Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

NETS

Lab Report

Using token models in graph networks

Trisrota Bose

Julius-Maximilians-Universität Würzburg
Center for Artificial Intelligence and Data Science
Lehrstuhl für Informatik - Machine Learning for Complex Networks

Abstract

Many data follows the graph structure, like social network or hierarchies however it comes with challenges like scalability and long range dependencies. However token based models provide an alternative to this. In this report we try to use the BERT model to effectively learn from random walks over graphs. We investigate two research questions:

(1) Can a Transformer encoder predict the label of the next node in a random walk given the sequence of previous node labels?

(2) Can a sequence-to-sequence Transformer predict the labels of subsequent nodes when conditioned on prior labels and structural features such as centralities?

In this report, we implement two models: a BERT style encoder (**BertEncoder**) and a sequence-to-sequence Transformer (**Seq2SeqBERT**).

Zachary club dataset is used for both the models.

Our results indicate that the **BERTEncoder** model performs well by incorporating the transformer model into the graph structure for moderately long graphs but lacks the structural information needed for more complex graphs.

The **Seq2SeqBERT** incorporates centrality measures for structural information that improves the token accuracy but struggles with longer sequences because the errors accumulate, and the unpredictable nature of random walks makes future paths uncertain. We conclude that transformer model with structural information can complement short traditional graph structures but it fails with longer graphs and global dependencies.

1 Introduction

Graph Neural network is not scalable, but token model is. So we can use token model for the Graph Neural Network.

In Graph Neural Network, the focus is on the system as a whole and not the isolated nodes. Through iterative message passing GNNs aggregate and transform information from immediate neighboring nodes, i.e. it is good with the local dependencies (like 2-3 layers of nodes). However when the network grows, GNN has to stack several layers. This leads to **over-smoothing** [4], which is that on constant aggregation the feature values become nearly identical and **vanishing gradients** [8] as depth increases, which means as the depth increases the loss gradient becomes negligible and layer stops learning .

Transformers [12] was a game changer in Natural Language Processing where all sequence positions could be processed simultaneously. It also used attention mechanism to have direct connections between any two inputs and preventing information decay over longer distances. BERT [2] is a bidirectional transformer which uses the context from both sides.

1.1 Motivation of the problem

GNN has scalability issues, however token models (like BERT) use attention algorithm which provides a more global context and , each token (or node) directly computes relationships with all other tokens in the sequence in a single layer thus making it more scalable. Transformers can solve the problems that GNN has. So in this report we try to find if transformers can sucessfully learn graph dynamics.

1.2 Structure of the problem

BERTEncoder takes the labels of the previous nodes as an input and outputs the label of the next node in a random walk while having BERT as an encoder.

Seq2SeqBERT takes labels of the previous nodes along with their centrality features to output the label of the next node while using BERT as an encoder and Transformer as a decoder.

We investigate two research questions:

1. **RQ1:** In a random walk can we predict the label of the next node given the sequence of previous nodes?
2. **RQ2:** In a random walk can we predict the labels of all the next nodes given the labels of previous nodes and centrality features?

1.3 Structure of the Work:

Chapter 2 is the declaration of Generative AI

Chapter 3 is Background.

Chapter 4 is Methodology.

Chapter 5 is Results and analysis.

Chapter 6 is discussion.

Chapter 7 is conclusion.

Chapter 8 is Limitations.

Chapter 9 is Future directions.

2 Use of Generative AI

Generative AI has been used with the following:

1. References (Organising and formatting references)
2. Background section (Drafting and structuring texts)
3. Framing the research questions (Brainstorm and refine the questions)
4. Making the language in the report sound more formal
5. Analysing the results
6. Structuring of the report and research for it
7. Literature review

All AI-generated content has been carefully reviewed, validated and, where necessary, edited by the author to ensure accuracy, technical correctness, and coherence. The final submission reflects the author's full understanding of the material and responsibility for the content presented.

3 Background

This section discusses the important concepts for the experiment.

3.1 Graph neural Network

Graphs are based on two things, entities (nodes) and their relationship with each other (nodes). GNNs aggregate information from the neighbouring nodes iteratively and keeps on updating themselves.

3.2 Random Walks for Graph Sampling

A random walk is a fundamental tool for sampling a graph's structure. The walk starts at a node and traverses the graph by randomly selecting a neighbor at each step till it reaches termination. Methods like DeepWalk [10] use these walks to learn node embeddings by treating them as sentences in a language model.

3.3 Transformer Architecture and BERT

The **Transformer** architecture [12] was an important step in NLP by introducing attention and further self attention. Self-attention allows the model to weigh the importance of all elements in a sequence when processing each element and then checking which element is most related to which element.

