2023 International Conference on Smart Applications, Communications and Networking (SmartNets) | 979-8-3503-0252-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/SmartNets58706.2023.10215926

Graph Neural Network based Approach for Rumor Detection on Social Networks

Daniel Hosseini
Department of Computer Science
California State University, Fullerton
Fullerton, CA, USA
behzadho@csu.fullerton.edu

Rong Jin
Department of Computer Science
California State University, Fullerton
Fullerton, CA, USA
rong.jin@fullerton.edu

***Abstract*—In today's society, social media usage has resulted in several challenges, including the widespread dissemination of rumors - unverified or false information that can significantly impact public perception and decision-making. As a result, detecting and preventing the spread of rumors is essential. Recent scientific studies have proposed various solutions that use both machine learning and deep learning techniques to identify rumors. In this paper, we investigate an approach that employs graph neural networks to detect rumors. Specifically, we represent posts and their responses as graphs, which are then processed through a Graph Attention Network (GAT) layer. The resulting representations are fed into a dense neural network for classification. Our experiments on the PHEME dataset show that our approach achieves satisfactory performance in identifying rumors. This study also provides a promising avenue for future research in the field of rumor detection using graph neural networks.**

***Index Terms*—Rumor Detection, Graph Neural Networks, Classification**

I. Introduction

Over the past decade, social media has emerged as an effective means of sharing information and ideas. However, with the growth of the internet and the ease of sharing information, rumors have spread rapidly across various social networks, resulting in significant harm to individuals and society. Consequently, identifying and preventing the spread of rumors has become crucial. Different definitions of the word "rumor" can be found in the literature. Some studies define rumor as misinformation, while others describe it as unverified information.

Recent advancements in artificial intelligence have led to the development of various techniques for automatically identifying textual rumors. Some studies have proposed identifying rumor characteristics and features from textual content or user information [1]. However, manually collecting attributes for traditional machine learning models is problematic because they do not fully capture rumor patterns. To tackle this problem, previous research works [2], [3] have utilized deep learning models such as Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) to extract feature representations from textual data, user information, and information diffusion patterns. Although the current models can capture local structural features effectively, they fail to account for the overall structural pattern for the purpose of rumor detection. Consequently, it is imperative to incorporate both local and global structural information to achieve accurate rumor detection. The global structural information captures all interactions between tweets. An example of how rumors spread is that if a tweet is originally a rumor, other tweets that come after it may align with the original tweet or previous posts. Thus, maintaining the source-replies graph structure for each conversation and utilizing the Graph Neural Network (GNN) is beneficial for extracting global features in addition to local features.

The effectiveness of Graph Neural Networks (GNNs) in analyzing graphs has made them increasingly popular in recent years. Specifically, they are capable of processing non-Euclidean and non-sequential data in the form of graphs. Various GNN variations have been developed based on various deep learning methods, including CNNs, RNNs, GANs, autoencoders, and reinforcement learning, to effectively capture information from graph structures. Among these, Convolutional GNNs (CGNNs) have become the most popular approach for graph-based applications. Convolutional GNNs utilize both convolution and readout functions to capture both local and global structural information present in graphs. Two types of graph convolutions are spectral and spatial, where spectral convolutions utilize the graph Fourier transform and its expansions to enable convolution through the translation of node representations into the spectrum domain [4]. The early CGNN method used a spectral approach that employed graph Laplacian [5] and spectral filters to build convolutional processes for graphs [6]. However, spectral techniques suffer from two primary limitations: the inability to share parameters among graphs of different sizes and shapes, and the considerable amount of time required for each forward and backward propagation. To overcome these issues, spatial approaches have been developed using message-passing architecture or neighborhood aggregation [7]–[9]. The primary objective of these methods is to aggregate the features of a node with its neighboring nodes and create a representation for the node. Early CGNNs consider all neighbors as equal and aggregate their node embeddings or feature vectors uniformly. In reality, however, neighbors have different impacts, and through training, they should be assigned different weights. The graph Attention Network (GAT) layer addresses this issue

by integrating the attention mechanism into CGNNs, which modifies the convolution function.

