Deep Learning versus Alternating Least Squares: Comparative Analysis of Trust Link Weight Prediction

ZHIQIANG YU   ZIYUN PAN   CHANGRONG LI   MANUSHREE TYAGI

zhyu | ziyunp | changrli | mtyagi @kth.se

June 7, 2024

Abstract

This study explores methods for predicting the strength of trust relationships within the Bitcoin Over-The-Counter (OTC) network. Trust, represented by weighted edges in a network, is crucial to facilitate transactions and mitigate fraud. We investigate two approaches for predicting link weights: Alternating Least Squares (ALS), a traditional network analysis method that leverages explicit trust ratings between users, and a Deep Learning Model inspired by recent research, which utilizes node embeddings to capture complex relationships and predict link weights. We evaluated both methods on a publicly available dataset, the Bitcoin OTC Web of Trust. Our findings demonstrate that both approaches achieve high accuracy in predicting link weights, with the Deep Learning Model exhibiting slightly better performance as measured by the Mean Squared Error (MSE). This research offers valuable insights into the dynamics of trust within the Bitcoin OTC network. The ability to predict trust relationships can benefit law enforcement agencies in combating criminal activity and stakeholders in enhancing security measures within the cryptocurrency ecosystem.

Keywords: Bitcoin OTC network, link weight prediction, trust networks, deep learning, alternating least squares.

1 Introduction

1.1 Background

The Bitcoin Over-The-Counter (OTC) network serves as a decentralized platform for peer-to-peer transactions, offering participants privacy and reduced market impact compared to centralized exchanges. Within this network, users engage in trust-based interactions, where they rate each other based on transaction experiences. These ratings establish a web of trust crucial for identifying reliable trading partners and mitigating fraud risks. Trust dynamics, represented by weighted edges in the network, play a fundamental role in facilitating transactions and ensuring security within the Bitcoin OTC ecosystem.

Predicting trust relationships and link weights in the Bitcoin OTC network is essential for enhancing security measures, combating fraudulent activities, and improving transactional efficiency. Some examples highlight the importance of predicting trust within such networks. For instance, in 2014, a major Bitcoin exchange, Mt. Gox, collapsed due to fraudulent activities and mismanagement, resulting in the loss of approximately $450 million worth of Bitcoin. Accurate trust prediction models could have flagged suspicious transactions and behaviour patterns, potentially preventing such significant losses [1]. Another example is the Silk Road marketplace, which was notorious for illegal transactions using Bitcoin [2]. Law enforcement agencies eventually shut it down, but better predictive models for trust could have earlier identified and flagged the illicit activities, leading to quicker intervention and reduced criminal activity .

Thus, this study aims to explore and compare the effectiveness of two different algorithms in predicting link weights within the Bitcoin OTC trust network, providing valuable insights into the dynamics of trust and security within decentralized trading environments, which is supposed to benefit stakeholders within the cryptocurrency ecosystem: Law enforcement agencies can benefit from improved methods for identifying suspicious activities and bad actors, while traders and investors can enhance their risk management strategies by identifying trustworthy counterparties.

1.2 Problem Definition

In the decentralized Bitcoin OTC network, accurately predicting trust relationships is essential yet remains a challenge. Classical methods such as Alternating Least Squares (ALS), while strongly interpretable and low-cost, may struggle to capture complex user behaviours. Deep learning approaches are usually more accurate, sacrificing some transparency. Balancing accuracy, interpretability, and transparency is essential for enhancing security and reliability in decentralized trading environments. *Ergo*, this study aims to explore and compare the effectiveness of ALS and deep learning approaches in predicting link weights within the Bitcoin OTC trust network, providing valuable insights into the dynamics of trust and security within decentralized trading environments.

2 Related Works and Research Question

2.1 Related Works

The prediction of link weights in complex networks has gained significant attention in recent years. This section reviews the literature on methodologies for link weight prediction, highlighting key contributions and advances. Graph neural networks (GNNs) have emerged as powerful tools for capturing the complex structures of graphs. Liang and Pu [3] propose the Line Graph Neural Network (LGNN) for link weight prediction, showcasing its ability to leverage the structural properties of graphs to improve predictive accuracy. Their work highlights the importance of considering the connectivity patterns and the local neighbourhood structure in predicting link weights. However, the application scenario is limited and can only handle undirected weighted graphs. Hou and Holder [4] extend the application of deep learning to link weight prediction, demonstrating significant improvements over traditional methods. Their approach integrates deep learning techniques to capture complex patterns within the data, providing a strong framework for weight prediction in diverse network types.

