ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Natural Language Processing

NEURAL CLUSTERING ON TREE STRUCTURED DATA: A CASE STUDY ON ARGUMENT MINING

CANDIDATE

Andrea Proia

SUPERVISOR

Prof. Paolo Torroni

CO-SUPERVISOR

Dott. Federico Ruggeri

Academic year 2022-2023

Session 1st

Contents

List of Figures

List of Tables

Abstract

This thesis explores the application of Graph Neural Networks based models to investigate the effectiveness in capturing and utilizing tree substructures, named fragments, to improve sentence classification in argument mining.

The analysis focuses on two sub-tasks: Argumentative sentence detection, which involves identifying sentences that contain arguments, and Argument component detection, which consists of identifying and classifying the different components within an argumentative text.

The dissertation presents a comprehensive study of three architecture variations for both sub-tasks: a classification baseline, a metric learning approach and a prototype network approach, evaluated on two separate datasets.

Experimental results reveal that the proposed models achieve satisfactory performance in terms of F1 score: a mean score of 88.35 for USElecDeb60To16 and 63.19 for Persuasive Essays Corpus. However, an analysis of the models' embeddings sparsity highlights that the performance of few models might not be entirely satisfactory and they require further refinement.

Chapter 1

Introduction

1.1 Motivation and Context

In recent years, the field of machine learning has witnessed significant advancements concerning models able to manage graph-structured inputs.

These architectures emerged as a powerful tool to manage data with complex relationships and have been widely studied and extended with numerous variants which have the capability to learn embedding representations of generic graphs. This is accomplished by leveraging aggregation layers that are usually stacked within deep neural networks, and the resulting embeddings can be effectively utilized in various high-level tasks.

Focusing on the application domain of this project, namely Natural Language Processing (NLP), this work explores some approaches to perform clustering of tree fragments applied to the field of Argument Mining that has emerged as a crucial area of study, focusing on the extraction and analysis of arguments from textual data. By understanding and analyzing arguments, valuable insights can be gained into the underlying structures and reasoning within a text, leading to improved information retrieval and text comprehension.

The classification of sentences as argumentative plays an important role in automating the analysis of textual data, allowing for the identification of argumentative fragments that contribute to the overall argumentative structure of a sentence. However, accurately determining the argumentative nature of a sentence is a complex challenge, primarily due to the diverse and often ambiguous nature of language.

Motivated by the need to automate the identification of argumentative fragments within textual data, this thesis proposes a novel approach that leverages GNNs and Pooling layers for clustering of tree fragments. To gain insights into the underlying structures of argumentative sentences, this work explores the application of graph neural networks (GNNs) to analyze the parsing trees of input sentences. By modeling the sentence structure as a graph and utilizing GNNs, we aim to capture the relationships and dependencies between different tree fragments. This approach enables us to uncover common substructures that may provide valuable insights for the classification task.

The research is driven by the growing demand for automated argument mining tools in various domains, such as legal analysis, political discourse analysis, and online debate monitoring. Extracting and analyzing arguments from large volumes of textual data is a labor-intensive and time-consuming task, often requiring the expertise of subject matter experts.

Moreover, while the most common used supervised learning methods have been extensively explored, there is a limited field of research focusing on the use of GNNs for argument mining and the identification of common substructures within argumentative sentences. This thesis aims to contribute to the existing knowledge by investigating the effectiveness of GNN-based models in capturing and utilizing these substructures for improved sentence classification in argument mining.

1.2 Contribution

This work is contextualized within the field of argument mining. Argument mining [19] aims to automatically extract arguments from collections of text:

one crucial aspect of this task involves identifying argument components, such as claims or premises, within sentences. In particular, we will focus on two sub-tasks: argumentative sentence detection with USElecDeb60To16 dataset and argument component detection with Persuasive Essays Corpus. More details regarding the datasets and approaches will be given in the following chapters.

Overall, the problem is defined as a binary classification task, and we propose three different approaches:

- a baseline classification network which serves as a reference and starting point for the other models' architectures. The network takes tree-structured input and processes it through GCN and pooling layers to learn meaningful representations which are then used to make a prediction about the sentence's classification. The baseline model provides a benchmark for evaluating the performance of other models and serves as a baseline comparison for their effectiveness.

- a metric learning approach that aims to improve the classification results by leveraging the triplet loss function. Instead of predicting only the label, this approach focuses on learning a distance metric in a latent space. The network is trained using triplets of sentences: an anchor sentence, a positive sentence (same anchor class), and a negative sentence (different from anchor class). The network learns to map the anchor and positive sentences closer together in the latent space while pushing the negative sentence farther away. By optimizing the triplet loss function, the model aims to improve the separation between different class sentences in the learned latent space, leading ideally to enhanced classification performance.

- a prototype network which is an instance-based learning model built on the concept of prototypes. The idea is to represent each class by a set of prototype instances. In this approach, the network is trained to identify the prototypes that best represent the characteristics of the different class sentences. During training, the network learns to update and refine these prototypes based on the encountered instances. When classifying a new sentence, the network compares it with the prototypes and assigns the class label based on the closest prototype. By leveraging prototypes, the network can capture the essential characteristics and substructures, potentially improving the classification performance.

1.3 Structure of the thesis

To provide a clear understanding of the topic discussed, a general overview of the dissertation is given. The structure foresees 3 chapters.

The first chapter has an introductory scope and focuses on the problem statement, introducing the proposed research questions, in order to give a clear explanation of context and motivations.

Chapter two provides an overview of the task with a specific focus on the background. In this section, we discuss the technical details of the set of technologies employed to implement the solutions, also with an in-depth analysis of the technical details related to the metric used in the metric learning approach.

Finally, in Chapter three, the experimental setup is covered, introducing the datasets that have been selected to perform the experiments and the baseline classification architecture. Then, a specific description of the Metric Learning and the Prototype Learning approaches is given and the results of the experiments are provided. Lastly, the results are compared to evaluate and discuss the key aspects.

Chapter 2

Background

This chapter provides a comprehensive overview of the technologies and concepts that form the foundation of the models and approaches presented in this dissertation, giving a thorough understanding of the tools and techniques will be used throughout the research.

Initially, we describe argument mining. Next, we focus on the technical details related to the Graph Neural Networks and to the concept of Differentiable Pooling. Finally, we explore more deeply the technical component related to the triplet loss that is used in the approach related to metric learning.

2.1 Argument Mining

First of all, let's focus on the application domain of the project.

Argument Mining in Natural Language Processing (NLP) refers to the process of automatically identifying and extracting arguments from textual sources. It involves applying computational techniques and algorithms to understand the structure, content, and relationships between different elements of an argument.

An argument is a unit of discourse that consists of a claim (i.e. main proposition) supported by one or more evidences, and potentially counter-arguments. The goal is to understand the relationships between claims and evidence to gain insights into the underlying reasoning employed in the text. This is achieved through an argument detection step, which employs a wide range of conventional machine learning techniques, followed by an argument structure prediction step [1]. The latter is needed to predict the connections between the previously detected argument components.

