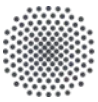

University of Stuttgart

Institute for Parallel and Distributed Systems
- Analytic Computing Department -

Universitätsstraße 32
70569 Stuttgart

Master Thesis

Explanation-based Learning with Feedforward Neural Networks

Jan Gemander

Study program:	M.Sc. Computer Science
1. Examiner:	Prof. Dr. Steffen Staab
2. Examiner:	Ph. D. Jim Mainprice
Advisors:	Mr. Zihao Wang
start date:	30.08.2021
end date:	28.02.2022

Abstract

The idea of this thesis is to adapt Možina et al.'s method [13] of exploiting experts' arguments to neural networks. This is done by incorporating the method of Ross et al. [17] to include an explanatory loss, which penalises attention on the wrong features. More specifically, we present a novel approach that in addition to recognising positive influencing features distinguishes between negative and neutral ones. Here we propose new variants of reinforcing correct explanations in our losses. Additionally, we want to improve results by using Shapley values contributions, which provides many desirable traits. In doing so we're concentrating the neural network to learn reasons for predictions that were specified in the experts' arguments. This leads to more predictable results of explanations generated on our network, which do not rely on unfamiliar dependencies.

Zusammenfassung[1]

Die Idee dieser Arbeit ist es, die Methode von Možina et al. [13] zur Nutzung von Expertenargumenten in neuronale Netze einzubinden. Dies geschieht durch Verwendung der Methode von Ross et al. [17], welche den Fehler des Netzes um einen Term erweitert, der die Aufmerksamkeit auf falsche Attribute bestraft. Genauer gesagt stellen wir einen neuen Ansatz vor, der neben der Erkennung von Attributen mit positivem Einfluss auch zwischen negativen und neutralen unterscheidet. Hier schlagen wir neue Varianten vor, welche die Fähigkeiten unseres Netzwerkes, spezifizierte Erklärungen zu lernen, verbessern sollen. Zusätzlich wollen wir die Ergebnisse durch die Verwendung von Shapley Werte verbessern, welche uns viele wünschenswerte Eigenschaften bieten. Dadurch konzentrieren wir das neuronale Netz darauf, Gründe für Vorhersagen zu lernen, die den Argumenten der Experten ähneln. Dies führt zu einer erhöhten Vorhersehbarkeit von auf unserem Netzwerk generierten Erklärungen, welche besser in menschlich verstehbaren Gründen fundiert sind.

[1] German translation of abstract.

Contents

1 Introduction

Within the last decades machine learning has entered into the everyday life of humans. The idea of artificial intelligence in form of neural network is of particular interest to the general populace. Most people carry an assistant such as Alexa or Siri around on their smartphones, and the concept of self driving cars is no longer fiction. Additionally, most people interact with recommender systems (Youtube, Facebook), used to maximise ad revenues and user retention, on a daily basis. Due to their adaptability neural networks have been deployed in a multitude of categories including natural language problems, healthcare, law and even military. They have been shown to outclass other approaches, including humans, in a lot of fields requiring fast and complicated reasoning.

Prevalence of artificial intelligence will most likely keep rising within the next years. Many experts have warned people of dangers that come with artificial networks. Leaving medical decisions or control of weapons to an algorithm with no morals seems to be bold at best, if not outright dangerous. Transparency of network's decisions is required in order to establish trust with humans. Teach et al. [24] have shown that medical personal is more likely to use a clinical consultation system, if it incorporated explanations. In 2017 DARPA initiated an Explainable Artificial Intelligence (XAI) program [7] in order to produce better explainable models. The goal is to produce explanation which can be translated into an understandable and useful domain for a human end user. In 2018 the EU has also passed GDPR article 15 (h), which gives a subject of automated decisions the right to receive "meaningful information about the logic involved".

Such explanations can help humans come to an understanding of the generated predictions. However, these networks still run the risk of learning wrong predictions or wrong explanations, which could wrongfully convince people. Additionally, in case of time sensitive problems, there's no time for a person to check the results. Možina et al. [13] provided a rule mining algorithm that incorporates positive and negative reasons for classes to improve generated rules. Similarly, for neural networks Teso et al. [25] introduced the concept of explainable interactive learning (XIL), where we interact with the explanations during training. This can be done by either supplying every training example with explanations [17], or by selectively correcting examples with wrong explanations [25].

1.1 Research Questions

We want to adapt the method by Možina et al.[13] to neural networks by selecting features as positive, neutral, or negative reasons for a prediction. Our method to do so is an adaption of papers [17, 20], which minimise explanations of features not relevant to the prediction. We hope that by additionally including negative reasons for the prediction the network can properly differentiate between features that are harmful to the prediction, and those that are just irrelevant. For example, a self driving car should recognise the different importances of driving towards a trash bag or a person. One of which is pretty much trivial, while the other case should be avoided at all cost.

Research Question 1 Can we sufficiently teach a network to properly identify features that should have positive, neutral and negative explanations, by extending the method of Ross et al. [17] to additionally distinguish between features with neutral and negative impact?

Research Question 2 Does correctly identifying explanations improve the performance of our network?

Additionally, we want to compare the methods of penalising unwanted explanations of features (eg. positive explanations of all features not selected as positive reasons) to rewarding/penalising contributions of features (e.g. reward/penalise positive/negative explanations of features selected as positive reasons).

Research Question 3 What difference does it make for our network to prevent false explanations or to reinforce correct explanations?

For this we want to use Shapley values [18], a concept first introduced to game theory in 1953, which has become a widely popular method for computation of explanations. This method computes explanations by looking at all possible feature combinations, thereby also learning synergies of features. Previous methods incorporating explanations in their loss used gradient [17] and integrated gradient [20] instead. Gradient is a naive approach only looking at a local explanation of a single feature, integrated gradient goes along that path. Both of these don't really include alterations of other features. Here we hope that our methods benefit from properties of Shapley values.

Research Question 4 Can we improve quality of explanations of our network using Shapley values?

1.2 Thesis Outline

Through the outline of this thesis we proceed to answer these research question, by taking the reader on a journey, which should convey the thoughts and ideas involved. In section 2 we want to introduce the reader to a selection of related research. In order to give the reader a rough overview on the topic, we'll cover popular ideas related to this thesis. Within section 3 we attempt to give the reader a better understanding on the topic by going into detail. More specifically, here we want to properly define the background foundations from other research that our work heavily depends on.

Finally in section 4 we propose our methods using these foundations. We first introduce our problem and then proceed to propose our novel approach. Furthermore, we want to discuss the implementations of our methods. Section 5 is used to discuss the datasets we're working on and the tests we intend to do in order to answer our research questions. Furthermore, it describes our proposed metrics. Results of these tests can be seen in section 6. Here we compare various metrics, losses and explanation functions. Moreover, we plan to do an ablation study of our proposed loss. Using these results we come to a conclusion in section 7, where we relate our results to the above stated research questions and propose future work that can be done to further research the topic.

2 Related Work

In this section we want to introduce ideas from related research. While this work is closely related to neural networks, this section won't introduce research on their general idea. Instead we have tried to describe their function more in detail in section 3.2.

Arguments Argument based machine learning (ABML) is about using arguments supporting and attacking an assertion for correct classification. Možina et al. [13] introduced a covering algorithm on annotated classification data such as [credit: approved, because: rich, despite: not paying regularly] to mine rules. Knowledge gained through arguments is used to improve performance and quality of the mined rules. These had to support positive arguments while omitting negative arguments.

Explainable Artificial Intelligence (XAI) XAI is the concept of explaining the decisions of artificial intelligence, namely neural networks. Due to their structure they are basically a black box systems for humans. Explanations have gotten a lot of track in recent years, and as a result multiple variants have been developed. Model agnostic methods generate explanations by simulating variants of the input. LIME [16] samples randomly locally around a point and attempts to create a linear separation of classes. Shapley [18], adapted to neural networks as baseline Shapley, attempts to simulate all possible feature permutations, where features are omitted by replacing them with a baseline value. Various methods have been derived on the ideas presented by Shapley values [11, 3].
Back-propagation methods portray another option to generate explanations. Here the idea is to use activations in the network to compute explanations generated from gradients of predictions with respect to the data. Popular methods include DeepLIFT [19], LRP [10] and IG [23].

Interactive Learning The field of explainable interactive learning (XIL) focuses on interacting with the explanations of a predictor model, to prevent cases where the model predicts the correct result because of the wrong reasons. Ross et al. [17] introduced an additional explanatory loss factor, which should restrict the networks explanations to the right reasons. The loss factor penalises the attention on wrong reasons, by summing up their gradients. Stammer et al. [20] expand the explanatory loss term to include explanations of both a concept and a reasoning module attempting to teach the network the correct concept. The reasoning module estimates the importance of symbolic representations using the integrated gradient method, while

the concept explanations are an attention mapping restricted by importance of the respective symbolic representation.
Teso et al. [25] interact with explanations computed with LIME. Wrong explanations are corrected, by adding generated counterexamples to the training dataset.
ELIXIR by Ghazimatin et al. [6] is a graph recommender system featuring explanations of recommendations that the user can interact with. More specifically a feedback matrix is stored for every user that incorporates how much a user likes <item, explanation> combination pairs to create further recommendations. Arous et al. [4] learned a model that incorporates reasoning, in form of attention on relevant areas, for text classification using a Bayesian inference framework. Attention values were derived from human annotators, including a learned reliability factor of the workers to counteract bad annotations.

Other related methods There have also been experiments incorporating explanations in training, without specific annotations. Sun et al. [21] have found an improved generalisation in networks guided by their explanations. These were computed for every feature using LRP and normalised to a range [-1,1] to create a weighing factor. Prediction errors are then computed on both normal and weighted data. The final loss contains a sum of these prediction errors.

3 Background

In this section we want to introduce the foundations from other research that this work heavily relies on.

3.1 Machine Learning

The field of machine learning (ML) is incredibly large. In order to constrain the scope of this work, we'll restrict ourself to only roughly covering the ideas appearing in this thesis. For a better understanding of the field we suggest reading Bishop's "Pattern Recognition and Machine Learning"[5]. ML itself is essentially the idea of teaching a computer to make predictions on data. More precisely the goal is to find patterns in the data, through usage of algorithms and heuristics, that determine the prediction. Data itself can be in any shape or form that a computer can manipulate. For proper results it should be as best correlated to the prediction as possible. In practice the original data is usually preprocessed to extract relevant features. Here a feature is defined as follows:

Definition 1 *A feature is an element used as input for a ML algorithm. For an input of data point $x = (x_1, \ldots, x_n)$ the attribute x_i is the i-th feature.*

Usually these features contain numerical values. For example for images we can use values of pixels as independent features. Meanwhile in most natural language problems (NLP) words are encoded as a vector describing the context in which the word usually appears.

The ML algorithm works on a set of data points in order to learn predictions in form of a mapping $f(x) \to y$. In this study we're working on supervised learning problems where for every point x used during training, we are previously aware of the correct result y^*. More specifically all of our problems will be classification tasks. We refer to classification task when y^* is categorial. Here the result is a one-hot vector $y^* = (y_1^*, \ldots y_m^*) \in \{0, 1\}^m$ with exactly one 1, in place of the correct class of a corresponding data point x. Consider the problem of classifying animal data into mammals, birds, reptiles and insects. In this case there are four possibilities for the class so $y^* \in \{0, 1\}^4$. A distinguishing feature could for example be the existence of feathers, in which case the corresponding class vector would be $y^* = [0, 1, 0, 0]$, where y_2^* represents the class of birds. The final ML algorithm computing the mapping $f(x) \to y$ is called a classifier.

There are many types of classifiers, among them simpler ones such as logistic regression or support vector machines. An especially popular classifier nowadays are artificial neural networks (ANN), first introduced in 1943 by McCulloch et al. [12].

Their idea is mimic a simplified version of neurons in brains, using a connected set of artificial neurons. We'll discuss them in detail in the next section.

3.2 Neural Networks

Neural networks are made up by a set of artificial neurons. Similar to biological neurons each of these artificial neurons has several inputs and computes one output. For an input of shape $z^{(in)} \in \mathbb{R}^k$ this is usually done by computing the following:

$$z^{(out)} = h(b + \sum_{c=1}^{k} w_c * z_c^{(in)})$$

Where $w = w_1, \ldots, w_k$ are the learned weights, b a learned bias and h the activation function of the neuron. An example for such a neuron can be seen in figure 1. Activation functions are used in order to introduce non-linearity to the network. An

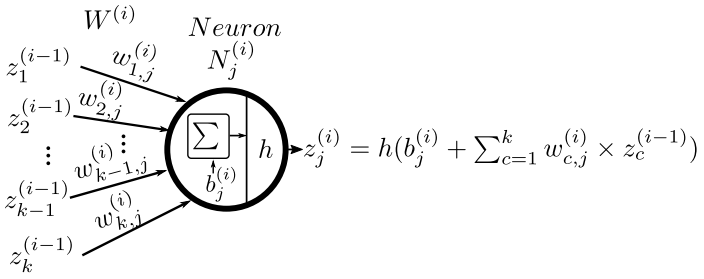

Figure 1: The *j-th* neuron in layer i of a network, with input $z^{(i-1)} = (z_1^{(i-1)}, \ldots, z_k^{(i-1)})$, parameterised by its weights $w_{1,j}, \ldots, w_{k,j} \in W^{(i)}$ and bias $b_j^{(i)}$, with activation function h.

overview of the most popular activation functions can be seen in the plot in figure 2. For example the rectified linear unit (ReLu) activation function is inspired by biological neurons, which depending on the input fires or not. The ReLu adapts this by either passing on a 0, or a positive value.

