Using Graph Convolutional Networks for Approximate Reasoning with Abstract Argumentation Frameworks: A Feasibility Study

Isabelle Kuhlmann and Matthias Thimm

University of Koblenz-Landau

Abstract. We employ graph convolutional networks for the purpose of determining the set of acceptable arguments under preferred semantics in abstract argumentation problems. While the latter problem is complexity-wise one of the hardest problems in reasoning with abstract argumentation problems, approximate methods are needed here in order to obtain a practically relevant runtime performance. This first study shows that deep neural network models such as graph convolutional networks significantly improve the runtime while keeping the accuracy of reasoning at about 80% or even more.

Keywords: neural network · reasoning · abstract argumentation.

1 Introduction

Computational models of argumentation [3] are approaches for non-monotonic reasoning that focus on the interplay between arguments and counterarguments in order to reach conclusions. These approaches can be divided into either *abstract* or *structured* approaches. The former encompass the classical abstract argumentation frameworks following Dung [9] that model argumentation scenarios by directed graphs, where vertices represent arguments and directed links represent attacks between arguments. In these graphs one is usually interested in identifying *extensions*, i. e., sets of arguments that are mutually acceptable and thus provide a coherent perspective on an outcome of the argumentation. On the other hand, structured argumentation approaches consider arguments to be collections of formulas and/or rules which entail some conclusion. The most prominent structured approaches are ASPIC+ [21], ABA [26], DeLP [13], and *deductive argumentation* [4]. These approaches consider a knowledge base of formulas and/or rules as a starting point.

In this paper, we are interested in approximate methods to reasoning with abstract argumentation approaches. Previous works on reasoning with abstract argumentation focus mostly on *sound* and *complete* methods, see e. g. [5] for a recent survey and the International Competition on Computational Models of Argumentation[1] (ICCMA) [25,12] for actual implementations. To the best

[1] `http://argumentationcompetition.org`

of our knowledge, the only incomplete algorithms for abstract argumentation are [24,22] that use stochastic local search. Here, we use deep neural networks to model the problem of deciding (credulous) acceptability of arguments wrt. preferred semantics as a classification problem. We train a graph convolutional neural network [17]—a special form of a convolutional neural network that is tailored towards processing of graphs—with data obtained by random generation of abstract argumentation frameworks and annotated by a sound and complete solver (in our case *CoQuiAAS* [19]). After training, the obtained classifier can be used to solve the acceptability problem in *constant* time. However, the obtained classifier provides only an approximation to the actual answer. Our experiments showed that approximation quality is about 80 % in general, while it can be up to 99 % in certain cases.

The remainder of this paper is structured as follows. In Section 2, the basic concepts of abstract argumentation and artificial neural networks are recalled. Section 3 explains the approach of representation the acceptability problems as a classification problem. Section 4 describes our experimental evaluation and discusses its results. We conclude in Section 5 with a discussion and summary.

2 Preliminaries

In the following, we recall basic definitions of abstract argumentation and artificial neural networks.

2.1 Abstract Argumentation

An abstract argumentation framework [9] AF is a tuple $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ where Arg is a set of arguments and $\rightarrow \subseteq \mathsf{Arg} \times \mathsf{Arg}$ is the attack relation.

Semantics are given to abstract argumentation frameworks by means of *extensions*. A set of arguments $E \subseteq \mathsf{Arg}$ is called an extension if it fulfils certain conditions. There are various types of extensions, however this paper will be focused on the four classical types proposed by Dung [9]. Namely, these are *complete*, *grounded*, *preferred*, and *stable* semantics. All of these types of extensions must be *conflict-free*. A set of arguments $E \subseteq \mathsf{Arg}$ in an argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ is conflict-free, iff there are no arguments $\mathcal{A},\mathcal{B} \in E$ with $\mathcal{A} \rightarrow \mathcal{B}$.

Moreover, an argument $\mathcal{A}$ is called *acceptable* with respect to a set of arguments $E \subseteq \mathsf{Arg}$ iff for every $\mathcal{B} \in \mathsf{Arg}$ with $\mathcal{B} \rightarrow \mathcal{A}$ there is an argument $\mathcal{A}' \in E$ with $\mathcal{A}' \rightarrow \mathcal{B}$. Based on these definitions, the four different types of extensions are defined for an argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ as follows:

1. **Complete extension:** A set of arguments $E \subseteq \mathsf{Arg}$ is called a complete extension iff it is conflict-free, all arguments $\mathcal{A} \in E$ are acceptable with respect to E and there is no argument $\mathcal{B} \in \mathsf{Arg} \setminus E$ that is acceptable with respect to E.
2. **Grounded extension:** A set of arguments $E \subseteq \mathsf{Arg}$ is called a grounded extension iff it is complete and E is minimal with respect to set inclusion.