In the domain of **weighted signed networks**, Kumar et al. [5] explore the prediction of edge weights by considering the positive and negative interactions within the network. Their research underlines the complexity of such networks and the need for specialized algorithms to handle signed weights effectively. Similarly, Shi et al. [6] address link weight prediction in signed networks, proposing models that account for both the sign and magnitude of the weights, thus offering a better understanding of relationship strengths and directions.

Specific to the Bitcoin OTC network, Tanevski et al. [7] use link prediction techniques to understand the transactional behaviours and trust relationships within the network . Additionally, Atiya and Nawaf [8] leverage community structures within the Bitcoin network to enhance prediction accuracy, illustrating the benefits of incorporating domain-specific features into the predictive models.

Supervised learning remains a popular approach for link weight prediction. The formative work by Hasan et al. [9] applies supervised learning to link prediction, setting a foundation for many subsequent studies. Rodrigues de Sa and Prudencio [10] further this line of research by focusing on weighted networks, where the prediction of link weights is treated as a regression problem. Zhu et al. [11] propose a neighbour set-based approach for weight prediction, highlighting the importance of local node properties in predictive modelling.

Overall, The extensive research on link weight prediction spans various methodologies and applications, from graph neural networks and deep learning to specialized studies in dynamic and signed networks. These contributions collectively enhance our understanding of link weight dynamics and provide strong frameworks for accurate prediction across different types of networks.

2.2 Research Question

Based on the above research on link prediction methods, we aim to compare two typical supervised learning methods, a kind of linear regression method with the most innovative one to determine which method is the most effective. This comparison will provide a reference for future research or practical applications. Therefore, our research question is: How effective are the two methods for predicting link weights in Bitcoin OTC trust networks using deep learning and traditional network analysis?

3 Methods

3.1 Data Acquisition and Description

The project utilizes the publicly available Bitcoin OTC web of trust network dataset from the Stanford Network Analysis Project (SNAP). This dataset offers valuable insights into user interactions:
Nodes: Represents Bitcoin user addresses (5,881)
Edges: Represents trust relationships between users, with a positive weight signifying trust and a negative weight indicating distrust (35,592 edges)
Time: Associated with each edge is a timestamp, allowing for analysis of trust dynamics over time.

3.2 Data Preprocessing

For efficient computation and to facilitate the regression task, we scaled the raw data to float values in the range of -1 to 1. This normalization step ensures that the data is suitable for input into our models, particularly for deep learning algorithms, which benefit from scaled input values for stability and performance. The data preprocessing steps are as follows:

Normalization: The 'RATING' values in the dataset, which originally ranged from -10 to 10, were normalized (i.e. scaled by 0.1 here) to the range of -1 to 1. This normalization helps to standardize the input data, reducing the risk of numerical instabilities and improving the convergence of the model during training.
Dataset Splitting: The normalized data was then split into three subsets: training (70%), validation (10%), and test (20%). The training set was used to train the model, the validation set to tune hyperparameters and prevent overfitting, and the test set to evaluate the final performance of the model.

We utilized Mean Squared Error (MSE) as the evaluation metric on the test set to assess the predictive accuracy of our models. MSE is defined as the average of the squared differences between the predicted and actual values, providing a measure of the model's performance in terms of prediction error.

When providing the final predictions, we mapped the predicted values back to their original range. This reverse mapping ensures that the output of our model is interpretable and comparable to the original dataset values, facilitating a meaningful evaluation of model performance and practical application of the results.

3.3 Algorithms Description

3.3.1 Alternating Least Squares

3.3.1.1 Least Squares

First, we briefly introduce the classic Least Squares (LS) method. LS is a mathematical optimization technique commonly used in machine learning. It finds the best fit for data by minimizing the sum of the squares of the differences (errors) between observed and predicted values. Essentially, LS aims to find the optimal unknown parameters that minimize the discrepancy between the observed data and the model predictions.

To illustrate LS, consider a simple linear regression model:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

where y is the dependent variable, x is the independent variable, β_0 and β_1 are the coefficients to be estimated, and ε is the error term. The LS method minimizes the sum of the squared errors:

$$\text{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i))^2$$

3.3.1.2 Alternating Least Squares

Next, we introduce the Alternating Least Squares (ALS) algorithm. In recent years, the model-based recommendation algorithm ALS has been successfully applied in Netflix, achieving significant performance improvements [12]. ALS uses machine learning algorithms to build interaction models between users and items, thus predicting new ratings.