In the upcoming sections, we will present a literature review of the relevant aspects of GNNs that are employed in our work. This includes the Graph Attention Layer, Jumping Knowledge Network, and Readout Operations.

A. Graph Attention Layer

Our study uses the Graph Attention Layer (GAT) in the GAT architecture to capture global structural information by computing attention coefficients. The Graph Attention (GAT) layer is responsible for converting input characteristics into more advanced output characteristics. It achieves this by employing self-attention on the nodes and evaluating the significance of adjacent nodes' features for the node under consideration. Softmax normalization is used to adjust the coefficients, and the resulting values are utilized to calculate a linear mixture of the associated features. This forms the final output features for every node. Attention scores are used to scale the gathered embeddings from neighboring nodes. Additionally, multi-head attention can be implemented to enhance the stability of self-attention learning.

B. Jumping Knowledge Network

To develop accurate representations, varying node radius size must be considered. The Jumping Knowledge Network (JKN) addresses this by customizing the neighborhood range for each node and allowing nodes to select the best representation from all intermediate representations in the final layer [10]. Aggregation methods such as concatenation, max-pooling, and LSTM-attention can be used to merge information [11].

C. Readout Operations

Graph convolution procedures learn valuable node features to solve node-level problems. However, graph-level problems require a readout operation to merge node information into a graph representation. Standard CNNs cannot be applied directly to graphs due to their non-grid structure, and readout operations must be order-invariant. Simple statistics such as summing, averaging, and max-pooling [4], fully connected layers [6], hierarchical clustering algorithms [12], and the addition of a pseudo node have been explored as possible solutions for merging representations and generating a graph representation [4].

In our research, we present an approach for classifying a post as either rumor or non-rumor. The approach employs a tree-like graph structure to represent the post and its associated responses. A graph neural network (GNN) algorithm is used to generate an embedding vector for each graph. We create the feature vector for every post using a language model and employ a Graph Attention (GAT) layer to carry out multiple graph convolutions, updating the node representations in the process. A global pooling operation is utilized to produce the overall graph representation, which consolidates the node embeddings to yield a fixed-size representation for the entire graph. To improve the precision of our method, we incorporate a jumping knowledge network (JKN) layer. This layer improves the learning process by allowing the model to leverage the learned representations of nodes from distant regions of the graph. After preprocessing and transforming the data, we develop several models and evaluate their performance on the PHEME dataset.

This article is structured in the following manner. Section II examines pertinent literature. The proposed approach is detailed in Section III. Section IV outlines the experimental procedures, while Section V showcases the findings. Concluding remarks are offered in Section VI, and potential future research works are discussed in Section VII.

II. Related works

Bai et al. [13] propose a deep ensemble neural network to learn global and local structural information from tweets using Source-Replies relation graphs. They use a nodes proportion allocation method (NPAM) to develop an ensemble deep neural network to extract characteristics from discussions of varying lengths [13]. Their study finds that for small graphs, local structural aspects and text characteristics are more crucial for rumor identification, while for complex graphs, both global structural and text properties are significant. They evaluate the impact of word embedding dimensions and compare their results with those of other studies.

Lotfi et al. [14] propose a strategy for rumor detection using Graph Convolutional Networks (GCN) [7]. They construct two graphs for each conversation: a tweet graph in the form of a tree structure and a user participation graph. GCNs are applied to both graphs to capture structural and content information, and the outputs are then combined and sent to a dense layer for identification. The model is evaluated on the PHEME dataset.

Vu et al. [15] propose a strategy for propagation graph embedding using a graph convolutional network, specifically GraphSAGE. The learned node embeddings capture both structural and node characteristics. They further utilized a Long short-term memory to construct the graph's representation from the node embedding vectors. The graph embedding vector is then classified as a rumor or not using a dense neural network. The proposed approach was evaluated on publicly available Twitter datasets.

III. Proposed Approach

Fig. 1 provides a visual summary of the process of the proposed approach. The aim is to classify conversations into two categories: rumor and non-rumor.

