Ministry of Higher Education
and Scientific Research
*** * ***
University of Carthage
*** * ***
National Institute of Applied Sciences and Technology

المعهد الوطني للعلوم التطبيقية و التكنولوجيا
Institut National des Sciences Appliquées et de Technologie

End of Year Project

4th Year of Computer Networks and Telecommunications Engineering

Subject:

Generalization of Transformer Networks for Graphs: State of the Art and Implementation of a Graph Transformer Architecture.

Prepared by :

TRIKI Achref

CHERIF Ahmed

OUESLATI Ahmed

ZIADA Oussama

Supervised by :

Dr.SOUID MILED Wided

Dr.CHAKCHOUK Faten

examined by :

Dr.HAMDI Sana

School year: 2023/2024

Acknowledgments

We would like to express our deepest gratitude to our primary supervisor, Dr. Wided Souid Miled, for her invaluable guidance, support, and encouragement throughout this project. Her expertise and insights have been instrumental in the successful completion of this research.

We are also profoundly grateful to our supervisor Dr. Faten Chakchouk for her constant help, valuable advice, and innovative ideas that significantly improved our research and helped us achieve our goals.

Finally, we would like to extend our sincere thanks to Dr. Sana Hamdi, our examiner, for the time and effort she put into studying our research report.

Abstract

This project was conducted as part of an end-of-year project focusing on the generalization of transformer networks to graph structures and their application in embryo classification. Our work centered on implementing a Graph Transformer architecture, a promising advancement in machine learning on graphs, which was applied to the classification of human embryos. The initial chapter introduces graph theory, providing the foundational knowledge necessary for understanding complex graph-based data representations. Subsequent chapters delve into transforming traditional transformer networks to operate on graphs effectively, discussing both theoretical frameworks and practical adaptations.

Contents

List of Figures

List of Tables

General Introduction

In recent years, the field of machine learning has witnessed significant advancements, particularly in the areas of natural language processing (NLP) and computer vision, largely due to the development of transformer networks. Transformers, introduced by Vaswani et al. in 2017, have revolutionized NLP by enabling models to capture long-range dependencies in sequences through a self-attention mechanism. This innovation has led to state-of-the-art performance in various NLP tasks.

However, many real-world problems involve data that is naturally structured as graphs rather than sequences or grids. Graphs, which consist of nodes (representing entities) and edges (representing relationships between entities), are a flexible and powerful representation for such data. Common applications of graph-based data include social networks, biological networks, and knowledge graphs.

Graph Neural Networks (GNNs) have emerged as a prominent approach for learning on graph-structured data. GNNs leverage the connectivity patterns and node features within graphs to perform tasks such as node classification, link prediction, and graph classification. Despite their success, GNNs face limitations, particularly in capturing complex and long-range dependencies within graphs.

To address these limitations, researchers have begun to explore the generalization of transformer networks to graph-structured data, resulting in the development of graph transformers. Graph transformers aim to combine the strengths of transformers' self-attention mechanism with the ability to process graph-structured data,

thereby overcoming the limitations of traditional GNNs. By doing so, graph transformers have the potential to achieve superior performance in various graph-based tasks.

This report delves into the state-of-the-art in graph transformers, exploring their foundational concepts, architecture, and practical implementations. We will also present a novel graph transformer architecture tailored for a specific application: embryo classification. In this application, images of embryos are transformed into graphs, which are then processed by the graph transformer for classification. This case study will demonstrate the efficacy of graph transformers.

This report summarizes and presents the different phases of our project. It is organized as follows:

- In chapter 1, we introduce the graph theory.
- Chapter 2 presents the key concepts of transformers and the generalization of this architecture to graphs.
- Chapter 3 presents a graph transformer architecture designed for embryo classification task.

Chapter 1

Introduction to Graph Theory

Introduction

Graph theory is a vital branch of mathematics focused on studying graphs, structures composed of nodes and edges, which originated from Leonhard Euler's 18th-century solution to the Königsberg bridges problem. It is extensively used across various fields such as computer science, biology, and social sciences to model and analyze complex systems. Graphs are particularly crucial in computer science for simulating networks, algorithms, and software processes. The theory offers powerful algorithms for managing network flows, finding shortest paths, and optimizing schedules, making it essential for both theoretical exploration and practical problem-solving.

Graphs are very useful in artificial intelligence (AI) for modeling complicated systems, including machine learning frameworks and neural networks, which need sophisticated data connections and network dynamics. In the field of artificial intelligence, graphs help create complex models like Graph Neural Networks (GNNs), which are intended to process and learn directly from graph-structured data.

1.1 What is a graph?

1.1.1 Definition

A graph is a structure consisting of a set $\{v_1, \ldots, v_n\}$ of objects and a set $\{e_1, \ldots, e_n\}$ of relationships between these objects. Objects are generally referred to as vertices or nodes; relationships as arcs or edges. Graphs are most commonly represented as diagrams in which vertices are depicted as points and edges as lines joining vertices, as shown in Figure 1.1 [12]

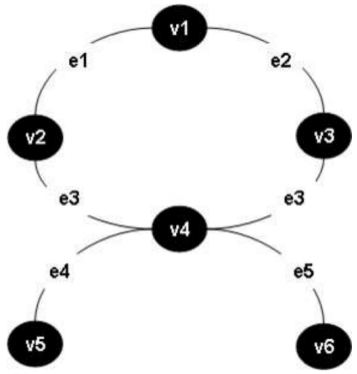


Figure 1.1: An example of a graph

1.1.2 Types of graph

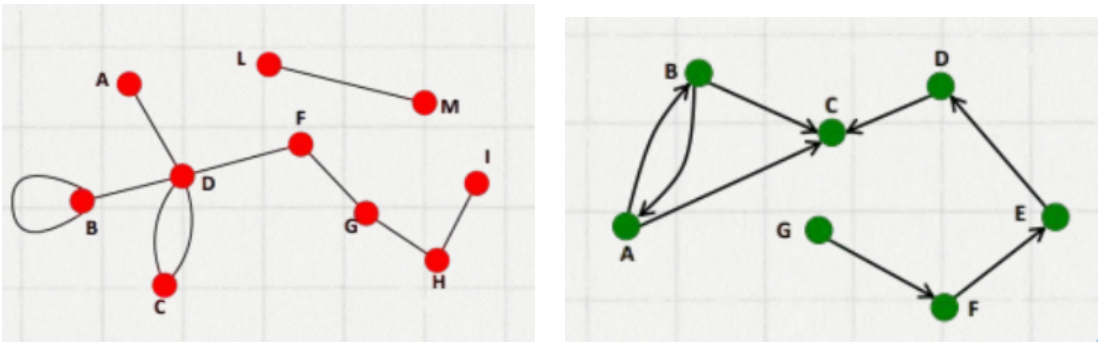
1.1.2.1 Undirected VS directed Graphs

- **Undirected Graphs**: Edges have no direction, indicating a bidirectional relationship between vertices.
- **Directed Graphs**: Edges have a direction, represented by arrows, indicating the relationship flows from one vertex to another.

1.1.2.2 Weighted vs Unweighted Graphs

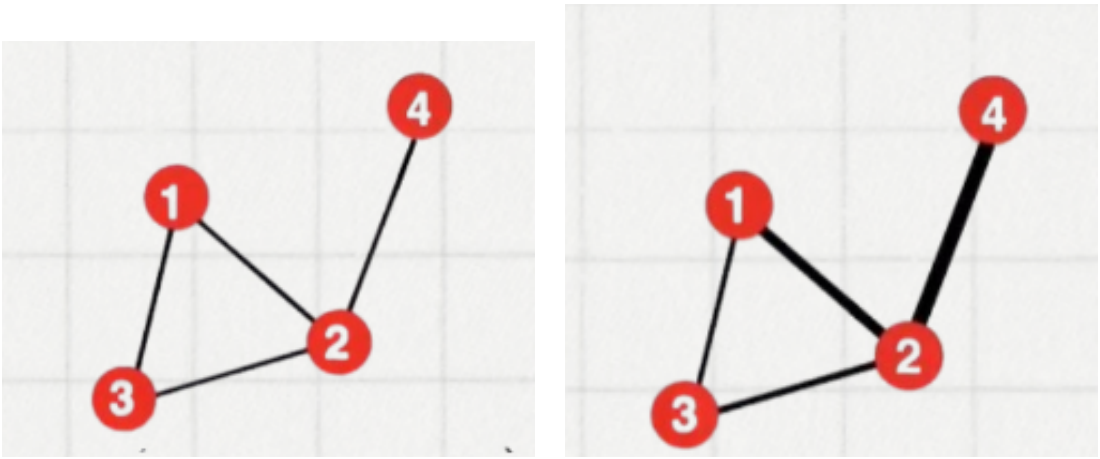
- **Unweighted Graphs**: Edges do not carry weights, representing connections where only the existence of a link matters.

- **Weighted Graphs**: Edges have weights that can represent various attributes like cost or distance.



(a) (b)

Figure 1.2: Types of a Graph (a) Undirected Graph (b) Directed Graph



(a) (b)

Figure 1.3: Example of (a) an Unweighted Graph (b) Weighted Graph

Complex domains have a rich relational structure, which can be represented as a relational graph By explicitly modeling relationships we achieve better performance!

Main question: How do we take advantage of relational structure for better prediction?

1.2 Representation of data in graphs

1.2.1 How we represent a graph ?

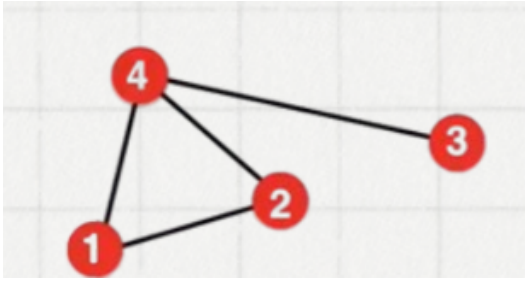
A heterogeneous graph is defined as $G = (V, E, R, T)$ where:

- Nodes with node types $v_i \in V$
- Edges with relation types $(v_i, r, v_j) \in E$
- Node type $T(v_i)$
- Relation type $r \in R$
- Nodes and edges have attributes/features

1.2.2 Adjacency Matrix

An adjacency matrix is a square matrix used to represent a graph. Rows and columns of the matrix represent vertices, and the entries indicate whether there is an edge between the corresponding vertices.

