Argflow: A Toolkit for Deep Argumentative Explanations for Neural Networks

Adam Dejl*
Imperial College London, UK
adam.dejl18@ic.ac.uk

Peter He*
Imperial College London, UK
peter.he18@ic.ac.uk

Pranav Mangal*
Imperial College London, UK
pranav.mangal18@ic.ac.uk

Hasan Mohsin*
Imperial College London, UK
hasan.mohsin18@ic.ac.uk

Bogdan Surdu*
Imperial College London, UK
george-bogdan.surdu18@ic.ac.uk

Eduard Voinea*
Imperial College London, UK
eduard-george.voinea18@ic.ac.uk

Emanuele Albini
Imperial College London, UK
emanuele@ic.ac.uk

Piyawat Lertvittayakumjorn
Imperial College London, UK
pl1515@ic.ac.uk

Antonio Rago
Imperial College London, UK
a.rago15@ic.ac.uk

Francesca Toni
Imperial College London, UK
ft@ic.ac.uk

ABSTRACT

In recent years, machine learning (ML) models have been successfully applied in a variety of real-world applications. However, they are often complex and incomprehensible to human users. This can decrease trust in their outputs and render their usage in critical settings ethically problematic. As a result, several methods for explaining such ML models have been proposed recently, in particular for black-box models such as deep neural networks (NNs), but these are predominantly explaining outputs in terms of inputs, disregarding the inner workings of the ML model computing those outputs.

We present *Argflow*, a toolkit enabling the generation of a variety of *'deep' argumentative explanations (DAXs)* for outputs of NNs on classification tasks. Argflow comprises a Python library for generating DAXs as well as a web portal for delivering these DAXs to users with differing requirements. The design of Argflow is based on principles of modularity and extensibility, ensuring that it is flexible enough to be used for a variety of applications.

KEYWORDS

Computational Argumentation, Explainable AI, Neural Networks

ACM Reference Format:
Adam Dejl, Peter He, Pranav Mangal, Hasan Mohsin, Bogdan Surdu, Eduard Voinea, Emanuele Albini, Piyawat Lertvittayakumjorn, Antonio Rago, and Francesca Toni. 2021. Argflow: A Toolkit for Deep Argumentative Explanations for Neural Networks. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), London, UK, May 3–7, 2021*, IFAAMAS, 4 pages.

1 INTRODUCTION

Recently, machine learning (ML) models have been successfully applied in a variety of real-world settings, including self-driving cars, automated translation, diagnostic engines, or job applicant screening. In many such deployments, understanding why certain outputs are generated can be critical. As a simple example, if an ML model is deployed in healthcare for treatment recommendation, a medic (or patient) may wish to know why one treatment was suggested over another. In other settings, explanations of ML systems may be needed to assess the presence of algorithmic bias.

For some ML models, such as decision trees or k-nearest neighbours, generating explanations is relatively straightforward; one may say that they are *intrinsically interpretable*. However, for some ML models, and in particular those based on modern machine learning algorithms such as deep artificial neural networks (NNs), it is often difficult to understand why a certain output is generated, even for experts in ML. The development of methods and systems for extracting human-interpretable descriptions of black-box model behaviour such as NNs has thus recently received much attention in the field of explainable artificial intelligence (XAI), e.g. with *post-hoc* approaches for explanation. These include feature importance methods (such as LIME [5] and GradCAM [6]), prototype-based methods (such as activation maximisation [3]), model extraction (such as [2]) and counterfactual explanations (such as [7]). However, the majority of research has hitherto been focused on explaining the output of machine learning models solely in terms of the input, without providing intuition regarding the inner workings of a given model.

Recently, a novel method of *deep argumentative explanations (DAXs)* has been proposed, drawing ideas from computational argumentation [1]. The advantage of DAXs over previous methods is that it constructs 'deep' explanations that reflect the internal influence structure of a model. In a convolutional neural network (CNN), this may correspond to how the detection of lower level features (such as linguistic or facial features) influence the detection of higher level features (such as text or face classification). Moreover, as the concepts of debating and argumentation are generally well-understood by human users, the explanations generated by computational argumentation can often be more intuitive than

*These authors contributed equally.

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, London, UK.

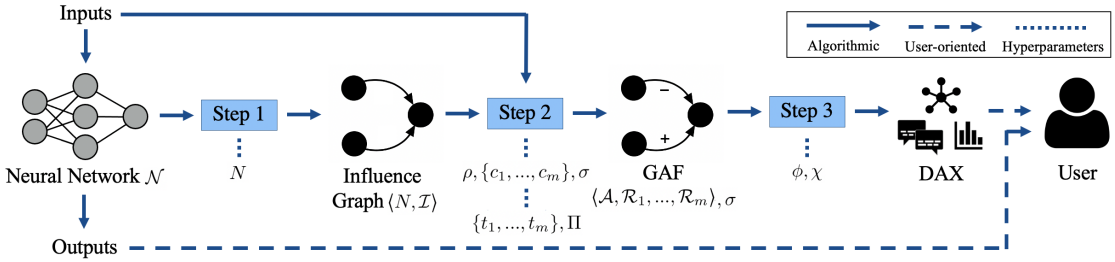

Figure 1: Adapted from [1]: DAX methodology (alongside the typical process of obtaining outputs from a neural model given its inputs) comprising steps: 1. Based on the chosen *nodes* N within $\mathcal{N}$, extract $\langle N, \mathcal{I} \rangle$, a directed graph of *influences* between nodes; 2. Extract a *Generalised Argumentation Framework (GAF)* from the output of the first step, based on choices of *argument mapping* ρ, *dialectical relation characterisations* $\{c_1, ..., c_m\}$ and *dialectical strength* σ. These choices are driven by the types $\{t_1, ..., t_m\}$ of dialectical relations to be extracted and the *dialectical properties* Π that σ should satisfy (on the GAF) to form the basis for explanations; 3. Generate a *DAX* from the GAF for user consumption in a certain *format* ϕ associating arguments with human-interpretable concepts through a *mapping* χ.

explanations generated using other methods. The overall DAX methodology is summarised in Figure 1. This involves constructing an *influence graph* (Step 1), converting it to a *generalised argumentation framework (GAF)* (Step 2) and then displaying the GAF to users in the relevant format with individual arguments visualised in a human-interpretable format (Step 3). We refer the reader to [1] for further details on the DAX methodology and its use of computational argumentation.