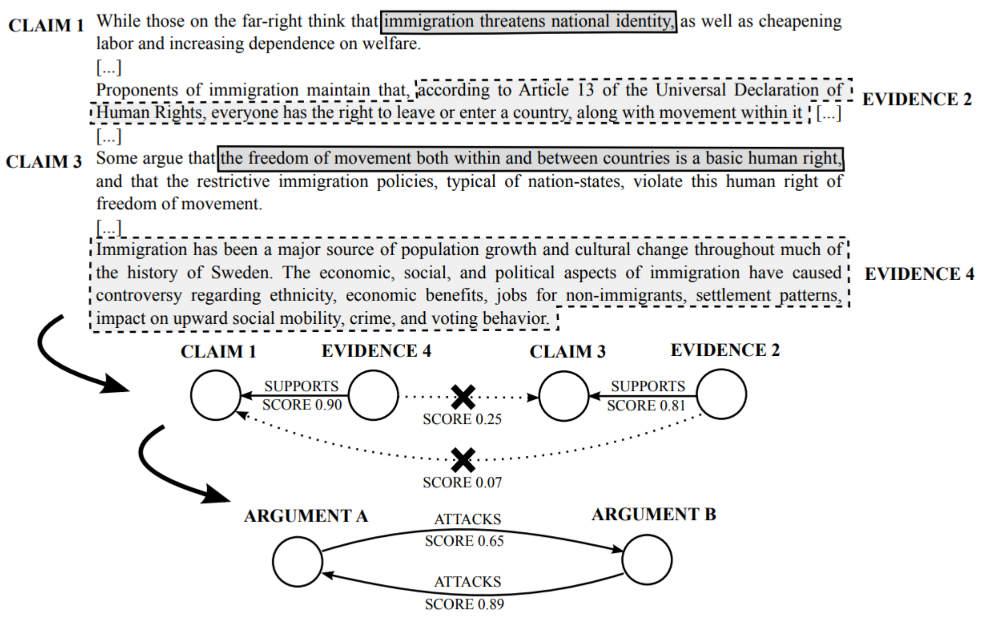

CLAIM 1
While those on the far-right think that immigration threatens national identity, as well as cheapening labor and increasing dependence on welfare.
[...]
Proponents of immigration maintain that, according to Article 13 of the Universal Declaration of Human Rights, everyone has the right to leave or enter a country, along with movement within it [...]
EVIDENCE 2
[...]
CLAIM 3
Some argue that the freedom of movement both within and between countries is a basic human right, and that the restrictive immigration policies, typical of nation-states, violate this human right of freedom of movement.
[...]
Immigration has been a major source of population growth and cultural change throughout much of the history of Sweden. The economic, social, and political aspects of immigration have caused controversy regarding ethnicity, economic benefits, jobs for non-immigrants, settlement patterns, impact on upward social mobility, crime, and voting behavior.
EVIDENCE 4
CLAIM 1
EVIDENCE 4
CLAIM 3
EVIDENCE 2
SUPPORTS
SCORE 0.90
SCORE 0.25
SUPPORTS
SCORE 0.81
SCORE 0.07
ARGUMENT A
ATTACKS
SCORE 0.65
ARGUMENT B
ATTACKS
SCORE 0.89

Figure 2.1: Automatic argument extraction pipeline [10]

We will now proceed giving an insight on the two sub-tasks related with the two datasets employed to perform the experiments: Argumentative sentence detection and Argument component detection.

2.1.1 Argumentative sentence detection

ASD is a NLP task that involves identifying sentences or text segments that contain arguments. The goal is to automatically classify sentences as either argumentative or non-argumentative based on their content and structure. The task of argumentative sentence detection plays a crucial role in argument mining: by identifying argumentative sentences, users can gain insights into the persuasive elements within a text and understand the structure of an argumentative discourse.

The detection process typically involves training a classifier using labeled data, where sentences are annotated as argumentative or non-argumentative. Features such as sentence structure, linguistic patterns, sentiment, and contextual information are often used to distinguish argumentative sentences from non-argumentative ones.

Applications of argumentative sentence detection can be found in various domains. For instance, in political debates or legal contexts [13], identifying argumentative sentences can help in understanding the key points of contention and the strategies employed by speakers or writers to persuade their audience.

For the ASD task, it is used the USElecDeb60To16 dataset.

2.1.2 Argument component detection

Argument component detection is a sub-task within the argument mining field that involves identifying and classifying the different components within an argumentative text or discourse [12].

The goal of argument component detection is to understand the structure and organization of arguments, enabling users to analyze the relationships between different elements and assess the strength or weakness of an argument.

The classification of argument components can vary depending on the specific task and annotation scheme. For example, claims represent the main assertions or conclusions of an argument, while premises provide supporting evidence or reasons.

Argument component detection can be employed in analyzing persuasive essays, online discussions, and many other forms of text where arguments are present.

For the ACD task, it is used the PersuasiveEssays dataset.

2.2 Graph Neural Networks

In this section we will dive into Graph Neural Networks (GNNs), which are a key component employed inside our models to handle tree-structured inputs. GNNs have emerged as a powerful framework for analyzing and modeling structured data that can be represented as graphs [20] [24].

Unlike traditional neural networks designed for grid-like data, such as images or sequences, GNNs operate directly on graph-structured data, making them well-suited for applications where the relationships between entities are critical.

The ability of GNNs to capture and leverage the structural dependencies between entities in graphs has led to significant advancements in various domains, including natural language processing.

Thanks to their ability to handle graphs of varying sizes and structures, they can adapt to graphs with different numbers of nodes and varying connectivity patterns, making them highly flexible and applicable to a wide range of real-world scenarios. Moreover, they can incorporate additional components, such as attention mechanisms or graph pooling layers, to further enhance their expressiveness and performance.

2.2.1 Parse trees

There are different methods to convert textual data into a tree representation. Two standard methods [21], that differ in the type of information they focus on, are:

- **Dependency tree**: represents the grammatical structure of a sentence by establishing dependencies between words. Each word is considered a node in the tree, and the dependencies are represented by directed edges which represent the grammatical relationships (e.g subject, conjunction, etc.). The root of the tree typically represents the main verb or the main clause in the sentence. Overall, dependency trees provide a more fine-grained analysis of the grammatical relationships and the syntactic dependencies between words.

- **Constituency tree**: represents the hierarchical structure of a sentence by grouping words into constituents. It aims to capture the hierarchical organization of a sentence by dividing it into smaller sub-phrases (e.g. noun phrases, verb phrases, etc.). Each node in the tree represents a constituent, which can be a word or a group of words, and the branches represent the relationships between the constituents. The root of the tree represents the entire sentence. Constituency tree focuses more on the hierarchical organization of the sentence.

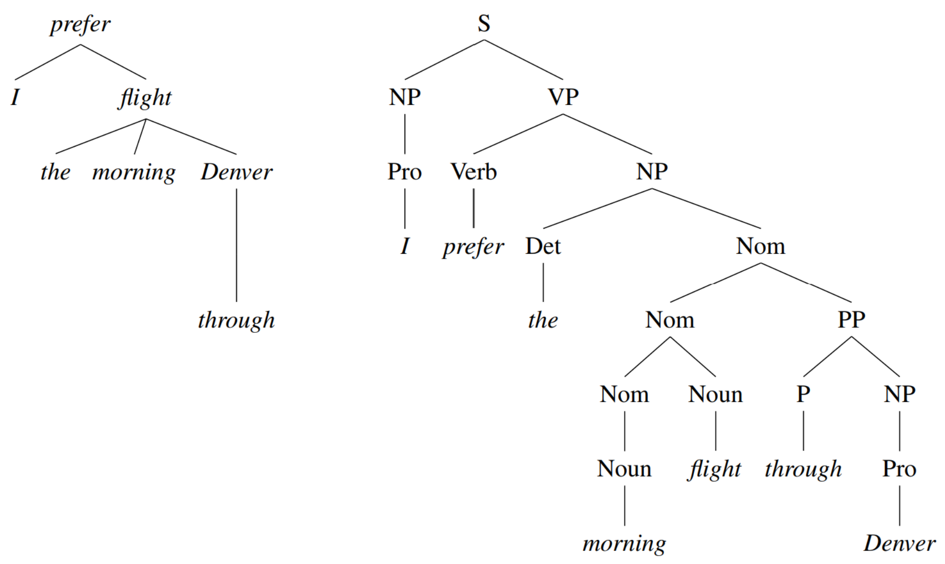


Figure 2.2: Dependency tree vs Constituency tree [1]

The experiments in Chapter 4 are performed using dependency tree as input. The motivation is that dependency trees tend to have a simpler and more straightforward structure compared to constituency trees. They also usually have fewer nodes and edges compared to constituency trees, resulting in more compact representations, and so leading to a more efficient training and faster processing times, especially when working with large-scale datasets.

Our models will consider tree fragments, i.e. smaller substructures or portions of a larger tree. In the context of tree-based models or algorithms, tree fragments are subsets of nodes and edges that capture local structures or patterns within a tree.

When working with tree-based data, such as constituency trees or dependency trees, analyzing the entire tree as a single entity might not be feasible or optimal for certain tasks. Instead, breaking down the tree into smaller fragments can provide more focused insights or facilitate specific operations.

Tree fragments can be defined based on various criteria, such as the depth of the nodes, the presence of specific types of nodes or edges, or the occurrence of certain patterns.

By extracting and analyzing tree fragments, our models can gain a better understanding of the local structures and relationships within a tree.

2.2.2 Graph Convolutional Networks

Among the different architectures within the GNN framework, Graph Convolutional Networks (GCNs) have gained considerable attention for their effectiveness in modeling graph data.