In this context, each rating's source and target correspond to the user and item in the original model, fitting well with the explicit feedback scenario. According to the collaborative filtering concept, the row vectors of R correspond to each user (U), and the column vectors correspond to each item (V). The core idea of ALS is to project users and items into a k-dimensional space, assuming there are k latent feature vectors. Each user and each item are represented by a k-dimensional vector, and their inner product approximates the rating value. This allows us to approximate the ratings.
Formally, the rating behaviour can be represented as:

$$R \approx UV^T$$

where:
- R is the rating matrix,
- $U \in R^{m \times k}$ is the user latent feature matrix,
- $V \in R^{n \times k}$ is the item latent feature matrix,
- m is the number of users,
- n is the number of items,
- k is the number of latent features.

The parameters of this model are U and V. Once these are estimated, we can approximate the ratings for unrated items. Under the explicit feedback condition, the cost function is defined as:

$$J(U,V) = \sum_{(i,j) \in \kappa} (R_{ij} - U_i^T V_j)^2 + \lambda (\|U\|_F^2 + \|V\|_F^2)$$

where:
- κ represents the set of user-item pairs for which the ratings are known,
- λ is the regularization parameter,
- $\|\cdot\|_F$ denotes the Frobenius norm.

3.3.1.3 Solution of Alternating Least Squares

As for solving the explicit cost function, it is a convex function, where the variables are coupled together, so the naive conventional gradient descent algorithm can hardly solve it. However, if we first fix U to solve for V, then fix V to solve for U, by alternating between these two steps, the algorithm converges to a local minimum of the cost function, resulting in the optimized U and V matrices. This is why this method is called Alternating Least Squares. Specifically:
1. Fix V and solve for U:

$$U = \arg\min_{U} \sum_{(i,j)\in\kappa} (R_{ij} - U_i^T V_j)^2 + \lambda \|U\|_F^2$$

2. Fix U and solve for V:

$$V = \arg\min_{V} \sum_{(i,j)\in\kappa} (R_{ij} - U_i^T V_j)^2 + \lambda \|V\|_F^2$$

To conclude, this approach leverages the explicit feedback in the dataset to predict user preferences effectively. Since the matrix is sparse and meets the approximate conditions, the time complexity is only $O(k^2N + k^3\sqrt{N})$ (detailed as per **Appendix**), where the latent factor dimension k=15 is quite small in this case according to the metrics we pick.

3.3.1.4 Metrics used in Alternating Least Squares Algorithm

In this section, we provide a detailed description of the six metrics used in our study: Common Neighbors (CN), Jaccard Coefficient (JC), Preferential Attachment (PA), Adamic-Adar Index (AA), Resource Allocation (RA), and Local Clustering Coefficient (CC). While the first three metrics are traditionally defined for undirected and unweighted graphs, we specifically extend their definitions to suit the directed and weighted nature of the network.

Common Neighbors (CN): For a directed and weighted graph, we define four variations of Common Neighbors based on the direction of edges:

$$\text{CNo;o}(x,y) = |\Gamma^+(x) \cap \Gamma^+(y)|, \quad \text{CNo;i}(x,y) = |\Gamma^+(x) \cap \Gamma^-(y)|,$$
$$\text{CNi;o}(x,y) = |\Gamma^-(x) \cap \Gamma^+(y)|, \quad \text{CNi;i}(x,y) = |\Gamma^-(x) \cap \Gamma^-(y)|,$$

where $\Gamma^+(x)$ denotes the set of successors (out-neighbors) of node x and $\Gamma^-(x)$ denotes the set of predecessors (in-neighbors) of node x.

Jaccard Coefficient (JC): Similarly, we define four variations of the Jaccard Coefficient:

$$\text{JCo;o}(x,y) = \frac{|\Gamma^+(x) \cap \Gamma^+(y)|}{|\Gamma^+(x) \cup \Gamma^+(y)|}, \quad \text{JCo;i}(x,y) = \frac{|\Gamma^+(x) \cap \Gamma^-(y)|}{|\Gamma^+(x) \cup \Gamma^-(y)|},$$
$$\text{JCi;o}(x,y) = \frac{|\Gamma^-(x) \cap \Gamma^+(y)|}{|\Gamma^-(x) \cup \Gamma^+(y)|}, \quad \text{JCi;i}(x,y) = \frac{|\Gamma^-(x) \cap \Gamma^-(y)|}{|\Gamma^-(x) \cup \Gamma^-(y)|}.$$

Preferential Attachment (PA): We extend Preferential Attachment to account for directed graphs:

$$\text{PAo;o}(x,y) = |\Gamma^+(x)| \cdot |\Gamma^+(y)|, \quad \text{PAo;i}(x,y) = |\Gamma^+(x)| \cdot |\Gamma^-(y)|,$$
$$\text{PAi;o}(x,y) = |\Gamma^-(x)| \cdot |\Gamma^+(y)|, \quad \text{PAi;i}(x,y) = |\Gamma^-(x)| \cdot |\Gamma^-(y)|.$$

Adamic-Adar Index (AA): For the Adamic-Adar Index, we consider only the common neighbors and their respective degrees:

$$\text{AA}(x,y) = \sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{\log(|\Gamma(z)|)},$$

where $\Gamma(x)$ denotes the set of neighbors of node x in an undirected sense for the common neighbors.