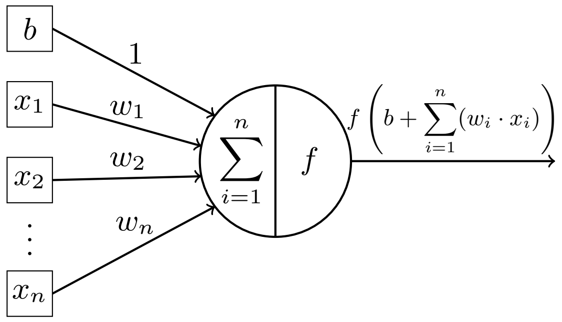

Fig. 1. Artificial neuron, adapted from `https://inspirehep.net/record/1300728/plots`

3. **Preferred extension:** A set of arguments $E \subseteq \mathsf{Arg}$ is called a preferred extension iff it is complete and E is maximal with respect to set inclusion.
4. **Stable extension:** A set of arguments $E \subseteq \mathsf{Arg}$ is called a stable extension iff it is complete and $\forall \mathcal{A} \in \mathsf{Arg} \backslash E : \exists \mathcal{B} \in E$ with $\mathcal{B} \rightarrow \mathcal{A}$.

2.2 Artificial Neural Networks and Graph Convolutional Networks

An *artificial neural network* (henceforth also referred to as *neural network* or simply *network*) generally consists of multiple artificial neurons that are connected with each other. In biology, a neuron is a nerve cell that occurs, for example, in the brain or in the spinal cord. Neurons are specialised on conducting and transferring stimuli [23]. In computer science, (artificial) neurons denote a data structure that was developed to work similarly to their biological example. It is to be noted that there exist different models of artificial neurons and neural networks. Due to its contextual relevance in this paper, solely the structure and functionality of the *multilayer perceptron* model [14] will be described.

An artificial neuron can have multiple inputs $x_i \in \mathbb{R}$ with $i \in \{1, \dots, n\}$ that form the input vector $\boldsymbol{x} = (x_1, \dots, x_n)^\top$. Each of the n inputs is multiplied by a weight w_i. In addition to the regular inputs, there are so-called *bias* inputs b. They serve the purpose of stabilising the computation. As visualised in Figure 1, an activation function $f(\cdot)$ is applied to the sum of all weighted inputs. The result of the function is the neuron's output [8,16].

Analogously to the biological prototype, artificial neurons are connected to networks. Such networks are usually arranged in layers that consist of at least one neuron. There is one input layer, one or more so-called *hidden layers*, and one output layer. It is to be noted that the input layer is considered a layer only for convenience, because it only passes the input values to the next layer without further processing [20,16]. Neural networks can be understood as graphs, with neurons as nodes and their connections as edges. For training neural networks, the *back-propagation* algorithm is used in most cases. Back-propagation is a supervised learning method, meaning that at all times during training, the output corresponding to the current input must be known. The goal is to find the most exact mapping of the input vectors to their output vectors. This is realised by adjusting the weights on the edges of the graph, see [16] for details.

In the context of graph theory, Kipf et al. [17] introduce graph convolutional networks that are able to directly use graphs as input instead of a vector of reals. More precisely, they introduce a layer-wise propagation rule for neural networks that operates directly on graphs. It is formulated as follows:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{1}$$

$H^{(l)} \in \mathbb{R}^{N\times D}$ denotes the matrix of activations in the l^{th} layer. $\sigma(\cdot)$ is an activation function, such as ReLU (*Rectified Linear Units*) [18]. Moreover, $D_{ii} = \sum_j \tilde{A}_i j$ and $\tilde{A} = A + I_N$, where A is the adjacency matrix of the graph and I_N is the identity matrix. $W^{(l)}$ denotes a layer-specific trainable weight matrix. Spectral convolutions on graphs are defined as

$$g_\theta * x = U g_\theta U^\top x. \tag{2}$$

A signal $x \in \mathbb{R}^N$ (a scalar for every node) is multiplied by a filter $g_\theta = \text{diag}(\theta)$, which is parameterised by θ in the Fourier domain. U is the matrix of Eigenvectors of the normalised graph Laplacian $L = I_N - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = U\Lambda U^\top$, where Λ is a diagonal matrix of the Laplacian's Eigenvalues. $U^\top x$ is the graph Fourier transform of x [17].