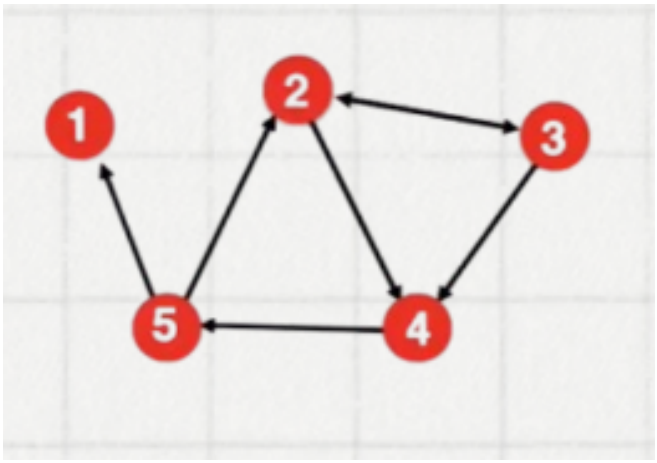
$$A_{ij} = \begin{cases} 1 & \text{if there is a link from node } i \text{ to node } j, \\ 0 & \text{otherwise} \end{cases}$$



$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

1.2.3 List of edges

A list of edges is a simple and compact way to represent a graph. It consists of a list of tuples, where each tuple represents an edge connecting two vertices.
Representation: Each tuple (u,v) in the list represents an edge from vertex u to vertex v.



$(2, 3)$

$(2, 4)$

$(3, 2)$

$(3, 4)$

$(4, 5)$

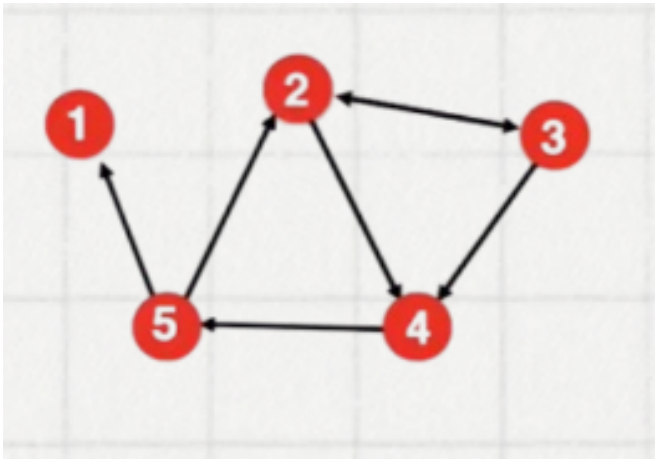
$(5, 2)$

$(5, 1)$

1.2.4 Adjacency list

An adjacency list is a data structure used to represent a graph. It consists of an array of lists, where each list contains the vertices adjacent to a particular vertex.
Representation:

- Each element of the array corresponds to a vertex in the graph.
- For each vertex v, the corresponding list contains the vertices adjacent to v.



1 :

2 :3, 4

3 :2, 4

4 :5

5 :1, 2

1.3 Applications of Graph Machine Learning

Graph Machine Learning aims to predict properties and behaviors of interconnected objects, which are represented as nodes and edges within a graph. GML has numerous applications and can be divided into supervised or unsupervised learning problems, each with individual challenges and objectives for the study of graph-based data. These predictive tasks can range from classifying individual nodes to estimating the properties of entire graphs.[2]

1.3.1 Supervised GML Tasks

Supervised Graph Machine Learning uses labeled data within graphs to predict node, link, or entire graph properties. It's particularly effective in contexts like fraud detection and bioinformatics, where relationships between data points critically influence outcomes.This approach uses the [9]

1.3.1.1 Node Property Prediction

Description: Node Property Prediction in graph machine learning analyzes how nodes are connected within a graph to predict their characteristics. This approach enhances prediction accuracy by integrating information about the nodes and their network relationships.

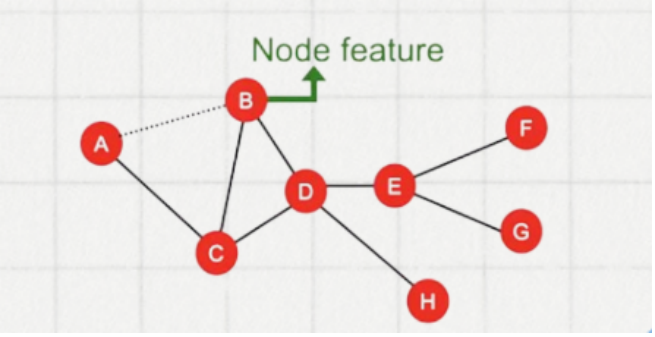


Figure 1.4: Example of node prediction

1.3.1.2 Link prediction

Description: Link prediction in graph machine learning evaluates potential connections between nodes based on their attributes and interactions within the graph. This method improves predictive insights by considering the relationships and similarities between entities.

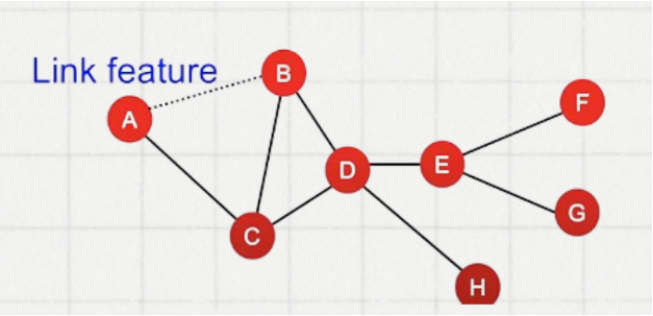


Figure 1.5: Example of link prediction

1.3.1.3 Graph property prediction:

Description: In graph machine learning, graph-level property prediction involves leveraging the attributes of an entire graph or subgraph for making predictions. By utilizing features derived from the complete structure of the graph, as well as information about its neighborhoods, the accuracy of predictions can be enhanced.

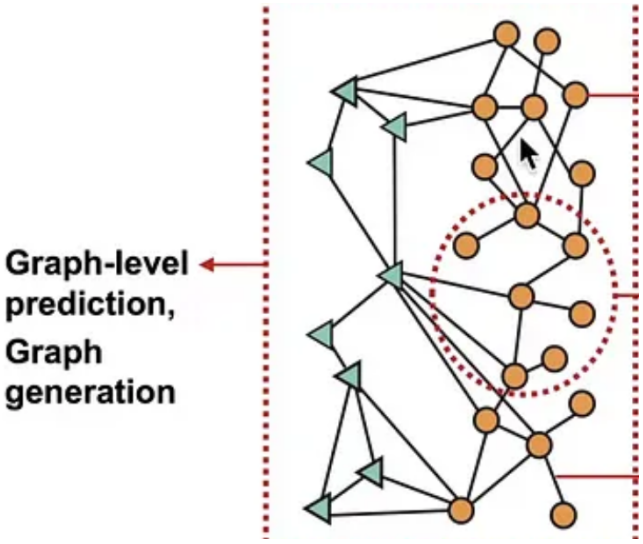


Figure 1.6: Example of graph level prediction

1.3.2 Unsupervised GML Tasks

Unsupervised Graph Machine Learning analyzes patterns in graph data without labeled examples, making it ideal for areas like social media and protein interactions where labels are scarce. This approach identifies communities, predicts missing links, and detects key influencers, uncovering new insights from complex, unlabeled data.

1.3.2.1 Community Detection (clustering for relationships)

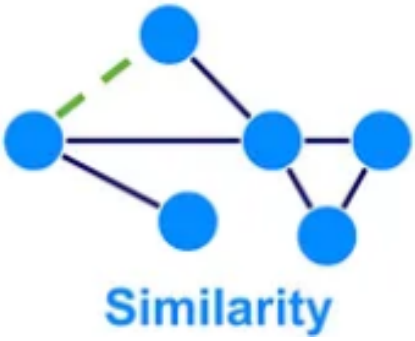
Description: Community detection identifies clusters of closely connected nodes within a graph, highlighting densely interconnected groups.

1.3.2.2 Representation Learning

Description: In Graph Machine Learning (GML) applications, a key goal is to reduce dimensionality while preserving essential information. Graph representation learning achieves this by creating compact, low-dimensional features from complex graph structures.

1.3.2.3 Similarity

Description: In Graph Machine Learning (GML), similarity involves identifying and measuring similar pairs of nodes in a graph . Common Similarity techniques include node similarity algorithms such K-Nearest-Neibor (KNN).



1.4 Graph Neural Networks

Graph Neural Networks (GNNs) are a class of neural networks designed to work with graph-structured data. They have become increasingly popular due to their ability to model the relationships and interactions within data that can be naturally represented as graphs, such as social networks, molecular structures, and recommendation systems.

1.4.1 Message Passing

Message passing is the core mechanism in GNNs where nodes exchange information with their neighbors to update their feature representations. In this project, we focus on *Message passing graph neural networks (MPNNs)*, which follow an iterative scheme [3, 4, 5, 6, 7]. Throughout, they maintain a representation (embedding) $h_v^{(t)} \in \mathbb{R}^{d_t}$ for each node $v \in V$. In each iteration t, they update each embedding $h_v^{(t)}$ as a function of its neighbors' embeddings and possible edge attributes:

$$h_v^{(0)} = x_v, \quad \forall v \in V \qquad \text{(Initialization)}$$

$$m_v^{(t)} = f_{\text{Agg}}^{(t)}\left(h_u^{(t-1)}, \left\{h_u^{(t-1)}, w(u,v) \mid u \in \mathcal{N}(v)\right\}\right), \quad 1 \le t \le T \qquad \text{(Aggregate)}$$

$$h_v^{(t)} = f_{\text{Upd}}\left(h_v^{(t-1)}, m_v^{(t)}\right) \qquad \text{(Update)}$$

The final node representation $f(v) = h_v^{(T)}$, $\forall v \in V$ is the last iterate, possibly concatenated with a linear classifier. $\mathcal{N}(v) \subseteq V$ denotes the neighborhood of $v \in V$, and $\{\cdot\}$ a multiset. Here, $h_v^{(t)}$ encodes the t-hop neighborhood of node v, i.e., the subgraph of all nodes reachable from v within t steps. The number of iterations T is also termed the GNN depth, and one iteration may be viewed as a layer.
Finally, if a graph-level prediction is desired, like in our case, all node representations can be aggregated by a permutation invariant *readout* function

$$F(G) = f_{\text{read}}\left(\{h_v^{(T)} \mid v \in V\}\right).$$

1.4.2 Applications of Graph Neural Networks

In this subsection, we explore the diverse range of applications where Graph Neural Networks (GNNs) have been successfully applied.