Resource Allocation (RA): Resource Allocation is similarly defined using common neighbors:

$$\mathrm{RA}(x,y) = \sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{|\Gamma(z)|}.$$

Clustering Coefficient (CC): The Clustering Coefficient is extended to account for the signed and weighted nature of the graph:

$$\mathrm{CC}(x) = \frac{2\cdot \text{number of triangles involving } x}{\text{number of connected triples centered on } x},$$

where a triangle involving x is considered only if the product of the weights of the edges in the triangle is positive.

Each of these metrics provides valuable insights into the network's structure and is utilized in our Alternating Least Squares (ALS) algorithm for link prediction and weight estimation in the directed, weighted graph.

3.3.2 Deep Learning

3.3.2.1 Model R

In addition to using traditional methods for link prediction, we also explore a deep learning approach Model R inspired by the research of Yuchen Hou and Lawrence B. Holder[4], which is particularly suitable for weighted directed graphs. The key aspect that differentiates Model R from other methods is its use of a fully connected neural network to directly map node pairs to link weights, leveraging the flexibility and power of deep learning.

A crucial process in this approach is representing nodes as vectors, as vectors can effectively capture and represent complex relationships and similarities between nodes. Unlike the popular skip-gram model used in Word2Vec[13], which deals with unstructured, implicit relations in natural language, and traditional Node2Vec[14], which focuses on preserving node proximity and structural roles through random walks, Model R learns vector representations directly using link weights as the supervisory signal. Based on this idea, we build a neural network model using node pairs as input and the weight of the link connecting the nodes as the output. The specific Deep Neural Network Architecture is as follows:

Model R Architecture

- **Mapping Layer:** A node dictionary is defined, mapping node IDs to node vectors. The goal is to find the best dictionary mapping that reflects the relations between node pairs and link weights through training.

- **Input Layer:** When inputting the node pair IDs, the corresponding two vectors are activated and fed into the neural network.

- **Multiple Hidden Layers:** One of the important features of deep neural networks is the multiple hidden layers to extract abstract information. In this context, multiple hidden layers are used to extract abstract weight-relevant information. The ReLU activation function mitigates the risk of gradient vanishing and explosion.

- **Output Layer:** The output layer uses a linear activation function to process the results from hidden layer units. The output is used to predict the link weight.

Like traditional deep neural network training methods, the model is trained using backpropagation, propagating the error gradients layer by layer. To accelerate and smooth the descent by minimizing error, this method employs mini-batch training with stochastic gradient descent. To reduce overfitting, early stopping is used as a regularization technique. If the training error decreases while the validation error stops decreasing or starts increasing, the training process is terminated.

For this neural network architecture, the key hyperparameters are chosen as follows: To eliminate the gradient vanishing problem and accelerate computation, ReLU is chosen as the activation function for the hidden layers. The size of the data and the layer size are positively correlated, and the layer size is defined by the formula $d = \log_2(n)$. The more complex the relationships between node pairs and link weights, the more layers are needed. The initial number of layers is set to 4, and it will be adjusted based on model performance.

3.3.2.2 Transformers

The inspiration to explore Transformers for link weight prediction stems from the success of Model R in capturing complex node relationships using deep learning. Model R demonstrated that node embeddings and fully connected networks could effectively predict link weights by learning intricate patterns in the data. The fundamental similarity between Model R and Transformers in mapping node IDs to vectors and leveraging these embeddings for prediction provided a strong motivation to explore Transformers. This structural similarity, combined with the advanced capabilities of Transformers, made them a promising candidate for improving upon Model R's performance.

Transformers are a type of neural network architecture that utilizes self-attention mechanisms to process input data[15]. This mechanism allows them to dynamically focus on the most relevant parts of the input data and capture complex relationships within it. Additionally, the parallel processing capabilities of Transformers make them suitable for handling large datasets, leading to faster training times and more accurate predictions. Initially designed for natural language processing tasks, Transformers have proven to be highly effective in various domains. Similarly, in graph-based tasks, we hypothesize that Transformers can capture complex relationships between nodes more effectively than traditional methods and accurately predict link weights. This hypothesis will be validated in subsequent experiments.