A. Data Preprocessing & Transformation

On social networks, users can respond to posts, creating "reply relationships" between the original post and its replies. To discover the relationships between the postings, the original post and its replies can be represented as tree graphs.

As part of our data preprocessing, we employ a Min-Max transformation, see Equation (1), to normalize the values of the features such that they fall within the range [0, 1]. This

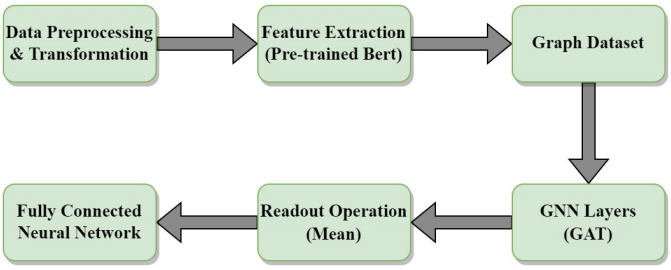

Fig. 1. An Overview Procedure of The Proposed Approach

normalization is done to ensure that the features are of similar scales, which can aid in gradient descent convergence to minima. The scaling of features is particularly helpful in neural network algorithms, as it can improve optimization, speed up training, and prevent the optimization from becoming trapped in local optima.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1}$$

where X is a feature's original value, X_{norm} is its normalized value, and X_{min} and X_{max} are its minimum and maximum values, respectively.

B. Feature Extraction

Given that the posts are text-based, we employ the pre-trained BERT model [17] to obtain feature vectors. We specifically utilize the "bert-base-cased" pre-trained English language model, which has a masked language modeling (MLM) objective. This model is case-sensitive, differentiating between English and english, and needs minimal modifications for various tasks, leading to a substantial enhancement in performance. Table I presents the parameters utilized for Bert.

TABLE I. Parameters of Bert Model

Padding	Max length
Truncation	true
Max length	512
Add special tokens	true

C. Graph Dataset

After processing the tabular datasets in CSV format, we generate an in-memory graph dataset to apply GNN. The graph dataset we create consists of a feature matrix with dimensions of [$|V|$, F], where $|V|$ represents the number of nodes, and F is the number of features for each node. Additionally, it includes an adjacency matrix in COO format with dimensions of [2, $|E|$], where $|E|$ denotes the number of edges. Numeric labels are assigned to each node, starting from zero. The node with label "0" represents the source tweet. Each node in the model has a feature vector comprising of pre-trained Bert model's extracted features and user profile features like user account age. User profiles offer insights into the credibility and reliability of information sources, considering factors like reputation, past behavior, and expertise. Hence, analyzing profiles helps identify patterns related to rumor spreading, like bot accounts or misinformation history, which helps enhance accuracy by incorporating contextual information beyond individual messages. The feature matrix consists of all the feature vectors for all nodes. As the graph is undirected, each pair of nodes has two edges.

D. GCN Layers and Readout

We utilize a GCN called GAT to perform a series of graph convolutions, which update the node representations through the incorporation of data from adjacent nodes within the graph. The GAT is based on the attention mechanism, which weighs neighbor nodes differently during the aggregation and updating of feature vectors. To learn the graph embedding vector or the graph representation, we utilize the global pooling operation like mean, which is also referred to as the readout operation. The vector representing the graph is subsequently classified using a fully connected neural network. Our method's main goal is to effectively combine the structure and node characteristics to create a graph representation that can be employed to differentiate between rumor posts.

To enhance the proposed approach's effectiveness in classifying discussions as rumor or non-rumor, we can integrate jumping knowledge network (JKN) layers into the graph convolutional networks. The JKN layer allows us to pass the node features learned by each GCN layer directly to the JKN layer, which then combines these features using a technique such as concatenation. By passing information from previous layers directly to the JKN layer, subsequent GCNs layers can incorporate information from deeper parts of the graph, potentially improving the model's performance in distinguishing between rumor and non-rumor conversations.