1.4.2.1 Social Network Analysis

GNNs have shown remarkable success in analyzing social networks[8]., where nodes represent individuals, and edges denote social connections. Tasks such as community detection, influence prediction, and anomaly detection benefit from GNNs' ability to capture complex relationships and structural patterns within the network.

1.4.2.2 Recommendation Systems

Graph-based recommendation systems[**?**] leverage GNNs to model user-item interactions as a bipartite graph. By learning latent representations of users and items, GNNs can effectively capture user preferences, item similarities, and contextual information, leading to personalized and accurate recommendations.

1.4.2.3 Biological and Chemical Graph Analysis

In bioinformatics and chemoinformatics, GNNs are used for analyzing molecular structures represented as graphs[16]. GNNs can predict molecular properties, identify molecular structures with desired characteristics, and assist in drug discovery by learning from large-scale molecular databases.

1.4.3 Limitations of Graph Neural Networks and Emergence of Graph Transformers

While Graph Neural Networks (GNNs) have demonstrated impressive capabilities in various graph-related tasks, they also exhibit a major limitation:

Difficulty with heterogeneous graphs: GNNs are designed primarily for homogeneous graphs, where all nodes and edges have the same type. Adapting GNNs to heterogeneous graphs with diverse node and edge types can be challenging and may

require specialized architectures.
Additionally, recent research has seen the emergence of Graph Transformers, which offer alternative approaches to graph representation learning.

Conclusion

In summary, while Graph Neural Networks (GNNs) have significantly advanced graph-related tasks, their struggle with heterogeneous graphs is a notable limitation. This challenge has spurred the emergence of Graph Transformers as a promising alternative. By leveraging self-attention mechanisms, Graph Transformers offer solutions to overcome GNNs' limitations.

Chapter 2

Generalization of Transformer Networks to Graphs

Introduction

Transformer networks, a groundbreaking development in the field of deep learning, have revolutionized natural language processing and are now extending their influence to the domain of graph theory. Originally introduced by Vaswani et al. in 2017, transformers leverage self-attention mechanisms to capture intricate dependencies within data sequences, proving exceptionally effective in tasks such as language translation and text generation.

By generalizing transformer architectures to handle graphs, it is possible to enhance the performance of tasks like node classification, link prediction, and graph generation. This chapter delves into the methodologies and advancements involved in adapting transformer networks to graphs, highlighting their significance in advancing the state-of-the-art in graph-based learning and applications.

2.1 Key concepts of Transformers

2.1.1 What is a transformer

Transformer neural networks are a revolutionary architecture in the field of machine learning that have significantly advanced the capabilities of artificial intelligence, particularly in processing sequential data such as text and speech. Introduced in the landmark paper "Attention Is All You Need" in 2017, transformers have set new benchmarks in tasks like language translation, text generation, and more,surpassing previous architectures like recurrent neural networks (RNNs) and long short-term memory networks (LSTMs) in many tasks. [1]

2.1.2 Core Components of Transformers

2.1.2.1 Token embedding

- **Introduction to Word Embeddings**: Token embedding is a fundamental process in natural language processing (NLP) that converts words or phrases from text into numerical representations, which are then used as input for neural network models. This process is crucial because neural networks, at their core, operate on numeric data rather than raw text. Traditional representations like one-hot encoding suffer from high dimensionality and lack of semantic information. In contrast, word embeddings capture both the semantic and syntactic essence of words, allowing models to process text data more effectively.

- **Embeddings in Transformers**: In transformer architectures, embeddings serve as the initial layer that processes input text data. Unlike recurrent neural networks (RNNs) that inherently understand sequence due to their architecture, transformers require embeddings to be combined with positional encodings. This addition informs the model of the order of words, which is crucial for understanding the contextual relationships within the text.

- **Comparison with other Embedding Techniques**: While traditional embedding techniques like Word2Vec provide a static representation of words based

on their linguistic contexts, transformer embeddings are contextually informed by the model's architecture. Each word's embedding can adjust based on its surrounding words, allowing for a deeper understanding of language nuances. Furthermore, as these embeddings pass through successive transformer layers, they accrue more abstract and comprehensive linguistic features, enhancing the model's interpretive ability.

2.1.2.2 Positional Encoding

- **Introduction to Positional Encoding**: Positional encoding is a technique used in transformer models to give them an understanding of the order of words in a sequence, a critical feature that's missing in the initial embedding layers. Unlike RNNs or CNNs, which inherently process data sequentially, transformers process input data in parallel and thus do not have a built-in way to recognize word order. Positional encoding solves this problem by adding extra information to each token's embedding, indicating its position in the sequence.

- **Concept and Calculation**: Positional encoding involves adding vectors to input embeddings that vary depending on the position of tokens in the input sequence. The most common method uses sinusoidal functions to generate these vectors, ensuring that each element of the sequence can be distinguished from others by its unique encoding . The functions are defined as follows:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

Here, pos represents the position in the sequence, i represents the dimension, and d_{model} is the dimensionality of the model. This pattern allows the model to use the relative or absolute position of the tokens in the sequence to enhance its understanding of the text.
[1]

- **Advantages of Sinusoidal Encoding**: The use of sinusoidal functions for positional encoding offers several benefits:

  - Differentiability: The continuous nature of sine and cosine functions facilitates gradient-based optimization, making the training process smoother.

  - Scalability: These functions allow the model to interpolate and extrapolate for sequence positions outside the training set, making models more flexible when dealing with texts of varying lengths.

2.1.2.3 Self-Attention Mechanism

- **Introduction to Self-Attention**: Self-Attention is a mechanism at the heart of Transformer architectures that fundamentally differentiates them from previous models based on recurrent or convolutional layers. Unlike these older approaches, which process data sequentially, Self-Attention allows the model to weigh and respond to different parts of the input data simultaneously. This feature enables the Transformer to capture complex dependencies and relationships within the data, irrespective of their distance in the input sequence.

- **Mechanics of Self-Attention**:

  - For each word vector, three different vectors are computed using trainable weight matrices: a Query vector (Q), a Key vector (K),and a Value vector (V). These vectors are used to determine the attention each word should pay to all other words in the input sequence.The attention weights can be formally expressed as[1]:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

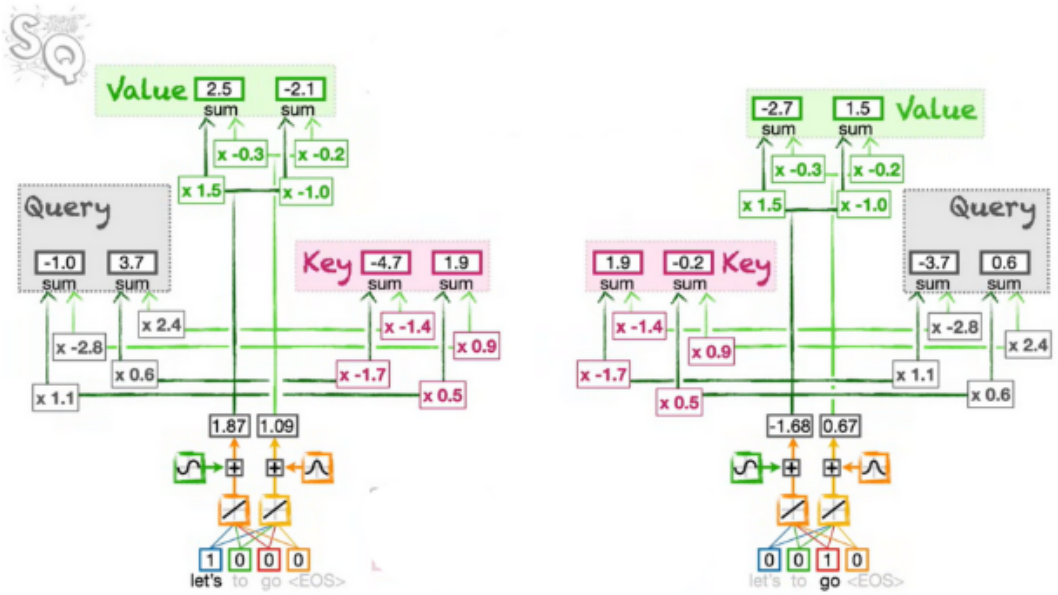


Figure 2.1: visualization of the computation of self attention mechanism

– The attention score between two words is calculated by taking the dot product of the Query vector of one word with the Key vector of another. This score indicates how much focus should be placed on other parts of the input when encoding the current word The attention scores for a word are then normalized using a softmax function.