GCNs [8] extend the concept of convolutional layers from grid-like structures to graph structures, enabling them to exploit the connectivity patterns and local neighborhood information of nodes in a graph, making them particularly suitable for tasks that involve node classification, link prediction, etc.

The key idea behind GCNs is to aggregate information from a node's neighboring nodes and use it to update the node's own representation. This process recalls the message passing scheme of graph theory, where nodes exchange information iteratively. By propagating and aggregating information across the graph, GCNs capture both local and global patterns, enabling them to learn rich representations that encode the graph's structural information.

In a typical GCN architecture, each layer performs a graph convolution operation, which involves a weighted aggregation of the features from neighboring nodes, followed by a non-linear activation function. The resulting node representations are then passed to the next layer, allowing the network to refine the representations through multiple iterations. By stacking multiple graph convolutional layers, the model can capture increasingly complex relationships and dependencies.

In this thesis, we will explore the application of GNNs with a specific focus on GCNs to test different network architectures.

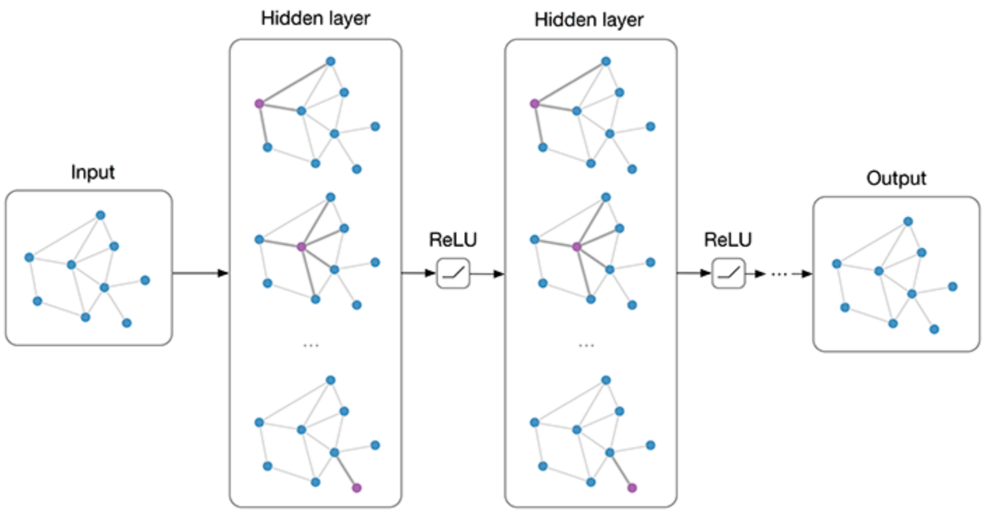


Figure 2.3: GCN architecture [2]

2.2.3 Mathematical Formulation

In order to introduce the theory underlying the GNNs, we first have give a preliminary explanation regarding graph components.

From the definition in [14], given a graph G = (V, E) with:

- V = $v_1, ..., v_n$ set of vertices
- E = $e_1, ..., e_m$ set of edges

a GNN generally employs the following message-passing aggregation function:

$$H^{t+1} = f(A, H^t; \theta^{t+1}) \tag{2.1}$$

where $H^t = \{h_1^t, ..., h_n^t\}$ is the node representation matrix $n \times d$ at time t, where d is the embedding dimensionality. Each node h_i^t is the node embedding vector and A is the binary adjacency matrix given (V, E) and θ^t are model parameters at time t.

This work employs the GCN architecture whose aggregation function can be expressed as follows:

$$H^{t+1} = ReLU(\tilde{A}H^tW^t) \tag{2.2}$$

where D is the degree matrix, $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is the degree normalized adjacency matrix, W^t is a trainable weight matrix.

2.3 Differentiable Pooling

To accomplish specific tasks like graph classification, a hierarchical representation of the input graph is utilized, where nodes are grouped together through node clustering. This process, referred to as node clustering, resembles message-passing and is applied iteratively to determine the building blocks of the graph.

In the field of neural networks, Differentiable Pooling (DiffPool) [25] represents a notable method for performing node clustering in a differentiable manner. DiffPool approaches node clustering as a soft node assignment problem [14], where n input nodes are associated with k node clusters. Given the input node embeddings H in matrix form ($n \times d$), the pooling layer is defined as follows:

$$P = softmax(HW_P) \tag{2.3}$$

where W_P $(d \times k)$ represents a trainable weight matrix, and P $(n \times k)$ denotes the pooling soft assignment matrix. Lastly, the node cluster embeddings $\tilde{H}$ $(k \times d)$ are determined using the following equation:

$$\tilde{H} = P^T H \tag{2.4}$$

Similarly, the adjacency matrix for the new node clusters, denoted as $\tilde{A}$ $(k \times k)$, is constructed based on the previous node adjacency matrix A as follows:

$$\tilde{A} = P^T AP \tag{2.5}$$

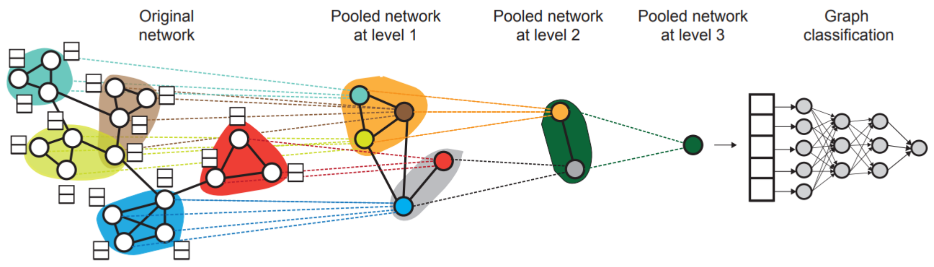

Original network
Pooled network at level 1
Pooled network at level 2
Pooled network at level 3
Graph classification

Figure 2.4: DiffPool [25] representation

2.3.1 Gumbel Softmax

As explained in the previous section, DiffPool views node clustering as a soft assignment task, where nodes are associated with clusters based on their probabilities.

On the other hand, Gumbel Softmax [7] is a method used for differentiable sampling from a categorical distribution. It introduces Gumbel noise and applies the softmax function to obtain a continuous relaxation of the discrete sampling process. Gumbel-Softmax allows for stochasticity in the sampling process while ensuring differentiability, which is crucial for end-to-end training in neural networks.

The idea is that we want to obtain a discretized network to facilitate the interpretation of fragments: a node is either clustered in a group or not at all, instead of having a soft assignment (e.g. 0.30 intensity), but at the same time we ensure that the entire model, including the pooling operation, can be trained jointly with other components since the gradients can flow through the Gumbel Softmax relaxation.

In section 4 we can find results of experiments performed with pooling soft clustering compared with the same performed using the Gumbel Softmax, in order to verify whether it can modify the learning process during training.

2.4 Triplet Loss

Since one of the approaches deals with metric learning, in this last section of the chapter, we need to introduce the concept in order to have a clear overview of the process and particularly on the metric used to optimize the model.

Metric learning [22] [17] focuses on learning a similarity or distance metric between data points. The objective of metric learning is to optimize a model such that it can measure the similarity or dissimilarity between data samples in a way that aligns with the proposed task.

Therefore, it becomes crucial to have an appropriate notion of similarity between data points, which should be specifically tailored to the task.

To address this need, we employ a loss function originally proposed in [15]: the triplet loss, which is a popular loss function used in several deep learning application scenarios to learn effective representations of data in an embedding space [23]. It is commonly used in tasks related to the Computer Vision (CV), such as face recognition [15], but it can be exploited for different kinds of similarity learning tasks.

The goal of the triplet loss is to encourage the network to map similar instances closer together while pushing dissimilar instances apart.

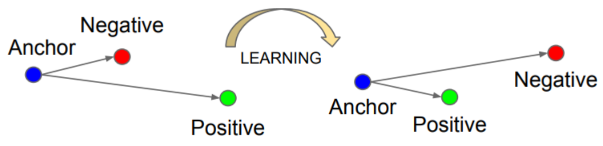


Figure 2.5: Learning goal of the triplet loss [15]

2.4.1 Mathematical Formulation

The theory behind the triplet loss formulation is quite simple, since it relies on a few elements.