BERT (Bidirectional Encoder Representations from Transformers) [2] is a Transformer-based model designed to pre-train deep bidirectional representations from unlabeled text. Its key innovation is the Masked Language Model (MLM) pre-training objective, which randomly masks tokens in the input sequence and trains the model to predict them based on their bidirectional context. This was a groundbreaking invention because before this only singular context was taken. Taking bidirectional context made this more accurate.

Our first model BERTEncoder uses Bert as an encoder to predict the next label.

3.4 Sequence-to-Sequence Learning

Sequence-to-Sequence (Seq2Seq) [11] learning involves transforming an input sequence into an output sequence. It is commonly used for tasks like machine translation and text summarization. A standard Seq2Seq model consists of an **encoder** that processes the input sequence and compresses it into a context vector, and a **decoder** that autoregressively generates the output sequence one token at a time, conditioned on the context vector and its previous outputs.

Our second model (**Seq2SeqBERT**) brings this concept to graphs. The encoder is fed a sequence of node features and prior node label, and the decoder outputs the corresponding sequence of node labels, learning the complex mapping from a node's structural location to its community affiliation.

3.5 Centrality Measures

[7] Centrality measures are used to numerically represent each node's structural importance in the graph. Here 6 centrality measures are used. **Degree centrality** measures the number of immediate connections and thus the local connectivity. **Betweeness** measures the frequency of the node appearing on the shortest path, thus identifying nodes which connects various paths. **Closeness centrality** measures how close a certain node is to other nodes. **Eigenvector** measures the importance of a node compared to other important nodes. **PageRank** assigns a numerical score to a node based on the quantity and quality of links pointing to it, with links from more important nodes counting more. **Community** is any centrality measure that evaluates a node's importance based on its relationship to the community structure of the network, rather than just its individual connections.

Since transformers do not have a adjacency matrix, they rely on embeddings and centrality measures help them determine which nodes are more globally important and which one more locally to understand their influence on other nodes.

4 Methodology

4.1 Dataset

In these experiments Zachary Karate Club dataset is being used. This dataset [13] was used to map the friendship between two members of the Karate club over the period of 1970-1972. This social network has **34 members (nodes) and 78 edges**. Following a dispute, the club divides into two groups **("Mr. Hi" (class 0) and "Officer" (class 1))** and these serve as our target class. To convert these categorical string labels into a numerical format that works for machine learning, we use a LabelEncoder from scikit-learn [9]. This changes the two club names into integer values of 0 and 1. Since transformers work on words, we converted the labels into sequential data by using random walks. So instead of individual labels like 0,1,1,0, we get a sequence (0110.....). These walks serve as graph sentences, where each node corresponds to a token.

4.1.1 Data Preparation for Model 1 (BertEncoder):

Each node is replaced by the label. Then the categorical node labels were encoded into integers using one hot encoding. Multiple random walks were generated from each node. For each walk we apply a sliding window of seq_length to generate input sequences and immediate next step serving as the target label. Data is divided into 80/20 for train and test. Inputs are converted into tensor format for transformer processing. One-hot encoding ensures labels are treated as independent categorical tokens like to words in a sentence, with no implicit numerical ordering.

Total walks = num walks $\times$ number of nodes

4.1.2 Data Preparation for Model 2 (Seq2SeqBERT):

For each node we calculate a feature vector containing the centrality measures like (**Degree, Betweeness, Closeness, Eigenvector, Pagerank, Community**). Each data sample is a pair: a sequence of S feature vectors (the input) and the corresponding sequence of S node labels (the target output). All numeric features were normalized to a [0, 1] range. Sequences were padded/truncated to fixed length. Data is divided into training, validation and test set (70/15/15). All sequences are converted to tensors suitable for a Transformer.

4.2 Model Architecture

4.2.1 Architecture of Model 1: (BertEncoder)

This model frames the question as: Given a sequence of S node labels, what is the next node's label?

This model inputs each node label is a one hot vector. A sequence of these one hot vectors make the input of the model.

The model consists of three primary components that work in sequence to transform input label sequences into class predictions. One hot vector has only (0,1) but BERT expects 768 dimensional numbers so we need projection layer to transform accordingly. As Transformers are permutation-invariant, positional encodings are added to the projected label embeddings to preserve the order of tokens within each random walk.

The second component is the BERT Encoder itself. It has 4 attention heads, this helps in learning about the specific patterns in how labels relate to each other. It further has 2 hidden layers, where each hidden layer has a multi head attention and feed forward network with residual connections and layer normalization applied throughout.

BERT produces the output [batch_size, seq_length, 768], where for each sequence of batch, and for each position in that sequence there is a 768 dimensional vector that determines what the label means in that specific context.