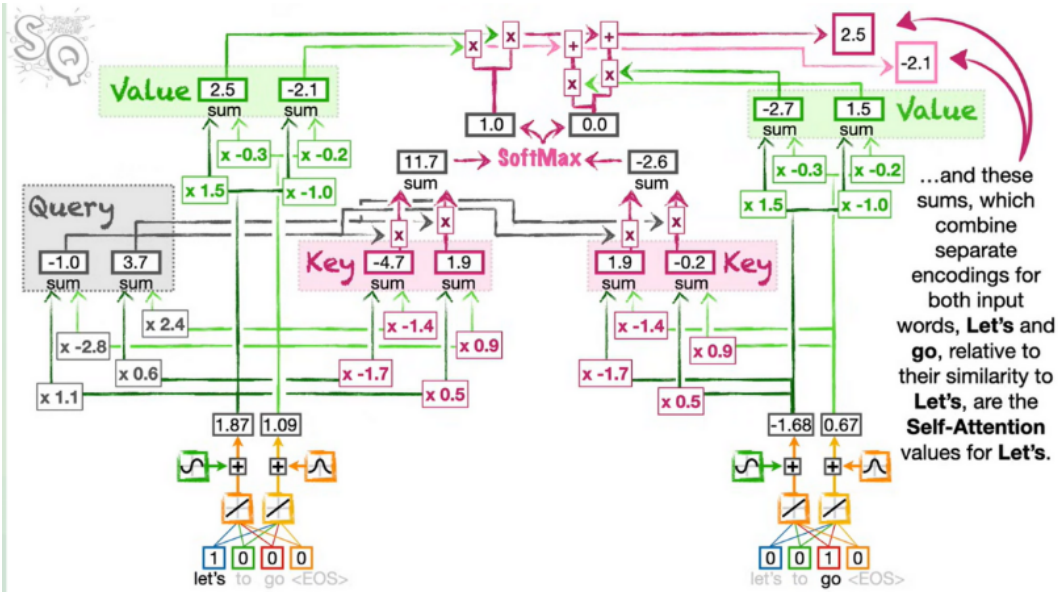


Figure 2.2: visualization of the computation of attention score between two words

2.1.2.4 multi-head attention

- **Introduction to Multi-Head Attention**: Multi-head attention is a sophisticated extension of the attention mechanism, pivotal in transformer architectures. Unlike traditional single-head attention, which considers the input sequence through a singular lens, multi-head attention allows the model to explore the input sequence under multiple representation subspaces at different positions. This feature is crucial as it significantly enhances the model's ability to discern varied nuances within the data.

- **Concept and Operation**: Multi-head attention operates by segmenting the model's attention into several "heads", each performing an independent attention operation. This segmentation is achieved by projecting the queries, keys, and values through multiple sets of linear transformations, thus allowing each head to focus on different aspects of the input: [14]
  Mathematical Formulation:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

  where

  – W_i^Q, W_i^K, W_i^V are the parameter matrices for the i-th head for queries, keys, and values respectively.

  – W^O is the output linear transformation matrix.

2.1.3 Transformer Model Architecture

2.1.3.1 Overall structure

Transformers use a unique architecture that allows for parallel processing, enabling faster and more efficient training than previous models like RNNs and LSTMs. In this architecture, the encoder processes the input sequence in parallel through multiple layers, each composed of self-attention and feed-forward neural network com-

ponents, enabling the model to efficiently handle dependencies between elements in the sequence. The encoded information is then passed to the decoder, which, also structured in layers, sequentially generates the output. The decoder's self-attention mechanism is nuanced, allowing each position in the output sequence to attend over all positions in the input sequence, ensuring that predictions for each token are informed by the entirety of the input. transformers excel in diverse tasks ranging from language translation to content generation, setting a new standard for performance in complex sequence-based problems.

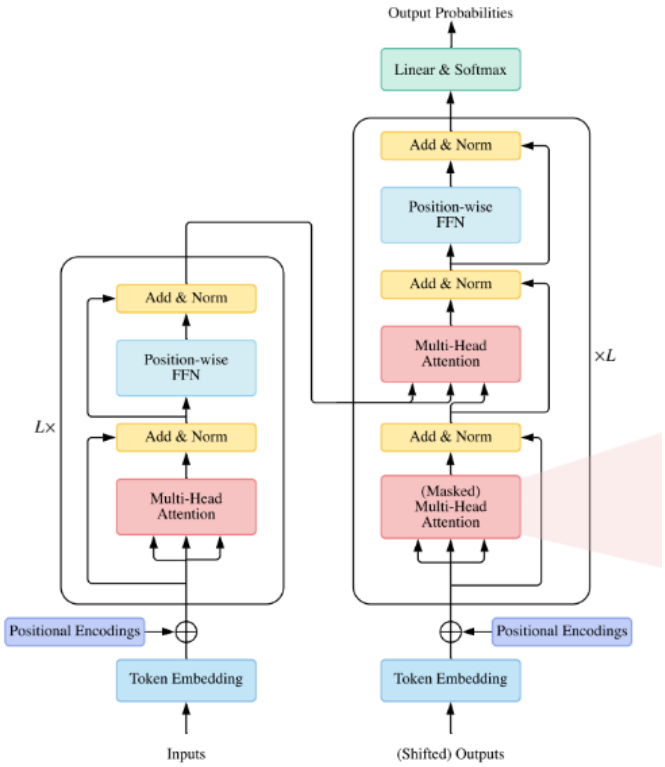


Figure 2.3: Transformer architecture

2.1.3.2 Encoder

The encoder takes the input sequence and processes it to generate a representation of the input. It consists of multiple layers of self-attention and feed-forward neural networks.

- **A multi-head self-attention mechanism**, which allows the encoder to consider different positions of the input sequence simultaneously.
- **A position-wise fully connected feed-forward network**, which transforms the output of the attention mechanism independently at each position.After obtaining the context vectors through self-attention, each word's representation passes through a position-wise feedforward neural network. This network consists of two linear transformations with a non-linear activation function (e.g., ReLU) applied in between. It operates independently on each position in the sequence, which helps capture local information effectively.

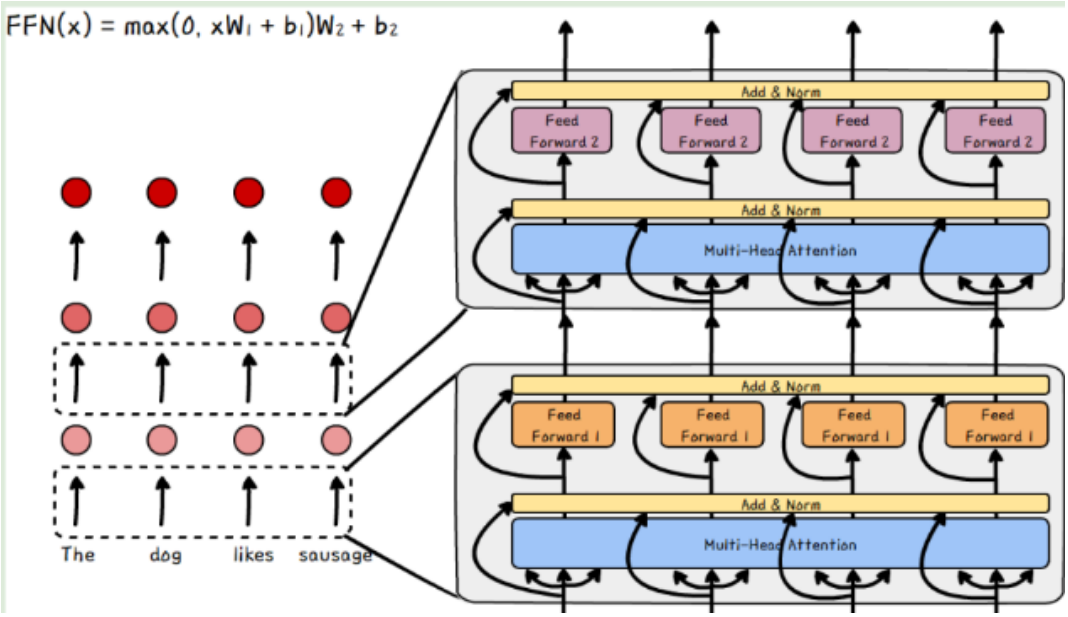


Figure 2.4: feed forward network example

The encoder processes the input sequence layer by layer. At each layer, the self-attention mechanism and position-wise feedforward networks are applied sequentially. Additionally, residual connections and layer normalization are employed to stabilize and speed up the training process.

2.1.3.3 Decoder

The decoder takes the encoded representation from the encoder and processes it to generate the output sequence. Like the encoder, the decoder consists of multiple

layers, each with its own self-attention and feed-forward neural networks. However, it also includes an additional attention mechanism to focus on relevant parts of the input sequence.

- **Masked Multi-Head Self-Attention Mechanism**: The decoder uses a masked multi-head self-attention mechanism to ensure that the predictions for a position in the output sequence depend only on the known outputs at preceding positions. This is achieved by masking future positions, preventing the model from peeking ahead and ensuring auto regressive properties.

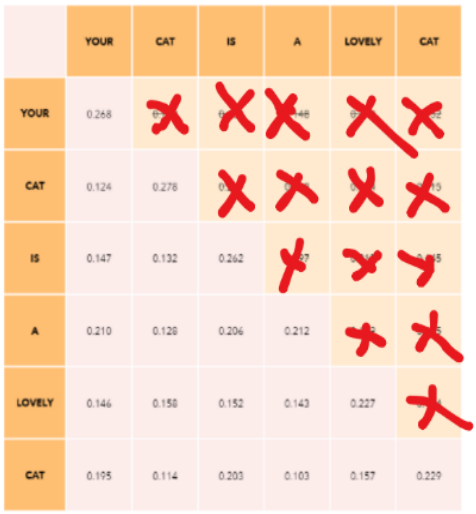


Figure 2.5: Example of masked attention

- **Encoder-Decoder Attention**: Encoder-Decoder Attention: In addition to self-attention, the decoder utilizes an encoder-decoder attention mechanism to incorporate information from the encoded input sequence. This mechanism allows the decoder to focus on relevant parts of the input sequence when generating each word in the output sequence. Similar to self-attention, this attention mechanism is also masked during training to prevent attending to future tokens.

- **Position-wise Fully Connected Feed-Forward Network**: Similar to the encoder, the decoder also has a position-wise fully connected feed-forward network. After incorporating the context from the encoder-decoder attention, each position in the sequence passes through this network, which applies two linear

transformations with a non-linear activation function (e.g., ReLU) in between. This helps to capture local information and refine the output sequence.

- **Residual Connections and Layer Normalization**: Residual Connections and Layer Normalization: Similar to the encoder, residual connections and layer normalization are employed after each sub-layer in the decoder to stabilize and speed up training.