For a number of reasons, evaluating equation (2) is computationally expensive. For example, computing the Eigendecomposition of L might become rather expensive for large graphs. Hammond et al. [15] suggest that $g_\theta(\Lambda)$ can be approximated by a truncated expansion in terms of Chebyshev polynomials in order to avoid this problem:

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) \tag{3}$$

$T_k(x)$ denotes the Chebyshev polynomials up to K^{th} order. The matrix Λ is rescaled to $\tilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - I_N$, where $\lambda_{\max}$ describes the largest Eigenvalue of L. Besides, $\theta' \in \mathbb{R}^K$ is now a vector of Chebyshev coefficients. Integrating this approximation into the definition of a convolution of a signal x with a filter $g_{\theta'}$ yields

$$g_{\theta'} * x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x, \tag{4}$$

with $\tilde{L} = \frac{2}{\lambda_{\max}}L - I_N$ [17]. Because this convolution is a K^{th}-order polynomial in the Laplacian, it is K-localized. This means, it depends only on a certain neighbourhood—more specifically: it only depends on nodes which are at maximum K steps away from the central node.

Stacking multiple convolutional layers in the form of equation (4) (each layer followed by a point-wise non-linearity) leads to a neural network model that can directly process graphs.

3 Casting the Acceptability Problem as a Classification Problem

In abstract argumentation there are several interesting decision problems with varying complexity [10]. For example, the problem $\textsc{Cred}_\sigma$ with σ being either complete, grounded, preferred, or stable semantics, asks for a given $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ and an argument $\mathcal{A} \in \mathsf{Arg}$, whether $\mathcal{A}$ is contained in at least one σ-extension of AF. For preferred semantics this is an NP-complete problem [10]. For our first feasibility study here, we will focus on this problem, i. e., $\textsc{Cred}_{PR}$.

In order to represent $\textsc{Cred}_{PR}$ as a classification problem, we assume that for any given input argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ we have an arbitrary but fixed order of the arguments, i. e., $\mathsf{Arg} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$. Moreover, let $\mathfrak{A}$ denote the set of all abstract argumentation frameworks and V the set of all vectors with values in [0,1] of arbitrary dimension. Conceptually, our classifier C then will be a function of the type $C : \mathfrak{A} \rightarrow V$ with $|C(\mathsf{Arg}, \rightarrow)| = |\mathsf{Arg}|$, i. e., on an input argumentation framework with n arguments we get an n-dimensional real vector as the result.[2] The interpretation of this output then is that the i-th entry of $C(\mathsf{Arg}, \rightarrow)$ denotes the likelihood of argument $\mathcal{A}_i$ being credulously accepted wrt. preferred semantics. Of course, a sound and complete classifier C should output 1 whenever this is true and 0 otherwise. However, as we will only approximate the true solution, all values in the interval [0,1] are possible.

The function C, in our case represented by a graph convolutional network, will be trained on benchmark graphs where the gold standard, i. e. the *true* solutions, is available, e. g., by means of asking a complete oracle solver. Given enough and diverse benchmark graphs for training, our main hypothesis is that C approximates the intended behaviour.

4 Experimental Evaluation

The framework for graph convolutional networks (GCNs) offered by Kipf et al. [17], which is realised with the aid of Google's *TensorFlow* [1], is designed to find labels for certain nodes of a given graph and is thus a reasonable starting point for examining if it is possible to decide whether an argument is credulously accepted wrt. preferred semantics by the use of neural networks.

4.1 Datasets

An essential part of any machine learning task is collecting sufficient training and test data. The probo[3] [7] benchmark suite can be used to generate graphs with different properties. A solver such as CoQuiAAS [19] can then be used to compute the corresponding extensions. The suite offers three different graph

[2] Note that implementation-wise this is not completely true as the size of the output vector has to be fixed.