A single neuron is pretty limited in what it can predict. In fact, a single neuron is about as powerful as logistic regression. However, by connecting multiple neurons in a network we can create any arbitrary function. In general a network consists of an input layer, any assortment of hidden layers, and an output layer. Figure 3 shows an example of such a network.

In a feedforward neural network neurons can be arranged in layers such that neurons in layer i pass their result to nodes in layers $> i$. The network in figure 3 additionally features fully connected layers, meaning that every neuron passes its values exactly to every neuron within the next layer. Number of neurons in each

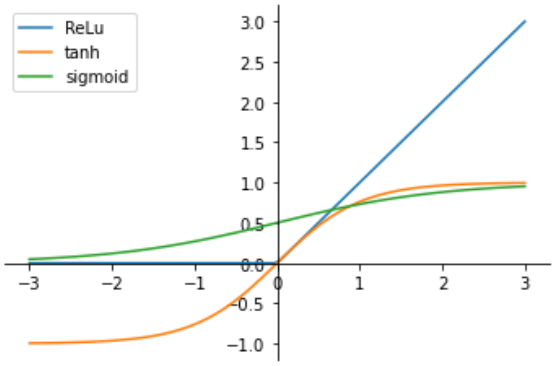

Figure 2: Plots for the activation functions ReLu(x),tanh(x) and sigmoid(x) on a range of $[-3, 3]$.

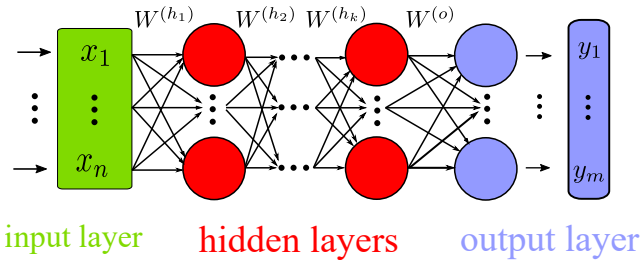

Figure 3: Feedforward neural network with weights W and fully connected hidden layers.

layer does not have to be the same, however once a model for a network is created, its shape usually does not change. There are other types of networks such as recurrent neural networks, that use backward edges to store information about the current prediction for upcoming tasks. NLP task often heavily rely on this method. In this thesis we'll focus on feedfoward networks specifically.

Assume that $l^{(i)}$ is the computation of layer i given by $l^{(i)}(z^{(i-1)}) = h(W^{(i)}z^{(i-1)} + b^{(i)})$, where h is the respective activation function of the layer. Then for a network f with n hidden layers the prediction $f(x) = y$ is computed by propagating an input x through the different layers:

$$y = l^{(n+1)}(l^{(n)}(\dots l^{(2)}(l^{(1)}(x)))) \tag{1}$$

Where layer $n + 1$ is the output layer. As described in section 3.1 the output y for a classifier with m classes is of shape $y \in \mathbb{R}^m$. In order to receive probabilistic outputs of the classes a softmax function is usually applied on the outputs. Consider

$z^{(o)} \in \mathbb{R}^m$ as result of neurons in the output layer, then:

$$y = softmax(z^{(o)}) = \frac{e^{z^{(o)}}}{\sum_{k=1}^{m} e^{z_k^{(o)}}}$$

Training is done by running the network on multiple data points and trying to reduce the error of predictions. There are different types of loss functions used to evaluate how well the prediction $f(x)$ of a data point (x, y^*) did. Regression tasks often employ the mean squared error (MSE), as the name suggests, computed by squaring the error and then taking the mean. For classification tasks the most popular choice is the cross entropy error. For two distributions $P = (p_1, \ldots, p_m)$ and $Q = (q_1, \ldots, q_m)$ the cross entropy (CE) is computed as the following:

$$CE = -\sum_{i=1}^{m} p_i * log(q_i)$$

Assuming that P is the distribution given by the one-hot vector y^*, with $y_j^* = 1$ being the target class, and Q the distribution given by the prediction of our network $f(x) = y$, this is equal to:

$$CE = -log(q_j)$$

The network learns from errors by using backpropagation to compute gradients of the error with respect to every trainable parameter, such as weights and biases. In a second step these parameters are updated using these gradients. The simplest method to do so is gradient descent, where the parameters are updated in negative direction of the gradient. Adam [9] has become especially popular method in recent years, by incorporating adaptive learning rates for every parameter using the first and second moments of the gradients.

3.3 Argument based machine learning

Another method for classifying data relevant to this thesis is the option of mining rules that can specify the output.

Definition 2 *Let D be a set of entities $\{d_1, \ldots, d_n\}$, each with their own attributes. Let I be the collection of all first order logic condition statements that can be created for attributes of our entities $I(D) = \{c_1, \ldots, c_m\}$. Then a rule is an implication of the form $A \implies B$ with $A \subset I, B \subset I$ and $A \cap B = \emptyset$. Support s is given by the percentage $s\%$ of entities d in D where the conditions $(A \cup B)$ hold true for every feature.*

The above definition is an extension of the rule definition by Agrawal et al. [1] from 1994. Here conditions may take any arbitrary form. A simple rule could be:

$$IF(age > 17) \qquad THEN \qquad adult = true$$

Where $age > 17$ and $adult = true$ are conditions for attributes of entities in D. Rules with high confidence measures are often used to automatically extend knowledge bases. In the context of classification conditions regarding both feature and class values for each data point are contained in the set of items. Rules then determine the class value based on the attributes of a data point, excluding the class attributes. There are multiple algorithms to do so efficiently. We're not really interested in the concept of rule mining itself, but more so the idea behind ABCN2 in the paper argument based machine learning(ABML) by Mozina et al. [13]. Here data points (x, y^*) are additionally augmented with arguments $A = a_1, \dots a_n$. A more detailed discussion regarding the formalism of arguments used in [13] can be found in [2]. For the purpose of this works arguments are defined as follows:

Definition 3 *An argument a is comprised of a conjunction of reasons $r_1 \wedge \cdots \wedge r_n$ that have an impact on the class value y^* of a data point x by either supporting ("speak for") or attacking ("speak against") it. Here an argument may only contain supporting or attacking reasons but not both. Arguments are then given by:*

$$\begin{array}{llll} supporting: & y^* & Because & reasons \\ or & & & \\ attacking: & y^* & Despite & reasons \end{array}$$

Here *reasons* $r_1, \dots, r_n$ are *rationales* with binary outputs. A rationale refers to a relationship $R \in \{=, <, >, +, -, \times, /\}$ of two variables x and $z : xRz$, where z is of the same domain as x. Variables can be either features, fixed values, or rationales themselves, however the final binary rational used as reason must be true for the related data point. For example consider a bank, that has made the decision to hand out a credit with a duration of 6 months and a monthly loan of 10,000 to a person of age 21, who currently does not hold a job, but has a deposit of 100,000 and so far has paid all his credits back on time. Simple arguments could then be:

$$\begin{array}{lll} loan\_granted = true & Despite & job = false \\ loan\_granted = true & Because & pays\_on\_time = true \end{array}$$

A more complicated argument containing multiple reasons and rationales could be:

$$loan\_granted = true \quad Because \quad age < 25 \wedge (duration * monthly\_loan) < deposit$$

This argument contains the two reasons $age < 25$ and $(duration * monthly\_loan) < deposit$. The latter reason is a combination of two rationals and $z < deposit$ where z is the rational given by $(duration * monthly\_loan)$. While in this example all named features are contained in arguments, this is usually not the case, as not all features have to be relevant for every class.

Arguments are given independently for each data point. While arguments and rules share similar properties, their purpose is different. An argument gives reasoning for the class of a single data point. Meanwhile, a rule of the form $A \implies B$ for data (X, Y), with $A \subseteq I(X)$ and $B \subseteq I(Y)$ the sets of possible conditions, is used to determine the class values of multiple data points $x \in X$. The authors in [13] use the knowledge gained through arguments in order to learn better rules that use the features relevant for the predicted class. Structure of a data point in ABML is as follows:

Definition 4 *An argumented example AE is a four-tuple (x, y^*, A^+, A^-), comprising an example (x, y^*), as well as a set of positive arguments A^+ and a set of negative arguments A^- with $A^+ \cap A^- = \emptyset$. A^+ comprises reasons that are important for the classification of y^*. A^- comprises reasons which "speak against the class value" [13].*

Where arguments are specified by an expert prior to training. For example, consider the above credit loan example where a bank has made the decision to hand out a credit, then the following argument sets are selected by an expert:

$$\begin{aligned} &Because && A^+ = \{age < 25 \wedge (duration * monthly\_loan) < deposit, \\ &&& \quad pays\_on\_time = true\} \\ &Despite && A^- = \{job = false\} \end{aligned}$$

Here $A+$ contains the two above specified positive arguments, and A^- the negative argument. Other features such as gender, age, or hair that are not relevant to the class are excluded from arguments.

Lets define that for a collection of attributes $x \in \mathbb{R}^n$ and a set of conditions $C = \{c_1, \ldots, c_m\}$, that $C \models x$ iff every condition c_i is true for the attributes of x. Then an $AE = (x, y^*, A^+, A^-)$ is covered by a rule $C_1 \to C_2$, if and only if all of the following are true:

$$\begin{gathered} C_2 \models y^* \\ C_1 \models x \\ \exists a : a \in A^+ \wedge a \in C_1 \\ \nexists a : a \in A^- \wedge a \in C_1 \end{gathered}$$

Remember our credit loan example of a person of age 21, who pays on time and is currently jobless. Consider the following rules for the above arguments:

$$\begin{aligned}
Rule1: &\quad IF \quad pays_on_time = true \wedge job = false \quad THEN \quad loan_granted = true \\
Rule2: &\quad IF \quad gender = female \wedge age > 20 \quad THEN \quad loan_granted = true \\
Rule3: &\quad IF \quad pays_on_time = true \wedge age > 20 \quad THEN \quad loan_granted = true
\end{aligned}$$

Rule 1 does not cover the example because it contains one of its negative arguments. Rule 2 cannot cover the example because it does not include either of the positive arguments. Rule 3 covers the example because it contains one positive argument and only contains conditions true for the example. Mining using the ABCN2 algorithm works by first evaluating arguments $a \in A^+$ as rules and then refining them. All the while rules that contain reasons in negative arguments are removed. This way the algorithm incorporates knowledge gained through arguments in order to learn improved rules depending on the correct features.

3.4 Shapley Value

Shapley values, first introduced by L. Shapley[18] in 1953 are a concept from game theory. Consider a set of n players $N = \{p_1, \ldots, p_n\}$ playing a game (v, N) in a coalitions $S \subseteq N$ with payout $v(S)$. Each player has their own impact on the payout by joining the coalition, including the possibility of negative contributions. Additionally a player can have different impacts depending on the existent coalition, for example by synergising with specific other players. Using the Shapley value it is possible to fairly compute the attribution of every player by simulating every possible ordering.