During inference (i.e., generating output sequences), the decoder operates autoregressively, where each word is generated one at a time based on previously generated words. At each step, the decoder attends to the encoded input sequence using the encoder-decoder attention mechanism and attends to preceding words in the output sequence using the self-attention mechanism. This process continues until an end-of-sequence token is generated or a predefined maximum sequence length is reached.

2.1.4 Transformer variants

BERT (Bidirectional Encoder Representations from Transformers): BERT is a Transformer variant that emphasizes bidirectional training of Transformers for language understanding. Unlike traditional Transformers that process text in a unidirectional manner, BERT considers the context from both directions, leading to improved comprehension and performance on tasks such as question answering and language inference.

GPT (Generative Pre-trained Transformer): GPT focuses on generating text and has been pre-trained on a large corpus of text data. It uses a unidirectional approach, predicting the next word in a sequence, making it particularly effective for tasks like text completion, summarization, and creative writing.

T5 (Text-To-Text Transfer Transformer): T5 treats every NLP task as a text-to-text problem, transforming inputs and outputs into text strings. This unified framework allows T5 to handle a wide range of tasks, including translation, summarization, and question answering, using the same model architecture.

2.2 Graph Transformer

2.2.1 Positional Encoding

In the realm of graph neural networks (GNNs), encoding the position of nodes presents unique challenges. Unlike sequences in text, graphs often exhibit symmetrical structures, which complicates the assignment of unique positional encodings to nodes. According to Murphy et al. (2019) [10], these symmetries obstruct the straightforward use of canonical node positional information,making it challenging to distinguish nodes based purely on their graph position without additional structural context.

The issue of positional embeddings has been explored in recent GNN works with a goal to learn both structural and positional features.In particular, Dwivedi et al. (2020) make the use of available graph structure to pre-compute Laplacian eigenvectors and use them as node positional information. Since Laplacian PEs are generalization of the PE used in the original transformers to graphs and these better help encode distance-aware information (i.e., nearby nodes have similar positional features and farther nodes have dissimilar positional features),

=> we use Laplacian eigenvectors as PE in Graph Transformer.

What are Laplacian Eigenvectors:

Before understanding Laplacian positional encoding, let's first talk about Laplacian eigenvectors. Laplacian eigenvectors are a set of eigenvectors of the Laplacian matrix of a graph. The Laplacian matrix of a graph is a square matrix that encodes the graph's topology. It is defined as the difference between the degree matrix (a diagonal matrix that shows the degrees of each node) and the adjacency matrix (a matrix that shows the connections between nodes). The Laplacian matrix is used in many applications, such as graph clustering, community detection, and graph embedding.

=>In simple terms, Laplacian eigenvectors are a way to break down a graph into its most fundamental components. They help in understanding the structure of the graph, the relationships between the nodes, and the importance of each node in the graph.

Laplacian Positional Encoding:
Laplacian positional encoding offers a sophisticated solution to the challenge of node position encoding in GNNs. By leveraging Laplacian eigenvectors, which are derived from the graph Laplacian matrix ,[14]

$$\Delta = I - D^{-1/2}AD^{-1/2} = U^T \Lambda U,$$

where A is the $n \times n$ adjacency matrix, D is the degree matrix, and Λ, U correspond to the eigenvalues and eigenvectors respectively. We use the k smallest non-trivial eigenvectors of a node as its positional encoding and denote by λ_i for node i.

This method generalizes the sinusoidal positional encodings used in traditional transformers to suit graph structures. This approach allows for encoding positional information that reflects the distances and relationships between nodes, enhancing the model's ability to differentiate node roles based on their structural positions.

- **Eigenvalues (Λ)**: These values indicate the "frequency" or the importance of the corresponding eigenvectors in representing the graph structure.
- **Eigenvectors (U)**: Each eigenvector provides a way to position nodes in a space where their geometric arrangement reflects their connectivity and roles in the graph.

2.2.2 Graph Sparsity

NLP Transformers consider fully connected graph and this choice can be justified for two reasons:

- Difficult to find meaningful sparse connections between words, i.e. absence of sparse structure.
- NLP Transformer sentences often contain less than tens or hundreds of words, hence scalable to consider full attention.

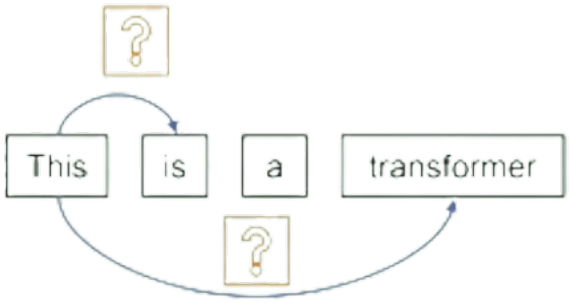


Figure 2.6: sequence of words in NLP

However, graphs have arbitrary connectivity structures, and full attention is not feasible due to the potential presence of millions or billions of nodes. Graph Neural Networks (GNNs) are state-of-the-art in several application domains as they take the sparse structure into account while learning feature representations. Sparsity provides a good inductive bias for generalization.

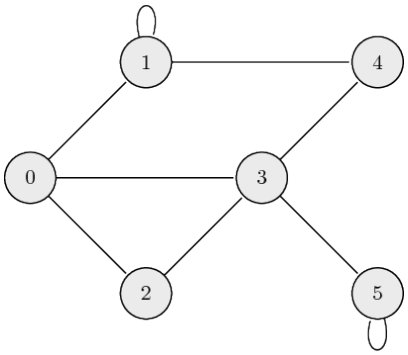


Figure 2.7: Example of sparse graph

2.2.3 Graph Transformer's Architecture

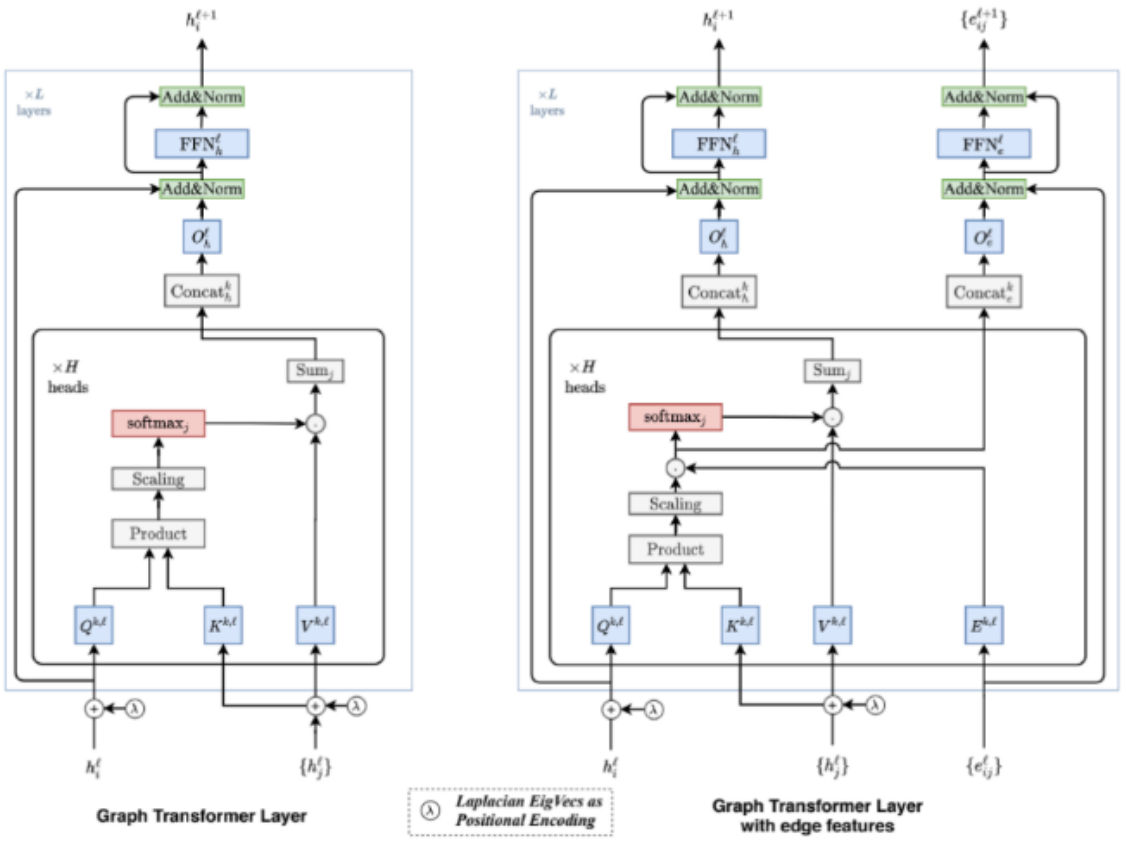


Figure 2.8: Graph Transformer's Architecture

Compared to the original transformer, the highlights of the proposed architecture are:[14]

- he attention mechanism is a function of neighborhood connectivity for each node.

$$\hat{h}_i^{\ell+1} = O_h^{\ell} \Big\|_{k=1}^{H} \left(\sum_{j \in \mathcal{N}_i} w_{ij}^{k,\ell} V^{k,\ell} h_j^{\ell} \right),$$

  where,

$$w_{ij}^{k,\ell} = \text{softmax}_j \left(\frac{Q^{k,\ell} h_i^{\ell} \cdot K^{k,\ell} h_j^{\ell}}{\sqrt{d_k}} \right).$$

- Positional encoding is represented by Laplacian PE-k smallest non-trivial eigenvectors of a node.

- The layer normalization is replaced by a batch normalization layer.
- The architecture is extended to have edge representation, which can be critical to tasks with rich information on the edges.