[3] `https://sourceforge.net/projects/probo/`

generators that each yield graphs with different properties. The first one, the *GroundedGenerator*, produces graphs that have a large grounded extension. The *SccGenerator* produces graphs that are likely to have many strongly connected components. Lastly, the *StableGenerator* generates graphs that are likely to have many stable, preferred, and complete extensions. To provide even more diversity in the data, we use *AFBenchGen*[4] [6] as a second graph generator. It generates random scale-free graphs by using the *Barabási-Albert model* [2], as well as graphs using the *Watts-Strogatz model* [27], and the *Erdős-Rényi model* [11].

In order to examine the impact of the training set size on the classification results, a number of different-sized datasets is generated. It is to be noted that each dataset contains the next smaller dataset in addition to some new data. This strategy is supposed to keep changes in the character of the dataset minimal. The test set is, of course, an exception from this rule. Moreover, each dataset (including the test set) is composed of equal shares of all six previously described types of graphs, and all graphs have between 100 and 400 nodes. Table 1 gives an overview.

In addition to the specifically generated test set, a fraction of the benchmark dataset used in the International Competition on Computational Models of Argumentation (ICCMA) 2017 [12] is used in order to examine how a trained model performs on external data. Said fraction consists of 45 graphs of group B (the only one designated for solvers of $\textsc{Cred}_{PR}$) that were chosen from all five difficulty categories.

ID	Number of graphs	Total number of nodes
5-of-each	30	5,461
10-of-each	60	12,056
25-of-each	150	32,026
50-of-each	300	73,717
75-of-each	450	108,050
100-of-each	600	149,130
test	120	30,603

Table 1. Dataset overview.

4.2 Experimental Setup

The GCN framework [17] was designed to perform node-wise classification on a single large graph in a semi-supervised fashion. In order to use the GCN framework in its intended way, three different matrices need to be provided: an $N \times N$ adjacency matrix (N: number of nodes), an $N \times D$ feature matrix (D:

[4] `https://sourceforge.net/p/afbenchgen/wiki/Home/`

number of features per node), and an $N \times F$ binary label matrix (F: number of classes).

For this work, the training process should be supervised rather than semi-supervised. However, the set of unlabeled nodes can be left empty. Because all nodes consequently have a known label, the training process becomes supervised instead of semi-supervised. Besides, instead of one single graph with some nodes to be classified, entire sets of graphs are supposed to provide the training and test sets. To realise this, the graphs in both training and test set are considered one big graph. This yields an adjacency matrix that essentially contains the adjacency matrices of all graphs. The graphs belonging to the test set make up the set of nodes that are to be classified.

The feature matrix can be used to provide additional information on the content of the nodes that could be used to improve classification. However, defining an appropriate feature matrix is a rather difficult matter in our application scenario, because the nodes do not contain any information, in contrast to, for example, social networks or citation networks. In Section 4.3, two different solutions are explored. The first one is a simple $N \times 1$ matrix that contains the same constant for every node (which means that no additional features are provided for the nodes). For the second option, the number of incoming and outgoing attacks per argument are used as features, resulting in an $N \times 2$ matrix (one column for each incoming and outgoing attacks).

4.3 Results

When dealing with artifcial neural networks, quite a few parameters can influence the outcome of the training process. The following section describes various experimental results in which the impact of different factors on the quality of the classification process is examined. Those factors include, for instance, the size and nature of the training set, the learning rate, and the number of epochs being used to train the neural network model. Finally, we report on some runtime comparison with a sound and complete solver.

Feature Matrix As explained in Section 4.2, there are two different types of feature matrix that may be used in the training process. While training with the feature matrix that does not contain any features (henceforth referred to as FM1) always results in an accuracy of 77.0%, training with the matrix that encodes incoming and outgoing attacks as features (henceforth referred to as FM2) offers slightly better results (up to 80.3%). *Accuracy* is measured by dividing the number of correct predictions by the total number of predictions. The accuracy value for class YES can also be viewed as the *recall* value, which is calculated by dividing the number of true positives by the sum of true positives and false negatives. Moreover, by calculating the *precision* (true positives divided by the sum of true positives and false positives), the *F1 score* can be obtained as follows:

$$\mathrm{F}_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

FM1		FM2	
Accuracy YES	Accuracy NO	Accuracy YES	Accuracy NO
0.0000	1.0000	0.1499	0.9846
0.0000	1.0000	0.2025	0.9810
0.0000	1.0000	0.2083	0.9803

Table 2. Accuracy per class for both feature matrix types.

