Foundations for Solving Classification Problems with Quantitative Abstract Argumentation

Nico Potyka¹[0000-0003-1749-5233]

University of Stuttgart, Germany nico.potyka@ipvs.uni-stuttgart.de

Abstract. Abstract Argumentation Frameworks represent arguments and their relationships like attack and support in a graph. Their simple structure makes them easily interpretable and therefore a potentially interesting tool for explainable machine learning. We discuss some ideas for modeling and solving classification problems as abstract argumentation problems. As opposed to previous approaches that built argumentation frameworks on top of the result of machine learning algorithms, our classifiers can be learnt in an end-to-end fashion. Our research is still in an early stage, but, hopefully, this position paper will inspire interesting discussions at the workshop.

1 Introduction

Abstract argumentation frameworks model arguments and their relationships in a graphical structure. The computational problem is to decide which arguments can be accepted. Dung's original framework [7] considered only attack relations, but has been extended in different directions. Perhaps the most interesting directions for our purpose are bipolar and quantitative extensions. Bipolar argumentation frameworks [2] add support relations. This extension seems vital for classification tasks since the decision about the final label should not only be based on contra arguments, but also on pro arguments. Quantitative argumentation frameworks measure the degree of acceptance by a numerical scale. For example, in probabilistic epistemic argumentation [11] arguments are interpreted by probabilities that reflect degrees of belief. In gradual argumentation [4, 23,1], more general numerical values are considered. However, the intuition remains the same: The classical notion of acceptance usually corresponds to the largest value and the classical notion of rejection to the minimal value. By considering a numerical scale, we get a fine-grained distinction between these two extremes. In this paper, we will discuss some ideas for how to apply this technology for solving classification problems in a transparent and explainable way.

2 Background on Abstract Argumentation

Our understanding of an argument in this work follows Dung's notion of *ab*stract argumentation: "an argument is an abstract entity whose role is solely

⁰ Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).



Fig. 1. Example BAG.

determined by its relations to other arguments" [7]. In particular, we abstract from the content of an argument and are only interested in its acceptability dependent on the acceptability of its attackers and supporters. We consider *bipolar argumentation graphs* (*BAGs*) of the form (\mathcal{A} , Att, Sup), where \mathcal{A} is a finite set of arguments, Att $\subseteq \mathcal{A} \times \mathcal{A}$ is the *attack relation* and Sup $\subseteq \mathcal{A} \times \mathcal{A}$ is the support relation. With a slight abuse of notation, we let Att(\mathcal{A}) = { $B \in \mathcal{A} \mid (B, A) \in Att$ } denote the attackers of \mathcal{A} and let Sup(\mathcal{A}) denote its supporters. Graphically, we denote attack relations by solid and support relations by dashed edges. Figure 1 illustrates the definition. The BAG models part of a decision problem from [17], where we want to decide whether to buy new or to sell existing stocks of a company. A1 corresponds to the statement of an expert that recommends selling. A2 and A3 correspond to statements by experts who contradict A1's premises and recommend buying. Since we do not want to accept both the selling and the buying decision, the corresponding decision arguments attack each other.

We will not talk about the classical interpretation of bipolar argumentation frameworks and move directly to quantitative approaches. Our focus is on two bipolar approaches that seem interesting for classification tasks. Both share a BAG as a common data structure. They enhance the BAG in different ways in order to derive numerical degrees of acceptance.

2.1 Probabilistic Epistemic Argumentation

Probabilistic epistemic argumentation [11] builds up on basic probability theory. Probabilities are assigned to arguments by means of probability functions $P: 2^{\mathcal{A}} \to [0, 1]$ such that $\sum_{w \in 2^{\mathcal{A}}} P(w) = 1$. Each $w \in 2^{\mathcal{A}}$ can be seen as one possible world, in which the arguments in w are accepted and the remaining ones are rejected. The probability of an argument $A \in \mathcal{A}$ under P is then defined by adding the probabilities of all worlds in which A is accepted, that is, $P(A) = \sum_{w \in 2^{\mathcal{A}}, A \in w} P(w)$. In order to restrict the set of all probability functions to those that respect the BAG, different constraints have been introduced [11]. We give a few examples here:

COH: P is called *coherent* if for all $A, B \in \mathcal{A}$ with $(A, B) \in A$ tt, we have $P(B) \leq 1 - P(A)$.

SFOU: P is called *semi-founded* if $P(A) \ge 0.5$ for all $A \in \mathcal{A}$ with $Att(A) = \emptyset$.

SOPT: P is called *semi-optimistic* if $P(A) \ge 1 - \sum_{B \in Att(A)} P(B)$ for all $A \in \mathcal{A}$ with $Att(A) \neq \emptyset$.