The triplet loss operates on triplets of data points: an anchor (A), a positive example (P), and a negative example (N). The anchor and positive example belong to the same class (having ideally similar attributes), while the negative example is from a different class (having ideally dissimilar attributes). The loss is computed based on the distances between these points in the embedding space. Let:

- d(A, P) represent the distance between the anchor (A) and positive example (P) in the embedding space
- d(A, N) represent the distance between the anchor (A) and negative example (N) in the embedding space

The triplet loss (L) is defined as:

$$L = max(d(A, P) - d(A, N) + margin, 0) \tag{2.6}$$

The margin is a hyper-parameter that specifies the minimum desired difference between the distances d(A, P) and d(A, N). This encourages the network to learn embeddings where similar instances are close together, while dissimilar instances are separated by a distance larger than the margin.

Concerning the distance *d* between data points, we have different options such as Euclidean distance, squared Euclidean distance or cosine similarity.

We decided to use the latter since it is computationally efficient to calculate, especially when working with large-scale datasets or high-dimensional feature spaces.

2.4.2 Variants

In the standard triplet loss, triplets are randomly selected during training, which may not provide the most informative examples for learning. For this reason, several variants have been developed to address its limitations and improve its effectiveness in different scenarios. The following list provides some of the commonly used variants:

- **Batch Hard Triplet Loss**: selects the hardest positive example (closest to the anchor) and the hardest negative example (farthest from the anchor) for each anchor within the same mini-batch. By focusing on the hardest examples, this variant encourages the network to learn more distinguished embeddings.
- **Semi-Hard Triplet Loss**: by considering that, in some cases, it can be challenging to find triplets that satisfy the condition

  $$d(A,P) < d(A,N) + margin \tag{2.7}$$

  (especially with large margins), the semi-hard variant addresses this issue by selecting a negative example that is harder than the positive example but still has a positive distance with the anchor. As result, it allows for a more relaxed constraint and avoids trivial solutions while maintaining training stability.

The image 2.6 shows a visual representation to give an insight of how we classify Hard, Semi-hard and Easy negatives in this context.

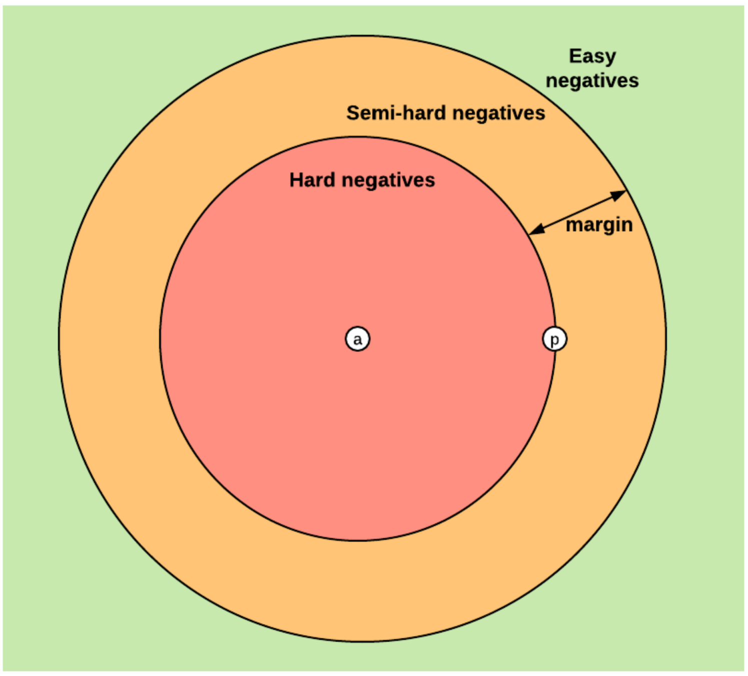


Figure 2.6: Visual representation of possible negatives [3]

Results in section 4 related to the ML_GCN model are relative to tests performed using the Batch Hard version.

2.4.3 Triplet Mining: Offline vs. Online

Understanding the usage of triplet loss means also to analyze the aspect of computational complexity in forming the triplets to be used for training. For this reason, it is important to explain the difference between offline and online triplet mining because they represent two distinct approaches for constructing triplets:

- **Offline triplet mining**: at the beginning of each epoch, we compute all the embeddings on the training set, and then only select hard or semi-hard triplets to train one epoch. Concretely, starting from a list of triplets (i,j,k), we create batches of size B, meaning that we have to compute 3B embeddings to compute the loss and backpropagate into the network. Overall, this technique is not particularly efficient since a full pass on the training set is needed to generate triplets and it also requires to update the offline mined triplets regularly.

- **Online triplet mining**: selecting informative triplets can be computationally expensive, especially in large-scale datasets with a large number of potential triplets. Online triplet mining dynamically selects hard positive and negative examples during training. It starts with an easy triplet, where the negative example is the closest one to the anchor, and iteratively selects harder triplets as the network learns. This approach reduces the computational burden by focusing on the most informative examples, leading to more efficient training.

Chapter 3

Experimental Setup

This chapter provides a comprehensive overview of the dataset used, the methodologies adopted, and the results obtained from the experiments. Furthermore, it includes a comparative analysis of the different approaches to gain insights into their effectiveness and performance.

3.1 Data

Concerning the aspect related to the data, different datasets have been employed to perform the experimental evaluation:

- **USElecDeb60To16 [5]**: political debates gathered from the Commission of Presidential Debates. It includes 39 different transcriptions for a total of approximately 6000 speech turns. (size: 29,621 sentences)

- **Persuasive Essays Corpus (PE) [16]**: set of 402 documents extracted from an online community regarding essays discussion and advise. Documents are annotated at token-level following an argument annotation schema that distinguishes 3 argumentative components: major claim, claim, and premise. (size: 6,089 sentences)

3.2 Baseline Classification

Starting with the baseline, the proposed basic approach is a classification network whose first layer encodes the input, which includes the set of tree nodes and adjacency information. This layer (EMB) processes the input data and prepares it for subsequent operations.

Next, a GCN layer provides graph-specific operations to extract meaningful features from the input. The GCN layer leverages the graph structure and performs aggregation and transformation operations on the node features and their neighboring nodes.

Following the GCN layer, a re-implementation of a pooling layer based on DiffPool [25], that employs a softmax function for sparse input extraction, is provided. This layer selectively extracts small fragments from the input, enhancing interpretability. The result of this pooling layer is an embedding representation of these fragments (F).

To further refine the extracted fragment embeddings, a layer for computing the average of the fragment embeddings is used. This layer aggregates the fragment representations employing a mean pooling operation to obtain a more compact and comprehensive representation of the fragments.

Finally, a dense layer is employed for the final classification task (CLF), which aims to classify the input into two classes. The dense layer takes the aggregated fragment embeddings and performs the classification task, making predictions based on the learned representations.

In this case the loss is straightforward:

$$L = CE \tag{3.1}$$

where *L* represents the total loss which is directly equal to *CE*, i.e.the standard Cross-Entropy for penalizing the misclassification.

Figure 3.1 provides a simple visual representation of the classification baseline (CLF-GCN) architecture described in this section.

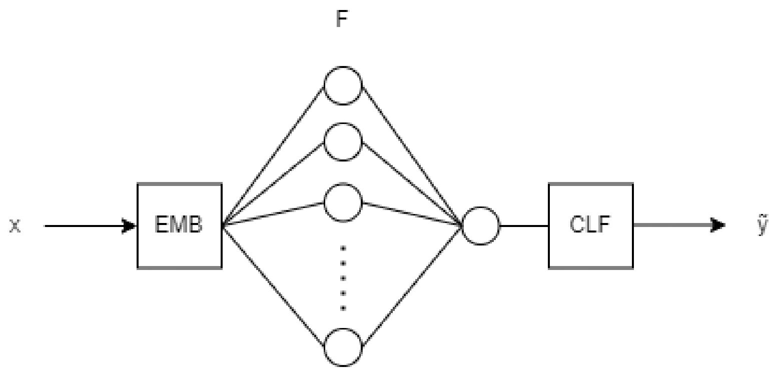


Figure 3.1: CLF-GCN architecture

3.3 Metric Learning

The second approach provides a modified version of the baseline classification model. In addition to the baseline architecture described above, this version of the neural network model incorporates a metric learning component that computes the triplet loss (see Section 2.4) and combines it with the classification result. This enhancement aims to further improve the network's performance in learning discriminative embeddings.