For numerical stability, the outputs after taking exponents of the terms inside softmax is clamped to a value between -5 to +5.
The attention outputs $\hat{h}_i^{\ell+1}$ are then passed to a Feed Forward Network (FFN) preceded and succeeded by residual connections and normalization layers, as:

$$\hat{h}_i^{\ell+1} = \text{Norm}(h_i^{\ell} + \hat{h}_i^{\ell+1}),$$
$$\tilde{h}_i^{\ell+1} = W_2^{\ell}\text{ReLU}(W_1^{\ell}\hat{h}_i^{\ell+1}),$$
$$h_i^{\ell+1} = \text{Norm}(\hat{h}_i^{\ell+1} + \tilde{h}_i^{\ell+1}).$$

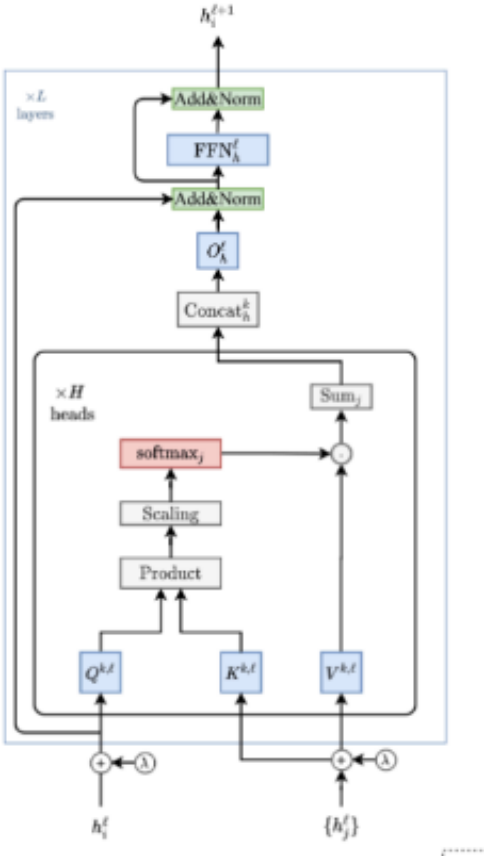


Figure 2.9: Graph Transformer Layer

For the graph transformer layer with edge features : the output is :

$$\hat{h}_i^{\ell+1} = O_h^\ell \,\|_{k=1}^{H} \left(\sum_{j \in \mathcal{N}_i} w_{ij}^{k,\ell} V^{k,\ell} h_j^\ell \right),$$
$$\hat{e}_{ij}^{\ell+1} = O_e^\ell \,\|_{k=1}^{H} \left(\hat{w}_{ij}^{k,\ell} \right), \quad \text{where,}$$
$$w_{ij}^{k,\ell} = \text{softmax}_j \left(\hat{w}_{ij}^{k,\ell} \right),$$
$$\hat{w}_{ij}^{k,\ell} = \left(\frac{Q^{k,\ell} h_i^\ell \cdot K^{k,\ell} h_j^\ell}{\sqrt{d_k}} \right) \cdot E^{k,\ell} e_{ij}^\ell.$$

For numerical stability, the outputs after taking exponents of the terms inside softmax is clamped to a value between -5 to +5. The outputs $\hat{h}_i^{t+1}$ and $\hat{e}_{ij}^{t+1}$ are then passed to separate Feed Forward Networks preceded and succeeded by residual connections and normalization layers, as:

$$\hat{h}_i^{t+1} = \text{Norm}(h_i^t + \hat{h}_i^{t+1})$$
$$\hat{h}_i^{t+1} = W'_{h,2}\text{ReLU}(W'_{h,1}\hat{h}_i^{t+1})$$
$$\hat{h}_i^{t+1} = \text{Norm}(h_i^t + \hat{h}_i^{t+1})$$
$$\hat{e}_{ij}^{t+1} = \text{Norm}(e_{ij}^t + \hat{e}_{ij}^{t+1})$$
$$\hat{e}_{ij}^{t+1} = W'_{e,2}\text{ReLU}(W'_{e,1}\hat{e}_{ij}^{t+1})$$
$$\hat{e}_{ij}^{t+1} = \text{Norm}(e_{ij}^t + \hat{e}_{ij}^{t+1})$$

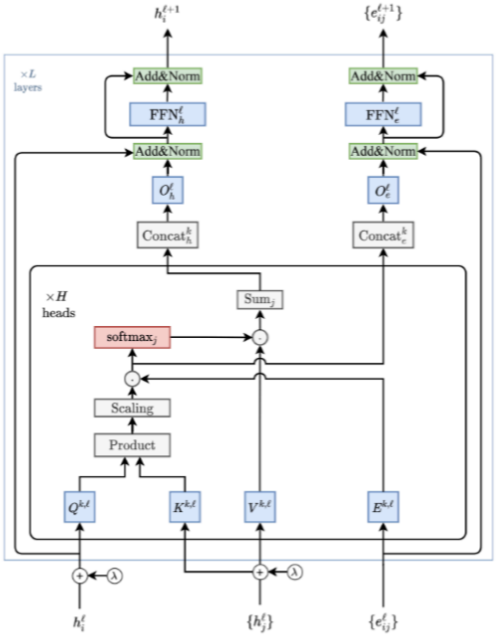


Figure 2.10: Graph Transformer Layer

2.3 Datasets

2.3.1 ZINC dataset

For our benchmark, we use a 12K subset of the 250K ZINC dataset to regress the constrained solubility of molecular graphs. Each graph's node features represent heavy atom types, while edge features represent bond types. Given the rich edge attribute information, we employ the 'Graph Transformer with edge features' for this task.

Splitting: ZINC has 10, 000 train, 1, 000 validation and 1, 000 test graphs.

Performance Measure: Mean absolute error (MAE) between the predicted and ground truth constrained solubility for each molecular graph.

2.3.2 PATTERN dataset

PATTERN is a node classification dataset generated using Stochastic Block Models (Abbe 2017) to classify nodes into two communities. It consists of 14K graphs without explicit edge features, so we use the simple 'Graph Transformer' for this task.

Splitting: 10,000 training, 2,000 validation, and 2,000 test graphs.

Performance Measure: Average node-level accuracy weighted by class sizes.

2.3.3 Cluster dataset

CLUSTER is a synthetic dataset generated using the SBM model, with the task of assigning one of six cluster labels to each node. Like PATTERN, CLUSTER graphs lack explicit edge features, so we use the simple 'Graph Transformer' for this task. The dataset contains 12K graphs.

Splitting: 10,000 training, 1,000 validation, and 1,000 test graphs.

Performance Measure: Average node-level accuracy weighted by class sizes.

2.4 Implementation and Results

The table below presents the results of the GraphTransformer (GT) model evaluated on three different datasets: ZINC, PATTERN, and CLUSTER. The performance measure for ZINC is Mean Absolute Error (MAE), whereas for PATTERN and CLUSTER it is Accuracy (Acc). The results, which are averaged over 4 runs with 4 different seeds, highlight the model's performance across two types of graph structures: Sparse Graph and Full Graph. In the Full Graph configuration, full connections are created among all nodes, and for the ZINC dataset, edge features are discarded to focus on the effects of the full graph without any sparse structure information. The best performing model for each dataset is highlighted in bold. These results are extracted from the paper "A Generalization of Transformer Networks to Graphs" by Vijay Prakash Dwivedi and Xavier Bresson.[14] This study will be referenced in my report, followed by a discussion of my own implementation on the ZINC dataset.

Dataset	LapPE	L	#Param	Sparse Graph				Full Graph			
				Test Perf.±s.d.	Train Perf.±s.d.	#Epoch	Epoch/Total	Test Perf.±s.d.	Train Perf.±s.d.	#Epoch	Epoch/Total
Batch Norm: False; Layer Norm: True											
ZINC	x	10	588353	0.278±0.018	0.027±0.004	274.75	26.87s/2.06hr	0.741±0.008	0.431±0.013	196.75	37.64s/2.09hr
	✓	10	588929	0.284±0.012	0.031±0.006	263.00	26.64s/1.98hr	0.735±0.006	0.442±0.031	196.75	31.50s/1.77hr
CLUSTER	x	10	523146	70.879±0.295	86.174±0.365	128.50	202.68s/7.32hr	19.596±2.071	19.570±2.053	103.00	512.34s/15.1hr
	✓	10	524026	70.649±0.250	89.305±0.528	130.75	200.55s/7.43hr	26.916±3.920	26.916±3.764	139.50	565.13s/22.37hr
PATTERN	x	10	522742	73.140±13.633	73.070±13.589	184.25	276.66s/13.75hr	70.854±0.111	50.906±0.056	108.00	540.85s/16.77hr
	✓	10	522982	71.005±11.831	71.125±11.977	192.50	294.91s/14.79hr	56.482±3.549	56.565±3.546	124.50	637.55s/22.69hr
Batch Norm: True; Layer Norm: False											
ZINC	x	10	588353	0.264±0.008	0.048±0.006	321.50	28.01s/2.52hr	0.724±0.013	0.518±0.013	192.25	50.27s/2.72hr
	✓	10	588929	0.226±0.014	0.059±0.011	287.50	27.78s/2.25hr	0.698±0.049	0.339±0.123	273.50	46.05s/3.50hr
CLUSTER	x	10	523146	72.139±0.405	88.571±0.555	181.00	205.85s/6.88hr	21.092±2.134	21.071±3.037	100.75	595.24s/17.10hr
	✓	10	524026	73.169±0.622	86.585±0.905	170.50	201.06s/7.20hr	27.192±8.471	27.192±8.485	133.75	552.06s/20.72hr
PATTERN	x	10	522742	83.940±3.803	83.864±4.089	236.50	299.54s/19.71hr	50.889±0.069	50.873±0.039	104.50	621.33s/17.53hr
	✓	10	522982	84.808±0.068	86.559±0.116	145.25	309.95s/12.67hr	54.941±3.739	54.915±3.769	117.75	683.53s/22.77hr

Table 2.1: Results of GraphTransformer (GT) on all datasets.