Coherence encodes one possible meaning of attack relations: if A attacks B, then the probability of A bounds the probability of B from above (and vice versa). The intuition behind the definition is that we do not want to accept both an argument and its attacker. Semi-foundedness encodes the intuition that an argument should not be rejected if there is no reason for doing so. Formally, this means that the degree of belief in this argument should not be lower than 0.5. Semi-optimism gives another lower bound for the degree of belief. If there are no attackers, the argument must be accepted (probability 1). Attackers will decrease the lower bound based on their own degree of belief. These constraints have been defined for attack-only graphs, but they can be easily extended to bipolar graphs. For example, dual to coherence, we could define a lower bound for support edges as follows.

S-COH: P is called *s-coherent* if for all $A, B \in \mathcal{A}$ with $(A, B) \in \text{Sup}$, we have $P(B) \ge P(A)$.

The previous examples are all special cases of *linear atomic constraints* [19] that are generally written in the normalized form

$$\sum_{i=1}^{n} c_i \cdot \pi(A_i) \le c_0,$$

where $A_i \in \mathcal{A}$, $c_i \in \mathbb{R}$ and π is a syntactic symbol for the probability of an argument. A probability function P satisfies such a linear atomic constraint iff $\sum_{i=1}^{n} c_i \cdot P(A_i) \leq c_0$. Note that all constraints above can be brought into this form. For example, the *Coherence* constraint can be rewritten as $1 \cdot \pi(A) + 1 \cdot \pi(B) \leq 1$ and the *S*-*Coherence* constraint as $1 \cdot \pi(A) + (-1) \cdot \pi(B) \leq 0$. In general, we can also consider more general *epistemic constraints* that allow non-linear combinations of probabilities and probabilities of complex formulas over arguments [10]. However, in order to keep things simple, we do not discuss the most general form. For future reference, we define a P-BAG as follows.

Definition 1. A *P*-BAG is a tuple (\mathcal{A} , Att, Sup, C), where (\mathcal{A} , Att, Sup) is a BAG and C is a set of epistemic constraints over the BAG.

Given a P-BAG (\mathcal{A} , Att, Sup, C), we are interested in solving the following *entailment problem*: compute tight upper and lower bounds on the probability of an argument $A \in \mathcal{A}$ among all probability distributions that satisfy the constraints in C. If we restrict to linear atomic constraints, this problem can be solved in polynomial time [19]. To illustrate the entailment problem, consider again the BAG in Figure 1. We consider a new constraint that we call *Balance*.

BAL: *P* is called *balanced* if $P(A) = \frac{1}{2} + \frac{\sum_{B \in \text{Sup}(A)} P(B) - \sum_{B \in \text{Att}(A)} P(B)}{1 + \max\{|\text{Sup}(A)|, |\text{Att}(A)|\}}$ for all $A \in \mathcal{A}$.



Fig. 2. Entailment results for BAL constraints (left) and SFOU, COH and S-COH constraints (right).

To improve readability, we do not present *Balance* in the normal form of linear atomic constraints. Based on our previous discussion, the reader hopefully recognizes that *Balance* can indeed be written in this form. Note, in particular, that an equality can be represented by two inequalities $(x = y \text{ iff } x \leq y \text{ and} y \leq x)$. Intuitively, *Balance* enforces probability 0.5 if attackers and supporters are equally strong and moves the probability towards 0 (1) if the attackers are stronger (weaker) than the supporters. The entailment results are shown in Figure 2 on the left. In general, we may get interval probabilities rather than point probabilities. To illustrate this, we can replace BAL with the constraints SFOU, COH and S-COH that we discussed before. The entailed probabilities are shown in Figure 2 on the right.

A polynomial-time implementation for solving the entailment problem for linear atomic constraints can be found in the Java library Probabble¹.

2.2 Gradual Argumentation

Gradual argumentation frameworks evaluate arguments by numerical strength values. We will focus on bipolar frameworks that use the interval [0, 1] to give a uniform presentation. A discussion of more general frameworks can be found in [3]. The basic building block is again a BAG. Now, additionally, every argument has an initial weight that can be seen as an apriori belief in the argument when ignoring all the others. For future reference, we define a *G*-*BAG* as follows.

Definition 2. A *G*-BAG is a tuple (\mathcal{A} , Att, Sup, w), where (\mathcal{A} , Att, Sup) is a BAG and $w : \mathcal{A} \to [0, 1]$ is a weight function.

The main computational problem is again to assign acceptance values to arguments. In this context, they are often called strength values. Strength values are often computed by applying an update function to the G-BAG repeatedly until the values converge. The update function often has a simple modular structure consisting of an aggregation and an influence function [14]. The aggregation function takes the strength values of all parents and combines them to a numerical value. Intuitively, an attacker should decrease the value based on its strength, whereas a supporter should increase the value based on its strength.