Let $sh_i(v)$ be Shapley value of player p_i computed on a game (v, N). Then the Shapley value is given by the unique attribution method satisfying all of the following axioms [18]:

- Efficiency: Sum off Shapley values for all players is equal to value function on all players: $\sum_{p_i \in N} sh_i = v(N)$
- Dummy: Players with no impact on the value function have a Shapley value of 0: $\forall p_i : \forall S \in (N \setminus p_i) : v(S \cup p_i) = v(S) \Leftrightarrow sh_i = 0$
- Symmetry: Two players with the same contributions towards every subset of players are interchangeable and have the same Shapley values:
  $\forall p_i, p_j \in N : \forall S \in (N \setminus \{p_i, p_j\}) : v(S \cup i) = v(S \cup j) \Leftrightarrow sh_i = sh_j$

- Additivity: Sum of Shapley values for two games with payouts v and w is the same as the Shapley values of their summed value functions:
  $sh(v) + sh(w) = sh(v + w)$

Often the efficiency and dummy axioms are combined under one carrier axiom. As a result of these axioms L. Shapley [18] derived the following equation:

Definition 5 *The Shapley value $sh_i(v)$ for a player p_i for a game (N, v) is defined by the players average contribution to the payout:*

$$sh_i(v) = \frac{1}{n!} \sum_{S \subseteq N \setminus p_i} |S|!(n - |S| - 1)!(v(S \cup p_i) - v(S)) \tag{2}$$

This function sums up the contributions of a player p_i when added to a coalition S for all possible subsets of N. The contribution is multiplied by the possible permutations of the current ($|S|!$) and remaining ($(n - |S| - 1)!$) players. Finally, the function averages over all possible permutations ($\frac{1}{n!}$).
For a three-player game with $v(\emptyset) = v(1) = v(2) = v(3) = 0, v(1,2) = 1, v(1,3) = 2, v(2,3) = 3$ and $v(1,2,3) = 6$ the Shapley value for player 1 would be computed as follows:

$$\begin{aligned} sh_1(v) = & \quad 1/3! * (2 * (v(1) - v(\emptyset)) + (v(1,2) - v(2)) \\ & + (v(1,3) - v(3)) + 2 * (v(1,2,3) - v(2,3)) \\ = & \quad 1/6 * (2 * (0 - 0) + (1 - 0) + (2 - 0) + 2 * (6 - 3) \\ = & \quad 1.5 \end{aligned}$$

The Shapley value vector for all players is then given by $sh(v) = (sh_1(v), \ldots, sh_n(v))^T$.
Another possibility for the computation of Shaley values is given by Owen's multilinear extension of games [15]. Assume we have a fixed set of players $N = \{p_1, \ldots, p_n\}$ and subsets $S_i \subseteq N \setminus \{p_i\}$ where the probability for occurrence of each player $p_j \in N \setminus \{p_i\}$ is exactly q such that:

$$\begin{aligned} &\mathbb{P}(p_j \in S_i) = q, && \forall i = 1, \ldots, n; j = 1, \ldots, n; i \neq j \\ &\mathbb{P}(p_i \in S_i) = 0, && \forall i = 1, \ldots, n \end{aligned}$$

Then, provided mutually independent sampling, the Shapley values for a value function v is given by the following equation [15]:

$$sh_i(v) = \int_0^1 \mathbb{E}(v(S_i \cup p_i) - v(S_i))dq \tag{3}$$

Here $\mathbb{E}$ refers to the expected value.

3.5 Explanations

Explanations, in the context of this work, are computed values that should convey the reasoning of a prediction. Specifically the goal is to describe the relation between input features and predictions of neural networks. Generally, this is done to convey this information to humans.

Definition 6 *Assume a network f with activations $\mathbb{A}$ for data points $d^i = (x, y^*)$ with $x \in \mathbb{R}^n$ trained on a set $T = \{d^1, \ldots, d^k\}$. Then an explanation method e uses metadata $\mathbb{M} = (m_1, \ldots, m_n)$ with $\mathbb{M} \subseteq \{f, \mathbb{A}, T\}$, in order to create an explanation E for a data point d^i: $e(d^i, \mathbb{M}) = E$. [8]*

There are various types of explanations E and explanation methods e. Explaining using examples from prior training instances is one option to do so. Similar to how a person may recognise how to act because of being aware of similar situations, a network can refer to previous instances in the training set for explanation the current prediction.

Most explanations methods map features to real values. Here the idea is to recognise which features have the biggest impact on predictions. An example of this is the Shapley value, which we have seen in the previous chapter. It estimated attributions of players by simulating all possibilities.

This can be adapted to neural networks by using the network function f itself as a black-box value function v. Using features $x_1, \ldots, x_n$ as players we can use the Shapley value to estimate feature's contributions to the predictions. As the Shapley value takes all permutations into account, synergies between features may be learned. The Shapley value computation for the contribution of feature x_i to a network's prediction $f(x)$ with set of all features $N = \{x_1, \ldots, x_n\}$ is then given by:

$$sh_i(f, x) = \frac{1}{n!} \sum_{S \subseteq N \setminus x_i} |S|!(n - |S| - 1)!(f(S \cup x_i) - f(S)) \tag{4}$$

Here features are omitted from a subset by replacing them with either 0 or a baseline value of the dataset. This is known as baseline Shapley and has been shown to satisfy the in section 3.4 defined axioms dummy, symmetry, and additivity [22]. A more detailed discussion of these axioms in the context of neural networks can be found in [22].
Another popular explanation variant are gradient methods that incorporate knowledge of the network and its activations $\mathbb{A}$. Notable mentions of such explanation methods, that were used in related papers are:

1. Gradient: This method was used by Ross et al. [17] as a first order explanation. It is a naive approach that computes gradients of the predictions with respect to inputs x_i:
   $\nabla_i(f, x) = \frac{\partial f(x)}{\partial x_i}$

2. Integrated gradient: This method first introduced in 2017 [23] is used in the reasoning module by Stammer et al. [20]. Satisfies dummy, symmetry, and additivity axioms [22]. This is done by integrating the above gradient over the feature space:
   $IG_i(f,x) = (x - x') \cdot \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha \times (x - x')}{\partial x_i} d\alpha$

Let the aforementioned explanation functions computing attributions be summarised as contribution functions $\phi(f,x)$. A network with multiple output values $f(x) = (y_1, \ldots, y_m)$ has contributions of each feature towards each predicted output y_j, such that:

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} \text{ and } f(x) = (y_1, \ldots, y_j, \ldots, y_m) \text{ then } \phi(f,x) = \begin{pmatrix} \phi_{1,1}(f,x) & \ldots & \phi_{1,j}(f,x) & \ldots & \phi_{1,m}(f,x) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \phi_{i,1}(f,x) & \ldots & \phi_{i,j}(f,x) & \ldots & \phi_{i,m}(f,x) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \phi_{n,1}(f,x) & \ldots & \phi_{n,j}(f,x) & \ldots & \phi_{n,m}(f,x) \end{pmatrix}$$

Here $\phi_{i,j}$ specifies the contribution of feature x_i to the prediction y_j. Then the vector specifying contributions of all features to one prediction y_j is given by $\phi_{:,j}$. Often times such real valued explanations are transformed into a visual domain, to provide better understandability for humans. For images this can be done by superimposing the real valued explanations using colouring. Similar for NLP problems one may highlight important words.

4 Methodology

This thesis attempts to adapt the idea of ABML to neural networks. In this section we first want to introduce the problem setting, then move on to our methods before finally talking about our implementation.

4.1 Problem Setting

ABML incorporated knowledge in form of arguments in mining to improve the quality of generated rules. Neural networks trained on restricted training sets can learn wrong reasons which could lead to wrong predictions on new examples. Arguably worse would be if correct reasons are identified with a wrong prediction. Here an expert using a network as guidance could be led astray. The idea behind our thesis is to adapt the approach of ABML to arguments in neural networks where arguments specify features relevant for a prediction. Namely the network should learn the correct explanations of its predictions.

ABML required conditional reasons to create rules incorporating these. In the scope of neural networks we want to use features themselves as reasons for the prediction. Therefore, we will refer to $x_i \in A^+$, if feature x_i is a reason contained within positive arguments, and $x_j \in A^-$ if feature x_j is a reason contained within negative arguments. An example of our goal can be seen in figure 4. Considering that we're

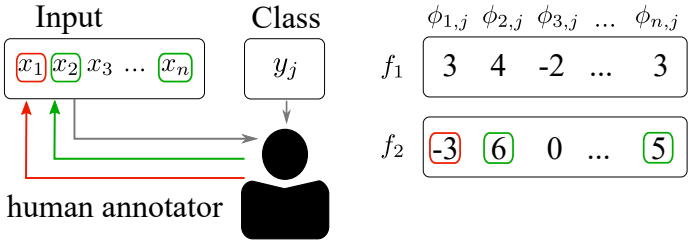

Figure 4: Annotator looking a data point (grey arrow) selects (coloured arrows) features for positive (green) and negative (red) arguments of y_j. f_1 is a network trained without the focus on selected features, f_2 focuses contributions (e.g. Shapley values) on selected features by maximising contributions of features in positive arguments and minimising contributions of features in negative arguments.

working on a classification problem, contributions in this figure are exaggerated. As a result of the probabilistic output through softmax, contributions of features are limited to a range of $[-1, 1]$.

To achieve this we want to incorporate explanations in form of contributions during training. A possibility to do so, is to incorporate the contributions within

the loss such that the network optimises in direction of better explanations. Table [1] provides an overview of all variables relevant to this chapter.

symbol	function	symbol	function
f	network	e	explanation method for y
W	weights of a network f	ϕ	contribution method to explain y
x	data point		with respect to x
x_i	i-th feature of data point x	sh	contributions using Shapley values
y^*	target class vector	IG	contributions using integrated gradient
y	prediction $f(x) = y$	l_{pred}	prediction loss between y and y^*
A^+	positive Arguments	l_{arg}	argumentation loss of contributions $\phi(x)$
A^-	negative Arguments	l_{reg}	regularisation loss of weights W
AE	argumented example (x, y^*, A^+, A^-)	L	extended loss containing $l_{pred}, l_{arg}, l_{reg}$

Table 1: Overview of our denotations

4.2 Incorporating contributions as loss

In this section we want to discuss how we plan to implement the focus on important features in neural networks akin to f_2 in figure 4. Important features are selected as arguments by a human annotator for every example individually. In order to learn from these arguments we intend to add an argument loss l_{arg} in addition to the default prediction error l_{pred} and a regularisation error l_{reg} similar to the RRR-loss defined by Ross et al. [17]. Here l_{reg} is a term regularising the weights W parameterising our network. Let L be the extended loss incorporating all of these:

$$L_f(x, y^*, W, A^+, A^-) = l_{pred}(y^*, f(x)) + \lambda_1 l_{arg}(x, y^*, f, A^+, A^+) + \lambda_2 l_{reg}(W) \quad (5)$$

Here λ are weighing factors used to adjust the scale of argument and regularisation loss terms. As were working on classification problems we're using the cross entropy as prediction error. For regularisation we plan to use L1 regularisation, to encourage smaller weights and avoid overfitting. Additionally, L1 regularisation leads to more sparse weights.

For every training sample (x, y^*) additional knowledge is given in the form of arguments, which specify the subset of important features. To ensure that our network incorporates this knowledge we can either focus on correctly using these features or penalise wrongly using others. Ross et al. [17] used the latter method in their RRR loss, by adding an additional loss factor that penalises attention on the wrong features. We want to adapt their method, but instead focus on rewarding correct usage of selected features. We will adjust the argument loss later during testing to compare various options.

For features in positive arguments $x_i \in A^+$ we want to reward positive contributions and penalise negative ones. This can be done by summing up contributions

of features in positive arguments and subtracting that from the loss. We multiply these with the class vector as arguments only give knowledge about the correlation of features and the correct class y^*. For a data point given by (x, y^*) with $x \in \mathbb{R}^n$ and one-hot vector $y^* \in \{0,1\}^m$ with contribution function ϕ this is given by:

$$-\sum_{j=1}^{m} y_j^* \sum_{x_i \in A^+} \phi_{i,j}(f, x)$$

Here negative contributions are added to the loss and positive contributions are subtracted. Similarly, for features in negative reasons $x_i \in A^-$ we want to reward negative contributions by subtracting them from the loss and penalise positive contributions by adding them to the loss:

$$\sum_{j=1}^{m} y_j^* \sum_{x_i \in A^-} \phi_{i,j}(f, x)$$

By combining these we finally receive our argument loss:

$$l_{arg}(x, y^*, f, A^+, A^+) = -\sum_{j=1}^{m} \left(y_j^* \sum_{x_i \in A^+} \phi_{i,j}(f, x) - \sum_{x_i \in A^-} \phi_{i,j}(f, x) \right)$$

We can use this to formulate our extended Loss L_f, where the prediction error is computed using the cross entropy and L1 is used as regularisation error. This equation uses weights W of our network, data $x \in \mathbb{R}^n$, class vector $y^* \in \{0,1\}^m$, contribution function ϕ, and arguments A^+/A^-.

$$\begin{aligned} L_f(x, y^*, W, A^+, A^-) =& l_{pred} + \lambda_1 l_{arg} + \lambda_2 l_{reg} \\ =& -\sum_{j=1}^{m} y_j^* log(f(x)_j) \\ & - \lambda_1 \sum_{j=1}^{m} \left(y_j^* \sum_{x_i \in A^+} \phi_{i,j}(f, x) - \sum_{x_i \in A^-} \phi_{i,j}(f, x) \right) \\ & + \lambda_2 \sum_{w \in W} |w| \end{aligned} \tag{6}$$

Where every line refers to the aforementioned losses respectively. For computation of contributions we're using Shapley values, gradients, or integrated gradients.