In addition to the dense layer used for classification, the triplet loss is computed to promote better class separation and to enhance the discriminative power of the learned representations. The triplet loss adjusts the embedding space by minimizing the distance between the anchor and positive examples while maximizing the distance between the anchor and negative examples.

By combining the classification output with the computed triplet loss, this version of the model should benefit from both the discriminative power of the embeddings learned through metric learning and the classification capability of the dense layer. This allows the network to not only classify the input into two classes but also generate embeddings that capture meaningful relationships between fragments, improving the overall performance of the task.

Technically, the triplet loss is computed using the angular distance (cosine similarity) between data points of the triplets with a *margin* parameter of 0.1. Moreover, the total loss is computed as:

$$L = CE + \lambda_{tl} \cdot TL \tag{3.2}$$

where *L* represents the total loss, *CE* is the standard Cross-Entropy for penalizing the misclassification, and *TL* is the Triplet Loss computed as in 2.6, with $\lambda_{tl} = 0.1$ as a real-valued hyperparameter.

3.4 Prototype Learning

The third approach is based on Prototype learning an foresees a quite different structure with respect to the previous, both from a conceptual and architectural point of view.

A Prototype neural network (PNN) [9] is a type of model that operates based on the concept of prototypes. It belongs to the family of instance-based learning algorithms, where the model stores a set of representative examples (prototypes) and uses them for classification or regression tasks.

In a PNN, each prototype represents a specific class or category in the dataset. During training, the model learns to associate the prototypes with their corresponding class labels. The prototypes can be thought of as reference points or representatives of each class in the input space.

The process of training a PNN usually involves two main steps: first a Prototype Selection step is required to let the model select a subset of prototypes from the training data that best represent the underlying classes. This can be done using various methods, such as random selection, clustering algorithms, or distance-based approaches.

This is followed by a Prototype Assignment step: once the prototypes are selected, the model assigns class labels to them based on their nearest neighbors in the training data, so when presented with a new input, the model compares it to the prototypes and assigns the class label of the nearest prototype as the predicted class for the input.

The distance metric used for prototype assignment can vary depending on the problem and data characteristics. Commonly used distance measures include Euclidean distance, Manhattan distance, or cosine similarity.

Prototype neural networks have the advantage of interpretability, as the prototypes can provide insights into the characteristics of each class.

However, PNNs may struggle with certain scenarios since they are sensitive to noise in the training data, as noisy prototypes can lead to misclassifications.

Our PNN implementation has the same architecture as the previous models up to the pooling layer. Once the fragments $F = \{f_1, ..., f_n\}$ are obtained, the prototypes $P = \{p_1, ..., p_m\}$ are defined. These are used to compute the distances from fragments F to obtain the contributions R1 and R2 used to determine the total loss.

Formally, the cost function, denoted by *L*, on train data *D* used to train our network, is given by:

$$L = CE + \lambda_1 \cdot R_1(p_1, ..., p_m, D) + \lambda_2 \cdot R_2(p_1, ..., p_m, D) \tag{3.3}$$

where λ_1, λ_2 are real-valued hyperparameters that adjust the ratios between terms, in this case set to $\lambda_1 = \lambda_2 = 0.1$ and *CE* the standard Cross-Entropy for penalizing the misclassification.

The two interpretability regularization terms *R1* and *R2* are formulated as follows:

$$R_1(p_1, ..., p_m, D) = \frac{1}{m} \sum_{j=1}^{m} \min_{i \in [1,n]} \|p_j - f(x_i)\|_2^2 \tag{3.4}$$

$$R_2(p_1, ..., p_m, D) = \frac{1}{n} \sum_{i=1}^{n} \min_{j \in [1,m]} \|f(x_i) - p_j\|_2^2 \tag{3.5}$$

Figure 3.2 provides a simple visual representation of the Prototype network (Proto-GCN) architecture described in this section.

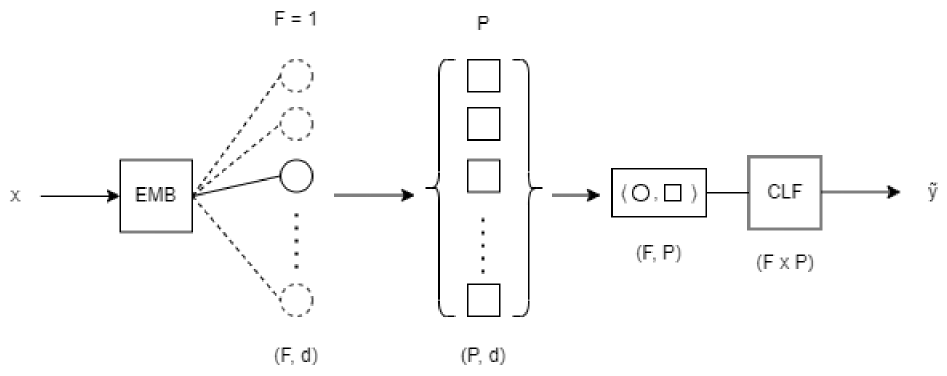


Figure 3.2: Proto-GCN architecture

Chapter 4

Discussion of results

The results found in the following tables are related to experiments performed using an EarlyStopping callback with a *patience* parameter of 5. All models were trained using a 10-fold cross-validation.

4.1 USElecDeb60To16

Table 4.1 presents the parameter configuration for the models tested on UsElecDeb60to16.

Table 4.1: Parameter configuration for UsElecDeb60to16

Param	**Value**
l2_regularization	1e-05
dropout_rate	0.4
input_dropout_rate	0.3

Table 4.2 reports the binary F1-score concerning the argumentative (*arg*) class.

Table 4.2: Classification performance on USElecDeb60To16

Model	**F1**	**Node selection ratio**
Soft		
CLF-GCN (F=2)	$88.79_{\pm 1.11}$	$13.48_{\pm 1.45}$
CLF-GCN (F=8)	$88.96_{\pm 1.09}$	$10.50_{\pm 2.80}$
ML-GCN (F=2)	$88.77_{\pm 0.98}$	$15.49_{\pm 1.24}$
ML-GCN (F=8)	$88.83_{\pm 1.18}$	$11.46_{\pm 3.32}$
Proto-GCN (F=1, P=2)	$87.92_{\pm 1.45}$	$1.03_{\pm 0.34}$
Proto-GCN (F=1, P=8)	$87.19_{\pm 1.20}$	$0.67_{\pm 0.16}$
Gumbel		
CLF-GCN (F=2)	$\mathbf{89.03}_{\pm 1.17}$	$60.08_{\pm 10.60}$
CLF-GCN (F=8)	$88.81_{\pm 1.19}$	$31.76_{\pm 8.00}$
ML-GCN (F=2)	$88.70_{\pm 1.39}$	$100.00_{\pm 0.01}$
ML-GCN (F=8)	$88.53_{\pm 1.35}$	$100.00_{\pm 0.00}$
Proto-GCN (F=1, P=2)	$87.38_{\pm 1.46}$	$1.02_{\pm 0.73}$
Proto-GCN (F=1, P=8)	$87.33_{\pm 0.92}$	$48.03_{\pm 6.03}$

4.2 Persuasive Essays Corpus

Table 4.3 presents the parameter configuration for the models tested on PersuasiveEssays:

Table 4.3: Parameter configuration for PersuasiveEssays

Param	**Value**
l2_regularization	1e-05
dropout_rate	0.4
input_dropout_rate	0.3

Table 4.4 reports the macro F1-score average over 10 fold runs.

4.3 Analysis

The results obtained from the evaluation of the classification networks showcase interesting findings.