¹ https://sourceforge.net/projects/probabble/

The influence function then adapts the initial weight based on the aggregate. For example, the aggregation function of the DF-QuAD algorithm uses multiplication [23]. While it has some nice properties, its particular definition has the disadvantage that the aggregate is necessarily (close to) 0 if both an attacker and a supporter have strength (close to) 1. The Euler-based semantics introduced in [1], uses addition instead of multiplication to overcome this problem. However, its influence function is not symmetric about 0, which causes a counterintuitive imbalance between attacks and supports. The quadratic-energy model from [16] uses a symmetric influence function in order to overcome this problem. Its aggregation function is defined by

$$\alpha(A) = \sum_{B \in \operatorname{Sup}(A)} s(B) - \sum_{B \in \operatorname{Att}(A)} s(B)$$

where $s : \mathcal{A} \to [0, 1]$ assigns the current strength value to every argument. Its influence function is defined by

$$\iota(A) = w(A) - w(A) \cdot h(-\alpha(A)) + (1 - w(A)) \cdot h(\alpha(A)),$$

where $h(x) = \frac{\max\{0,x\}^2}{1+\max\{0,x\}^2}$ squashes its input to the interval [0, 1]. Intuitively, a positive aggregate will move the weight towards 1, while a negative aggregate will move the weight towards 0. The strength values are initialized with the initial weights. Then the aggregation and influence function are applied repeatedly until the values converge.

Figure 3 shows, at the top, possible initial weights for the example BAG on the left and the resulting final strength values on the right. At the bottom of Figure 3, the left graph shows how the final strength values evolve over time. For acyclic graphs, they can be computed in linear time by a single forward pass [16]. In acyclic BAGs, the convergence theory is actually not complete. [14] gave some examples for cyclic G-BAGs where the strength values under different approaches start cycling and do not converge. In the known cases, these convergence problems can be overcome by continuizing the discrete update approach [16] as illustrated in the right graph at the bottom in Figure 3. In all cases, where convergence guarantees are known, it is actually guaranteed that the discrete and continuized algorithm will converge to the same strength values [18]. Empirically, the continuous algorithm still converges in subquadratic time [16]. The plots of the evolution of strength values may also add transparency and explainability to gradual argumentation frameworks.

Implementations for computing strength values with different gradual argumentation approaches and plotting the evolution of their strength values can be found in the Java library Attractor².

3 Abstract Argumentation Classifiers

We will now look at how classification problems can be solved by means of argumentation frameworks. The goal of classification is to map inputs \mathbf{x} to outputs y.

² https://sourceforge.net/projects/attractorproject/



Fig. 3. Top: Initial weights (left) and final strength values (right). Bottom: evolution of strength values for A1 (blue), A2 (green), A3 (red), Buy (violet) and Sell (yellow) under discrete (left) and continuous (right) quadratic-energy semantics.

We think of the inputs as feature tuples $\mathbf{x} = (x_1, \ldots, x_k)$, where the i-th value is taken from some domain D_i . The output y is taken from a finite set of class labels L. A classification problem $P = ((D_1, \ldots, D_k), L, E)$ consists of the domains, the class labels and a set of training examples $E = \{(\mathbf{x}_i, y_i) \mid 1 \le i \le N\}$.

By a numerical classifier, we mean a function $c: (X_{i=1}^{k} D_i) \times L \to \mathbb{R}$ that assigns to every pair (\mathbf{x}, y) a numerical value. An important special case is a probabilistic classifier $p: (X_{i=1}^{k} D_i) \times L \to [0, 1]$ where $\sum_{j=1}^{|L|} p(\mathbf{x}, y_i) = 1$. Then $p(\mathbf{x}, y) \in [0, 1]$ can be understood as the confidence of the classifier that an example with features \mathbf{x} belongs to the class y. Let us note that every numerical classifier c can be turned into a probabilistic classifier p_c by normalizing the label outputs by a softmax function. That is, $p_c(\mathbf{x}, y) = \frac{\exp(c(\mathbf{x}, y_i))}{\sum_{j=1}^{|L|} \exp(c(\mathbf{x}, y_j))}$.

Figure 4 shows the high-level architecture of our classifiers. The input is first encoded in a BAG that models the classification problem. If the resulting model is not a probabilistic classifier, the acceptance degrees of arguments corresponding to the labels are then normalized by a softmax function. We will discuss these steps in more detail in the following sections.



Fig. 4. High-level Architecture of Argumentation Classifiers.

3.1 Input and Output Arguments

In order to solve a classification problem $P = ((D_1, \ldots, D_k), L, E)$ with argumentation technology, we first transform the domains and class labels into arguments. A categorical feature with domain $D = \{d_1, \ldots, d_l\}$ can be transformed into l arguments $A_{D,1}, \ldots, A_{D,l}$. Intuitively, we can identify the value d_i with accepting $A_{D,i}$ and rejecting the remaining arguments $A_{D,j}, j \neq i$, of the feature. A continuous feature with domain $D \subseteq \mathbb{R}$ can be discretized by partitioning D into l intervals that can then be treated like discrete features. We denote the resulting input arguments by \mathcal{A}_{in} .

We proceed analogously for the class labels. For multiclass classification problems, we create one argument for every label analogous to discrete features. However, if the classification problem is binary, $L = \{c_0, c_1\}$, we create a single output argument. We denote the resulting output argument(s) by \mathcal{A}_{out} .