We tested the graph-transformer with edge features architecture using Laplacian PE and batch normalization on the Zinc dataset and we got the following results:

Dataset	LapPE	Layers	#params	test perf (MAE)	train perf (MAE)	#epoch	time epoch/total	Batch normalization
ZINC	TRUE	10	588929	0.2099	0.0373	359	45.37s / 4.55 hrs	TRUE

Table 2.2: Performance metrics for the ZINC dataset

Model	ZINC	CLUSTER	PATTERN
GNN BASELINE SCORES from (Dwivedi et al. 2020)[13]			
GCN	0.367 ± 0.011	68.498 ± 0.976	71.892 ± 0.334
GAT	0.384 ± 0.007	70.587 ± 0.447	78.271 ± 0.186
GatedGCN	0.214 ± 0.013	76.082 ± 0.196	86.508 ± 0.085
OUR RESULTS			
GT	0.226 ± 0.014	73.169 ± 0.622	84.808 ± 0.068

Table 2.3: Performance comparison of different models on various datasets.[14]

Conclusion

Graph transformers represent a significant advancement in machine learning by extending transformer networks to handle graph-structured data. Incorporating self-attention mechanisms and techniques such as Laplacian positional encodings, graph transformers effectively capture long-range dependencies and global context, overcoming the limitations of traditional Graph Neural Networks (GNNs).

One key innovation in graph transformers is their architecture, which supports both node and edge features, allowing for a more comprehensive modeling of complex relationships within graphs. The use of Laplacian positional encodings enhances the ability of graph transformers to understand the structure of graphs by providing additional spatial context.

Overall, graph transformers mark a pivotal step forward in processing and analyzing graph-structured data, opening new possibilities for research and applications in numerous fields. Their ability to integrate detailed feature information and capture extensive dependencies makes them a powerful tool for future developments in machine learning.

Chapter 3

Implementation of Graph Transformer in Embryo Classification

Introduction

In this section, we delve into the application of the Graph Transformer architecture for the classification of human embryos. This model is specifically engineered to handle structured data, such as graphs, which naturally encapsulate the relational information present in embryo images. By representing each embryo image as a graph, where nodes denote distinct regions or features within the embryo and edges capture the connections between these features, the Graph Transformer model can effectively learn to classify embryos based on their structural characteristics.

We provide an extensive overview of the Graph Transformer architecture and its integral components, which include graph attention mechanisms and transformer layers. Following this, we explain the preprocessing steps essential for converting embryo images into graph representations conducive for input into the model. Additionally, We talk about the details of the training procedure,discussing pertinent topics such as data augmentation techniques and optimization strategies tailored specifically for the Graph Transformer mode.

3.1 Technical environment

This project was executed on a local computer running Ubuntu, providing a stable and customizable development environment, equipped with an NVIDIA RTX 3050 GPU to leverage CUDA for accelerated computing. All project files, from datasets to scripts and their outcomes, were stored and managed locally. For machine learning and high-performance numerical computing, we utilized PyTorch, a powerful open-source machine learning library developed by Facebook's AI Research lab. PyTorch supports a wide range of classification and regression algorithms as well as deep learning and neural networks, making it ideal for our Graph Transformer project. The project is written in Python version 3.10. The main libraries used include DGL (Deep Graph Library) for scalable graph neural networks, Numpy for numerical computations, Scikit-learn for additional machine learning utilities, Pandas for data manipulation.

3.2 Human Embryo Dataset

In this project, we utilize a dataset of grayscale images representing human embryos at the blastocyst stage, which is critical for assessing embryo viability for implantation in IVF treatments (in vitro Fertilization , a manual fertilization procedure in which embryos are cultured in a laboratory under very strict control conditions) . we worked on a public dataset containing 249 images [11]

Each image captures an embryo cultivated in an incubator equipped with a camera, aimed at observing the embryo's development up to the fifth day of culture, as shown in figures.3.1.

(a)

(b)

Figure 3.1: Different structures of human embryo at blastocyst stage

Our dataset is divised into 3 classes based on Blastocyst development stage ,Inner cell mass (ICM) quality and Trophectoderm (TE) quality.
The three classes are :

- Good
- Fair
- Poor

The primary aim of this project is to use Graph Transformers for the task of embryo classification. Graph Transformers can capture the inherent relational structure within the embryo images by representing each embryo image as a graph

3.3 Graph construction

The methodology for constructing graphs from images of embryos, as detailed in this report, was significantly inspired by existing research in the field of digital pathology. Our approach adapts and extends the innovative techniques introduced in the paper "A Graph-Transformer for Whole Slide Image Classification" by Yi Zheng, Rushin

H. Gindra, Emily J. Green, Eric J. Burks, Margrit Betke, Jennifer E. Beane, and Vijaya B. Kolachalama.[15]

The division of images into patches, the utilization of convolutional neural networks (CNNs) for extracting detailed features from these patches, and the employment of contrastive learning strategies to enhance the feature embedding process are all derived from the methodologies discussed in this seminal work. By integrating these sophisticated techniques, we aim to enhance the accuracy and efficiency of our graph construction process, thereby facilitating more advanced analyses of embryo images.

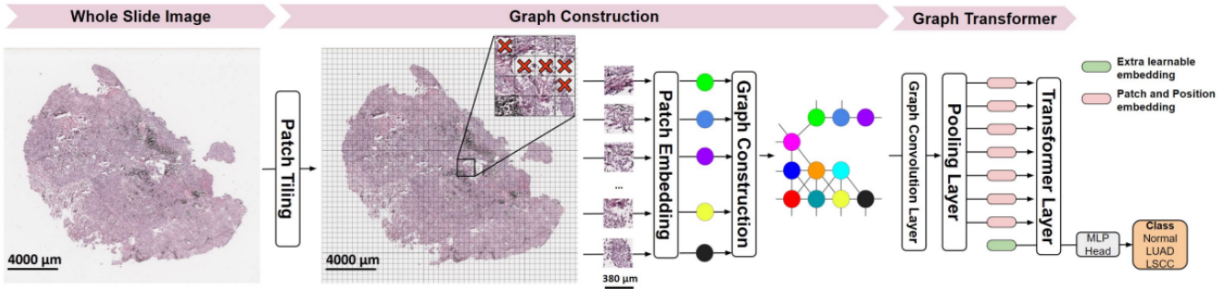


Figure 3.2: Schematic of the graph-transformer.

3.3.1 Division into patches

Our approach to constructing graphs from embryo images involves an initial step of dividing these images into smaller, manageable patches. This method is derived from techniques commonly employed in digital pathology for Whole Slide Images (WSIs), as discussed by Zheng et al. in their exploration of graph-transformer architectures for image classification.[15]

Image Preparation:

High-resolution JPEG images of embryos are prepared by ensuring they are free of artifacts and are of consistent quality to facilitate uniform patch extraction.

Patch Extraction:

- Similar to the technique used for WSIs, each embryo image is systematically divided into non-overlapping patches. This process is crucial as it reduces computational overhead and allows for more focused analysis on smaller sections of the image, which is vital given the detailed and varied morphology of embryos.

- The size and shape of patches can be adjusted based on experimental needs. For our purposes, we utilize square patches of 64 pixels, which simplify the handling and processing of image data.

Refinement of Patch Selection through Variance Thresholding:
To enhance the quality of data for analysis, we exclude patches with a variance in intensity below 200. This step identifies and removes patches that predominantly contain homogeneous or background areas, which are less informative for our analysis.

- *Variance Calculation:* Calculate the variance of pixel intensities for each patch.
- *Threshold Application*: Exclude patches with variance below 200 to focus computational resources on more informative regions of the embryo images.

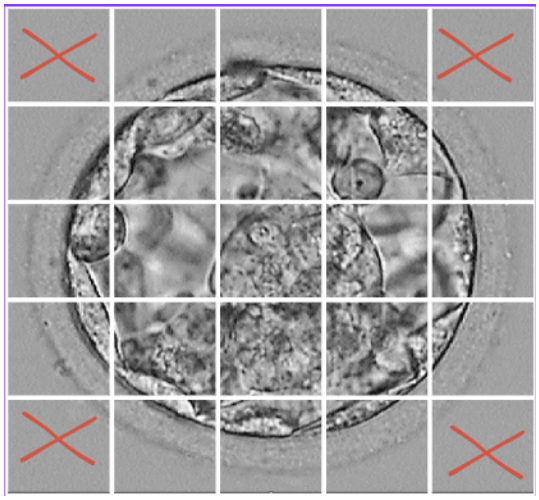


Figure 3.3: Patch Extraction

3.3.2 Extraction of patch features

Each patch is passed through a Convolutional Neural Network (CNN) to extract its features. CNNs are powerful deep learning models commonly used for image analysis tasks due to their ability to automatically learn hierarchical features from raw pixel values.

During this process, the CNN applies a series of convolutional, pooling, and activation layers to the input patch, gradually transforming the raw pixel values into a high-level feature representation.

The features extracted by the CNN capture important visual patterns and structures present in the patch, which can be crucial for subsequent analysis and classification tasks.

1.Contrastive Learning:

We use contrastive learning to train a CNN. This involves maximizing agreement between two differently augmented views of the same image patch via a contrastive loss in the latent space. The goal is to ensure that the embeddings produced by the CNN are consistent and informative.

2.Training Setup:

- We begin by tiling the image from the training set into patches and then randomly sampling a mini-batch of K patches.
- Each patch undergoes two different data augmentation operations, resulting in two augmented patches (pi and pj). These pairs of augmented patches from the same original patch are considered positive pairs. For a mini-batch of K patches, there are a total of 2K augmented patches.

3.Positive and Negative Samples:

Given a positive pair, the other $2K - 1$ augmented patches in the mini-batch are considered as negative samples. This setup allows the network to learn to discriminate between positive and negative pairs.

4.Extraction of Embedding Vectors:

The CNN is then used to extract representative embedding vectors (fi, fj) from each augmented patch (pi, pj).