Transformer Architecture

- **Input:** Pairs of nodes are used as input, with each node represented by an embedding vector. Optionally, positional encodings can be added to provide additional information about the order or position of nodes within the graph.
- **Core Architecture:** The model architecture consists of initial embedding layers, followed by multiple Transformer encoder layers equipped with multi-head self-attention mechanisms. These encoder layers are designed to refine the node embeddings by dynamically integrating information from other nodes in the graph. The final component of the architecture is a prediction head that outputs the predicted link weight for each pair of nodes.
- **Output:** The model's output is the predicted link weight for each input pair of nodes, which is compared to the actual link weight.

The training process uses Mean Squared Error (MSE) as the loss function to measure prediction accuracy, employs the Adam optimizer to update model parameters by minimizing the loss, and incorporates periodic validation with early stopping to prevent overfitting.

Several hyperparameters can be adjusted to optimize the model's performance, including the embedding dimension(typically 64, 128, or 256), number of attention heads (commonly 4, 8, or 16), number of Transformer encoder layers(usually 2, 4, or 6), feed-forward dimension(often 128, 256, or 512), and dropout rate(commonly 0.1, 0.2, or 0.3). These adjustments influence the model's ability to capture node features, focus on various data aspects, and prevent overfitting.

4 Results and Findings

4.1 Alternating Least Squares

4.1.1 Experimental Setup

The experiment of ALS was conducted on a local PC (lightweight consumer-grade office laptop) with the following configuration, only CPUs used during runtime.

Hardware	Model	Software	Version
CPU	16x AMD Ryzen 7 5800H	Windows 11 Home	64-bit (10.0.22631.3593)
GPU	AMD Radeon(TM) Graphics	Python	3.11.6
RAM	2x 8GiB Samsung RAM	sklearn	0.0.post11
		networkx	3.3

Table 1: Configuration Table for ALS Algorithm Environment

4.1.2 Results

ALS: Average MSE is 0.125. It should be noted that in the scatter plot, since most of the predicted points that match the true value overlap, it does not look obvious, but the actual prediction accuracy is about 70%. The same is true in the subsequent deep learning methods.

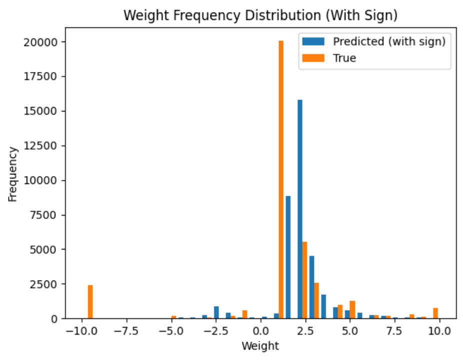

Figure 1: Distribution of Prediction Errors

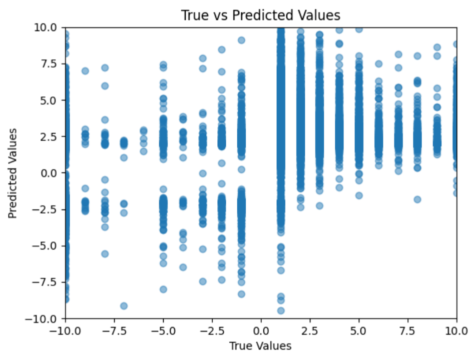

Figure 2: Actual vs Predicted Link Weights

4.1.3 Analysis

Overall, this approach correctly predicted the edge trust weight for about 70% of the edges. As shown here, the ALS method does well on trust edges near the peak (the mode of the ratings) and gives a good enough baseline performance with just limited CPU computing resources. The program only uses a short total core time when running, which is consistent with the **linear time complexity** of the algorithm. In addition, we use the **BLAS library to parallelize the main body of computing**, finally showing great scalability.

4.2 Deep Learning

4.2.1 Experimental Setup

The deep learning experiment was conducted on Google Colab, using its GPU capabilities for accelerated training. The following libraries and versions were used:

Software	Version
Python	3.10.12
PyTorch	2.3.0+cu121
Scikit-Learn	1.2.2
Matplotlib	3.7.1

Table 2: Libraries and Versions Used for Deep Learning Experiment

To evaluate the performance of our model and ensure robustness and reliability, we conducted 25 independent trials with the following procedure:

Mounting Google Drive To access the dataset stored on Google Drive.
Model Training Defined the model R architecture and trained it using the specified configurations:
Learning Rate: 0.001, Batch Size: 32, Layer sizes: 14, Hidden layers: 3, Epoch: 20
During training, the embeddings for 'SOURCE' and 'TARGET' were learned, where a node-to-vector mapping was supervised by the link weight. This process resulted in a node dictionary that included each node ID and its corresponding vector representation. The source and target vectors were combined to predict the link weight. The best model was saved based on validation loss.
Evaluation The best model from the training phase was loaded and evaluated on the test set. We calculated the Mean Squared Error (MSE) as the metric for prediction accuracy. Additionally, we visualized the results by plotting actual vs. predicted link weights and the distribution of prediction errors.
Hyperparameter Tuning We varied the layer size and number of hidden layers and repeated the experiments to ensure the robustness and reliability of our results.