The third component is the classification head, which now maps the contextualized BERT representations to class predictions for the next label. This is implemented as a simple linear layer (nn.Linear) that transforms from the 768-dimensional BERT hidden space to the number of output classes (2 in this case). So now the model takes the first position's representation with all the context and predicts the next node label.

The output logits are transformed via softmax to produce a probability distribution over the two possible community labels.

For optimisation, cross-entropy loss is used which penalises the model for wrong predictions. AdamW [5] is preferred over traditional Adam due to its decoupled weight decay formulation, which improves generalization and stabilizes convergence for optimisation.

The model is trained for 10 epochs. Throughout training, the total loss is calculated across all batches, and the average loss per batch is computed and printed at the end of each epoch.

Standard metrics are calculated like Accuracy, Precision, Recall and F1-score. Further we systematically vary the num_walks and walk_length parameters across predefined ranges

to understand their impact on model performance. Number of walks varies from 5-75 increasing by 10 and walk length increasing by 5 from 5 to 30.

4.2.2 Architecture of Model 2: (Seq2SeqBERT)

This model uses centrality features in addition to the labels of the nodes and therefore integrates graph structural information through node-level centrality features. After computing these six centrality measures for all 34 nodes, the features are assembled into a feature matrix of shape [34 nodes, 6 features], where each row represents a node and each column represents one of the computed centrality metrics. Random walks are generated as before however each position now includes the label and the centrality information. Sequences are padded and truncated to fit the transformer. Feature normalization is performed using MinMaxScaler [9] from scikit-learn so that all the features are in the scale of [0,1].

The model has 3 parts: Encoder, Decoder and classifier. The encoder begins with a source projection layer (nn.Linear) that maps the 6-dimensional input features to a hidden dimension of 128, like the model before. Following this projection, a BERT encoder is configured with a hidden size of 128 dimensions, 4 attention heads, 2 layers, and an intermediate feed-forward network size of 512. Similar to the model above, BERT produces the output [seq_length, batch_size, 128] now enriched with the information. The self-attention output is then passed through a positionwise feed-forward network with a GELU (Gaussian Error Linear Unit) activation [3], residual connections, and layer normalization. Dropout regularization (rate = 0.1) is applied throughout to prevent overfitting.

To preserve sequential information positional embeddings are added in the random walk.

For the decoder, we use a transformer model so that only the previous nodes are responsible for the current node. Since the encoder processes continous tokens (centrality features) and the decoder needs discrete tokens (0,1,BOS), the decoder begins with a token embedding layer (nn.Embedding) that maps discrete label tokens to 128-dimensional dense vectors. A special BOS (Beginning-of-Sequence) token marks the start of generation.

The transformer autoregressively generates the corresponding sequence of community labels. During training teacher forcing [1] is use which means it receives the ground-truth label sequence from the original database shifted by one position however during inference it uses its own previously generated predictions as input. Transformer also has positional embeddings like the encoder to preserve the structure.

Each decoder layer has masked self-attention ensures that predictions at a certain time step can only depend on tokens from previous time steps, maintaining the causal structure

of the sequence.

The decoder further includes a cross-attention [12] mechanism that allows it to attend over the encoder outputs.

The decoder's final hidden states are passed through a linear projection layer that maps the 128-dimensional hidden representation to two output logits, one for each possible label. Applying the softmax function to these logits yields class probabilities for every position in the output sequence, just like we did in the first model).

The model is trained to minimize Weighted Cross-Entropy Loss, where class weights are set inversely proportional to class frequencies to compensate for possible imbalance between the two communities and prevent bias towards the dominant class.

Optimization is performed using the AdamW [5] optimizer with a learning rate of 10^{-4} and weight decay of 0.01, and default momentum parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$). The training procedure uses a batch size of 16–32 depending on available memory and applies gradient clipping (max norm = 1.0) to stabilize learning.

The model is trained through 60 epochs with a patience of 10 epochs.

The decoder predicts the probability distribution over labels for the first position, selects the most likely label, appends it to the sequence, and repeats the process for subsequent steps. In this experiment greedy decoding (beam width=1) [6] is used, where the model selects the most probable label at each step. However beams of different widths can be used to. Since we only has 2 classes the beam width greater than 1 was not efficient. Each predicted sequence is then compared to ground value to judge the evaluation metrics.

For evaluation of this model we use Token-level accuracy and sequence-level accuracy. In addition, precision, recall, and F1-score are computed per class and averaged to assess class-specific performance.

Even for this model we vary the num_walks and walk_length parameters across predefined ranges to understand their impact on model performance. Number of walks varies from 5-75 increasing by 10 and walk length increasing by 5 from 5 to 30.