4.3 Computation of contributions

Computing the Shapley value using equation 4, should be straightforward:

$$sh_i(f,x) = \frac{1}{n!} \sum_{S \subseteq N \setminus x_i} |S|!(n - |S| - 1)!(f(S \cup x_i) - f(S))$$

However, considering that the Shapley value has to compute the result for every possible subset of features, this gets computationally exponentially expensive with increased feature count. As we'll see later, computing these for larger datasets becomes quite unfeasible. An alternative is given with Owen's multilinear extension of games [15] in equation 3, which adapted to a neural network f and data x is as follows:

$$sh_i(f,x) = \int_0^1 \mathbb{E}(f(S_i \cup x_i) - f(S_i))dq$$

Where $\mathbb{E}$ is the expectation parameter and q is the probability $\mathbb{P}$, such that $\forall i, j : i \neq j : \mathbb{P}(x_j \in S_i) = q$ with $\mathbb{P}(x_i \in S_i) = 0$. Computing the integral itself is unreasonable, instead we'll approximate it using the Riemann sum. The Riemann sum splits up the integral into m sections, approximates their area and sums up over all sections. In case of the above equation this is done by sampling $m + 1$ subsets $0, \ldots, k, \ldots m$, where the probability of features occurring in the $k\text{-}th$ subset is $q = \frac{k}{m}$. For the area each of those subsets would have to be multiplied with step range $1/(m)$. As we're using a constant step size it is also possible to compute the average of all approximations and multiply it with the total spanned distance, which in this case is 1. Finally, the accuracy of the expectation parameter given by averaging over multiple of such sampling $runs$. This results in the following function that we'll refer to as sampling Shapley [14]:

$$SS_i(f,x) = \frac{1}{(m+1)runs} \sum_{j=1}^{runs} \sum_{k=0}^{m} f(S_i \cup x_i) f(S_i) \quad where \quad \mathbb{P}(x_j \in S_i) = \frac{k}{m} \tag{7}$$

$$and \qquad \mathbb{P}(x_i \in S_i) = 0 \tag{8}$$

Computing the gradient function is straightforward and poses no problem. For the integrated gradient function we once again run into the problem of computing an integral:

$$IG_i(f,x) = (x - x') \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha$$

This is again approximated using the Riemann sum. As the gradient is a continuous function we'll use the trapezoidal rule given by $\int_a^b f(x)dx \approx (b-a)\frac{1}{2}(f(b) + f(a))$ to approximate the area of each step. Here we can again omit scaling with $(b - a)$,

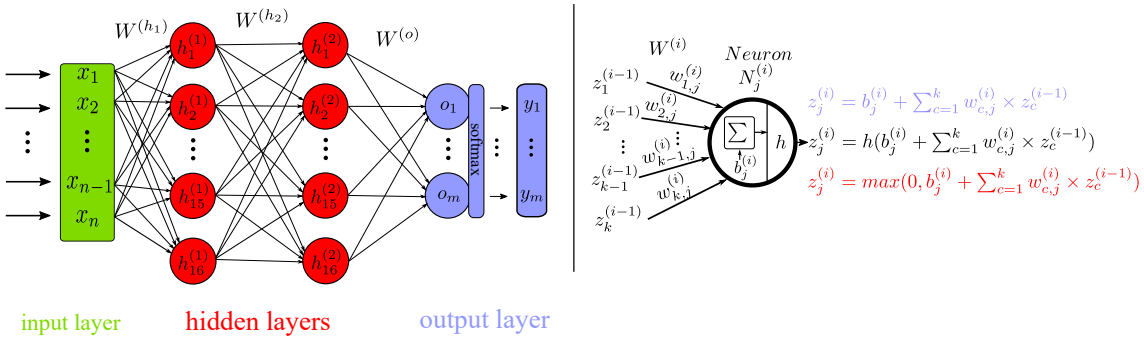

Figure 5: Representation of our network f (left), and a single neuron (right). Every input and neuron is connected to every neuron within the next layer. ReLu is used in hidden layers as activation function. Softmax is applied on predictions of the output layer. Weights $w_{c,j} \in W_i$, Bias b, Layer i, Intermediate layer results z_i, Activation function σ

as the total integral has again a width of 1. We numerically compute the integrated gradient function using the following equation:

$$IG_i(f,x) = \frac{(x-x')}{m} \sum_{k=1}^{m} \frac{1}{2} \left(\frac{\partial f(x' + \frac{k-1}{m})(x-x')}{\partial x_i} + \frac{\partial f(x' + \frac{k}{m})(x-x')}{\partial x_i} \right) \tag{9}$$

4.4 Implementation

Until now we only described the theory behind our problem. In this section we want to properly introduce our network, and how we implement the task in python.

4.4.1 Network

Implementation of our experiments is done in python 3.6, running tensorflow 2.6.0 and numpy 1.17.3.
All of our experiments are done on a simple feed forward neural network, composed of an input layer, two hidden layers and an output layer. A visual representation of our network can be seen in figure 5.

Input Layer The input layer passes the data immediately to the first hidden layer. More specifically for an input $x \in \mathbb{R}^n$ each unit of the input layer passes its assigned value x_i to each neuron of the first hidden layer.

Hidden Layers Our network contains two hidden layers with 16 neurons each. These neurons compute the activation of inputs multiplied with weights and an added bias akin to the right of figure 5. The first hidden layer receives its values from the input layer. For an input $x \in \mathbb{R}^n$ the weights are $W^{(h_1)} \in \mathbb{R}^{n \times 16}$ for the first hidden layer and $W^{(h_2)} \in \mathbb{R}^{16 \times 16}$ for the second hidden layer. Added biases are for both layers of shape $\mathbb{R}^{16}$. As activation function we use a rectified linear unit (ReLu), which only passes values greater than 0 to the next layer: $ReLu(z) = max(0, z)$. A neuron $N_j^{(h_i)}$ in the hidden layer h_i thus computes $z_j^{(h_i)} = max(0, b_j^{(h_i)} + \sum_{c=1}^{k} \left(w_{c,j}^{(h_i)} \cdot z_c^{(h_{i-1})} \right)$ where $w_c^{(h_i)} \in W^{h_i}$ and $k = n$ for first and $k = 16$ for second hidden layers respectively.
In theory one hidden layer should be sufficient to model most cases. We chose to go with two hidden layers to account for possible problems during training.

Output Layer The output layer for data point with m possible output classes has m neurons. The shape of its weights is $W^{(o)} \in \mathbb{R}^{16 \times m}$ with a bias $b^{(o)} \in \mathbb{R}^m$. Outputs $z^{(o)}$ of these neurons are computed without an activation function: $z_j^{(o)} = b_j^{(o)} + \sum_{c=1}^{16} w_{c,j}^{(o)} \times z_c^{(h_2)}$. A softmax is applied on final $z^{(o)}$ to create probabilistic outputs: $y_j = \frac{e^{z_j^{(o)}}}{\sum_{m}^{k=1} e^{z_k^{(o)}}}$

Additionally, all of our networks have L1 kernel regularisation applied, unless specifically stated otherwise. Training of the network is done using the Adam optimiser [9]. This network might not be optimal for the datasets we'll use, however it should be sufficient to receive comparable results. Tensorflow gives us the possibility to implement custom losses that are used for training. This enables us to implement our extended losses including the argument loss. We will describe these custom losses in detail in section 4.4.3.

4.4.2 Contribution Functions

We'll first introduce the contribution methods, that we've described above.

Gradient Our simplest contribution function is the gradient function used in [17]. As the name suggests we're only looking at the gradient of the result $f(x)$ with respect to the input x. The algorithm can be seen below:

Algorithm 1: Gradient

Input: x, f
Output: Gradients for predictions w.r.t input
1 $\nabla = \frac{\partial f(x)}{\partial x}$
2 **return** (∇)

Input x and network f are passed as arguments to the function. Line 1 computes our gradients using tensorflows gradient tape function. This operation provides an API for automatic differentiation. When "watching" x we can use this to request the gradients for $f(x)$. Line 3 returns the resulting gradients.

Integrated Gradient Related to the gradient is the integrated gradient function used in [20]. Here we're instead integrating over the above gradient. In addition to x and y^*, this function also incorporates a baseline x' of the dataset as a starting point for the integration. A zero baseline $x' = 0^n$ is assumed if its not specified. For computation of the integrated gradient we're using the in section 4.3 derived numerical equation 9, for a network f and a data point x :

$$IG_i(f,x) = \frac{(x-x')}{m} \sum_{k=1}^{m} \frac{1}{2} \left(\frac{\partial f(x' + \frac{k-1}{m})(x-x')}{\partial x_i} + \frac{\partial f(x' + \frac{k}{m})(x-x')}{\partial x_i} \right)$$

The algorithm implementing this equation can be seen below:

Algorithm 2: Integrated Gradient

Input: x,$x' = 0$, f, $m = 100$
Output: Integrated gradients for prediction w.r.t input

1 **for** $k = 0, \ldots, m$ **do**
2 $\quad interpolated\_x_k = x' + \frac{k}{m} * (x - x')$
3 $\quad \nabla_k = \frac{\partial f(interpolated\_x_k)}{\partial x_k}$
4 **for** $k = 0, \ldots, m-1$ **do**
5 $\quad integral_k = (\nabla_k + \nabla_{k+1})/2$
6 $IG = \frac{1}{m} \sum_{k=0}^{m-1} integral_k$
7 $IG = (x - x')IG$
8 **return** (IG)

Lines 1-2 compute $m+1$ linear interpolated data points starting from baseline x' to x. In line 3 gradients for predictions of these interpolated points are computed. Integrals are then approximated in lines 4-6 using the trapezoidal rule in line 5 and taking the mean in line 6 over our m steps. The integrated gradient is scaled in line 7 by $(x - x')$ to keep it in terms of the original input. Finally, the integrated gradients are returned in line 8.

Shapley The default Shapley computation iterates over all possible subsets. Equation 4 computes Shapley values $sh(f,x)$ on a network f for a data point $x \in \mathbb{R}^n$

with set of all features $N = \{x_0, \ldots, x_{n-1}\}$ as follows:

$$sh_i(f, x) = \frac{1}{n!} \sum_{S \subseteq N \setminus x_i} |S|!(n - |S| - 1)!(f(S \cup x_i) - f(S))$$

Features are omitted by replacing them using a baseline x' value, which we assume to be zero if not specified. Our implementation can be seen below:

Algorithm 3: Shapley

Input: x,$x' = 0$,f
Output: Shapley values of inputs on a network f

```
1  n = len(x)
2  PS = possiblePermutations(elements = [0, 1], length = n)
3  sh = 0
4  for S ∈ PS do
5      for i = 0, ..., n − 1 do
6          S2 = S
7          S2_i = 1
8          xo = S · x + (1 − S) · x'
9          xu = S2 · x + (1 − S2) · x'
10         d = f(xu) − f(xo)
11         |S| = sum(S)
12         sh_i = sh_i + |S|!(n − |S| − 1)! · d
13 sh = sh / n!
14 return (sh)
```

Here PS refers to all possible subsets in N. A subset is of the form $S \in \{0, 1\}^n$, where a 1 describes that the feature is within the subset and a 0 that the feature is omitted. For example for data with $x \in \mathbb{R}^2$ the set of possible subsets would be $[\{\}, \{x_0\}, \{x_1\}, \{x_0, x_1\}]$ with corresponding PS: $[[0, 0], [1, 0], [0, 1], [1, 1]]$. We iterate over these subsets in line 4 and our features in line 5.

Lines 6-7 generate a copy of subset S where feature x_i is included: $S2 = S \cup x_i$. We apply these subsets by multiplying them with x in lines 8 and 9. Here we additionally add a subset with opposite features $(1 - S)$ and multiply these with our baseline vector x'. This way we replace the values of omitted features to the baseline. Line 10 finally computes the difference between our subsets. The difference is scaled by the amount of permutations of the current $(= |S|)$ and remaining features $(= (n - |S| - 1)!)$ and then added to the Shapley values in line 12. Line 13 then divides by the number of total permutations $(= \frac{1}{n})$ for the average, and line 14 returns the Shapley values.

While our equation would only look at subsets excluding x_i implementing so would make it more difficult. If x_i is already included in a subset S, then it follows

that $S = S2$, $xo = xu$, $f(xu) = f(xo)$ and therefore $d = 0$. Adding 0 to our Shapley values makes no difference, therefore our implementation should be sufficient.

Sampling Shapley We have derived an equation for sampling Shapley based on Owen's multilinear extension in section 4.3 in equation 7. For m Riemann steps averaged over multiple *runs* sampled Shapley values for a data point x on a network f are given by:

$$SS_i(f,x) = \frac{1}{(m+1)runs} \sum_{j=1}^{runs} \sum_{k=0}^{m} f(S_i \cup x_i) f(S_i) \quad where \quad \mathbb{P}(x_j \in S_i) = \frac{k}{m}$$
$$and \quad \mathbb{P}(x_i \in S_i) = 0$$

Our implementation is an adaption of the repository[2] by Ohkrati et al. [14]. The pseudocode can be found below:

Algorithm 4: Sampling Shapley

Input: x,$x' = 0$, f, $m = 100$, $runs = 2$
Output: Sampled Shapley values of inputs on a network f

$n = len(x)$
$sh = 0$
for $i = 0, \ldots, runs - 1$ **do**
  for $k = 0, \ldots, m$ **do**
    $q = \frac{k}{m}$
    $sampledS_{i(m+1)+k} = Bernoulli(p = q, length = n)$
for $S \in sampledS$ **do**
  for $j = 0, \ldots, n - 1$ **do**
    $S1 = S$
    $S1_j = 0$
    $S2 = S$
    $S2_j = 1$
    $xo = S1 \cdot x + (1 - S1) \cdot x'$
    $xu = S2 \cdot x + (1 - S2) \cdot x'$
    $d = f(xu) - f(xo)$
    $sh_j = sh_j + d_j$
$sh = \frac{sh}{(m+1)runs}$
return (sh)

[2] https://github.com/aldolipani/OwenShap

Here $runs$ defines the accuracy of our expectation parameter $\mathbb{E}$ and m refines the Riemann sum. Thus, lines 3-6 define our sampled subsets $sampledS$ with a length of $runs * (m + 1)$ as Bernoulli distributions of length n with probabilities $q = \frac{k}{m}$. Lines 7-8 iterate over subsets and features. Within the loop we create a subset where feature x_j is omitted ($S1$) and one where it is included ($S2$). Similar to the previous Shapley code we than apply them on our data point x and replace omitted values with baseline values x' in lines 13-14. The difference for predictions on subsets of x is computed in line 15 and added to the corresponding Shapley value in line 16. Averages are then taken in line 17, and approximated values are returned in line 18. Generally, we're using $runs = 2$ and $m = 100$ in our tests.