Inspired by how the node-to-vector mapping method is similar to word2vec and image2vec, we conducted experiments on the dataset using the Transformer model following the steps above as well.

4.2.2 Results

model R: Average MSE is 0.1026.

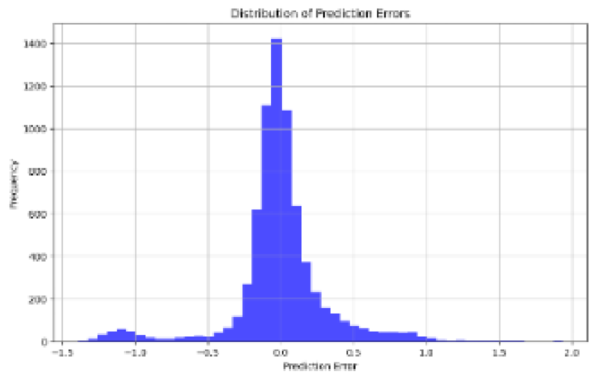

Figure 3: Distribution of Prediction Errors

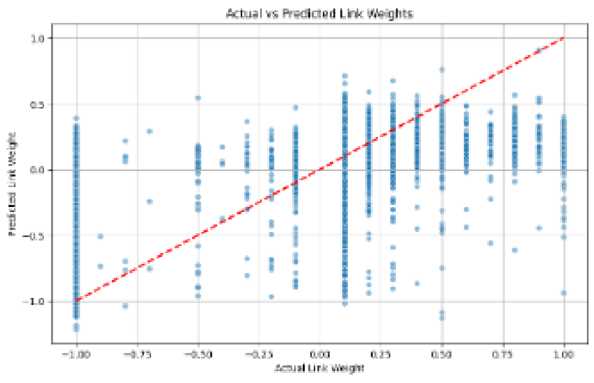

Figure 4: Actual vs Predicted Link Weights

Transformer: Average MSE is 0.0928.

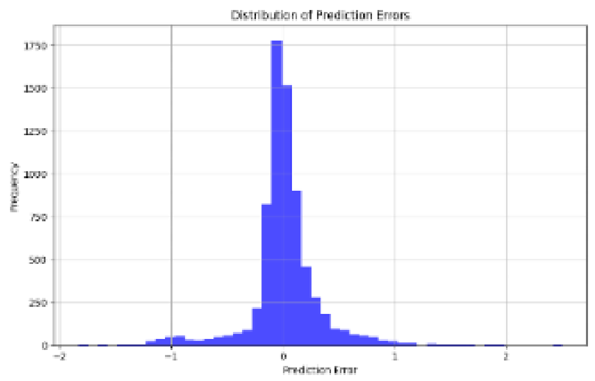

Figure 5: Distribution of Prediction Errors

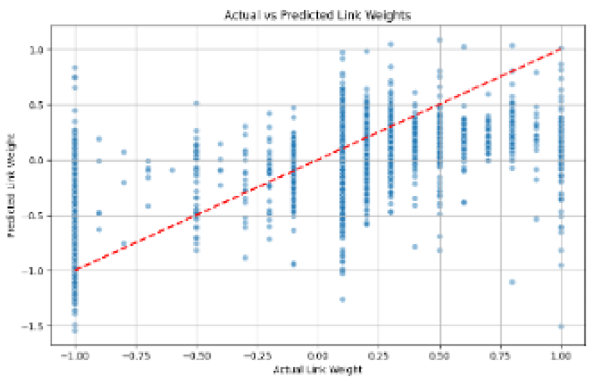

Figure 6: Actual vs Predicted Link Weights

4.2.3 Analysis

Since embeddings were used to convert the nodes into continuous vector representations, which served as inputs, both model R and Transformer could learn complex patterns and relationships between nodes. As shown in Figure 3 and Figure 4, the bar charts of prediction errors are concentrated around 0 and the distributions are quite narrow. On the other hand, as shown by Figure 5 and Figure 6, the scatter plots of the distribution of actual versus predicted values, containing a red line where y=x, indicates that the data points are closely clustered around the baseline. This indicates high accuracy in the predictions.

With the MSE at 0.1026 and 0.0928, it is evident that deep learning methods for link weight prediction are highly accurate and well-performing.