4.3 Experiments

A series of experiments were conducted to examine how the number of walks (num_walks) and length of walks (walk_length) were affecting the model. These are the two most important factors in this context because the length of walk determines the sequence of data to be predicted and number of walks determine the volume of data to be predicted.

We consider the following:

Number of walks per node: {5, 15, 25, 35, 45, 55, 65, 75}

Walk length: {5, 10, 15, 20, 25, 30}

For each combination of (num_walks, walk_length), we trained the models and evaluated it on the following metrics:

For BERTEncoder:

We contrast the **walk length** and **number of walks** with **Accuracy, F1 score, Training time, Precision, Recall** and **Final Loss** to create a heatmap.

Further we see graphs for Accuracy vs Walk length, F1 score vs Walk length, Training time vs Number of walks and Number of samples vs Accuracy (Sample efficiency).

For Seq2SeqBERT:

We contrast **walk length** and **number of walks** with **Accuracy, F1 score, Token Accuracy, Sequence Accuracy** and **Training time** to create a heatmap.

Further we see graphs of Accuracy vs Walk Length, Token and Sequence accuracy vs Walk length, Training time vs number of walks and Accuracy vs Number of samples (Sample efficiency).

5 Results and Analysis

5.1 Heatmaps for Model BERTEncoder

Heatmap 1,2,4,5: num_walks, walk_length and Accuracy, F1 Score Precision and Recall and F1Score:

Accuracy peaks around the intermediate region instead of extremes (num_walks= 25, walk_length=20) and (num_walks= 45, walk_length=10). Walks with number of walks above 55 or walk length above 20 mostly plateaus in terms of performance or performs worse. F1 performs the same way

Heatmap 3: num_walks, walk_length and Training Time

Training time grows linearly with both the parameters.

Heatmap 6: num_walks, walk_length and Final Loss

As walk length increases the model gets more context so the loss also decreases. Short sequences underfit and longer sequences overfit so the best results we get in the moderate area. We can also see that the walk length is more important for good accuracy than number of walks.