Not all of our implementations are exactly as the above pseudocodes. Vector operations often enable us to omit some of the loops. We have additionally implemented the possibility to calculate contributions for multiple data points in our functions to enable the possibility of batch sizes. Array broadcasting was used to achieve this.

4.4.3 Extended Loss

Next we want to introduce our extended loss L that incorporates the argument loss. Tensorflow gives us the ability to create our own custom loss that is executed during training. We have specified our extended loss L for an AE (x, y^*, A^+, A^-) with $x \in \mathbb{R}^n$, one-hot vector $y^* \in \{0, 1\}^m$ for a network f with weights W using a contribution function ϕ in section 4.2 as follows:

$$\begin{aligned} L_f(AE, W) = &- \sum_{j=1}^{m} y_j^* log(f(x)_j) \\ &- \lambda_1 \sum_{j=1}^{m} \left(y_j^* \sum_{x_i \in A^+} \phi_{i,j}(f, x) - \sum_{x_i \in A^-} \phi_{i,j}(f, x) \right) \\ &+ \lambda_2 \sum_{w \in W} |w| \end{aligned}$$

Where the first line defines the prediction loss in form of cross entropy, the second line the argument loss and the third line the L1 regularisation error. Explanations are incorporated in form of contribution function ϕ, such as the Shapley computation. As a feature can not be in both positive and negative arguments, we simplify A^+ and A^- into one vector $A^* \in \{-1, 0, 1\}^n$ where A_i^* describes if x_i is in an argument:

$$\begin{aligned} A_i^* = 1 &\iff x_i \in A^+ \\ A_i^* = 0 &\iff x_i \notin (A^+ \cup A^-) \\ A_i^* = 1 &\iff x_i \in A^- \end{aligned}$$

An AE (x, y^*, A^+, A^-) can thus be described by (x, y^*, A^*). Reconsider the argument loss given for $x \in \mathbb{R}^n$ and $y^* \in \{0,1\}^m$:

$$l_{arg}(x, y^*, f, A^*) = -\sum_{j=1}^{m} \left(y_j^* \sum_{x_i \in A^+} \phi_{i,j}(f, x) - \sum_{x_i \in A^-} \phi_{i,j}(f, x) \right)$$

This function can now be simplified using the following function:

$$l_{arg}(x, y^*, f, A^*) = -\sum_{j=1}^{m} \left(y_j^* \sum_{i=1}^{n} (\phi_{i,j}(f, x) \cdot A_i^*) \right)$$

Within the brackets contributions of features in positive arguments are multiplied with 1 and added. Contributions of features in negative arguments are multiplied with -1 and thus subtracted. The contributions of all other features are multiplied with 0 and thus omitted.

A custom loss in tensorflow receives a vector y_{pred} that is computed as $f(x) = y_{pred}$ during training and a y_{true} vector that is the one-hot class vector. In order to compute the contributions and to determine which features our loss should focus on we have to include information about the data point x, and its arguments A^*. For this we padden the true classification vector y^* with additional information. More specifically, for an AE with $x = (x_1, \ldots, x_n)$, corresponding arguments $A^* = (A_1^*, \ldots, A_n^*)$ and $y^* = (y_1^*, \ldots, y_m^*)$ we generate a padded class vector $\tilde{y}$ that incorporates these informations:

$$\tilde{y} = [y_1^*, \ldots, y_m^*, x_1, \ldots, x_n, A_1^*, \ldots, A_n^*]$$

This vector is then split up within the custom loss function to retrieve the original information. The feature vector x is required to compute our contribution functions, be it the Shapley function or the approximations thereof.

With these informations we can now create our own custom loss as follows:

Algorithm 5: Extended-Loss

Global: f,x', λ_1,λ_2, n, m
Input: y_{pred}, $\tilde{y}$
Output: Loss = cross entropy loss + argument loss + regularisation loss

1 $y* = \tilde{y}_{0:m}$
2 $x = \tilde{y}_{m:m+n}$
3 $A^* = \tilde{y}_{m+n:m+2n}$
4 $loss = CrossEntropy(y_{pred}, y^*)$
5 $contr = \phi(x, x', f) \cdot y^*$
6 $argLoss = sum(contr \cdot A^*)$
7 $loss = loss - \lambda_1 \cdot argLoss$
8 **for** $W \in f.weights$ **do**
9 $\quad loss = loss + \lambda_2 \cdot sum(abs(W))$
10 **return** $loss$

Lines 1-3 separate our previously defined $\tilde{y}$ into x, y^* and the arguments. In line 4 we computes the cross entropy classification error using a function supplied by tensorflow. The argument loss is computed in lines 5-6 and subtracted from the loss in line 7. We can multiply the contributions with the vectors representing arguments and class vector to zero out unwanted contributions. Here ϕ is any of the above defined contribution functions. In the pseudocode we supply it with data x, baseline x', class vector y^* and the network f. Depending on the contribution function slight alternations have to be done. For one our gradient function does not support baselines. Integrated gradient and sampling Shapley can accept additional values specifying number of steps and runs. Lines 8-9 apply L1 regression by iterating over the weights and adding their absolute values to the loss.

RRC-Reasoning is a loss from [20] that we also want to discuss here. It is similar to our custom loss in that it focuses on contributions. However, instead of using Arguments, they instead use an annotation matrix for features that should not have a positive contribution, and then penalise such positive contributions. We adapt this to an AE with (x, y^*, A^+, A^-) on a network f with weights W as penalising all features not in positive arguments $x_i \notin A^+$:

$$L_f^{RRC}(AE, W) = l_{pred} + \lambda_1 \sum_{j=1}^{m} \left(y_j^* \sum_{x_i \notin A^+} max(0, IG_{i,j}(f, x)) \right) + \lambda_2 l_{reg}$$

The algorithm for this is as follows:

Algorithm 6: RRC-Reasoning-Loss

Global: f,x', λ_1,λ_2, n, m
Input: y_{pred}, $\tilde{y}$
Output: Loss = cross entropy loss + argument loss + regularisation loss

1 $y* = \tilde{y}_{0:m}$
2 $x = \tilde{y}_{m:m+n}$
3 $A^* = \tilde{y}_{m+n:m+2n}$
4 $A^* = 1 - max(0, A^*)$
5 $loss = CrossEntropy(y_{pred}, y^*)$
6 $contr = max(IG(x, x', y*, f), 0) \cdot y^* \cdot A^*$
7 $loss = loss + \lambda_1 \cdot sum(contr)$
8 **for** $W \in f.weights$ **do**
9 $\quad loss = loss + \lambda_2 \cdot sum(abs(W))$ **return** $loss$

This algorithm is mostly the same to our algorithm 5. In line 4 negative arguments are omitted through the max function. Zeros and ones are then swapped by subtracting A^* from 1, such that we have a 1 for all features $x_i \notin A^+$. This should be similar to the annotation matrix used by Ross et al. [17], which was adapted by [20]. By multiplying this with contributions we only keep the contributions of unwanted features. Another difference can be found in line 6 where we specifically use the integrated gradient method, and only keep positive contributions through the max function. A sum of these contributions is then added to the loss thereby penalising them.
Other adaptions can be created similarly on the fly.

5 Experiments

In order to analyse our method we run multiple tests. In conclusion the pipeline of our network is as follows:

1. Annotator selects features for A^+/A^-
2. Optimise network by maximising (A^+)/minimising (A^-) contributions of selected features
3. Evaluate new network and explanations generated on it

In this section we want to introduce the datasets we're experimenting on, the metrics were using and the tests we'll be running.

5.1 Datasets

For the evaluation of the framework, we want to use varying datasets where annotators can create arguments for examples with respect to their classes. The following two datasets were used in [13] which enables us to use them as a reference point:

1. Japanese Credit Screening[3] dataset includes 125 instances with 7 attributes. This dataset was mentioned multiple times in above examples.
2. Zoo dataset[4] features 101 animals with 17 attributes that are classified into 7 types (mammals, reptile, etc.). Various features are correlated to different classes in only specific examples (e.g. laying eggs), which makes this one challenging.

The following datasets were not used in [13] but include features that can be used for arguments:

3. A Synthetic Integer dataset with 3 features x_1, x_2, and x_3, where $f(x) = x_1 > x_2$. Values are randomly sampled from a range of $[-100, 100]$, for an average of 0 for every feature. Any amount of samples with varying data ranges can be created for testing.
4. The South German Credit[5] dataset includes 1000 datapoints with 20 features, predicting if a credit is paid back. This dataset additionally includes credit scores[6] for the feature value-ranges.

A comprehensive overview of our datasets and their characteristics can be seen in table 2.

[3] `https://archive.ics.uci.edu/ml/datasets/Japanese+Credit+Screening`
[4] `https://archive.ics.uci.edu/ml/datasets/zoo`
[5] `https://archive.ics.uci.edu/ml/datasets/South+German+Credit`
[6] `https://data.ub.uni-muenchen.de/23/1/DETAILS.html`

dataset	# features	# data points	class distribution
Integer	3	800	[400,400]
Japanese Credit	7	125	[85,40]
German Credit	20	1000	[700,300]
Animal	16	101	[41,20,5,13,4,8,10]

Table 2: A comprehensive overview of our datasets

5.1.1 Annotation

All of the above datasets have not been annotated with positive or negative Arguments. The annotations were done as follows:
The paper in [13] supplies rules for the classification of the Zoo and Japanese Credit Screening datasets. As an animal can't be of a type "despite" having a feature, this dataset only includes positive arguments. Features in rules that determine the type of the animal are used as positive arguments. The Japanese Credit Screening dataset only included rules for when a credit should be denied. We used these to create positive and negative arguments for denied and approved credits respectively. We additionally included some positive arguments for granted credits, such as a stable job or a larger wealth than the requested credit. For example:

Animal dataset contains the rule:
IF $has\_fins = true \wedge breathes = false$ THEN $fish = true$
Thus for a fish where these two conditions are true, both has_fins and $breathes$ are selected as positive arguments.

Japanese credit data set contains the rule:
IF $jobless = true \wedge male = true$ THEN $credit = false$ Thus for a data points where the credit was denied, and the conditions hold true, $jobless$ and $male$ are selected as positive arguments. On the other hand if the credit was still granted, these features were marked as negative arguments.

For the synthetic Integer dataset features x_1 and x_2 were selected as positive and negative arguments depending on the result. If $x_1 > x_2$ then x_1 is a positive argument if it's greater than 0, otherwise it is a negative argument. This is mirrored for x_2 and the case that $x_1 \leq x_2$. Features x_3 are never selected as a argument. For example:

$$
\begin{aligned}
x1 > x2 : x &= [5, -10, 5] \Rightarrow A = [1, 1, 0] \\
x &= [55, 40, 80] \Rightarrow A = [1, -1, 0] \\
x &= [-10, -43, -24] \Rightarrow A = [-1, 1, 0]
\end{aligned}
$$

As mentioned above the South German Credit dataset features value ranges for each feature, where a score is assigned depending on how positive it is. These Scores can then be directly used to create positive and negative annotations. For example the amount of credit is separated into ten intervals with 10 points for less than 500 DM, to 1 point for values larger than 20,000DM, where more points refers to a better credit scoring. For every feature the top 30% of scores are used as positive argument if the credit was paid back, and as a negative argument if not. Simultaneously, the lower 30% of scores are used as positive argument if the credit was not paid back, and as a negative argument if it was.

5.2 Metrics

In this section we want to describe the metrics we are going to use in the next chapter. Obviously we want to estimate the performance of our network. We will do so by computing the accuracy of predictions.