4.3 Comparison

As shown in the above graphs and results, the deep learning methods trained on GPU achieve more accurate results and better performance compared to the Alternating Least Squares (ALS) method. The prediction errors for deep learning methods are more concentrated around 0, indicating higher accuracy and fewer large errors. Additionally, the scatter plots of actual versus predicted values show that the points are more closely clustered around the y=x line, suggesting better prediction alignment with the true values. Table 3 presents the comparison of Mean Squared Error (MSE) among these models.

Model	MSE
ALS	0.1250
model R	0.1026
Transformer	0.0928

Table 3: Comparison of Mean Squared Error (MSE) Among Models

5 Discussion

In this section, we discuss the findings, acknowledge the limitations of our research, and suggest directions for future work.

Findings of Results ALS method has the advantage of being relatively computationally efficient, parallelizable, scalable and less resource-intensive compared to deep learning methods despite its lower prediction accuracy and difficulties with more complex datasets. On the other hand, deep learning methods such as model R and Transformer can capture complex, non-linear relationships in the data. However, deep learning models come with their own set of challenges. They are computationally expensive, requiring significant resources such as powerful GPUs and longer training times. Furthermore, these models are more complex to implement and require a higher level of expertise in machine learning and neural networks.

Limitations and Challenges The Bitcoin OTC dataset has a predominance of positive ratings (89% positive edges), which may lead to biased model performance favouring positive predictions, though the distribution accorded with the conclusion of human nature from the Balance Theory and Status Theory from Social Psychology[16]. Additionally, the model's ability to generalize to other trust-based networks remains untested, and further validation on diverse datasets is needed to ensure the robustness of the model. While deep learning methods achieved a lower MSE of 0.0928 compared to the ALS method with an MSE of 0.125, the difference in performance is relatively small. Deep learning models are computationally expensive and require significant resources. In contrast, the ALS method, being less computationally intensive, may be sufficient for practical purposes in the context of the Bitcoin OTC dataset.

Future Work Addressing the data imbalance issue by implementing techniques such as SMOTE (Synthetic Minority Over-sampling Technique) or GANs (Generative Adversarial Networks) to generate synthetic data and balance the dataset would be one of our future directions. Moreover, enhancing model scalability to handle larger, more complex networks with additional features is crucial. We should also consider a detailed cost-benefit analysis comparing the methods to balance specific use-case requirements and available resources.

6 Conclusion

The study compared two methodologies for predicting trust dynamics within the Bitcoin Over-The-Counter (OTC) network: the Alternating Least Squares (ALS) algorithm and deep learning models (Model R and Transformers). Our results indicated that deep learning models outperform ALS in terms of prediction accuracy, with Mean Squared Error (MSE) values of 0.1026 for Model R and 0.0928 for Transformers, compared to 0.1250 for ALS. While deep learning models provide higher accuracy, they are computationally expensive and require significant resources. In contrast, ALS is more computationally efficient and suitable for environments with limited resources.
Future research should address data imbalances and test the models' generalizability to other networks. For the next steps, we would consider techniques to balance the dataset and enhance model scalability with more features. This research provides a foundation for improving security and reliability in decentralized trading environments.

Individual Contribution

Zhiqiang Yu is responsible for theoretical research, literature research, revising research papers to confirm the research direction, and verifying the accuracy and effectiveness of deep learning methods in collaboration with Ziyun.
Ziyun Pan applied the deep learning methods of model R and Transformer to predict the link weights after data preprocessing and dataset splitting.
Changrong Li coordinated the project and took charge of literature research and theoretical derivation, development, deployment, result analysis, performance optimization of ALS algorithm, writing corresponding chapters and appendix of the report.
Manushree Tyagi is responsible for writing an initial proposal, doing theoretical research and working on ALS with Changrong Li. Also, wrote various sections for the report.

References

[1] Investopedia, "Mt. gox," 2023, accessed: 2024-06-07. [Online]. Available: https://www.investopedia.com/terms/m/mt-gox.asp#:~:text=Mt.%20Gox%20once%20accounted%20for%20over%2070%25%20of,Gox%20rehabilitation%20plan%2C%20closing%20a%20seven-and-a-half-year%20legal%20battle.

[2] BBC News, "Bitcoin's mt gox exchange goes offline," 2014, accessed: 2024-06-07. [Online]. Available: https://www.bbc.com/news/technology-24371894

[3] J. Liang and C. Pu, "Line graph neural networks for link weight prediction," 2020.

[4] Y. Hou and L. B. Holder, "Deep learning approach to link weight prediction," 2020.

[5] S. Kumar, F. Spezzano, V. Subrahmanian, and C. Faloutsos, "Edge weight prediction in weighted signed networks," 2020.

[6] S. Shi, T. Yang, and Z. Bie, "Link weight prediction in signed networks," 2020.

[7] O. Tanevski, I. Mishkovski, and M. Mirchev, "Link prediction on bitcoin otc network," 2020.