To illustrate the idea, we consider a small toy classification problem inspired by the Census dataset from the UCL machine learning repository³. We consider three features that correspond to age (continuous), education (3 categories) and work class (3 categories) and two class labels that correspond to an 'average or below' or 'above average' salary. Figure 5 shows one potential BAG built up from the corresponding input (bottom) and output (top) arguments. For simplicity, we chose a coarse discretization into three bins. Of course, determining the number and boundaries of the bins can also be made part of the learning process with the usual advantages (flexibility) and disadvantages (learning complexity).

3.2 Abstract Argumentation Classifiers

The most straightforward way to build a classifier is to take the input arguments and output arguments and to connect them via edges in such a way that a good classification performance on the examples is obtained.

Definition 3 (Naive Classification BAG). A Naive Classification BAG for a classification problem $P = ((D_1, \ldots, D_k), L, E)$ is a BAG (\mathcal{A} , Att, Sup) such that $\mathcal{A} = \mathcal{A}_{in} \cup \mathcal{A}_{out}$ is the set of input and output arguments for P, Att, Sup $\subseteq \mathcal{A}_{in} \times \mathcal{A}_{out}$ and Att \cap Sup $= \emptyset$.

Figure 5 shows one possible naive classification BAG for our census example. While a naive classification BAG is easy to interpret, it may not have sufficient degrees of freedom to capture more complicated relationships. In order

³ https://archive.ics.uci.edu/ml/datasets/Census+Income



Fig. 5. Hypothetical naive classification BAG for census example.



Fig. 6. Hypothetical deep classification BAG for census example.

to overcome the problem, we propose adding hidden layers of arguments between the input and the output layer inspired by the architecture of feedforward neural networks [8]. Similar to the intuition of deep feedforward neural networks, the hope is that deeper layers will form more sophisticated patterns from the patterns detected in earlier layers. Interpretability will probably suffer with increasing depth. However, due to the simple mechanics of the introduced argumentation approaches, a deep abstract argumentation classifier may still be easier to interpret than 'black box approaches' like neural networks or support vector machines. Figure 6 shows a possible deep classification BAG for our census example. The meaning of hidden arguments in early layers can still be intuitively explained. For example, $H_{1,1}$ can be roughly interpreted as saying that an above average salary is unlikely in the age group from 20 to 60 and if the education category is 1 unless the person is in working class 3.

Definition 4 (Deep Classification BAG). A Deep Classification BAG with k layers for a classification problem $P = ((D_1, \ldots, D_k), L, E)$ is a BAG (\mathcal{A} , Att, Sup) such that $\mathcal{A} = \bigcup_{i=0}^{k+1} \mathcal{A}_{\langle i \rangle}$ consists of the input arguments $\mathcal{A}_{\langle 0 \rangle} = \mathcal{A}_{in}$ and output arguments $\mathcal{A}_{\langle k+1 \rangle} = \mathcal{A}_{out}$ for P and additional layers of arguments $\mathcal{A}_{\langle i \rangle}$ such that $\mathcal{A}_{\langle i \rangle} \cap \mathcal{A}_{\langle j \rangle} = \emptyset$ for $i \neq j$. Furthermore, Att \cap Sup $= \emptyset$ and Att, Sup $\subseteq \bigcup_{i=0}^{k} \bigcup_{j=i+1}^{k+1} (\mathcal{A}_{\langle i \rangle} \times \mathcal{A}_{\langle j \rangle})$, that is, edges can only be directed towards deeper layers.

In order to classify an example with a classification BAG, we have to specify how to generate an output label from the input features. We will discuss some ideas for P-BAGs and G-Bags in the next sections.

3.3 Classification P-BAGs

In order to solve the classification problem with a P-BAG, we have to add a set of constraints to our classification BAG. We divide the constraints into **Classification Constraints:** encode the meaning of support and attack edges. **Instance Constraints:** encode the input features of an example.

The classification constraints are example independent, whereas the instance constraints change for every example (they correspond to the input transformation in Figure 4). We start discussing the instance constraints. The idea is simple: for every feature, we constrain the probability of the input argument that corresponds to the input value to 1 and the probability of the remaining input arguments for this feature to 0. For example, if $D_i = \{d_{i,1}, \ldots, d_{i,l}\}$ and $x_i = d_{i,2}$, we add the constraints $\pi(A_{i,2}) = 1$ and $\pi(A_{i,j}) = 0$ for $j \neq 2$.