Additionally, we want to evaluate how similar explanations generated on our network are in comparison to the arguments. Unfortunately there is a lack of proper metrics to evaluate the quality of explanations. Often times experts are presented an explanation in order to evaluate its quality. This is no option for us, instead we suggest measuring the quality of our explanations by comparing them to their respective arguments. We do so by computing the similarity of an argument vector for a data point given by $A^* \in \{-1, 0, 1\}^n$ to the computed explanation vector for the correct class using the cosine similarity. For two vectors a and b cosine similarity is given by: $\textit{cos-sim}(a, b) = \frac{a \cdot b}{||a|| \cdot ||b||}$. Where $||a||$ is the euclidean norm of a vector a. Assume that the correct class is y_j. Then we adapt function to our explanations by computing the similarity $\textit{cos-sim}(A^*, \phi_{:,j}) = \frac{A^* \cdot \phi_{:,j}}{||A^*|| \cdot ||\phi_{:,j}||}$. Using this we may be able to estimate how much our methods pulls the explanations towards our arguments. Cosine similarity can also be computed between contribution functions such as Shapley and integrated gradient to estimate their similarity.

Furthermore, we can attempt to categorise explanations into positive, neutral and negative, depending on threshold values. These can then be compared to ar-

guments, in order to estimate accuracies of our explanations. For a data point (x, y^*) of class y_j^*, we classify a feature x_i as positive explanation if it has a contribution of greater than $\frac{\phi_{i,j}(f,x)}{max(abs(\phi_{:,j}(f,x)))} \geq th$ and as a negative explanation if below $\frac{\phi_{i,j}(f,x)}{max(abs(\phi_{:,j}(f,x)))} \leq -th$. Features x_i with contributions $-th < \frac{\phi_{i,j}(f,x)}{max(abs(\phi_{:,j}(f,x)))} < th$ are assumed to be neutral. Here we divide by maximum absolute contributions towards y_j in order normalise contributions to a range of $[-1, 1]$. By applying a threshold we can create categorised explanations similar to arguments. Using this method it should be possible to evaluate different categories separately. To compare them we want to compute the F1 score given by $F_1 = \frac{TP}{TP+(FP+FN)}$, where TP are correctly identified features "true positives", FP are falsely identified features "false positives", and FN are missing features of categories "false negatives". By altering our methods we expect to impose improved recognition between negative and neutral explanations. However, the idea of categorisation is quite ambiguous. There is no definitive method to do so, and our implementation has varying results for different threshold values.

5.3 Tests

We want to explore our developed method using multiple tests. In order to ensure reproducible results within tests we sample random states before starting the training, and use the same states for every method. Additionally, results are averaged over multiple training attempts. As weighing factors we have used $\lambda_1 = 3$ for argument loss, in order to reinforce their importance, and $\lambda = 0.02$ for regularisation loss. In the following we'll summarise the tests we intend to do.

Explanation Functions and Time Complexity As using the Shapley computation itself for our tests is computationally unfeasible we first want to compare the time complexities of our functions ϕ. Additionally, we want to take a look at cosine similarities of the computed contributions. Differences hear do not necessarily translate to worse contribution functions, it might be possible that different attributions have similar success in our model.

Methods Next we'll want to compare the results of the networks themselves when using our methods. Here we want test the losses as implemented in section 4.4.3, using each of our contribution functions. For comparison purposes we'll include a network trained without the argument loss and one trained using the RRC-Reasoning loss [20]. Here we want to look at the accuracies of their predictions, and the cosine

similarities between computed explanations and arguments.

Impact of Explanation function Generally we use sampling Shapley to generate final explanations of our methods. In the previous test we want to interchange the explanations within the losses, by swapping the contribution functions. Optimising for Shapley values and integrated gradient, but only evaluating results using Shapley values could lead to an unwanted bias in the results towards Shapley values. Here we want to compare how well the integrated gradient explanation method performs in comparison.

Baseline Per default our networks omit features in the contribution calculation by replacing them with zero. Our assumption is, that a better baseline value for the datasets could help in improving the results. Therefore, this is something we will want to look at.

Ablation Study Going further in depth of our loss functions we want to do a ablation study in order to analyse the different parts of our loss functions. We do so by removing cross entropy loss, argument loss, or L1 loss respectively. This is important to find out which role every part plays in the total result.

Argument-Loss alterations Additionally, the RRC-Reasoning module used in [20] works by penalising false positive contributions. Our method instead interacts with features in the arguments, by subtracting contributionss of features in positive arguments and adding those in negative arguments. Here we're specifically interested in different alterations of our argument loss, and their role in correctly imposing constraints on explanations.

Arguments Moreover we intend to inspect the role of our arguments. Here we want to see if the performance of our network gets worse by intentionally using worse arguments.

6 Results

For our network we want to evaluate both the performance of the predictions as well as the quality of their explanations. We want to additionally compare the performance of different Shapley approximations.

6.1 Explanation Functions and Time Complexity

As mentioned in subsection 4.2 the Shapley computation gets exponentially expensive. Therefore, we suggested multiple alternatives. In this section we want to compare the default Shapley computation, the gradient approach, the integrated gradient method and finally the sampling Shapley variant described in section 4.4.2.

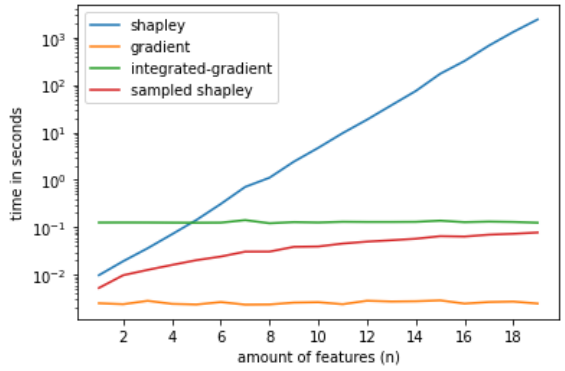

Figure 6: Log-Scale comparison of computation time for contributions of a single datapoint $x \in \mathbb{R}^n$

The plot in figure 6 has a Log-scale, therefore the linear Shapley value line represents an exponential growth. At the same time gradient and integrated-gradient method durations remain roughly the same, while the sampling Shapley approximation represents a linear growth. An extension of this plot with linear scaling omitting the Shapley values can be seen in figure 7. As we can see the required time to compute the sampled Shapley variant overtakes the integrated gradient method around 30 features, and keeps on growing. This might get problematic when applying these methods on datasets with a lot of features, such as images.

We additionally compare similarity of the contribution methods in table 3. Here our four contribution functions were used to explain predictions of a network trained on the South German credit dataset. Cosine similarities, with a range of $[-1, 1]$, were then computed in comparison to the Shapley values. Values were averaged

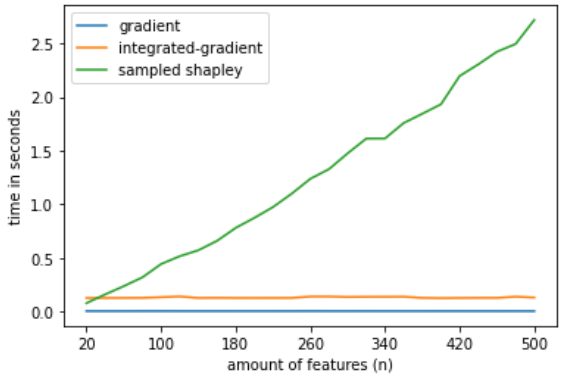

Figure 7: Comparison of computation time for contributions of a single datapoint $x \in \mathbb{R}^n$

Method	Gradient	Integrated-Gradient	Sampled-Shapley
cos-sim	-0.021	0.557	0.990

Table 3: Cosine similarity of the approximation methods in comparison to the default Shapley computation, averaged over 10 data points

over 10 data points and rounded to 3 decimals.
As we can see the gradient is almost orthogonal (cosine similarity of 0) in comparison to Shapley values. The integrated gradient method is better with a similarity above 0.5. As expected the sampled Shapley method is the best with a cosine similarity of almost 1. An explanation for the difference of gradient and integrated gradient can be seen in figure 8. Assume that the plot is the gradient of predictions with respect to input features. Here the gradient for $x = 3$ is almost 0, meanwhile the integrated gradient (marked in red) describes the history ranging from a baseline 0 to value 3.

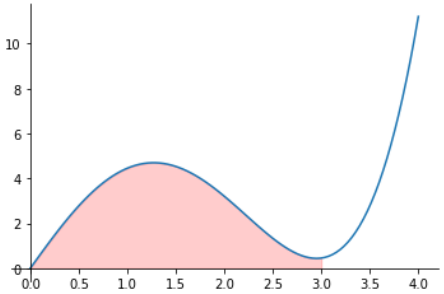

Figure 8: Visual comparison of gradient and integrated gradient.

This is not to say, that these explanations are necessarily worse than Shapley

values, just that they have different properties. Rather it backs us in swapping out Shapley values with sampled Shapley values for more complicated data sets. In that regard it makes sense that sampling Shapley is the closest to shapley values as it is the only contribution method actually attempting to approximate the Shapley values. A big difference between gradient and Shapley methods is that the latter simulates changes in the other features as well. Therefore, Shapley methods can recognise correlations between features, which is not really the case for gradient methods.

6.2 Model Comparison

In the previous section we compared the approximations themselves. Now we want to compare their results when using them in Equation 6 as an extended loss model incorporating the argument loss. For comparison we have included a Cross-Entropy loss and the RRC-Reasoning loss from [20] as baselines. Due to computation times, the Shapley loss is included only in the Integer dataset tests and otherwise represented by the sampled Shapley values. An overview of the losses included can be seen in table below:

Loss	Contribution Function	Loss contains
Cross Entropy Loss	-	CE, l_{reg}
Shapley Loss	Shapley	CE, l_{arg}, l_{reg}
Gradient Loss	Gradient	CE, l_{arg}, l_{reg}
Integrated Gradient Loss	Integrated Gradient	CE, l_{arg}, l_{reg}
Sampling Shapley Loss	Sampling Shapley	CE, l_{arg}, l_{reg}
RRC-Reasoning Loss	Integrated Gradient	CE, l_{arg}, l_{reg}

Table 4: Overview of losses contained within tests of section 6.2

Here loss contains refers to the parts of our extended loss that are included, where CE refers to the classification error given by the cross entropy, l_{arg} to our previously defined argument loss, and l_{reg} to a regularisation term penalising the weights of our network.

6.2.1 Prediction Accuracies

A comparison of the prediction accuracies of the selected methods can be seen in Figure 9. Accuracies on the training and testing sets are remarkably similar. There's a slight trend where accuracies on training sets are slightly better than those on the testing set. This is to be expected considering that the network was fit to the training set.

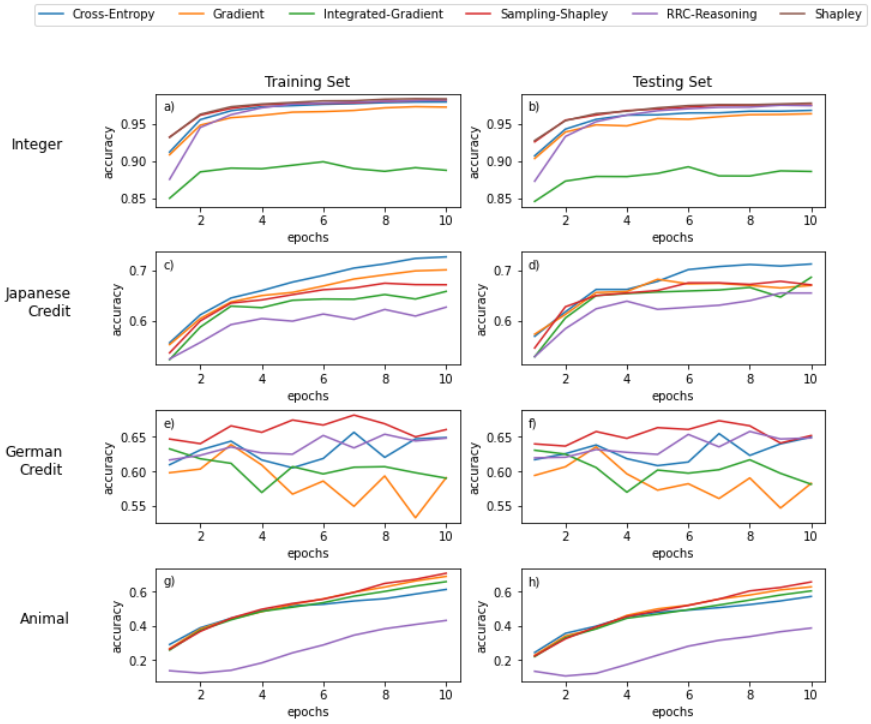

Figure 9: Accuracies on our datasets for different contribution methods in losses for training and testing sets respectively, for 1 to 10 training epochs. Averaged over 40 simulations.

Usually we can identify a growth for better accuracies with more training epochs. An exception to this can be found in the German Credit dataset, where growth is negligible, if not getting worse. This dataset seems to be the most difficult for predicting the result, which is understandable considering that it is difficult to predict if people are going to pay their credit back by inspecting their status. Most methods hover slightly above 50% accuracies meaning that they do hardly better than simply guessing.