	Barabási-Albert	Erdős-Rényi	Grounded	Scc	Stable	Watts-Strogatz
Accuracy YES	1.0000	0.0000	0.0771	0.0000	0.0000	0.0000
Accuracy NO	0.0000	1.0000	0.9950	1.0000	1.0000	1.0000
Accuracy total	0.8421	0.8152	0.7109	0.9886	0.8421	0.9988
F1 Score	0.0000	0.0000	0.1417	0.0000	0.0000	0.0000

Table 3. Training results for individual graph types and parameter settings for training. Additional parameters were set as follows: Number of epochs: 500, learning rate: 0.001, dropout: 0.05.

Moreover, because it seems unusual that multiple different training setups all return the same value, it is important to also look into the class-specific accuracies. Table 2 reveals that the network only learned to classify all nodes as NO when trained with FM1. Incorporating FM2 into the training process leads to an accuracy of class YES of up to 20.8%. Wheras this result still needs optimisation, it shows that using FM2 is the more promising approach. In all following experiments, FM2 is used.

Graph Types In order to further investigate the background of the prior results, the different graph types are examined. Six additional datasets that consist of one graph type each, are created. Each one contains 100 graphs for training and 20 graphs for testing. Essentially, the 100-of-each training set and the test set are split into six subsets consisting of only one graph type per set.

In Table 3, the training results, alongside the settings that were used to retrieve these values, are presented. Several observations can be made from the results. Firstly, a set of parameter settings does not work equally well on all graph types. While four out of six graph types only learn to decide on one class for all instances, Grounded and Stable graphs show first signs of a deeper learning process. Increasing the number of epochs to 1000 yields exactly the same accuracies for Barabási-Albert, Erdős-Rényi, Scc, and Watts-Strogatz graphs, but improves the values for Grounded and Stable. This leads to the assumption that the graph types are of different difficulty for the network to learn. The fact that 98.86% (Scc) or even 99.89% (Watts-Strogatz) of the graphs' nodes belong to one class supports this assumption. Classifying such unevenly distributed classes is quite a difficult task for a neural network.

Dataset	Accuracy Yes	Accuracy No	Accuracy total	F1 Score
5-of-each	0.0000	1.0000	0.7701	0.0000
10-of-each	0.1869	0.9795	0.7972	0.2976
25-of-each	0.2025	0.9810	0.8020	0.3199
50-of-each	0.2170	0.9797	0.8043	0.3377
75-of-each	0.2174	0.9793	0.8041	0.3380
100-of-each	0.2210	0.9786	0.8044	0.3419

Table 4. Classification results after training with different-sized training sets. Parameter settings: epochs: 500, learning: 0.01, dropout rate: 0.05 However, a difference in training set size might require different settings. For example, a larger dataset might need more epochs to converge than a smaller ones.

Another observation is that the set of Barabási-Albert graphs is the only one where the majority of instances is in the class Yes. This might help creating a dataset with more evenly distributed classes. Generally, it is certainly helpful to have some graphs with more Yes instances in a dataset in order to generate more diversity. Having a diverse dataset is a vital aspect when training neural networks. Otherwise, the network might overfit to irrelevant features or might not work for some application scenarios.

Dataset Size Besides the influence of a dataset's diversity, the amount of data also has an impact on the training process. Table 4 shows some classification results for the different datasets described in Section 4.1. As expected, it indicates that bigger training sets have a greater potential to improve classification results. Nonetheless, utilizing more training data does not automatically mean better results. As displayed in Table 4, adding more than 50 graphs of each type does not yield a significant increase in accuracy. The values for overall accuracy and accuracy for class No do not change much at all (both less than 3.5%) when adding more training data. It is, however, crucial to look into the accuracy of class Yes as well as the F1 scores, because it indicates that the network actually learned some features of a preferred extension, instead of guessing No for all instances. Training with 25 graphs per type (150 in total) already results in 20.25% accuracy of class Yes—only 1.85% less than a training with a total of 600 graphs yields. Training with 50 graphs per type increases the accuracy for Yes by another 1.45%, which may still be regarded as significant when considering that the difference to the next bigger training set is merely 0.04%. In summary, the increase in accuracy for class Yes rather quickly starts stagnating when more data is added.