There are various ways to define the classification constraints. Defining them could even be made part of the learning process. However, this would complicate the learning problem further. We therefore discuss some concrete options. Our initial proposal is a variation of the *Balance* constraint that we discussed before. We slightly generalize it by adding weights to edges. We will consider these weights as parameters that have to be learnt. For every argument $A \in \mathcal{A} \setminus \mathcal{A}_{in}$ that is not an input argument, we introduce one constraint of the form

$$\pi(A) = \frac{1}{2} + \frac{\sum_{B \in \operatorname{Sup}(A)} \theta_{B,A} \cdot \pi(B) - \sum_{B \in \operatorname{Att}(A)} \theta_{B,A} \cdot \pi(B)}{2}, \qquad (1)$$

where we demand $0 < \theta_{B,A}$, $\sum_{B \in \text{Sup}(A)} \theta_{B,A} = 1$ and $\sum_{B \in \text{Att}(A)} \theta_{B,A} = 1$. Let us note that, due to the simple structure of the BAG, the probabilities of the arguments can be computed in a single forward pass in linear time. The probabilities of the input arguments can be set immediately according to the instance constraints. We then go to the next layer. Since edges can only be directed towards deeper layers, all probabilities in this layer are determined by the probabilities of the previous layer. We can continue in this way until we set the probabilities in the output layer.

However, we note that the resulting classifier is necessarily a linear classifier. By induction, we can see that the probability at every argument is just a linear combination of the inputs: For the first layer, this is obvious. For the subsequent layers, it follows by induction, since a linear combination of linear combinations is a linear combination again. Therefore, the output probability is a linear function of the inputs. As a consequence, the classifier is only able to learn linearly separable functions. For example, it is not possible to learn to classify the XOR (exclusive or) function correctly when using the linear balance constraints in (1).

Fortunately, if we keep the structure sufficiently simple, we can still deal with non-linear constraints efficiently by performing a single forward pass through the BAG as before. We therefore propose the following non-linear variant of *Balance*.

$$\pi(A) = \theta_A + (1 - \theta_A) \cdot \max\{\sum_{B \in \operatorname{Sup}(A)} \theta_{B,A} \cdot \pi(B) - \sum_{B \in \operatorname{Att}(A)} \theta_{B,A} \cdot \pi(B), 0\} - \theta_A \cdot \max\{\sum_{B \in \operatorname{Att}(A)} \theta_{B,A} \cdot \pi(B) - \sum_{B \in \operatorname{Sup}(A)} \theta_{B,A} \cdot \pi(B), 0\}$$
(2)



Fig. 7. Classification P-BAG for XOR (left): Edges and non-input arguments are annotated with their weights. For example, $\theta_Y = 0$, $\theta_{X_1=0,H_{1,1}} = 1$. Classification of a positive example (right): Arguments are annotated with the entailed probabilities.

where again $0 < \theta_{B,A}$, $\sum_{B \in \text{Sup}(A)} \theta_{B,A} = 1$, $\sum_{B \in \text{Att}(A)} \theta_{B,A} = 1$ and furthermore $0 \leq \theta_A \leq 1$. The new parameter θ_A replaces 0.5 and allows learning a bias for the probability of this argument. The first term moves the probability towards 1 if the supporters are stronger, the second term moves the probability towards 0 if the attackers are stronger. To illustrate that this is sufficient to capture non-linear relationships, Figure 7 shows, on the left, a P-BAG for the XOR function. The graph on the right illustrates the classification process for a positive example. The output is 0.75 (the positive label is accepted). By going backward through the graph, the result can be explained. Y is accepted because its supporter $H_{1,1}$ is accepted. $H_{1,1}$, in turn, is accepted because $X_1 = 0$ and $X_2 \neq 0$ (the inputs differ). When using the BAG in Figure 8 with edge weights N = 1, the XOR function will actually be perfectly reproduced. That is, in the table in Figure 7, we will have 1 instead of 0.75 and 0 instead of 0.25. However, the P-BAG in Figure 7 also classifies every example correctly (when the decision threshold for a positive example is 0.5) and is easier to interpret.

3.4 Classification G-BAGs

For G-BAGs, we have to decide how to define the initial weights and which update function we use. For the update function, we propose a variant of the quadratic energy model that we explained before. Our variant again adds edge weights that are supposed to be learnt during the training process. The aggregation function is defined by

$$\alpha(A) = \sum_{B \in \operatorname{Sup}(A)} \theta_{B,A} \cdot s(B) - \sum_{B \in \operatorname{Att}(A)} \theta_{B,A} \cdot s(B),$$

where $s : \mathcal{A} \to [0, 1]$ assigns the current strength value to every argument as before. The influence function is defined by

$$\iota(A) = \theta_A - \theta_A \cdot h(-\alpha(A)) + (1 - \theta_A) \cdot h(\alpha(A)),$$

with $h(x) = \frac{\max\{0,x\}^2}{1+\max\{0,x\}^2}$. We demand $0 < \theta_{B,A}$ and $0 \le \theta_A \le 1$.



Fig. 8. Classification G-BAG for XOR. Edges and non-input arguments are annotated with their weights, where N = 100.

The weights of the input arguments can again be set based on the input. Since they do not have any ingoing edges, this weight will also be their final strength. For setting the weights, we proceed similar as before. For every feature, we set the weight of the argument corresponding to the input value to 1 and the weight of the remaining input arguments for this feature to 0.