Overall Heatmap analysis

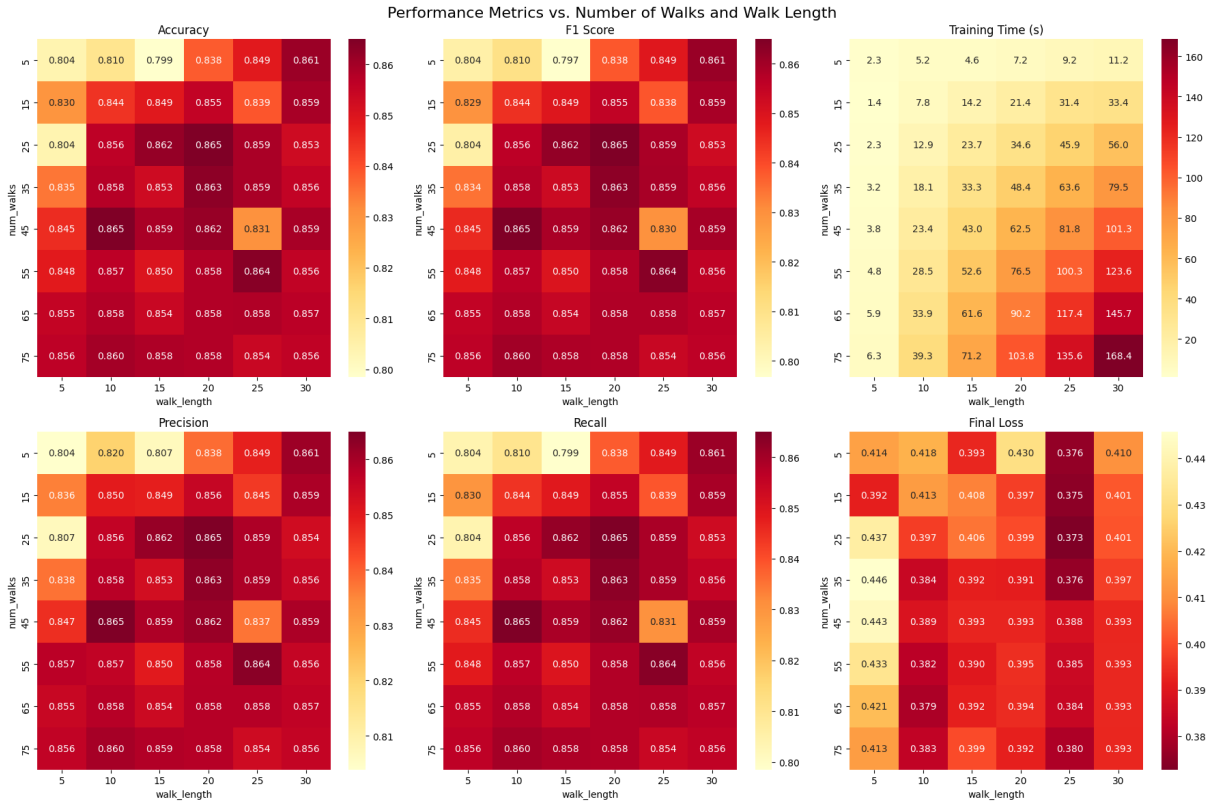

Figure 1: Heatmap for BERTEncoder

1. Performance improves with increase in walk length and number of walks upto a certain extent and then plateaus. 15-35 walks, length 10-20 (30-90 seconds, optimal performance)

2. Training loss cannot be directly correlated with accuracy.

3. More walks or higher number of walks do not automatically mean better results.

5.2 Graph for BertEncoder

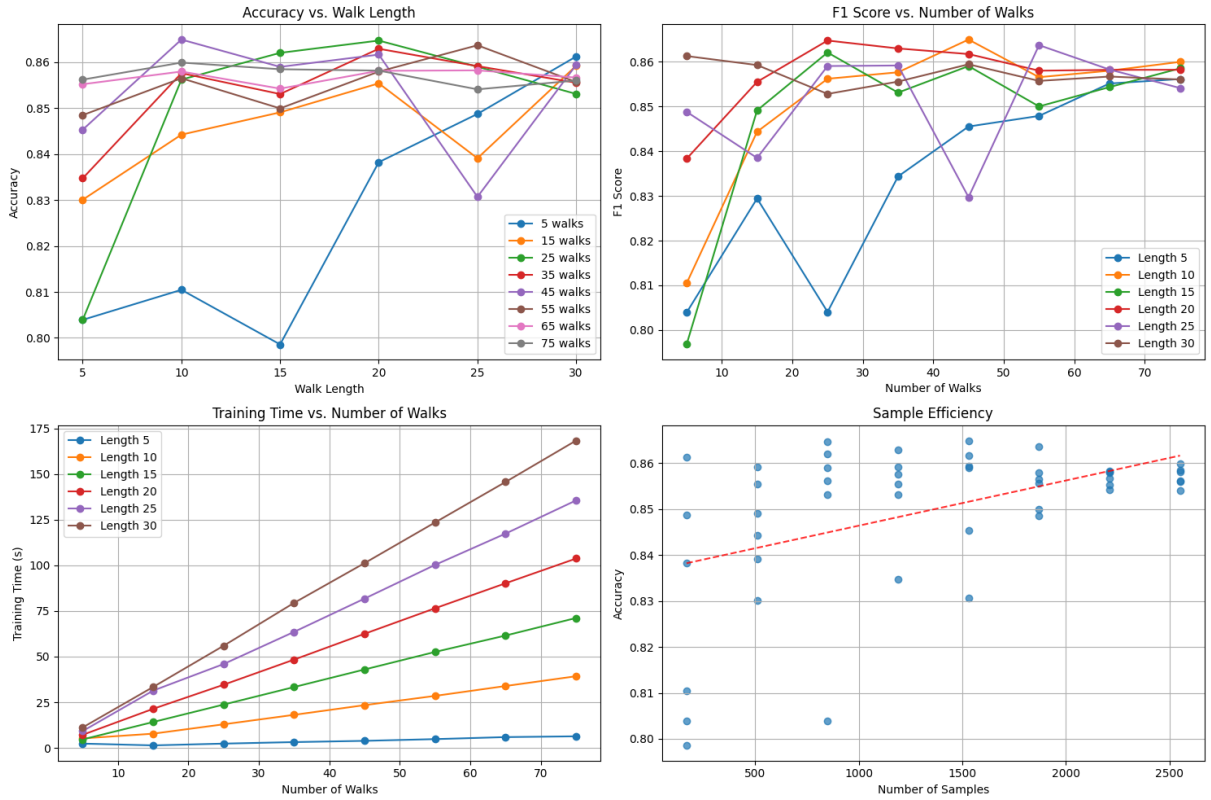

Figure 2: BERTEncoder

Graph 1: Walk Length in the graph further supports the heatmap, after a certain length the accuracy remains constant or decreases.

Graph 2: F1 score increases sharply with more walks between 5 to 35. After 35, it plateaus.

Graph 3: Training time and walk length are linearly correlated.

Graph 4: Accuracy improves with more samples. After 1500 samples it does not matter if we add more samples.

5.3 Heatmaps for Seq2SeqBERT

Heatmap 1,2 and 3 : Accuracy increases when number of walks increases specially for moderately long walks. However the accuracy decreases when walk length increases. F1 and token accuracy also follows the same pattern.