Optimisation Training a neural network is a task that demands careful adjustment of various parameters and other aspects. This section describes several approaches that may optimise the results gathered so far.

The main problem with the previous results is that the model seems to underfit. A reason for that might be that the training set is badly balanced in

Number of Epochs	Learning rate	Dropout	Accuracy Yes	Accuracy No	Accuracy total	F1 Score
500	0.1	0.05	0.2488	0.9705	0.8045	0.3693
500	0.01	0.05	0.2589	0.9669	0.8041	0.3781
500	0.001	0.05	0.2372	0.9735	0.8042	0.3578
250	0.01	0.05	0.2659	0.9644	0.8037	0.3839
750	0.01	0.05	0.2728	0.9622	0.8037	0.3899
500	0.01	0.01	0.2682	0.9637	0.8038	0.3859
500	0.01	0.1	0.2494	0.9697	0.8041	0.3693

Table 5. Classification results after training with a more balanced dataset in regard to instances per class.

terms of number of instances per class. A dataset where the two classes are about equally distributed might lead to an improvement. Therefore, an additional training set is generated, which conists of 100 Barabási-Albert graphs and a total of 100 graphs of the other types (20 graphs of each). The results for training with this dataset under different parameter settings (regarding the learning rate, number of epochs, and dropout rate) are displayed in Table 5. It becomes clear that the overall accuracy does not improve significantly in comparison to the previous results. Nevertheless, the accuracy of class Yes increased to values between 23.72% (500 epochs, learning rate 0.001, dropout 0.05) and 27.28% (750 epochs, learning rate 0.01, dropout 0.05). So, these results might be considered a slight improvement, because they are more evenly distributed than the former ones. Another observation is that changes in number of epochs, learning rate, or dropout rate do not lead to any significant improvements in total accuracy. In fact, most alterations in parameter settings yield slightly worse results.

Looking into the actual numbers of instances of Yes and No reveals that instances of the latter class are still the majority (54.4%). To further equalize the number of instances per class, the training set is augmented by 27 more Barabási-Albert graphs (7300 arguments). The distribution of ground truth labels is now 50.6% YES and 49,4% No, respectively. Training the neural network with this dataset (parameters are set to 500 epochs, a learning rate of 0.01, and a dropout rate of 0.05) results in a total accuracy of 80.0%. However, the accuracy of class Yes increased to 29.7%, while the corresponding value for class No marginally decreased to 95.0%. This demonstrates that using a more balanced training set (in respect of instances per class) also leads to more balanced results. Since the test set consists of 77.0% instances of class No, the total accuracy does not increase, though.

Competition data In order to get a sense of how the training results transfer to other data, two differently trained models are tested on the competition data (see Section 4.1). The first model is trained with the 50-of-each dataset. The learning rate is set to 0.01, dropout to 0.05, and number of epochs to 500. The second model uses the same settings, but is trained with the more balanced

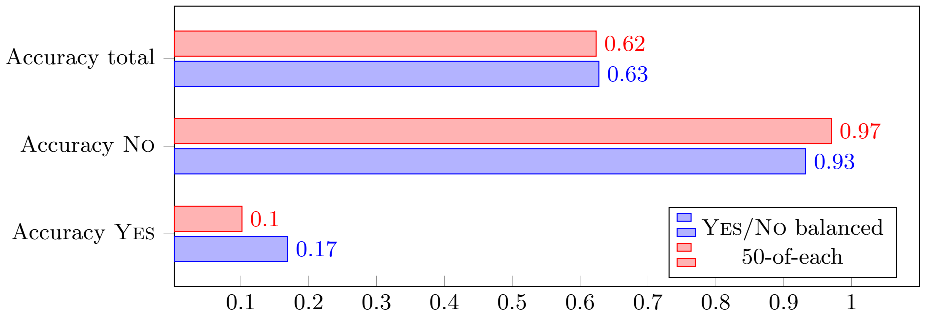

Fig. 2. Results for testing with benchmark data.

dataset containing 127 Barabási-Albert graphs and 100 others as illustrated above. Figure 2 displays a comparison of the results. The overall accuracy is very similar for both training sets: about 17% lower than for the regular test set, and the class-specific accuracy values are lower, too. This might be due to the benchmark dataset containing graphs that are smaller or larger than the ones in the training set. Also, additional types of graphs are included in the benchmark dataset.