Due to the acyclicity of the classification BAG, the final strength values can again be computed by a single forward pass through the graph in linear time. The result is guaranteed to be equal to the result of the iterative and continuous update approach [16]. Figure 8 shows a G-BAG for the XOR function. The more compact BAG in Figure 7 could also used for a G-BAG for XOR. In particular, as we increase the parameter values at the edges from 1 to ∞ , the outputs for positive (negative) examples will move to 1 (0). We will come back to the BAG in Figure 8 later because it illustrates an idea to classify arbitrary discrete functions with classification BAGs.

4 Learning

We finally discuss some ideas for learning classification BAGs from data. We divide the learning problem into parameter and structure learning. For parameter learning, we suppose that the classification BAG is already given and we only have to learn the weights. For structure learning, both the classification BAG (hidden arguments, edges) and the parameters have to be learnt.

4.1 Parameter Learning

One common way to learn the parameters of a model is to minimize a loss function that measures the discrepancy between the desired label and the output of the classifier. Recall from Figure 4 that our classifiers are probabilistic classifiers (if not by default, then due to the softmax normalization). A standard loss function for such classifiers is the logistic loss (a.k.a. cross-entropy loss). Given a classifier c_{Θ} with parameter vector Θ and examples $E = \{(\mathbf{x}_i, y_i) \mid 1 \le i \le N\}$, the logistic loss of the current parameters is $L(\Theta) = -\frac{1}{N} \sum_{i=1}^{N} \log c_{\Theta}(\mathbf{x}_i, y_i)$. Intuitively, it simply takes the negative logarithm of the classifier's confidence for the right label for every example. If the confidence for the right label is 1 (and, thus, the confidence for the wrong labels is 0), the loss is 0. As the confidence goes to 0 the loss goes to infinity. If we do not use the softmax normalization, we have to be careful that the output for the desired label is non-zero, but this can be guaranteed by some modifications of the actual implementation.

The loss is often minimized by using optimized variants of gradient descent. In general, the loss function for classification BAGs may be non-differentiable. However, due to the simple structure of our proposed models, we can guarantee differentiability in several cases. The key observation is that the degrees of acceptance can be computed by a simple forward pass for both the P-BAGs and G-BAGs that we discussed. Therefore, the acceptance degree at the label arguments is just a composed function of the inputs. If the involved functions are differentiable, the loss functions are differentiable and the gradient can be computed by automatic differentiation as implemented in libraries like Tensorflow. For our proposed P-BAGs, the loss function is non-differentiable at some points due to the use of the maximum. This may actually not cause any problems, but we could also make the loss function differentiable by replacing the maximum in our constraints with a squared maximum. For our G-BAGs, the loss function is already differentiable.

In general, we cannot apply gradient methods naively because our parameter ranges are constrained. However, the constraints are rather simple and can be dealt with by doing the following after every update step:

- 1. If a parameter θ exceeds a threshold b, set the parameter to $\frac{\theta_0+b}{2}$, where θ_0 is the previous value of the parameter.
- 2. Normalize parameters that have to be normalized (attack and support parameters for P-BAGs).

For the classification BAGs that we introduced here, we can actually get rid of the non-negativity constraint by considering edges with negative weights as attacks and edges with positive weights as supports. This is also beneficial because it allows to learn the nature of an edge (attack or support) from data. However, for other instantiations, like a G-BAG with product for aggregation, this may not be possible as easily. Furthermore, for more general classification BAGs, it may not be possible to obtain a differentiable loss functions. We are planning to look at two strategies for these cases:

- Heuristic Gradients: compute a heuristic gradient of the loss function by replacing the partial derivative for the i-th parameter with the approximation $\frac{L(\Theta')-L(\Theta)}{\epsilon}$, where Θ' is obtained from Θ by increasing the i-th parameter by a small constant $\epsilon > 0$.
- Meta-heuristics: apply meta-heuristics for solving numerical optimization problems that do not require gradient information like Differential Evolution or Particle Swarm Optimization [6].

4.2 Structure Learning

In principle, we could just build a huge classifier BAG and apply a parameter learning algorithm in order to solve a classification problem. Similar to the idea of the approximation theorems for neural networks [8], a classification BAG with a single hidden layer can learn to approximate arbitrary discrete functions. The intuitive explanation is that we can just generate one hidden argument for every possible input (or region of input values) that is supported by the input arguments that agree with this input and attacked by the remaining input arguments. This argument then supports the desired label and attacks all other labels. The classification BAG for XOR in Figures 8 illustrates the idea (we eliminated some redundancy, though). However, of course, this is not a very meaningful model and will probably overfit the noisy dataset after training. It will also be very difficult to interpret such a model.