Heatmap 4: Sequence accuracy is much lower than token accuracy, while model is good at predicting individual tokens, it is bad at predicting entire sequences. Has a strong dependence on both the parameters. Works best with short walks and low number of

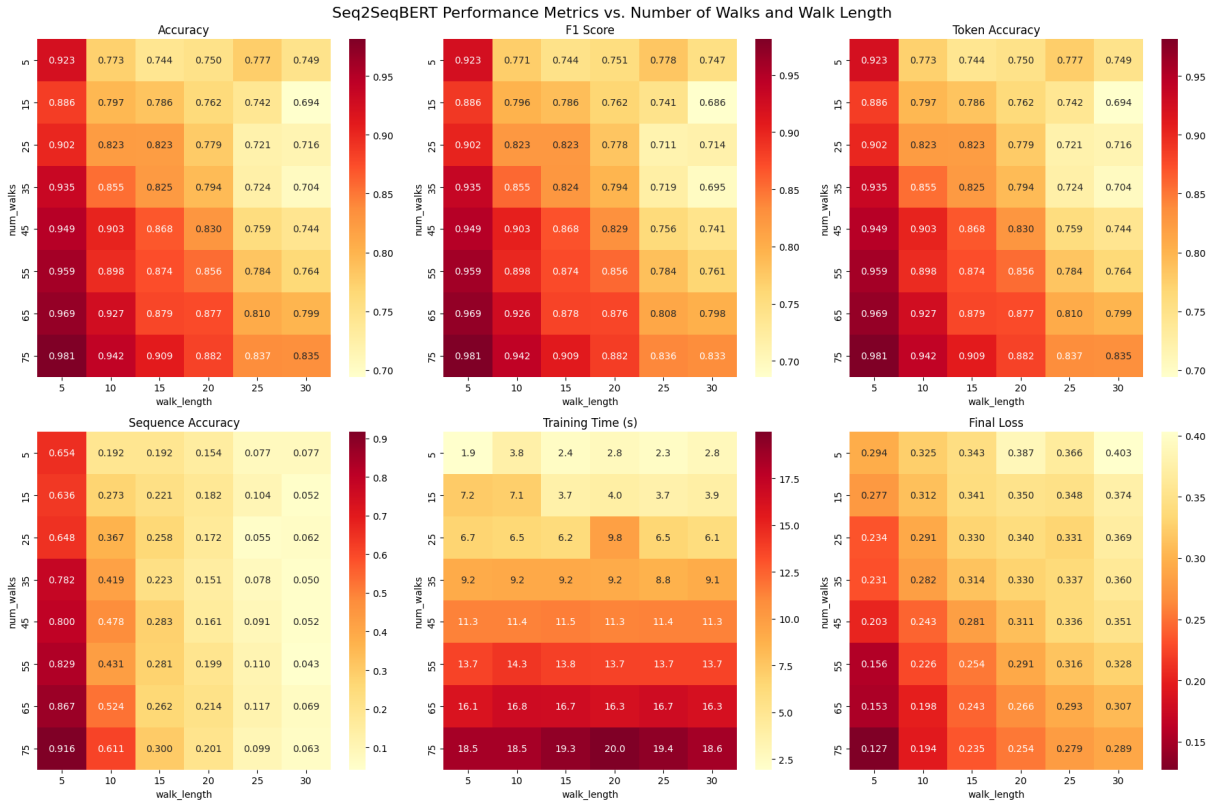

Figure 3: Seq2SeqBERT

walks.

Heatmap 5: Training time scales more with number of walks and walk length. The lowest training times are found at low walk_length and low num_walks.

Heatmap 6: Final loss decreases (lower values are better) with higher num_walks and shorter walk_length.

Overall Heatmap analysis: The optimal model performance (highest accuracy, F1, and lowest loss) is achieved with a **high number of walks and a short walk length**. Increasing number of walks is generally beneficial but is bad for training time. Increasing walk length is often not very beneficial because several parameters either plateau or decrease.

5.4 Graph for Seq2SeqBERT

Graph 1: As walk length increases, accuracy generally decreases for all numbers of walks. For every fixed walk length, higher numbers of walks consistently result in higher accuracy.