Regarding the USElecDeb60To16, the F1 score indicates that all three

Table 4.4: Classification performance on PersuasiveEssays

Model	**F1**	**Node selection ratio**
Soft		
CLF-GCN (F=2)	$64.76_{\pm 0.90}$	$17.40_{\pm 8.82}$
CLF-GCN (F=8)	$64.36_{\pm 0.91}$	$21.25_{\pm 7.23}$
ML-GCN (F=2)	$65.41_{\pm 1.10}$	$30.97_{\pm 5.29}$
ML-GCN (F=8)	$\mathbf{65.72}_{\pm 1.23}$	$34.04_{\pm 4.34}$
Proto-GCN (F=1, P=2)	$61.38_{\pm 0.62}$	$16.08_{\pm 3.45}$
Proto-GCN (F=1, P=8)	$64.75_{\pm 0.74}$	$25.67_{\pm 0.78}$
Gumbel		
CLF-GCN (F=2)	$64.47_{\pm 1.32}$	$32.12_{\pm 15.84}$
CLF-GCN (F=8)	$65.33_{\pm 0.95}$	$21.84_{\pm 6.33}$
ML-GCN (F=2)	$61.13_{\pm 11.27}$	$100.00_{\pm 0.00}$
ML-GCN (F=8)	$55.50_{\pm 12.03}$	$99.94_{\pm 0.17}$
Proto-GCN (F=1, P=2)	$61.63_{\pm 2.31}$	$74.91_{\pm 29.42}$
Proto-GCN (F=1, P=8)	$63.89_{\pm 1.35}$	$100.00_{\pm 0.00}$

tested models perform comparably well. The similarity in F1 scores across the models suggests that they are quite equally effective in correctly classifying the samples.

On one hand, this is a positive outcome, as it demonstrates the robustness and consistency of the classification task across different model architectures. On the other hand, it is an evidence of the fact that the additional contributions from the triplet loss in the ML-GCN model and R1, R2 in the Proto-GCN model do not represent a contribution of such importance as to cause a marked improvement in performance.

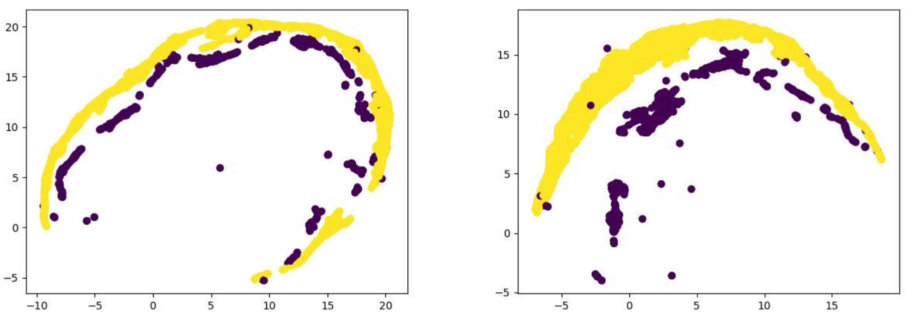
Overall, from a strictly quantitative point of view, the best F1 performance was achieved by the CLF-GCN model with a value of fragments F=2 and using the Gumbel-softmax.

Regarding the latter, we can make an interesting consideration related to the Node selection ratio. In fact, it is evident that almost all the experiments performed using the *Gumbel* function instead of the *Soft* alternative show a significantly higher node selection ratio. The biggest difference can be seen by taking into consideration the comparison between the results of the ML-GCN models that, although showing an almost identical F1 value, differ considerably in the node selection ratio (F=2: 15.49 $\rightarrow$ 100.00, F=8: 11.46 $\rightarrow$ 100.00).

The results of the experiments reveal significant variations in the node selection ratio across the different models, indicating the need for an in-depth analysis of the embeddings to gain insights into the underlying factors driving these discrepancies.

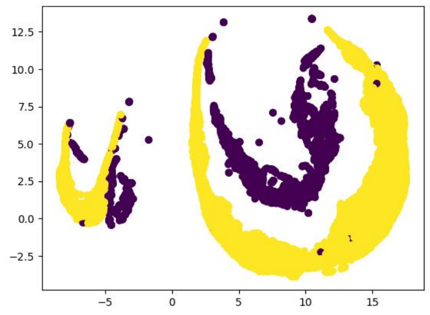
In order to visualize the embeddings produced by each model, a dimensionality reduction techniques is needed. In this case, UMAP [11, 4] is employed to project the high-dimensional embeddings into a lower-dimensional space. The resulting visualizations enable a qualitative assessment of the embeddings' distribution to understand the separability characteristics. We color the embeddings based on their class labels to better identify patterns inside the embedding projections of the model.

The images reported show a *Soft* vs. *Gumbel* comparison between the embeddings of the models. They refer to the first fold of the test set of USElec dataset.



(a) CLF-GCN (F=2 , *Soft*) (b) CLF-GCN (F=2 , *Gumbel*)

Figure 4.1: Comparison between CLF-GCN embeddings (F=2, USElec)


(a) CLF-GCN (F=8 , *Soft*)

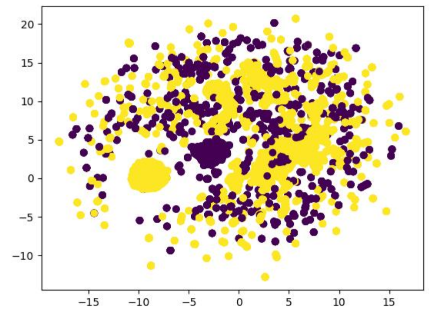
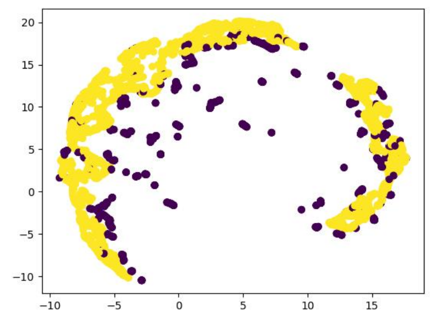

(b) CLF-GCN (F=8 , *Gumbel*)

Figure 4.2: Comparison between CLF-GCN embeddings (F=8, USElec)


(a) ML-GCN (F=2 , *Soft*)

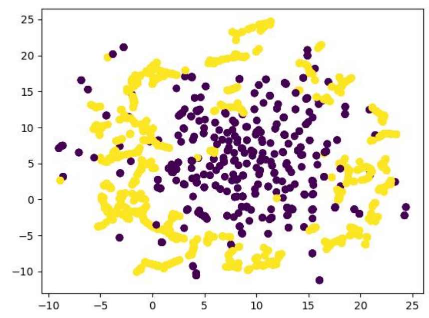
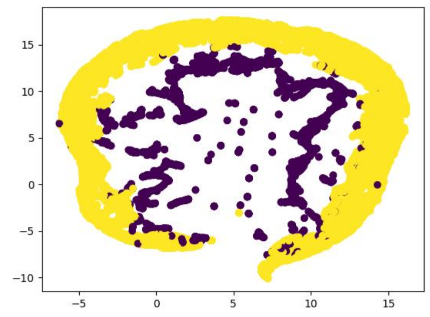

(b) ML-GCN (F=2 , *Gumbel*)

Figure 4.3: Comparison between ML-GCN embeddings (F=2, USElec)


(a) ML-GCN (F=8 , *Soft*)

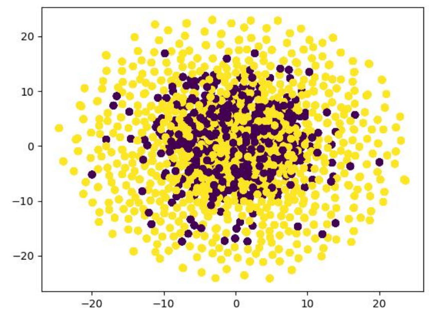
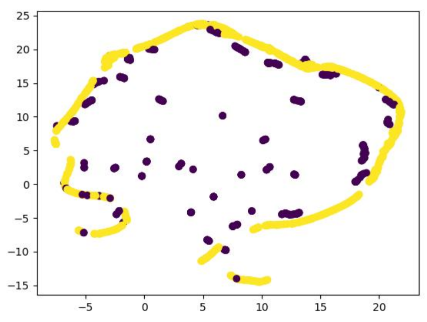
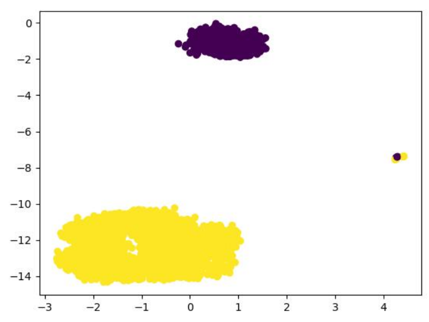

(b) ML-GCN (F=8 , *Gumbel*)

Figure 4.4: Comparison between ML-GCN embeddings (F=8, USElec)