IV. Experimental Setup

A. Dataset

We use the PHEME dataset, which is publicly available as part of the PHEME project [16]. This dataset comprises not only tweets but also responses and dissemination structures. Each source tweet within the PHEME dataset is labeled as either a rumor or a non-rumor. The dataset contains a total of 6,425 conversations. We exclude one sample that does not belong to either class. We use nine sets of real-life discussions related to breaking news, including the Charlie Hebdo massacre, the Ferguson unrest, the Ottawa shooting, the Sydney hostage crisis, and the Germanwings crash. The PHEME dataset provides detailed information on each tweet in JSON files. The data is organized into directories, with each event having its directory containing two subdirectories named "rumors" and "non-rumors." These directories contain additional directories labeled with corresponding tweet IDs. The source tweet can be found in the directory labeled "source-tweet," and the collection of responses to that tweet can be found in the directory labeled "reactions." We parse all JSON files and generate separate tabular datasets for source and replied tweets, collecting post and user profile information,

such as "Is the user verified?", followers count, friends count, tweet count, and account age. Table II summarizes the information in the PHEME dataset.

TABLE II. Statistics of the PHEME dataset.

No. of Source Tweets	6424
No. of Response Tweets	97909
No. of non-rumor tweets	4022
No. of rumor tweets	2402
Min No. of nodes / graph	1
Max No. of nodes / graph	346

B. Models

To evaluate our proposed approach for categorizing conversations as rumor or non-rumor, we build a total of 18 unique models based on our method, each with a different configuration of graph convolutional network (GCN) layers and global pooling layers. Specifically, we experiment with one GAT layer, two GAT layers, and two GAT layers with jumping knowledge network layers. For each configuration, we also experiment with six different global pooling layers: max, $mean$, add, $mean\_max$, $mean\_add$, and $mean\_max\_add$. We use a five-layer fully connected neural network to classify the graph embeddings for each model. The input layer of the neural network has the same number of units as the graph embedding vector, and the sigmoid activation function is used in the last layer. To ensure a fair comparison, we use the same hyperparameter values across all of the models. We compare the performance of the 18 models and select the one that provides the greatest overall performance as our final model for categorizing conversations as rumor or non-rumor. By experimenting with different configurations of GAT layers and global pooling layers, we can identify the optimal architecture for our proposed method and improve its performance in distinguishing between rumor and non-rumor conversations.

C. Implementation Details

To address the unbalanced nature of the dataset, we utilize the random feature vector masking method to balance the training set for our graph analysis. We begin by performing a stratified shuffle split on the data to create a training set (90%) and a test set (10%) that preserves the percentage of samples in each class. We then calculate the number of rumor and non-rumor samples in the training set and randomly duplicate rumor samples with more than 10 nodes while masking 50% of their feature vectors. This ensures that the number of samples in each class is equal. Afterward, we perform another stratified shuffle split to divide the training set into a training set (80%) and a validation set (20%). To optimize our base models, we use the Adam optimization algorithm with an initialization learning rate of 0.0001, 20 epochs, and an embedding size of 712. We set the number of heads in GAT to 1, the batch size to 32, and the dropout to 0.5. The GAT layer has 512 hidden dimensions, and rectified linear units are used for non-linearity. The output layer's activation function in the FC neural network is sigmoid. Our models are implemented in PyTorch [19], and we utilize PyTorch Geometric [20] for message-passing frameworks.

D. Performance Metrics

We use loss, accuracy, and F1 metrics as evaluation criteria for the classifier. These metrics are widely used in the literature as indicators of classifier performance. The binary cross-entropy loss function, Equation (2), is utilized in this study as follows:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i * \log \hat{y} + (1 - y_i) * \log(1 - \hat{y}) \quad (2)$$

where $\hat{y}$ is the predicted value, y is the actual value, and N is the sample size.