[8] H. R. Atiya and H. N. Nawaf, "Prediction of link weight of bitcoin network by leveraging the community structure," *IOP Conference Series: Materials Science and Engineering*, 2020.

[9] M. Hasan, V. Chaoji, S. Salem, and M. J. Zaki, "Link prediction using supervised learning," 2006.

[10] H. Rodrigues de Sa and R. B. C. Prudencio, "Supervised link prediction in weighted networks," 2020.

[11] B. Zhu, Y. Xia, and X.-J. Zhang, "Weight prediction in complex networks based on neighbor set," 2020.

[12] G. Takács and D. Tikk, "Alternating least squares for personalized ranking," in *Proceedings of the sixth ACM conference on Recommender systems*, 2012, pp. 83–90.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[14] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016. doi: 10.1145/2939672.2939754 pp. 855–864.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[16] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Signed networks in social media," in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2010, pp. 1361–1370.

A Time Complexity Analysis of the Alternating Least Squares (ALS) Algorithm

A.1 Introduction

The Alternating Least Squares (ALS) algorithm is a popular technique for matrix factorization, often used in collaborative filtering for recommendation systems. It decomposes a user-item interaction matrix R into two lower-rank matrices U (users' latent factors) and V (items' latent factors). The primary goal is to minimize the reconstruction error, typically measured by the Frobenius norm:

$$\min_{U,V} \|R - UV^T\|_F^2 \tag{1}$$

ALS alternates between fixing U and solving for V, and vice versa. This section delves into the time complexity of this algorithm, analyzing the computational costs at each step.

A.2 ALS Algorithm Overview

The ALS algorithm iteratively performs the following steps until convergence:

1. Fix U and solve for V.
2. Fix V and solve for U.

For a given user-item matrix R of size $m \times n$ and a chosen latent factor dimension k, the steps involve solving a series of least squares problems.

A.3 Time Complexity per Iteration

Let's denote the number of observed ratings in R as N. The time complexity analysis for each step in a single iteration is as follows:

A.3.1 Fixing U and Solving for V

To solve for V when U is fixed, we perform the following operations:

- Construct the normal equation for each item j:

$$V_j = (U^T W_j U + \lambda I)^{-1} U^T W_j R_j \tag{2}$$

  Here, W_j is a diagonal matrix indicating the presence of ratings for item j, and λ is the regularization parameter.

- The cost of forming $U^T W_j U$ is $O(k^2 d_j)$, where d_j is the number of non-zero entries in column j of R.

- The inversion of the $k \times k$ matrix $(U^T W_j U + \lambda I)$ is $O(k^3)$.

- The multiplication $(U^T W_j R_j)$ costs $O(k d_j)$.

  Summing these costs over all items, the complexity becomes:

$$O\left(\sum_{j=1}^{n} (k^2 d_j + k^3)\right) = O(k^2 N + k^3 n) \tag{3}$$

A.3.2 Fixing V and Solving for U

Similarly, to solve for U when V is fixed, we perform:

- Construct the normal equation for each user i:

  $$U_i = (V^T W_i V + \lambda I)^{-1} V^T W_i R_i \tag{4}$$

  Here, W_i is a diagonal matrix indicating the presence of ratings for user i.

  The cost of forming $V^T W_i V$ is $O(k^2 d_i)$, where d_i is the number of non-zero entries in row i of R.

- The inversion of the $k \times k$ matrix $(V^T W_i V + \lambda I)$ is $O(k^3)$.

- The multiplication $(V^T W_i R_i)$ costs $O(kd_i)$.

  Summing these costs over all users, the complexity becomes:

  $$O\left(\sum_{i=1}^{m}(k^2 d_i + k^3)\right) = O(k^2 N + k^3 m) \tag{5}$$

A.4 Overall Time Complexity

The time complexity for one complete iteration of ALS, which includes updating both U and V, is the sum of the complexities for fixing U and solving for V, and vice versa. Hence, the overall complexity per iteration is:

$$O(k^2 N + k^3 n + k^2 N + k^3 m) = O(k^2 N + k^3 (m+n)) \tag{6}$$

Assuming $m \approx n \approx \sqrt{N}$, the complexity simplifies to:

$$O(k^2 N + k^3 \sqrt{N}) \tag{7}$$

A.5 Conclusion

In summary, the ALS algorithm's time complexity per iteration is $O(k^2 N + k^3 (m+n))$, which, under the assumption $m \approx n \approx \sqrt{N}$, further simplifies to $O(k^2 N + k^3 \sqrt{N})$. This analysis highlights the efficiency of ALS in handling sparse matrices, making it a suitable choice for large-scale recommendation systems where N is much larger than k, m, and n.