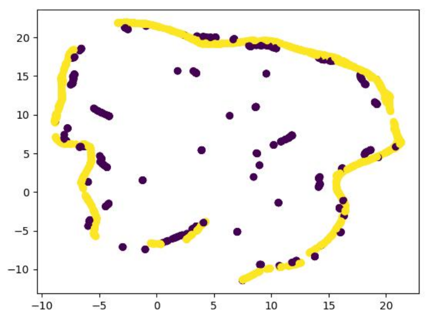


(a) Proto-GCN (*F*=1, *P*=2, *Soft*)

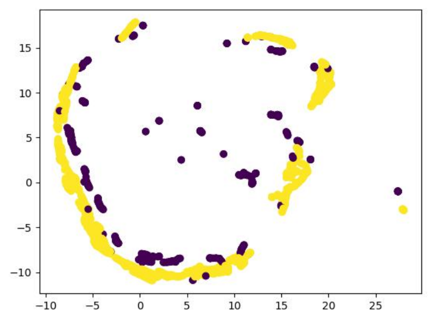


(b) Proto-GCN (*F*=1, *P*=2, *Gumbel*)

Figure 4.5: Comparison between Proto-GCN embeddings (*F*=1, *P*=2, USElec)



(a) Proto-GCN (*F*=1, *P*=8, *Soft*)



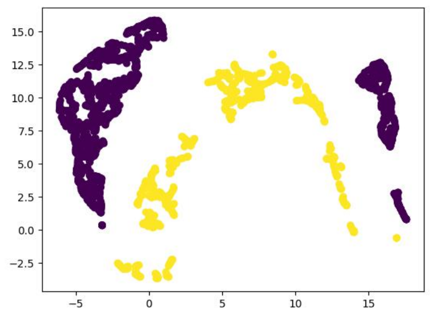
(b) Proto-GCN (*F*=1, *P*=8, *Gumbel*)

Figure 4.6: Comparison between Proto-GCN embeddings (*F*=1, *P*=8, USElec)

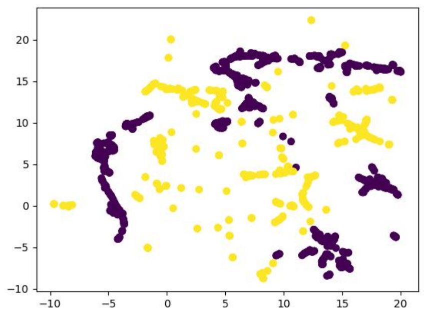
The results for Persuasive Essays instead show an F1 score that tends to be stably lower with respect to USElecDeb60To16, although quite similar among all models as in the previous case. This may be due to the intrinsic characteristics of the dataset itself as well as the fact that it may be less suitable for this kind of task performed with the specific models defined before.

In addition, the same consideration previously made regarding the node selection ratio can also be made in this case: we find significantly higher values in the experiments carried out using the *Gumbel*. In this specific scenario, an additional consideration must be made since, on average, the values of node selection ratio are higher in comparison to those of USElecDeb60To16, particularly considering the ”*Soft*” section of the two tables. In fact, for US-ElecDeb60To16, we find an average node selection ratio of 8.77 while for PersuasiveEssays the same parameter has a value of 24.24.

The images reported show a *Soft* vs. *Gumbel* comparison between the embeddings of the models. They refer to the first fold of the test set of PE dataset.

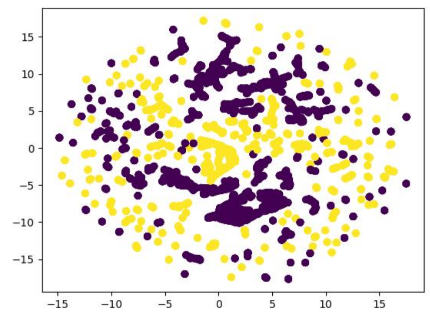


(a) CLF-GCN (*F*=2 , *Soft*)

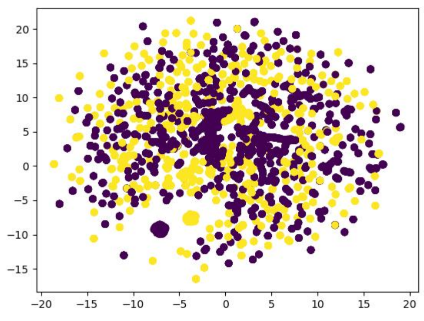


(b) CLF-GCN (*F*=2 , *Gumbel*)

Figure 4.7: Comparison between CLF-GCN embeddings (*F*=2, PE)

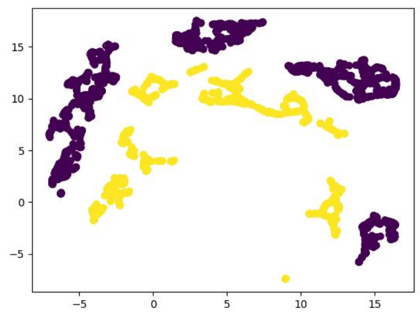


(a) CLF-GCN (*F*=8 , *Soft*)

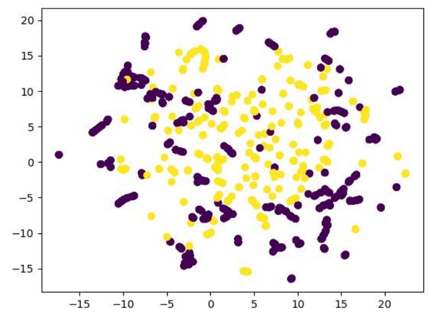


(b) CLF-GCN (*F*=8 , *Gumbel*)

Figure 4.8: Comparison between CLF-GCN embeddings (*F*=8, PE)

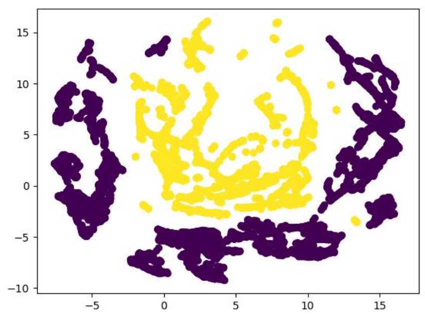


(a) ML-GCN (*F*=2 , *Soft*)

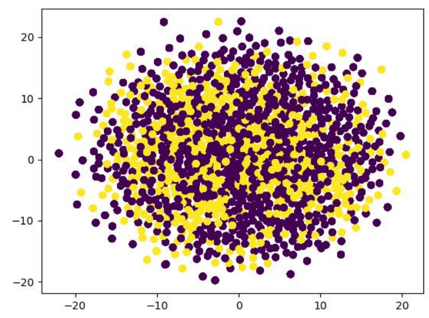


(b) ML-GCN (*F*=2, *Gumbel*)

Figure 4.9: Comparison between ML-GCN embeddings (*F*=2, PE)

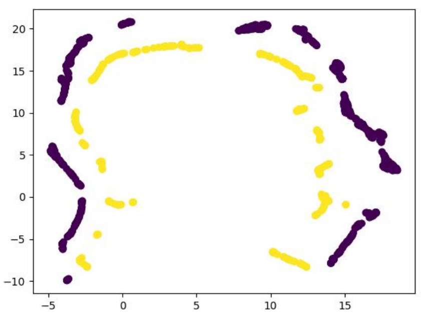


(a) ML-GCN (*F*=8 , *Soft*)

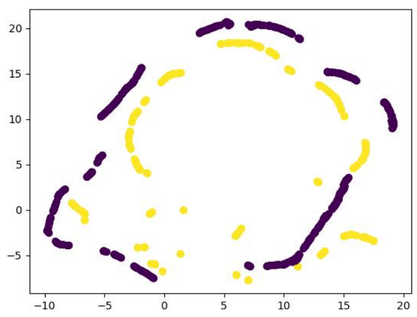


(b) ML-GCN (*F*=8, *Gumbel*)

Figure 4.10: Comparison between ML-GCN embeddings (*F*=8, PE)