V. Results

We use a train-test split approach and a 10-fold cross-validation strategy for evaluating our models. We report the average values of the evaluation metrics for comparative purposes. The loss and F1 score for training and validation using one GAT layer and six different global pooling methods are presented in Fig. 2 and Fig. 3. Meanwhile, the results for two GAT layers are shown in Fig. 4 and Fig. 5, and the results for two GAT layers with a jumping knowledge layer are presented in Fig. 6 and Fig. 7.

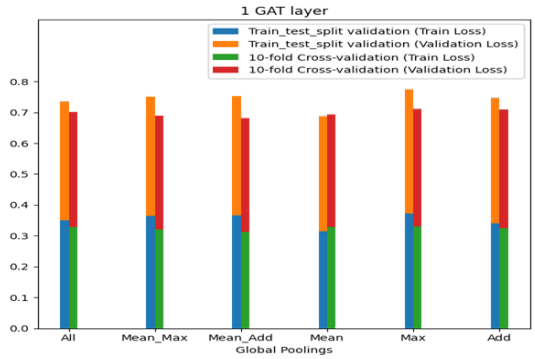

Fig. 2. One GAT Layer (Loss)

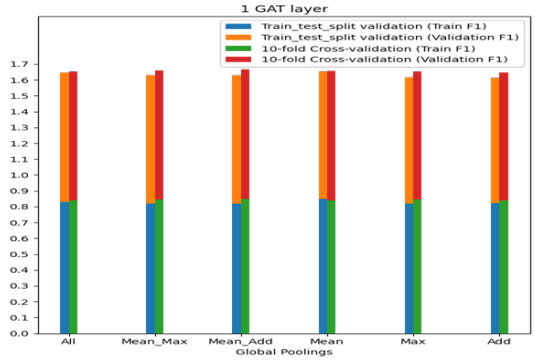

Fig. 3. One GAT Layer (F1 Score)

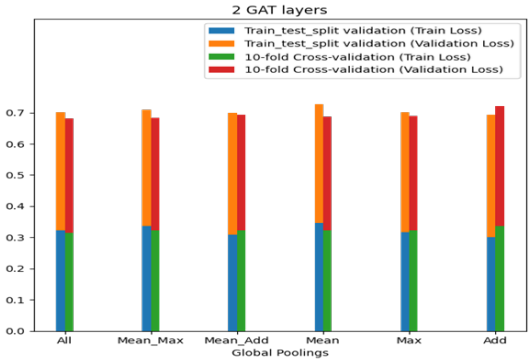

Fig. 4. Two GAT Layers (Loss)

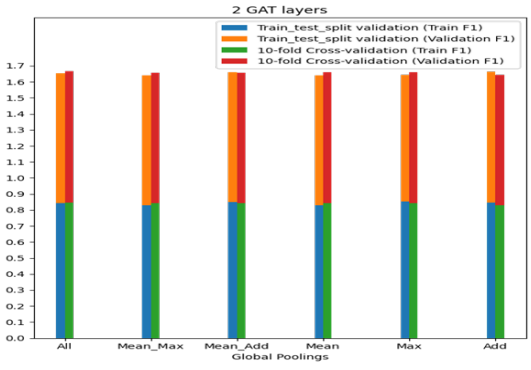

Fig. 5. Two GAT Layers (F1 Score)

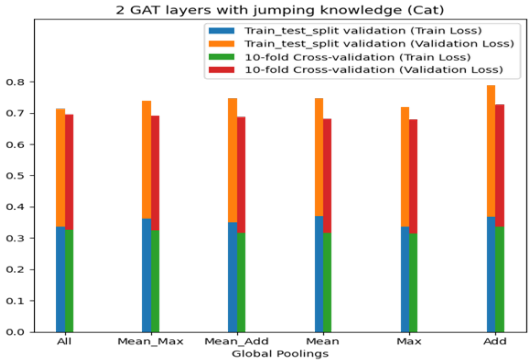

Fig. 6. Two GAT Layers with Jumping Knowledge (Loss)