2 ARGFLOW

We develop a generic toolkit for constructing DAXs for neural networks. The code is available at https://gitlab.com/argflow, and a video of experiments can be found at https://youtu.be/LPz4QbmLaxs.

Python Library. We collapse the first two steps into a single *GAF extraction step* handled by `GAFExtractor` class which implements dependency injection pattern. The `GAFExtractor` constructor takes an a `StrengthMapper`, an `InfluenceMapper`, a `CharacterisationMapper` and a `Chi` object as arguments corresponding roughly to σ, a slightly embellished ρ, c_j and χ in [1] respectively. The `GAFExtractor` class exposes a single `extract()` method which, given a model and its input, will return a GAF (represented by the `GAF` class) for the model run on its input.

To extract the influence graph (which is represented by the `InfluenceGraph` class) and arguments, we provide an abstract class `InfluenceMapper`. Developers implement the `apply()` function which extracts the necessary `InfluenceGraph` given a model and some input. The `InfluenceGraph` class is similar to a standard graph implementation with the nodes being able to store any data useful for the next steps. Moreover, in order to decide what constitutes a 'strong' argument, we use the `StrengthMapper` abstract class. Like for the `InfluenceMapper` class, developers implement the `apply()` function which returns a strength value for a given node in the `InfluenceGraph` object. To assign dialectical relations between arguments, we provide the `CharacterisationMapper` abstract class. Again, developers implement the `apply()` function, this time returning the dialectical relations (represented by an enumeration) between two nodes. Note that this is slightly different from c_j in that `apply(u, v)` on two nodes `u` and `v` returns the relation $\mathcal{R}_i$ for which $c_i = true$.

In order to visualise arguments in a human-interpretable modality, we provide the `Chi` abstract class which generates a visualisation given a specific node, a model and its input. These visualisations are stored in `Payload` objects associated with each node. Argflow provides several out-of-the-box concrete implementations of the `Chi` class (GradCAM and activation maximisation for convolutional filters), as well as the other abstract classes mentioned above.

Finally, Argflow provides utilities to serialise the generated GAFs as JSON files for use in the web portal using the `Writer` class.

Web Portal. This provides users with the ability to visualise GAFs in different formats (corresponding to ϕ in Figure 1). We use a typical web app architecture, with the frontend implemented as a React app using JavaScript, and a Python server for the application's backend. Note that the portal is not designed to be deployed publicly over a network. Rather, the server is intended to be run locally on a single machine, by a single user. Several parts of the portal have been designed to be customised and extended by the user.

The portal provides a graphical interface to quickly import some classes of model and generate DAXs for them. However, this functionality can be extended with the `ExplanationGenerator` abstract class which contains a method that can be called to generate explanations, taking in any necessary parameters. Developers can create their own `ExplanationGenerator` implementation in a plugins folder and, in the portal's GUI, a menu allows users to select which generator implementation they would like to use.

The visualisation system itself is also extensible. It uses an abstracted interface that enables plugging in additional visualiser implementations at runtime to allow for new types of visualisations. We provide two built-in visualisation types: graph-based and conversation-based. The former was capable of comfortably rendering up to 3000 nodes and 6000 edges on an Intel i7-6500U CPU with 16GB RAM, though in practice rendering so many arguments may lead to information overload for the user and is not recommended.

Demo Application. We present two demos generating explainations for VGG-16 [4] and a feed-forward NN.

REFERENCES

[1] Emanuele Albini, Piyawat Lertvittayakumjorn, Antonio Rago, and Francesca Toni. 2020. DAX: Deep Argumentative eXplanation for Neural Networks. *CoRR* abs/2012.05766 (2020). arXiv:2012.05766 https://arxiv.org/abs/2012.05766

[2] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. 2017. Interpreting Blackbox Models via Model Extraction. *CoRR* abs/1705.08504 (2017). http://arxiv.org/abs/1705.08504

[3] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2009. *Visualizing Higher-Layer Features of a Deep Network.* Technical Report 1341. University of Montreal. Also presented at the ICML 2009 Workshop on Learning Feature Hierarchies, Montréal, Canada.

[4] Shuying Liu and Weihong Deng. 2015. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR).* 730–734. https://doi.org/10.1109/ACPR.2015.7486599

[5] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16).* Association for Computing Machinery, New York, NY, USA, 1135–1144. https://doi.org/10.1145/2939672.2939778

[6] Ramprassath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *2017 IEEE International Conference on Computer Vision (ICCV).* 618–626. https://doi.org/10.1109/ICCV.2017.74

[7] Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. 2017. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. *CoRR* abs/1711.00399 (2017). http://arxiv.org/abs/1711.00399

REQUIREMENTS

For an ‘in-person’ demo using a laptop, a screen equipped with an HDMI lead and a power extension cord will suffice.