. 1097–1105.

Li, K., Guo, S., Du, N., Gao, J., Zhang, A., 2013. Learning, analyzing and predicting object roles on dynamic networks, in: Data Mining (ICDM), 2013 IEEE 13th International Conference On. IEEE, pp. 428–437.

Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R., 2015. Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493.

Liben-Nowell, D., Kleinberg, J., 2007. The link-prediction problem for social networks. journal of the Association for Information Science and Technology 58, 1019–1031.

Manessi, F., Rozza, A., Manzo, M., 2017. Dynamic Graph Convolutional Networks. arXiv preprint arXiv:1704.06199.

Masci, J., Boscaini, D., Bronstein, M.M., Vandergheynst, P., 2015. Geodesic Convolutional Neural Networks on Riemannian Manifolds, in: Proceedings of the 2015 IEEE International Conference on Computer Vision Workshop (ICCVW), ICCVW '15. IEEE Computer Society, Washington, DC, USA, pp. 832–840. doi:10.1109/ICCVW.2015.112

Neville, J., Jensen, D., 2000. Iterative classification in relational data. Presented at the Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data, pp. 13–20.

Niepert, M., Ahmed, M., Kutzkov, K., 2016. Learning convolutional neural networks for graphs. Presented at the International Conference on Machine Learning, pp. 2014–2023.

Pei, Y., Zhang, J., Fletcher, G.H., Pechenizkiy, M., 2016. Node classification in dynamic social networks, in: Proceedings of AALTD 2016: Second ECML/PKDD International Workshop on Advanced Analytics and Learning on Temporal Data. p. 54.

Perozzi, B., Al-Rfou, R., Skiena, S., 2014. DeepWalk: Online Learning of Social Representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14. ACM, New York, NY, USA, pp. 701–710. doi:10.1145/2623330.2623732

Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G., 2009. The graph neural network model. IEEE Transactions on Neural Networks 20, 61–80.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T., 2008. Collective classification in network data. AI magazine 29, 93–93.

Simonyan, K., Zisserman, A., 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs].

Sutskever, I., Vinyals, O., Le, Q.V., 2014. Sequence to Sequence Learning with Neural Networks. CoRR abs/1409.3215.

Vishwanathan, S.V.N., Schraudolph, N.N., Kondor, R., Borgwardt, K.M., 2010. Graph kernels. Journal of Machine Learning Research 11, 1201–1242.

Werbos, P.J., 1990. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE 78, 1550–1560.

Yao, Y., Holder, L., 2014. Scalable svm-based classification in dynamic graphs, in: Data Mining (ICDM), 2014 IEEE International Conference On. IEEE, pp. 650–659.

Zhu, X., Ghahramani, Z., Lafferty, J.D., 2003. Semi-supervised learning using gaussian fields and harmonic functions, in: Proceedings of the 20th International Conference on Machine Learning (ICML-03). pp. 912–919.