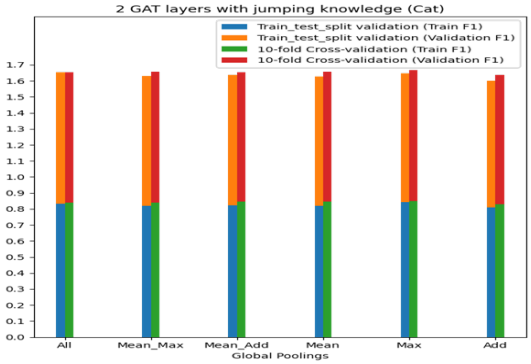

Fig. 7. Two GAT Layers with Jumping Knowledge (F1 Score)

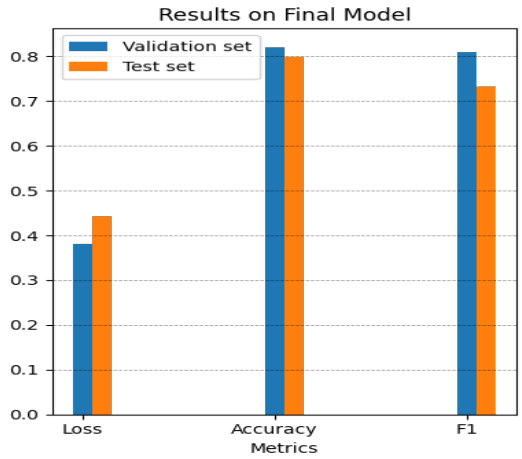

Fig. 8. Results of Test Set (Final Model)

The experiments demonstrate that the model with a single GAT layer and a mean readout method achieves a high level of accuracy (82%) and an F1 score (80%). Consequently, we have selected this model as our final model and evaluated its effectiveness on both the test and validation sets, as displayed in Fig. 8.

A. *Hyperparameter Tuning*

We use Optuna [21], an open-source hyperparameter optimization framework, to automate the process of hyperparameter search. We fine-tune various parameters, including the number of layers and neurons in the fully connected layer. The parameter settings are based on Table III and configured in Optuna. To identify the optimal hyperparameters for our model, we employ the Tree-structured Parzen Estimator (TPE) sampler, which fits a Gaussian Mixture Model (GMM) to the parameter values. We perform the optimization for 200 trials to efficiently find the optimal model and save time in the model development process.

TABLE III. Configuration for tuning hyperparameters.

No. of Epochs	[10, 15, 20, 25, 30, 35, 40, 50, 100]
Batch size	[2, 4, 8, 16, 32, 64]
No. of layers	[1, 6]
No. of units / layer	[128, 1024]
Dropout rate	[0.0, 0.6]
Learning rate	[0.0000001, 0.01]
Weight decay	[0.00000001, 0.1]
Optimizer	Adam
GAT layer	1
GAT layer output size	1024
Sampler	TPESampler [22]
Objectives	Validation accuracy, validation F1

After tuning hyperparameters, we develop a model with the optimized parameters presented in Table IV and evaluate its performance on both the validation and test sets. The outcomes are shown in Fig. 9, indicating that the adjusted model outperforms the original model. Specifically, we observe an

increase of 1% in both accuracy and F1 score, demonstrating the effectiveness of the parameter tuning process.

TABLE IV. Tuned parameters.

No. of Epochs	30
Batch size	32
No. of layers	4
No. of units / layer	1024, 987, 463, 654
Dropout rate	0.121, 0.024, 0.342
Learning rate	0.0000313
Weight decay	0.00221
Optimizer	Adam
GAT layer	1
Size of output in GAT layer	1024

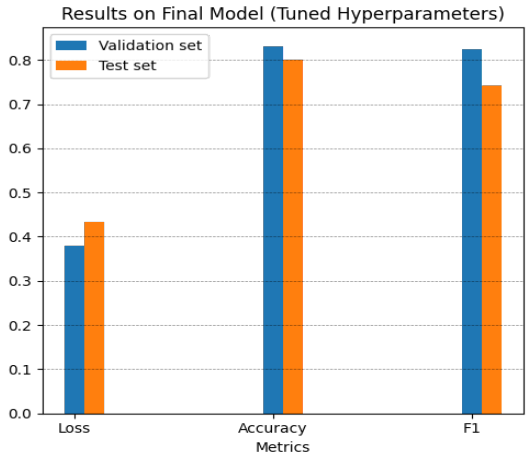

Fig. 9. Results of Test Set (Tuned Final Model)