Since interpretability is our main motivation, we want to learn a sparse classification BAG. The situation is similar for Bayesian networks, where one usually wants to learn a compact network with as few spurious edges as possible [13]. Bayesian networks have some local decomposition properties that we cannot exploit. However, there are some general learning ideas that we can immediately apply. One way to learn the structure of Bayesian networks is by means of a local search prodedure that starts from some random graph and then repeatedly

- 1. creates a neighborhood of the graph by means of search operators,
- 2. evaluates (a subset of) the graphs in the neighborhood by minimizing the loss function for this graph,
- 3. picks the best neighbor for the next iteration

until the graph cannot be improved anymore [13]. Step 2 can be easier for Bayesian networks because often closed-form solutions for the optimal parameters exist. We will usually have to perform a parameter search instead and may end up with parameters that are only locally optimal. An interesting question is therefore if we can set up classification BAGs such that we obtain closed-form solutions or, at least, loss minimization problems with a unique minimum. For example, for P-BAGs with the linear constraints from (1), this may be possible.

Common search operators for Bayesian networks are adding, deleting and reversing edges. Reversing edges currently does not play a role for us, but may be interesting in order to obtain more expressive classifiers. Turning an attack edge into a support edge may be another useful operator. We also want to add hidden arguments. Since just adding arguments without any edges cannot change the classification outcome, operators may introduce new arguments between existing connections and summarize these connections.

The search can be guided by local search heuristics like simulated annealing or beam search [6]. In order to simplify the search process, to avoid overfitting and to improve interpretability, it is reasonable to restrict the possible structures. In addition to the forward structure that we assumed throughout this paper, it seems also reasonable to restrict the number of layers, the number of arguments per layer and the number of ingoing and outgoing edges. We are currently working on an implementation to evaluate different strategies.

5 Related Work

There has been growing interest in combining argumentation technology and machine learning in recent years. [25] proposed some ideas for solving classification problems by means of structured argumentation. As opposed to abstract argumentation, structured argumentation explicitly takes the structure of arguments like their premises and conclusion into account. The idea in [25] is to apply a rule mining algorithm first in order to learn structured arguments. A structured argumentation solver can then be applied in order to derive a label for given inputs and to explain the outcome. While this is a very interesting idea for explainable classification, it relies very much on the success of the underlying rule mining algorithm. In particular, it is currently unclear how to train such a model in an end-to-end fashion such that the rule mining process is guided by the classification outcome of the reasoner.

Gradual argumentation frameworks have been combined successfully with machine learning methods in order to add explainability to problems like product recommendation [22] or review aggregation [5]. However, the argumentation framework and the machine learning method are again not trained in an end-to-end fashion. Instead, a machine learning method is applied first and the argumentation framework is applied on top of the result.

[9] recently proposed some ideas for learning epistemic constraints for P-BAGs from data. The constraints are more general than what we considered here and have the form of rules. The premise of these rules consists of a conjunction of atomic probability statements and the conclusion is an atomic probability statement as well. Using a constraint learning algorithm like that may be an alternative to learn the structure of a classification BAG without repeatedly calling a parameter learning algorithm.

There also has been some work on learning classical argumentation frameworks from sets of accepted arguments [24, 12]. However, the motivation is very different from our motivation. While argumentation usually asks, given an argumentation graph, which arguments can be accepted, the authors in these works ask, given arguments accepted by users, what is the underlying argumentation graph? Therefore, these approaches do not allow for a distinction between input and output arguments and the investigated frameworks are restricted to attack-relations only. In particular, the training procedure is currently based on argumentation rationales, rather than based on a classification outcome.

In principle, we could also instantiate classification BAGs with classical bipolar frameworks [2, 15, 20]. The degree of acceptance at an argument can then be defined as the relative frequency of labellings that accept the argument, where we restrict to such labellings that accept the input arguments. The resulting loss function will be non-differentiable, however, and computing all labellings can be very expensive. It is interesting to note, though, that the relative frequencies can often be computed by encoding the argumentation problems as a Markov network [21]. This relationship may also be helpful to apply learning algorithms for Markov networks for learning classical classification BAGs and other probabilistic classification BAGs discussed in [21].

6 Conclusion

We presented some conceptual ideas for solving classification problem by means of abstract argumentation technology. One important difference to previous combinations of argumentation and machine learning is that our framework can be trained in an end-to-end fashion.

Our classification BAGs are structurally similar to feedforward neural networks and Bayesian networks and we took a lot of inspiration from these fields. Naive classification BAGs are to deep classification BAGs as logistic regression is to deep multilayer perceptrons and as a naive Bayes classifier is to complex Bayesian networks. As we argued before, from a classification perspective, deep classification BAGs can be seen as universal function approximators that can theoretically approximate arbitrary discrete functions.

Our hope is that Classification BAGs can offer better transparency and explainability than other numerical classifiers. For example, for deep neural networks, transparency is often lost due to the deep and dense structure of the network. Bayesian networks look very intuitive, but are frequently misinterpreted. For example, practitioners often assume that edges encode causal relationships, while the actual theory only assumes that missing edges encode independencies [13]. Classification BAGs are less susceptible to misinterpretations because the meaning of attack and support edges is very intuitive. The calculations of degrees of acceptance seem also easier to grasp than the marginal probability computations that underlie Bayesian networks.