Graph 2: Token accuracy is consistently higher than sequence accuracy at all parameter settings. Both metrics decline with increasing walk length, but sequence accuracy drops much faster and to lower values than token accuracy. Higher numbers of walks improve

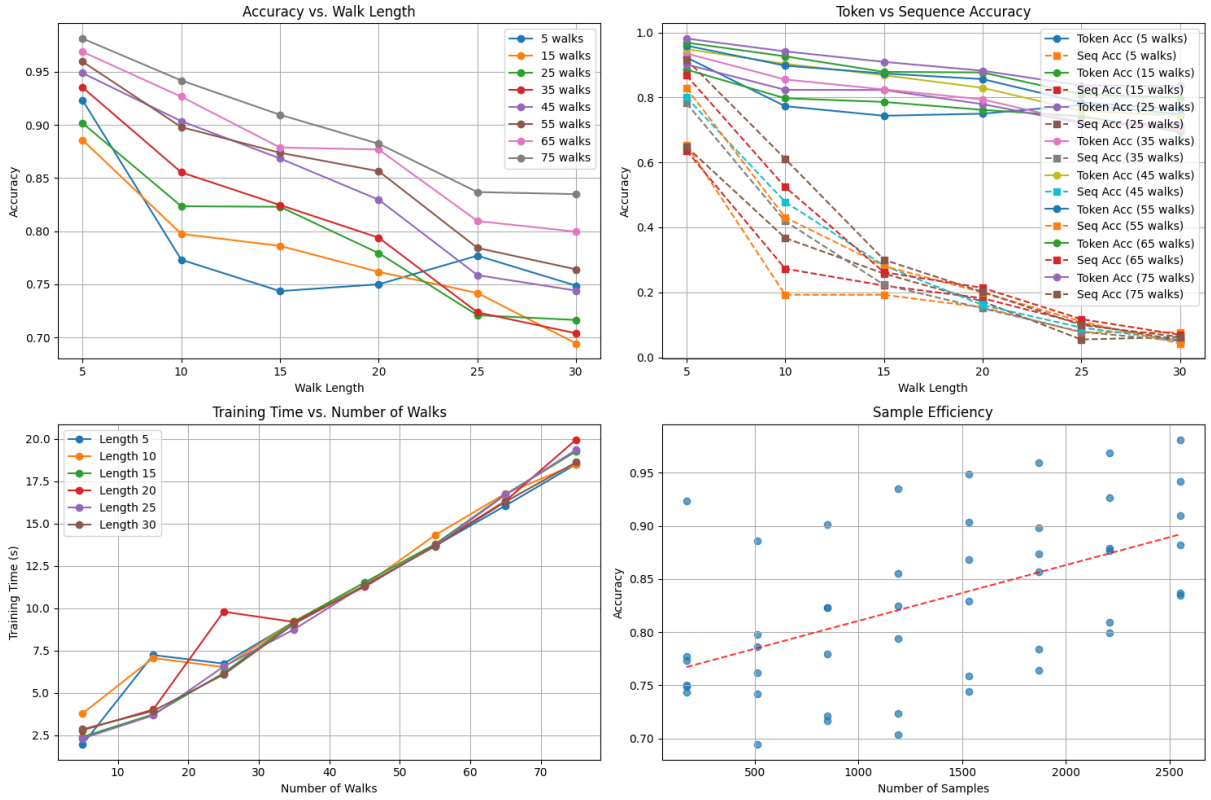

Figure 4: Graph for Seq2SeqBERT

both metrics, but sequence accuracy remains much more sensitive to increases in walk length.

Graph 3: Training time increases approximately linearly with the number of walks for all walk lengths. The curves for different walk lengths are closely grouped, suggesting that the primary determinant of time is num_walks rather than walk_length, though there is a modest separation at higher values.

Graph 4: Accuracy improves with more samples. After 2000 samples adding more walks/longer walks still increases sample count.

5.5 Key Takeaways from BertEncoder

BERTEncoder achieves best performance in moderate configuations. Lowest training loss does not guarantee highest accuracy.

5.6 Key Takeaway for Seq2SeqBERT:

This model reaches peak performance by using a large number of short random walks, which significantly boosts both token and sequence accuracy while minimizing final loss and managing computational cost. Increasing walk length tends to reduce accuracy, espe-

cially for entire sequences, and results in longer training times without clear performance benefits, so shorter walks are preferable.

6 Discussion

The goal of this study was to see if transformers can effectively work on graph data and incorporate the structural and sequential data. We are mainly working with two research questions:

RQ1: In a random walk can we predict the label of the next node given the sequence of previous nodes?

This task basically is the next node prediction in NLP. BertEncoder has shown moderately good results in this aspect. The model performs best for moderate configurations instead of extreme ones, so it can effectively work on small to medium sized graphs.

However, since the graph only has structural context, it decreases in performance or plateaus after the graph reaches a certain size.

Therefore, we can say that BERTEncoder can predict the label of the next node given the sequence of previous nodes for small to moderate graphs with satisfying accuracy.

RQ2: In a random walk can we predict the labels of all the next nodes given the labels of previous nodes and centrality features?