5.Projection into Latent Space and Loss Calculation:

The embedding vectors are mapped by a projection head to a latent space (zi, zj) where contrastive learning loss is applied. The contrastive learning loss function for a positive pair of augmented patches (i, j) is defined as a function of the similarity

between their embeddings in the latent space.

$$l_{i,j} = -\log\left(\frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2K} \mathbf{1}_{[k\neq j]} \exp(\text{sim}(z_i, z_k)/\tau)}\right) \tag{3.1}$$

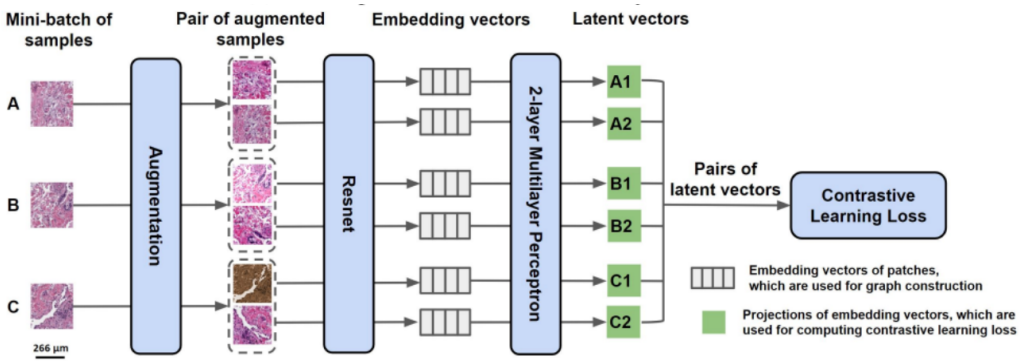


Figure 3.4: Feature generation and contrastive learning

3.3.3 Graph Construction

After training, we utilize the feature extractor to compute the feature vectors of the patches from the image. These feature vectors serve as node features in the graph construction phase. We obtain a node-specific feature matrix F, where each row corresponds to the feature vector of a patch.

We define edges between pairs of nodes based on the spatial location of their corresponding patches on the embryo image. If patch i is a neighbor of patch j, an edge is created between node i and node j, and the corresponding elements in the adjacency matrix are set to 1; otherwise, they are set to 0.

The feature node matrix F and the adjacency matrix A are then used to construct a graph to represent each image.

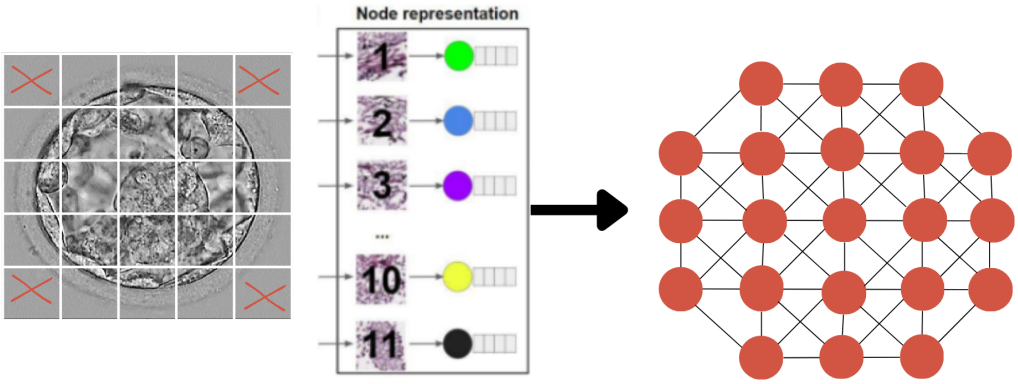


Figure 3.5: Graph Construction

3.4 Classification results with the graph transformer

In the course of evaluating the performance of our graph transformer model, several key metrics were recorded, including training accuracy, validation accuracy, precision, recall, and F1 score. These metrics provide a comprehensive overview of the model's effectiveness in classifying the given data, enabling us to assess both its overall accuracy and its ability to balance between precision and recall.

Metric	**Value**
Train Accuracy	0.9269
Validation Accuracy	0.7600
Precision	0.7197
Recall	0.6151
F1 Score	0.6250

Table 3.1: Performance Metrics

The results indicate that the graph transformer model performs well on the training data with an accuracy of 0.9269, demonstrating effective learning. The validation

accuracy of 0.7600 suggests that the model generalizes well to unseen data, indicating robust performance.

The precision of 0.7197 shows that the model is good at correctly identifying positive instances among its predictions. However, the recall of 0.6151 reveals that the model misses a significant number of actual positives. The F1 score of 0.6250, which balances precision and recall, suggests that the model tends to favor precision.

Overall, the model demonstrates strong performance in several key metrics, but the imbalance between precision and recall indicates room for improvement. Future work should focus on strategies to achieve a better balance between these metrics to enhance the overall effectiveness of the graph transformer model.

3.5 Relative Advantages

The Graph Transformer architecture offers several advantages, especially when dealing with structured data such as graphs.

- Capturing Relational Information: Graph Transformers are designed to handle graphs, making them ideal for applications where the relationships between data points are as important as the data points themselves. This is particularly useful in tasks like molecular analysis, social network analysis, and in your case, embryo classification, where the spatial or structural relationships within the data are crucial.

- Attention Mechanisms: Graph Transformers utilize attention mechanisms that allow the model to focus on the most relevant parts of the graph. This helps in effectively capturing long-range dependencies and complex interactions within the graph, improving the model's ability to learn and generalize from the data.

- Scalability: By leveraging transformer layers, Graph Transformers can scale to larger graphs compared to traditional graph neural networks (GNNs). This scalability makes them suitable for handling large and complex datasets without a significant drop in performance.

- Flexibility: The architecture can be easily adapted to various types of graph-structured data. Whether the nodes and edges represent molecular structures, social connections, or image regions, the Graph Transformer can be customized to fit the specific characteristics of the dataset.
- State-of-the-Art Performance: Graph Transformers have shown competitive performance across various tasks, often outperforming traditional GNNs and other methods in benchmarks. Their ability to model complex interactions and dependencies contributes to their high accuracy and effectiveness.
- Parallel Processing: Like traditional transformers, Graph Transformers can take advantage of parallel processing. This leads to more efficient training and inference, especially on modern hardware like GPUs and TPUs.

3.6 Limitations

While the Graph Transformer architecture offers many advantages, it also has several limitations:

- Computational Complexity: The attention mechanism in Graph Transformers can be computationally expensive, especially for large graphs, leading to high memory and processing requirements.
- Data Requirements: Graph Transformers often require large amounts of data to train effectively, which might not always be available, especially in specialized domains like embryo classification.
- Overfitting: Due to their high capacity, Graph Transformers are prone to overfitting, particularly if the dataset is small or not diverse enough. Proper regularization techniques are essential to mitigate this risk.
- Training Stability: Training Graph Transformers can sometimes be unstable, requiring careful tuning of hyperparameters and optimization strategies to achieve the best performance.

- Complexity of Implementation: Implementing Graph Transformers can be more complex compared to simpler models, requiring a deeper understanding of both graph theory and neural network architectures.

Conclusion

In this section, we successfully implemented a Graph Transformer architecture for the classification of human embryos. By converting embryo images into graph representations, where each patch of the image is treated as a node with features extracted using a CNN (ResNet-18), we were able to leverage the structural information inherent in the images. This approach enabled us to effectively capture the relationships between different regions of the embryo.

The constructed graph was then processed using the Graph Transformer model to predict the state of the embryo. The model achieved an accuracy of 0.76, demonstrating the potential of using Graph Transformers for this complex task. Despite the relatively high computational demands and the need for extensive data, the results indicate that Graph Transformers can provide meaningful insights and improved performance in the classification of embryo images compared to more traditional methods. This work paves the way for further refinements and optimizations, aiming to enhance accuracy and robustness in embryo viability assessments.

General Conclusion

This project has demonstrated a comprehensive exploration of transformer networks and their extension to graph-based structures, culminating in the implementation of a novel Graph Transformer architecture. Our study navigated through the foundational theories of graph theory, the intricacies of transformer mechanisms, and the practical challenges of adapting these models for graph data.

Our results underscore the robustness and versatility of Graph Transformers, particularly in handling complex data structures where relational information is pivotal. This was exemplified in our application to embryo classification, where the Graph Transformer adeptly modeled the relational nuances of spatial structures within the data.

Throughout this project, we leveraged cutting-edge methodologies and technologies, illustrating the potential of Graph Transformers to not only advance academic and research endeavors but also to offer substantial improvements in practical applications across various domains such as bioinformatics, social network analysis, and beyond.

As we conclude, it is clear that while challenges such as computational demands and data scarcity persist, the advancements in graph-based learning—specifically through the lens of transformer networks—herald a promising direction for future research and application. The potential for these models to transform data analytics and machine learning landscapes remains vast and largely untapped.

Bibliography

[1] Niki Parmar Jakob Uszkoreit Llion Jones Aidan N Gomez Łukasz Kaiser Illia Polosukhin Ashish Vaswani, Noam Shazeer. Attention is all you need. 2017.

[2] Zach Blumenfeld. Graph machine learning: An overview. April 2023.

[3] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. 2017.

[4] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. 2005.

[5] M. Grohe and M. Otto. Pebble games and linear equations. *Journal of Symbolic Logic*, 80(3):797–844, 2015.

[6] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. 2016.

[7] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. 2017.

[8] Thomas N. Kipf and Max Welling. Graph convolutional networks for text classification. 2018.

[9] Jure Leskovec. Cs224w: Machine learning with graphs. 2023.

[10] Vinayak Rao Ryan L. Murphy, Balasubramaniam Srinivasan and Bruno Ribeiro. Relational pooling for graph representations. 2019.

[11] Parvaneh Saeedi, Dianna Yee, Jason Au, and Jon Havelock. Automatic identification of human blastocyst components via texture. *IEEE Transactions on Biomedical Engineering*, 64(12):2968–2978, 2017.

[12] Antonio Sanfilippo. Graph. pages 2–3, December 2006.

[13] Anh Tuan Luu Thomas Laurent Yoshua Bengio Xavier Bresson Vijay Prakash Dwivedi, Chaitanya K. Joshi. Benchmarking graph neural networks. 2020.

[14] Xavier Bresson Vijay Prakash Dwivedi. A generalization of transformer networks to graphs. 2021.

[15] Emily J. Green Eric J. Burks Margrit Betke Jennifer E. Beane Yi Zheng, Rushin H. Gindra and Vijaya B. Kolachalama. A graph-transformer for whole slide image classification. 2022.

[16] Amir Barati Farimani Yuyang Wang, Zijie Li. Graph neural networks for molecules. 2023.