Of course, the applicability of classification BAGs will depend on the availability of reliable learning algorithms and an experimental evaluation is necessary in order to evaluate their performance. We are currently implementing different methods in a Python library and will start an experimental evaluation soon.

References

- Amgoud, L., Ben-Naim, J.: Evaluation of arguments in weighted bipolar graphs. In: European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU). pp. 25–35. Springer (2017)
- Amgoud, L., Cayrol, C., Lagasquie-Schiex, M.C.: On the bipolarity in argumentation frameworks. In: International Workshop on Non-Monotonic Reasoning (NMR). vol. 4, pp. 1–9 (2004)
- Baroni, P., Rago, A., Toni, F.: How many properties do we need for gradual argumentation? In: AAAI Conference on Artificial Intelligence (AAAI). pp. 1736–1743. AAAI (2018)
- Baroni, P., Romano, M., Toni, F., Aurisicchio, M., Bertanza, G.: Automatic evaluation of design alternatives with quantitative argumentation. Argument & Computation 6(1), 24–49 (2015)
- Cocarascu, O., Rago, A., Toni, F.: Extracting dialogical explanations for review aggregations with argumentative dialogical agents. In: International Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 1261–1269 (2019)
- Du, K.L., Swamy, M.: Search and optimization by metaheuristics. Techniques and Algorithms Inspired by Nature; Birkhauser: Basel, Switzerland (2016)

- 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)
- 8. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
- Hunter, A.: Learning constraints for the epistemic graphs approach to argumentation. In: Computational Models of Argument (COMMA). p. to appear. IOS Press (2020)
- Hunter, A., Polberg, S., Thimm, M.: Epistemic graphs for representing and reasoning with positive and negative influences of arguments. Artificial Intelligence 281, 103236 (2020). https://doi.org/10.1016/j.artint.2020.103236
- Hunter, A., Thimm, M.: On partial information and contradictions in probabilistic abstract argumentation. In: International Conference on Principles of Knowledge Representation and Reasoning (KR). pp. 53–62. AAAI Press (2016)
- Kido, H., Okamoto, K.: A bayesian approach to argument-based reasoning for attack estimation. In: Sierra, C. (ed.) International Joint Conference on Artificial Intelligence (IJCAI). pp. 249–255 (2017)
- Koller, D., Friedman, N.: Probabilistic graphical models: principles and techniques. MIT press (2009)
- Mossakowski, T., Neuhaus, F.: Modular semantics and characteristics for bipolar weighted argumentation graphs. arXiv preprint arXiv:1807.06685 (2018)
- Oren, N., Norman, T.J.: Semantics for evidence-based argumentation. In: Computational Models of Argument (COMMA). vol. 172, pp. 276–284. IOS Press (2008)
- Potyka, N.: Continuous dynamical systems for weighted bipolar argumentation. In: International Conference on Principles of Knowledge Representation and Reasoning (KR). pp. 148–157 (2018)
- 17. Potyka, N.: A tutorial for weighted bipolar argumentation with continuous dynamical systems and the java library attractor. International Workshop on Non-Monotonic Reasoning (NMR) (2018)
- Potyka, N.: Extending modular semantics for bipolar weighted argumentation. In: International Conference on Autonomous Agents and MultiAgent Systems (AA-MAS). pp. 1722–1730 (2019)
- Potyka, N.: A polynomial-time fragment of epistemic probabilistic argumentation. International Journal of Approximate Reasoning 115, 265–289 (2019). https://doi.org/10.1016/j.ijar.2019.10.005
- 20. Potyka, N.: Abstract argumentation with markov networks. In: European Conference on Artificial Intelligence (ECAI). p. to appear (2020)
- Potyka, N.: Bipolar abstract argumentation with dual attacks and supports. In: International Conference on Principles of Knowledge Representation and Reasoning (KR). p. to appear (2020)
- Rago, A., Cocarascu, O., Toni, F.: Argumentation-based recommendations: Fantastic explanations and how to find them. In: International Joint Conference on Artificial Intelligence (IJCAI). pp. 1949–1955 (2018)
- Rago, A., Toni, F., Aurisicchio, M., Baroni, P.: Discontinuity-free decision support with quantitative argumentation debates. In: International Conference on Principles of Knowledge Representation and Reasoning (KR). pp. 63–73 (2016)
- Riveret, R., Governatori, G.: On learning attacks in probabilistic abstract argumentation. In: Jonker, C.M., Marsella, S., Thangarajah, J., Tuyls, K. (eds.) Proceedings of AAMAS'16. pp. 653–661. IFAAMAS (2016)
- Thimm, M., Kersting, K.: Towards argumentation-based classification. In: Logical Foundations of Uncertainty and Machine Learning Workshop. vol. 17 (2017)