The largest total growth in prediction accuracy can be seen for the Animal dataset, where we start at around 30% and improve to around 70% on training set and 60% on testing set. Here the possibility of multiple classes make it more difficult for the classifier at the beginning, while the features non-ambiguity leads to decent results towards the end. Meanwhile simplicity of the Integer dataset produces good results after already the first epoch. Here Shapley and sampled Shapley have additionally pretty much exactly the same results.

On average all of the models have approximately the same results in this category. The only real outliers are the integrated gradient method on the Integer data set and the RRC-Reasoning module loss falling off in the Animal dataset. A possible

explanation for the behaviour on the Integer dataset could be that the class values on this dataset are directly coupled between two of the datasets features. Instead of looking at other features the integrated gradient method only alters the feature it tries to evaluate. This should be similar for the gradient variant, however it could be possible that local gradients pose not much of a reasoning, which is then evaluated as a cross entropy function with added noise. In fact gradient method in many ways appears to have a similar curve as the cross entropy method, just a bit worse. Meanwhile an explanation for the behaviour on the Animal data set could be that penalising contributions for all features $x_i \notin A^+$ constrains the network to much. Here animals often only have a single feature selected as reason. Thus the RRC-Reasoning loss would penalise positive contributions for all but one feature in each example.

In total our Shapley variant performs alright. It is not necessarily better than other methods for every dataset, but it is usually a top contender.

6.2.2 Cosinus Similarities

Here we compute the cosine similarity between arguments $A^* = \{-1, 0, 1\}^n$ and the explanation vector $\phi_{:,j}$ towards the correct class y_j^* by computing $\textit{cos-sim}(A^*, \phi_{:,j}) = \frac{A^* \cdot \phi_{:,j}}{||A^*|| \cdot ||\phi_{:,j}||}$. Results for explanations computed using sampling Shapley can be seen in figure 10. Using cosine-similarity we should be able to describe how much our argument loss draws explanations of predictions towards arguments. The network should learn explanations in form of positive contributions for features in positive arguments, negative contributions for features in negative arguments, and negligible contributions otherwise.

Once again a minor difference between training set and testing set can be seen, which is to be expected. Additionally, we can see an improvement in similarities with more epochs. In general the sampled Shapley Argument loss does substantially better than other methods. The Integer dataset is a special case, where the simplicity leads to decent results after only a few epochs, from where on the similarity stagnates and is overtaken by most other losses. Overall the cross entropy loss is the worst among our selection of losses. With the exception of the Integer data set, it ranks as the worst for every dataset.

The integrated gradient method once again struggles on the Integer dataset. Our assumption from the previous test is, that integrated gradient struggles with recognising the synergies between features x_1 and x_2. On the other hand, the sampled Shapley Argument loss outperformed the other methods by a wide margin in this section. The aforementioned advantage of simulating different feature subsets and thus including their interactions might be a reason for this. For the simpler dataset the classical cross entropy approach seems to outperform most of our methods without even focusing on arguments. It is possible, that we constrain the learner more

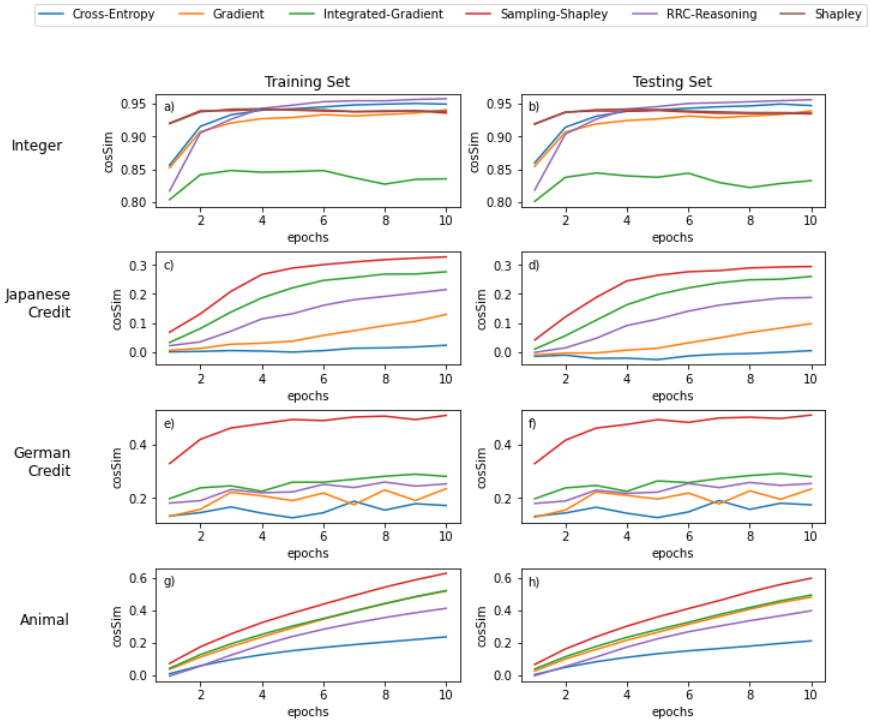

Figure 10: Cosinus Similarities between arguments and explanations computed using sampled Shapley on our datasets for different contribution methods in losses for training and testing sets respectively, for 1 to 10 training epochs. Averaged over 40 simulations.

than helping it in simpler cases.

6.3 Explanation function

The previous section evaluated explanations of our methods, that were computed using sampling Shapley. As some of our argument loss methods optimise towards Shapley values, this could have an unwanted bias effect for the results. For the two losses incorporating sampling Shapley contributions and integrated gradient contributions we want to compare both options of evaluating with sampling Shapley and integrated gradient explanations. We omit incorporating normal gradients as explanations as these only explain local differences of features. Explanations are once again evaluated by computing their similarity to arguments. Table 5 summarises our methods used in this section.

Figure 11 shows our results of these tests. In total sampled Shapley explanation have better performance than their integrated gradient counterpart in every test we have made. Even when working with a loss using integrated gradient contributions

Contribution Function	Explanation function
Integrated Gradient	sampling Shapley
sampling Shapley	sampling Shapley
Integrated Gradient	Integrated Gradient
sampling Shapley	Integrated Gradient

Table 5: Overview of losses contained within tests of section 6.3

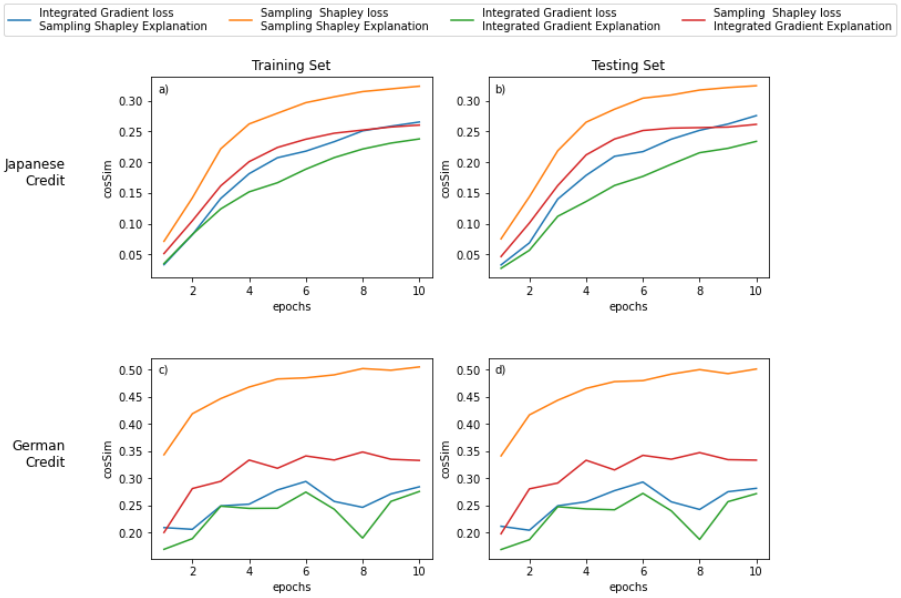

Figure 11: Cosinus Similarities between arguments and explanations, computed using IG and SS, on our datasets for training and testing sets respectively, for 1 to 10 training epochs. Averaged over 40 simulations.

we see an improvement in similarities to arguments. However, the difference between these explanations is larger for sampled Shapley contributions than for integrated gradient ones. So there might still be an unfair advantage for sampled Shapley contributions when also using it for explanations. Considering that explanations on integrated gradient are still improved when using sampling Shapley, specifically adapting explanations to their contribution functions does not make sense.

6.4 The role of Dataset-Baselines

In previous experiments, features for Shapley computation were omitted by replacing them with zeros. In this experiment we want to compute an average baseline $x' = (x'_1, \ldots, x'_n)$ for the datasets, which will replace the feature values instead. Previous tests computed for a subset $S = \{x_1, x_3\}$ of a data point $x = (x_1, x_2, x_3)$

the prediction $f(x_1, 0, x_3)$. The baseline variant instead uses the prediction for $f(x_1, x_2', x_3)$. Meanwhile for integrated gradient we'll change from integrating over 0 to x, to integrating from x' to x instead. Here we'll omit the Integer dataset, as it was created with a baseline of 0 in mind, and the Animal dataset, where due to the multiple classes creating a proper baseline poses a challenge. For the Japanese and German credit datasets we'll generate a baseline by computing the weighted average over the respective classes. Final explanations are computed using sampling Shapley, also incorporating the respective baselines of 0 and x'.
Figure 12 provides an overview for accuracies of predictions and cosine similarity of explanations towards arguments on these datasets. In our tests the baseline variants did not improve either of the accuracies. In fact they mostly got worse. Our cosine similarities mirror these results. Albeit not quite as fluctuating they still provide worse results. An explanation for this behaviour could be that many features in

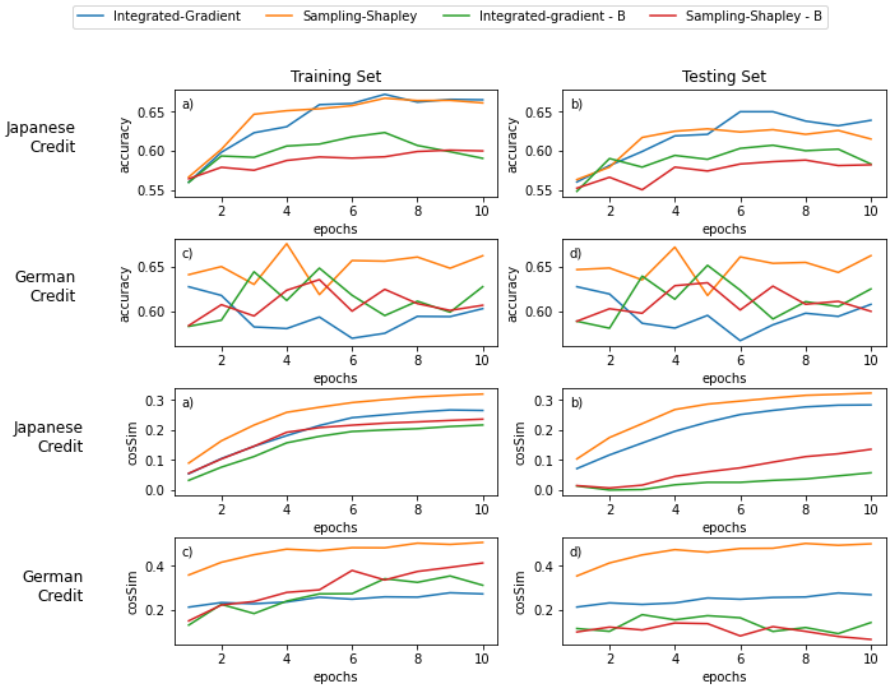

Figure 12: Accuracies and cosine similarities for predictions on Japanese and German Credit datasets for training and testing sets respectively, for 1 to 10 training epochs. Averaged over 40 random simulations. Attached B refers to included Baseline.

these datasets use bucket value ranges, wherein the features do not have proper continuous values. In such cases using zero baselines is usually the better idea. Another possibility is that our option of using a weighted average as baseline is a bad

idea, where a neutral state of our network differs from our computed baselines.

6.5 Ablation Study

In this section we want to analyse the importance of the different parts of our extended Loss. Our loss function consists of the prediction loss, which is computed using the cross entropy, the argument loss, and a regularisation parameter. In this section we want to compare how the loss holds up when removing specific parts of the loss function. In particular we want to compare the variants specified in table 6.

Loss	Contribution Function	Loss contains
Removed Arguments	-	CE, l_{reg}
Default (SS)	Sampling Shapley	CE, l_{arg}, l_{reg}
Removed Regularisation	Sampling Shapley	CE, l_{arg}
Removed Cross Entropy	Sampling Shapley	l_{arg}, l_{reg}
Removed Cross Entropy and Regularisation	Sampling Shapley	l_{arg}
Removed Arguments and Regularisation	-	CE

Table 6: Overview of losses contained within tests of section 6.5

Once again a combined overview of the accuracies can be seen in Figure 13. We have seen both the Default loss, previously named sampling Shapley, and the Removed Arguments loss, previously named cross entropy, before. These are included for comparison purposes, and we're more interested in the remaining four losses.