Runtime performance Aside from the quality of the classification results, another aspect that needs to be considered is the time efficiency. In order to put GCN's efficiency into perspective, it is compared to CoQuiAAS, the SAT-based argumentation solver used to provide ground truth labels for the training and test sets.

For the GCN approach, only the time for evaluating the test set is measured, since a neural network can, once it is trained, classify as many arguments as one wishes. Both methods are evaluated on classifying the entire test set (see Section 4.1) using the same hardware. The difference is enormous: While the GCN classifies the entire test set within < 0.5 seconds, CoQuiAAS needs about an hour (60.98 minutes). It is to be noted that the value for testing using a trained GCN varies a bit depending on the training conditions. For example, a measurement taken after training with the biggest training set (600 graphs) is 0.22 seconds. Training with half the data lead to 0.13 seconds.

Table 6 reveals the big fluctuations in the amounts of time CoQuiAAS needs to decide for a single argument whether it is included in a preferred extension or not. While the lowest value is at 0.002 seconds, the highest one is at 19.27 seconds—which is about 8674 times as much. It is also worth noting that, if evaluating the whole test set takes the GCN 0.22 seconds, it takes an average of $7 \cdot 10^{-6} = 0.000007$ seconds. That means, the minimal amount of time CoQuiAAS needed to evaluate an argument is still 317 times as much as the average amount of time the GCN takes. We only report on the mean runtime for the GCN approach as classification is independent of the instance, it is only polynomial in

Method	Property	Time in seconds
CoQuiAAS	maximum	19.274452
CoQuiAAS	mean	0.119561
CoQuiAAS	minimum	0.002222
GCN	mean	0.000007

Table 6. Time measurements in comparison.

the size of the trained network. It follows that the GCN approach has constant runtime wrt. the size of the instance.

Of course, one needs to consider that a neural network also needs time for training and possibly for preprocessing. Using the GCN framework, the training process took approximately between 20 minutes and two hours—depending on the dataset size and the parameter settings such as number of epochs or learning rate. For other network models and frameworks, training might take a lot longer. Nonetheless, once sufficient data is provided and the network is trained, it can be used for any test set and it is extremely fast.

5 Conclusion

All in all, the attempt of training a graph-convolutional network on abstract argumentation frameworks in order to decide whether an argument is included in a preferred extension or not was rather moderate. The overall accuracy did under no circumstances exceed 80.5%. When testing with benchmark data, it was even lower (63%). However, extending the diversity of the training set, for instance, by adding different-sized graphs or by adding new types of graphs, might improve this result.

Furthermore, training a neural network model involves adjusting a great number of parameters. Also, some of these parameters depend on each other. Considering that training a neural network requires careful adaption of the training data, the parameter settings, and the network architecture itself, and that some aspects also affect others, examining all reasonable possibilities exceeds the extent of this work.

The training results are moderate: On the one hand, the overall classification accuracy does not exceed 80.5%, which is not good enough for practical applications, but on the other hand, it proves that the network learned at least some rudimental features of a preferred extension. The fact that instances from both classes can be classified correctly reinforces this statement. The accuracy for class Yes is far lower ($< 30\%$) than the accuracy for class No ($> 90\%$) in all training procedures. A reason for this effect may be that the majority of the training data is not included in an extension and thus labelled as No. Using a training set where the distribution of instances per class is more balanced, counteracts this effect to some degree. Using benchmark data for testing leads to an overall accuracy of about 63%. The decrease in accuracy in comparison to the specifically generated test set might be due to graph sizes and types that are unknown to the network model, as they were not included in the training data.

Moreover, a GCN's classification process is very time efficient: the entire test set (30,603 arguments) is classified in < 0.5 seconds. For comparison: the SAT solver CoQuiAAS takes about an hour for the same dataset.

Generally, neural networks seem to be suited to perform the task of classifying arguments as "included in a preferred extension" or "not included in a preferred extension". After all, it did work to a certain degree. Nevertheless, the chosen network architecture seems to be inadequate for the task of abstract argumentation. It is quite possible that a different network architecture leads to better results. For example, an increased number of layers in a network or more neurons per layer may increase the network's ability to learn more complex features. The results gathered in this paper show signs of underfitting, so a deeper network would be a plausible strategy. Besides, GCNs were originally constructed to process undirected graphs, yet argumentation frameworks are represented as directed graphs. If a better suited neural network is found, the next step could be to expand the classification problem to a regession problem by training the network to predict entire extensions, or even all possible extensions of an argumentation framework.