VI. Conclusion

Our proposed solution for detecting rumors in social networks involves creating a graph representation of the conversation, with each post represented as a vertex and each connection between two posts as an edge. This graph-based approach incorporates various features, including content, user profiles, and dissemination patterns. We employ a GAT layer to integrate neighboring information with node features, resulting in an embedding vector for every node in the graph. We use the mean readout technique to derive a representation vector for the whole graph and then apply a fully connected neural network for classification. In our experiments with the PHEME dataset, we discovered that our selected model outperforms other experimented baseline models. This suggests that our model is highly effective in detecting rumors in social networks. Nevertheless, our study has limitations stemming from insufficient data and the readout operation.

VII. Future Work

We plan to create a diversified dataset since limited data in the current rumor analysis datasets can cause overfitting. Additionally, We will focus on dataset balancing and graph representation learning to enhance the model's performance.

References

[1] S. Vosoughi, D. Roy, and S. Aral, "The spread of true and false news online," Science, vol. 359, no. 6380, pp. 1146-1151, 2018, doi: doi:10.1126/science.aap9559.

[2] J. Ma et al., "Detecting rumors from microblogs with recurrent neural networks," 2016.

[3] A. Alsaeedi and M. Al-Sarem, "Detecting Rumors on Social Media Based on a CNN Deep Learning Technique," Arabian Journal for Science and Engineering, vol. 45, no. 12, pp. 10813-10844, 2020/12/01 2020, doi: 10.1007/s13369-020-04839-2.

[4] Z. Zhang, P. Cui, and W. Zhu, "Deep Learning on Graphs: A Survey," IEEE Transactions on Knowledge and Data Engineering, vol. 34, no. 1, pp. 249-270, 2022, doi: 10.1109/TKDE.2020.2981333.

[5] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," Advances in neural information processing systems, vol. 14, 2001.

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

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

[8] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," Advances in neural information processing systems, vol. 30, 2017.

[9] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," arXiv preprint arXiv:1710.10903, 2017.

[10] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in International conference on machine learning, 2018: PMLR, pp. 5453-5462.

[11] Z. Liu and J. Zhou, "Introduction to graph neural networks," Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 14, no. 2, pp. 1-127, 2020.

[12] J. Ma, P. Cui, X. Wang, and W. Zhu, "Hierarchical taxonomy aware network embedding," in Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 1920-1929.

[13] N. Bai, F. Meng, X. Rui, and Z. Wang, "Rumour Detection Based on Graph Convolutional Neural Net," IEEE Access, vol. 9, pp. 21686-21693, 2021, doi: 10.1109/ACCESS.2021.3050563.

[14] S. Lotfi, M. Mirzarezaee, M. Hosseinzadeh, and V. Seydi, "Detection of rumor conversations in Twitter using graph convolutional networks," Applied Intelligence, vol. 51, no. 7, pp. 4774-4787, 2021/07/01 2021, doi: 10.1007/s10489-020-02036-0.

[15] D. T. Vu and J. J. Jung, "Rumor detection by propagation embedding based on graph convolutional network," International Journal of Computational Intelligence Systems, vol. 14, pp. 1053-1065, 2021.

[16] E. Kochkina, M. Liakataand A. Zubiaga, "PHEME dataset for Rumour Detection and Veracity Classification". figshare, 10-Jun-2018, doi: 10.6084/m9.figshare.6392078.v1.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2018.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[19] A. Paszke et al., "Pytorch: An imperative style, high-performance deep learning library," Advances in neural information processing systems, vol. 32, 2019.

[20] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," arXiv preprint arXiv:1903.02428, 2019.

[21] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, 2019, pp. 2623-2631.

[22] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," Advances in neural information processing systems, vol. 24, 2011.