For this model we include the centrality features, so now we also use both the structural and sequential data of the graph. This boosts the token accuracy higher by roughly 10%, thus proving that structural data helps the model specially with nodes with similar label patterns but distinct topological positions.

Optimal region again is moderate configurations and not extreme ones. But it has higher sample efficiency, which means it reaches peak accuracy with lower number of samples as compared to the previous model.

However, the sequence accuracy can be considered low. So while the model can work well on local dependencies and predict individual nodes, it struggles with global dependencies and maintaining coherence over longer random walks. Minor token level inaccuracies creep in and keep on compounding over the sequence. This is a known fallacy of autoregressive decoding.

Furthermore, this is too stochastic a task to be performed because for random walks there actually is no deterministic mapping between a sequence of nodes to the next node. There can be multiple future sequences from a node and even a perfect model cannot predict the future nodes because the ground truth itself is probabilistic. Hence, the model fails

because the task is too random and not because the model is weak.

7 Conclusion

We test transformer based architecture, build for sequential data and language processing for graphical networks through random walk based tokenization. Using Zachary Karate Club dataset, we try to build two models analogous to the tasks of next token prediction **(BERTEncoder)** and sequence to sequence translation **(Seq2SeqBERT)**.

BERTEncoder is used to predict the label of the next node given the label of the previous nodes. This model depends only on the sequential data of the graphs. This gets around 85% accuracy and the evaluation metrics either plateau or decrease between moderately long to long walks. It fails for longer and more complex graphs because it lacks structural information.

Seq2SeqBERT is used to predict the labels of the next sequence of nodes given the previous labels and the centrality measures, which now incorporates the structural information too. This improved the token accuracy by 10% compared to BERTEncoder proving that include structural data helps the model. However, even in this case the performance peaks at medium sized graphs, thus not improving for larger graphs similar to the first model.

However the sequence accuracy is low, which shows that while the model can learn local dependencies clearly, it has a problem with learning global dependencies and maintaining sequential coherence. This can be because of two main reasons, one being that a small error in token prediction later compounds making the whole sequence incorrect. Secondly the research question in itself is too stochastic to be answered.

Concluding, it is possible to use transformer architecture to predict the label of the next node especially for small to moderately long graphs. The accuracy increases when structural features are included along with the sequential features. However the performance declines on longer and larger graphs in both the cases.

Sequence accuracy cannot be predicted accurately and therefore transformer architecture cannot be used reliably for global dependencies in a graph and over longer sequences.

8 Limitations

There are mainly two limitations to the experiments conducted. Karate club dataset is a small dataset with two classes, this limits the capacity and makes the models prone to overfitting. Further, the framing of the second research question makes it too stochastic to be accurately answered.

9 Future Directions

The models need to be tested on more data to correctly assess the ability to learn more complex dependencies. Further, instead of predicting a deterministic random walk path, the model can be made to predict the probability distribution of the possible paths. Instead of greedy search, beam search can be used to explore multiple sequences together. Finally in this report only structural and sequential features are looked into. Other features like temporal features can be looked into.

References

[1] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks, 2015. URL https://arxiv.org/abs/1506.03099.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

[3] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2016. URL https://arxiv.org/abs/1606.08415.

[4] Qimai Li, Zhichao Han, and {Xiao Ming} Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 32nd AAAI Conference on Artificial Intelligence, AAAI 2018, pages 3538–3545. AAAI press, January 2018. URL https://arxiv.org/abs/1801.07606. 32nd AAAI Conference on Artificial Intelligence, AAAI 2018 ; Conference date: 02-02-2018 Through 07-02-2018.

[5] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

[6] Clara Meister, Tim Vieira, and Ryan Cotterell. Best-first beam search. *Transactions of the Association for Computational Linguistics*, 8:795–809, 2020. doi: 10.1162/tacl_a_00346. URL https://aclanthology.org/2020.tacl-1.51/.

[7] Mark E. J. Newman. *Networks: An Introduction*. Oxford University Press, Oxford, 2010. ISBN 978-0199206650.

[8] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification, 2021. URL https://arxiv.org/abs/1905.10947.

[9] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python, 2011. URL https://arxiv.org/abs/1201.0490.

[10] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference*

on Knowledge discovery and data mining, KDD '14, page 701–710. ACM, August 2014. doi: 10.1145/2623330.2623732. URL http://dx.doi.org/10.1145/2623330.2623732.

[11] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL http://arxiv.org/abs/1409.3215.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

[13] Wayne W. Zachary. An Information Flow Model for Conflict and Fission in Small Groups. *Journal of Anthropological Research*, 33(4):452–473, 1977. doi: 10.1086/jar.33.4.3629752. URL https://doi.org/10.1086/jar.33.4.3629752.