Acknowledgements The research reported here was partially supported by the Deutsche Forschungsgemeinschaft (grant KE 1686/3-1).

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: a system for large-scale machine learning. In: OSDI. vol. 16, pp. 265–283 (2016)
2. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Reviews of modern physics **74**(1), 47 (2002)
3. Atkinson, K., Baroni, P., Giacomin, M., Hunter, A., Prakken, H., Reed, C., Simari, G.R., Thimm, M., Villata, S.: Toward artificial argumentation. AI Magazine **38**(3), 25–36 (October 2017)
4. Besnard, P., Hunter, A.: Constructing argument graphs with deductive arguments: a tutorial. Argument & Computation **5**(1), 5–30 (2014)
5. Cerutti, F., Gaggl, S.A., Thimm, M., Wallner, J.P.: Foundations of implementations for formal argumentation. In: Baroni, P., Gabbay, D., Giacomin, M., van der Torre, L. (eds.) Handbook of Formal Argumentation, chap. 15. College Publications (February 2018)
6. Cerutti, F., Giacomin, M., Vallati, M.: Generating challenging benchmark afs. COMMA **14**, 457–458 (2014)
7. Cerutti, F., Oren, N., Strass, H., Thimm, M., Vallati, M.: A benchmark framework for a computational argumentation competition. In: COMMA. pp. 459–460 (2014)
8. Ding, B.N.K.L.: Neural network fundamentals with graphs, algorithms and applications. Mac Graw-Hill (1996)
9. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial intelligence **77**(2), 321–357 (1995)

10. Dvořák, W., Dunne, P.E.: Computational problems in formal argumentation and their complexity. In: Baroni, P., Gabbay, D., Giacomin, M., van der Torre, L. (eds.) Handbook of Formal Argumentation, chap. 14. College Publications (February 2018)
11. Erdos, P., Rényi, A.: On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci **5**(1), 17–60 (1960)
12. Gaggl, S.A., Linsbichler, T., Maratea, M., Woltran, S.: Summary report of the second international competition on computational models of argumentation. AI Magazine **39**(4) (2018)
13. García, A.J., Simari, G.R.: Defeasible logic programming: Delp-servers, contextual queries, and explanations for answers. Argument & Computation **5**(1), 63–88 (2014)
14. Gardner, M.W., Dorling, S.: Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric environment **32**(14), 2627–2636 (1998)
15. Hammond, D.K., Vandergheynst, P., Gribonval, R.: Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis **30**(2), 129–150 (2011)
16. Jain, A.K., Mao, J., Mohiuddin, K.M.: Artificial neural networks: A tutorial. Computer **29**(3), 31–44 (1996)
17. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
18. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
19. Lagniez, J.M., Lonca, E., Mailly, J.G.: Coquiaas: A constraint-based quick abstract argumentation solver. In: Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on. pp. 928–935. IEEE (2015)
20. Michie, D., Spiegelhalter, D.J., Taylor, C.C.: Machine learning, neural and statistical classification. Citeseer (1994)
21. Modgil, S., Prakken, H.: The ASPIC+ framework for structured argumentation: A tutorial. Argument & Computation **5**, 31–62 (2014)
22. Niu, D., Liu, L., Lü, S.: New stochastic local search approaches for computing preferred extensions of abstract argumentation. AI Communications **31**(4), 369–382 (June 2018)
23. Schmidt, R.F., Lang, F., Heckmann, M.: Physiologie des menschen: mit pathophysiologie. Springer-Verlag (2011)
24. Thimm, M.: Stochastic local search algorithms for abstract argumentation under stable semantics. In: Modgil, S., Budzynska, K., Lawrence, J. (eds.) Proceedings of the Seventh International Conference on Computational Models of Argumentation (COMMA'18). Frontiers in Artificial Intelligence and Applications, vol. 305, pp. 169–180. Warsaw, Poland (September 2018)
25. Thimm, M., Villata, S.: The first international competition on computational models of argumentation: Results and analysis. Artificial Intelligence **252**, 267–294 (2017)
26. Toni, F.: A tutorial on assumption-based argumentation. Argument & Computation **5**(1), 89–117 (2014)
27. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world'networks. nature **393**(6684), 440 (1998)