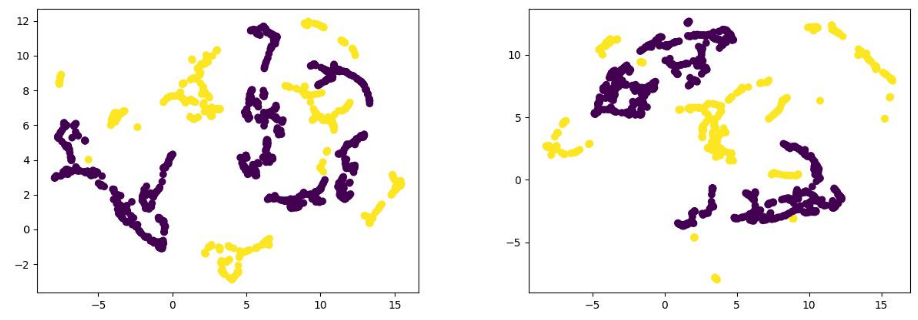


(a) Proto-GCN (*F*=1, *P*=2, *Soft*)



(b) Proto-GCN (*F*=1, *P*=2, *Gumbel*)

Figure 4.11: Comparison between Proto-GCN embeddings (*F*=1, *P*=2, PE)



(a) Proto-GCN (*F*=1, *P*=8, *Soft*) (b) Proto-GCN (*F*=1, *P*=8, *Gumbel*)

Figure 4.12: Comparison between Proto-GCN embeddings (*F*=1, *P*=8, PE)

Interestingly, when considering the sparsity of the embeddings, a notable observation which is referable to the ML-GCN results of both datasets, is the near-zero sparsity when using the Gumbel-Softmax trick. This indicates that the approach with metric learning might not be entirely satisfactory and requires further refinement. The lack of sparsity suggests that the embeddings obtained through the metric learning framework are not effectively separating the nodes into distinct clusters. This highlights the need to investigate alternative variations of the model architecture or loss functions to address this limitation and enhance the discriminative power of the embeddings.

Conclusion

We presented different architectures based on GNNs and pooling layers: a classification baseline, and two distinct variations with conceptual and architectural differences.

We considered two sub-tasks of argument mining to perform experiments on two distinct datasets in order to explore the potentiality of our models. This scenario came up as an interesting and challenging case study for our approaches, showing that it is not easy to produce a remarkable enhancement using the proposed architectures. In the future, it is feasible to extend the work in several directions.

First of all, it is possible to modify the model using other variations of GNNs such as Graph Attention Networks (GATs) [18] that allow to selectively focus on relevant nodes during information aggregation assigning different importance weights to different neighbors of a node. This could potentially enable capturing more fine-grained and context-aware information from the graph-structured input, allowing to better capture long-range dependencies.

Second, concerning the metric learning methodology, we can use the Semi-Hard triplet loss instead of the Hard variant, to see whether better results with more significant embeddings can be achieved. In addition, other loss functions can be explored. Investigate alternative loss functions, such as contrastive loss [6], can be useful to compare their effectiveness in learning discriminative embeddings, evaluating their impact on the model's performance in order to assess their suitability for this specific application domain.

Lastly, we can expand the scope of our approach by testing it on other domains or sub-tasks. Applying our models to diverse contexts can help assess their generalizability and potential for broader applicability.

By exploring these future directions, we aim to enhance the effectiveness and versatility of our architectures while advancing the understanding of graph-based models in different domains.

Acknowledgements

I would like to express my deepest gratitude to my esteemed supervisor, Prof. Torroni, and my co-supervisor, Dott. Ruggeri, for their invaluable guidance and support throughout the entire process of completing this thesis. Their expertise and encouragement have been instrumental in the successful completion of this research.

The original idea behind this thesis emerged from Prof. Torroni and Dott. Ruggeri's projects, and I am honored to have had the chance to give a small contribution to their research field.

I am grateful to Dott. Ruggeri for his invaluable contributions to this thesis. His extensive knowledge and ideas have been essential in expanding the scope and enhancing the quality of this work. His valuable insights and continuous support in guiding me through the research process have been truly significant.

I extend my deepest appreciation to Prof. Torroni for his profound impact on my academic journey. His way of teaching has offered me a great deal of knowledge and raised my interest in the subject matter to the extent that I decided to expand my knowledge, exploring deeper into the concepts through this research project.

Bibliography

[1] URL: https://virtuale.unibo.it/pluginfile.php/1060903/mod_resource/content/1/NLP_course_slides_2021-09.pdf.

[2] URL: https://tkipf.github.io/graph-convolutional-networks/.

[3] URL: https://omoindrot.github.io/triplet-loss.

[4] URL: https://github.com/lmcinnes/umap.

[5] S. Haddadan, E. Cabrio, and S. Villata. Yes, we can! mining arguments in 50 years of US presidential campaign debates. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4684–4690, Florence, Italy. Association for Computational Linguistics, July 2019. DOI: 10.18653/v1/P19-1463. URL: https://aclanthology.org/P19-1463.

[6] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742, 2006. DOI: 10.1109/CVPR.2006.100.

[7] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax, 2017. arXiv: 1611.01144 [stat.ML].

[8] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks, 2017. arXiv: 1609.02907 [cs.LG].

[9] O. Li, H. Liu, C. Chen, and C. Rudin. Deep learning for case-based reasoning through prototypes: a neural network that explains its predictions, 2017. arXiv: 1710.04806 [cs.AI].

[10] M. Lippi and P. Torroni. Argumentation mining: state of the art and emerging trends. *ACM Trans. Internet Technol.*, 16(2), March 2016. ISSN: 1533-5399. DOI: 10.1145/2850417. URL: https://doi.org/10.1145/2850417.

[11] L. McInnes, J. Healy, and J. Melville. Umap: uniform manifold approximation and projection for dimension reduction, 2020. arXiv: 1802.03426 [stat.ML].

[12] J.-C. Mensonides, S. Harispe, J. Montmain, and V. Thireau. Automatic Detection and Classification of Argument Components using Multi-task Deep Neural Network. In *3rd International Conference on Natural Language and Speech Processing*, Trento, Italy, September 2019. URL: https://hal.science/hal-02292945.

[13] M.-F. Moens, E. Boiy, R. Mochales, and C. Reed. Automatic detection of arguments in legal texts. In pages 225–230, June 2007. DOI: 10.1145/1276318.1276362.

[14] F. Ruggeri, M. Lippi, and P. Torroni. Tree-constrained graph neural networks for argument mining. *CoRR*, abs/2110.00124, 2021. arXiv: 2110.00124. URL: https://arxiv.org/abs/2110.00124.

[15] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: a unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, 2015. DOI: 10.1109/CVPR.2015.7298682.

[16] C. Stab and I. Gurevych. Parsing argumentation structures in persuasive essays. *Computational Linguistics*, 43(3):619–659, September 2017. DOI: 10.1162/COLI_a_00295. URL: https://aclanthology.org/J17-3005.

[17] J. L. Suárez-Díaz, S. García, and F. Herrera. A tutorial on distance metric learning: mathematical foundations, algorithms, experimental analysis, prospects and challenges (with appendices on mathematical background and detailed algorithms explanation), 2020. arXiv: 1812.05944 [cs.LG].

[18] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks, 2018. arXiv: 1710.10903 [stat.ML].

[19] Wikipedia contributors. Argument mining — Wikipedia, the free encyclopedia, 2023. URL: https://en.wikipedia.org/w/index.php?title=Argument_mining&oldid=1136432600. [Online; accessed 4-July-2023].

[20] Wikipedia contributors. Graph neural network — Wikipedia, the free encyclopedia, 2023. URL: https://en.wikipedia.org/w/index.php?title=Graph_neural_network&oldid=1153360761. [Online; accessed 4-July-2023].

[21] Wikipedia contributors. Parse tree — Wikipedia, the free encyclopedia, 2023. URL: https://en.wikipedia.org/w/index.php?title=Parse_tree&oldid=1159350944. [Online; accessed 4-July-2023].

[22] Wikipedia contributors. Similarity learning — Wikipedia, the free encyclopedia, 2023. URL: https://en.wikipedia.org/w/index.php?title=Similarity_learning&oldid=1160866705. [Online; accessed 4-July-2023].

[23] Wikipedia contributors. Triplet loss — Wikipedia, the free encyclopedia, 2023. URL: https://en.wikipedia.org/w/index.php?title=Triplet_loss&oldid=1159315260. [Online; accessed 4-July-2023].

[24] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, January 2021. DOI:

10.1109/tnnls.2020.2978386. URL: https://doi.org/10.1109%2Ftnnls.2020.2978386.

[25] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling, 2019. arXiv: 1806.08804 [cs.LG].