It is expected that the network performance drastically drops when removing the cross entropy, as there would be no longer a loss that teaches the network what the correct answer would be. We can see this behaviour in most datasets, besides the German credit dataset, where when keeping the regularisation term we have comparable prediction rates as the other loss variants. More remarkable is that even without the prediction error teaching the correct class, these variants still perform substantially better than guessing, which should have an accuracy for $1/m$ for sets with m classes. An explanation for this observation could be that by telling the network the class that the contributions should be maximised for, we indirectly also teach it the correct class. An exception to this is the Japanese dataset where the results for these two lines are just above 50% classification rate, which is about as good as guessing. Considering that in this dataset the losses without argument integration perform better than the others, it could be possible, that selected arguments to not reflect its classes properly.
Correlation between regularisation and predictions is not really clear. In most cases additionally removing the regularisation does not have much of an impact, or leads

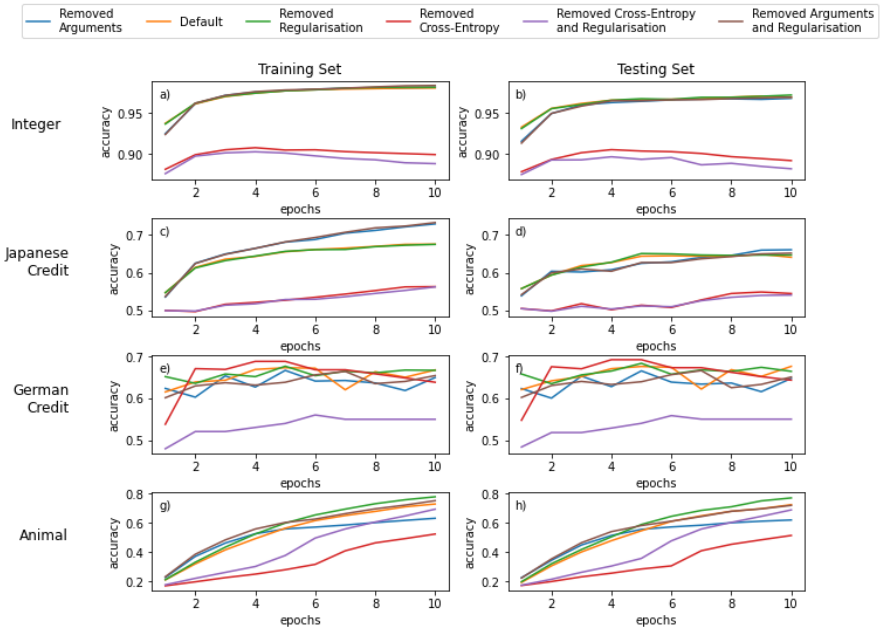

Figure 13: Accuracies of predictions for different loss variants on our datasets with training and testing sets respectively, for 1 to 10 training epochs. Averaged over 40 simulations.

to slightly worse results. An exception to this is the animal dataset where regularised variants perform worse than their counterparts. This is probably because of the non ambiguity of the correlation between features and predictions for this dataset. Every animal can be definitely categorised using their features, and thus using regularisation might slow down learning.

In Figure 14 we can see the similarity of explanations generated on our network in comparison to arguments. As expected, methods with the argument loss, usually outperform methods omitting it. In fact the only dataset where this is not the case is the Integer dataset. Here cross entropy suffices to learn correct correlations, similar to what we've observed in previous tests. Removing regularisation does not appear to have much of an impact on similarities, minor differences can be seen depending on dataset.

Overall the argument loss did surprisingly well on its own. While it was expected that it was good at generating the correct explanations, it exceeded expectations for prediction accuracies. We assume that by teaching the network which contributions it should maximise for the correct output, we indirectly also maximise the likelihood of said output. Performance of regularisation did depend on the dataset, we assume that it helps to prevent overfitting on datasets with ambiguous annotations. On the plus side, including the regularisation additionally leads to smaller weights and

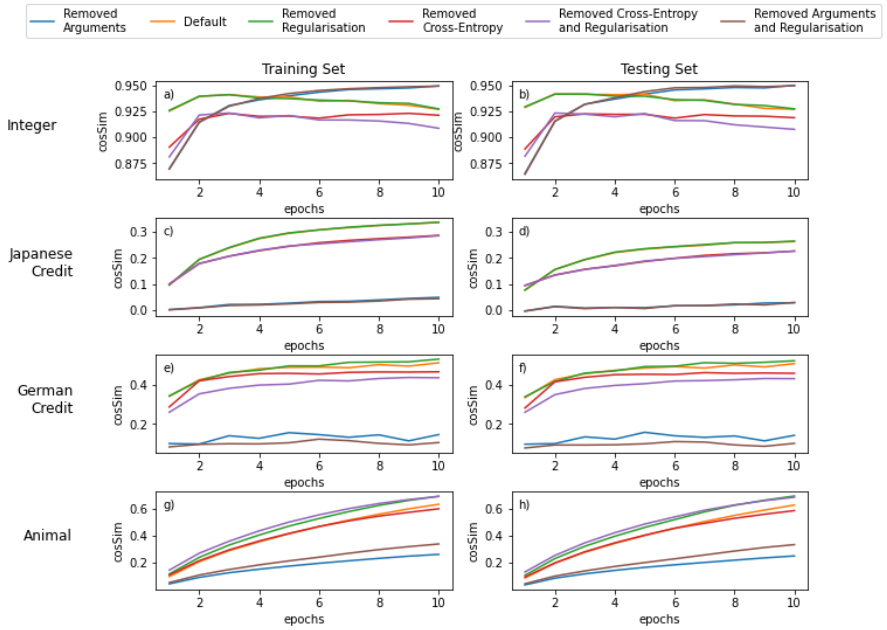

Figure 14: Cosine similarities between arguments and explanations for different loss variants on our datasets with training and testing sets respectively, for 1 to 10 training epochs. Averaged over 40 simulations.

more sparse networks.

6.6 Comparison of Argument loss variants

Our current system rewards correct contributions and penalises wrong contributions of features in arguments. The RRR method that the RRC-Reasoning incorporates instead penalises all positive contributions of features that are not supposed to have a positive contribution. In this section we want to compare a multitude of things. Specifically we're looking at adapting our method to penalising wrong contributions, such that:

$$l_{arg}^{pen} = \sum_{j=1}^{m} \left(y_j^* \sum_{x_i \notin A^+} max(0, \phi_{i,j})(f, x) - \sum_{x_i \notin A^-} min(0, \phi_{i,j}(f, x)) \right)$$

Here we want to prevent positive contributions of features not in positive arguments, and negative contributions of features not in negative arguments. In this section we want to additionally compare the impact of negative arguments. For this we'll also look at a comparable loss only looking at contributions of features in positive

arguments:

$$l_{arg}^{pos} = -\sum_{j=1}^{m} \left(y_j^* \sum_{x_i \in A^+} \phi_{i,j})(f, x) \right)$$

Specifically we'll also be looking at how well these implementations do in correctly identifying positive, negative and neutral explanations, by incorporating the categorisation method explained in section 5.2. Here we'll restrict ourselves to using tables as the amount of graphs required would not help with visibility. Additionally, we'll only cover results on respective testing sets. Training was done over four epochs and explanations are generated using sampling Shapley. An overview of the used methods can be seen in table 7.

Loss	Contribution Function	Losses contained
Cross Entropy Loss (CE)	-	CE, l_{reg}
Sampling Shapley Loss (SS)	Sampling Shapley	CE, l_{arg}, l_{reg}
Sampling Shapley Loss Pos (SS Pos)	Sampling Shapley	CE, l_{arg}^{pos}, l_{reg}
Sampling Shapley Loss Pen (SS Pen)	Sampling Shapley	CE, l_{arg}^{pen}, l_{reg}
RRC-Reasoning loss	Integrated Gradient	CE, l_{arg}^{RRC}, l_{reg}

Table 7: Overview of losses contained within tests of section 6.6

Accuracy results of predictions can be seen in 8. Generally, the differences in these methods are only minor. Specifically including contributions of negative arguments in the loss does not seem to make a difference in prediction quality (SS and SS Pos). For the Animal dataset, where only positive arguments are used, it must not make a difference. Using penalisation appears to have the same pitfall as the RRC-Reasoning loss. Both have substantially worse results on the Animal dataset, which we attribute to the fact, that our implementation also penalises contributions of features in neutral arguments. Doing so might constrain the features used for predictions in these methods to much.

Testing-Set	CE	SS	SS Pos	SS Pen	RRC-Reasoning
Integer	0.964	0.965	0.969	**0.973**	0.965
Jap Credit	0.58	0.588	**0.602**	0.564	0.558
Ger Credit	0.625	0.677	**0.68**	0.609	0.611
Animal	0.46	**0.457**	**0.457**	0.183	0.193

Table 8: Accuracies of predictions on testing splits of our datasets. Averaged over 40 simulations.

Testing-Set	CE	SS	SS Pos	SS Pen	RRC-Reasoning
Integer	0.93	0.944	0.938	**0.979**	0.937
Jap Credit	-0.001	**0.236**	0.2	0.118	0.113
Ger Credit	0.124	**0.479**	0.452	0.213	0.161
Animal	0.108	**0.319**	**0.319**	0.285	0.174

Table 9: Cosinus similarities of generated explanations towards arguments on testing splits of our datasets. Averaged over 40 simulations.

The scores given by similarities of generated explanations and arguments can be seen in table 9. Our default variant outperforms the others on each of the more complicated datasets. It makes sense that including negative arguments in contributions outperforms not doing so, when measuring similarity to arguments that contain them. Cross entropy performs obviously rather bad, as we've seen in previous experiments. It is surprising, that the functions penalising contributions perform this bad here. We would have assumed that penalising all features not in positive contributions is similar to focusing correct usage of features in positive contributions. Our penalising Shapley variant performs a bit better on the Animal dataset, but also fails pretty hard on the German credit dataset. This is probably due to penalising both positive and negative contributions of features $x_i \notin (A^+ \cup A^-)$, which forces them to be near 0. For ambiguous annotations (Credit datasets), this can constrain the learning to not include all relevant features.

Next we want to categorise our explanations into negative, neutral and positive categories, in order to evaluate how well our methods are at recognising each of these. F1 scores are computed for each of these categories and summarised in table 10. F1-scores for negative labels on the Animal dataset are omitted, as there do not exist any negative Arguments that these could be compared to. The results show that the positive argument variant has worse performances for neutral and negative explanation differences. This is expected, as it only forces positive contributions of features in positive arguments. However, in this experiment the penalisation Shapley variant also excels in some of the datasets. Basically on Integer and Animal datasets with complete and definite rules for annotations, it forces correct contributions of correct features. For datasets where this is not the case, it struggles with correctly identifying either of positive and negative explanations. However, these results are quite ambiguous. They only specify a correlation between normalised contributions and a threshold. Using different thresholds leads to other results. During testing we have seen that by increasing the threshold values we increase precision given by $\frac{TP}{TP+FP}$, and decrease recall, given by $\frac{TP}{TP+FN}$ of features in positive and negative arguments. Here $TP/FP/FN$ refer to the correctly identified explanations "true positives", features wrongly recognised in a category "false positives", and features missing from a category "false negatives".

Integer	CE	SS	SS Pos	SS Pen	RRC-Reasoning
negative	0.822	0.837	0.829	**0.922**	0.851
neutral	0.759	0.786	0.759	**0.932**	0.677
positive	0.932	0.944	0.941	**0.967**	0.936
Jap Credit	CE	SS	SS Pos	SS Pen	RRC-Reasoning
negative	0.211	**0.366**	0.304	0.236	0.255
neutral	0.736	**0.755**	0.753	0.747	0.734
positive	0.263	**0.393**	0.361	0.265	0.313
Ger Credit	CE	SS	SS Pos	SS Pen	RRC-Reasoning
negative	0.16	**0.364**	0.323	0.136	0.146
neutral	0.638	**0.653**	0.617	0.619	0.627
positive	0.261	0.573	**0.579**	0.208	0.213
Animal	CE	SS	SS Pos	SS Pen	RRC-Reasoning
negative	-	-	-	-	-
neutral	0.876	0.888	0.888	**0.918**	0.885
positive	0.201	0.307	0.307	**0.313**	0.243

Table 10: F1 scores of categorised explanations on testing sets splits for our datasets. Results averaged over 40 simulations.

6.7 Importance of correct Arguments

In previous tests we have identified different performances of arguments on the datasets. Therefore, we also want to measure the impact of our arguments on the results. For this we want to compare results of our argument loss on proper arguments to wrong ones. This includes random arguments, where for every attribute it is randomly selected if it is a positive/negative argument or neither, and inverted arguments, where we swap the contents of A^+ and A^-. An overview of our results on the testing splits can be seen in figure 11.

Testing-Set	Normal Arguments	Random Arguments	Inverted Arguments
Integer	**0.8**	0.502	0.215
Japanese	**0.547**	0.5	0.493
German	**0.654**	0.533	0.357
Animal	**0.251**	0.213	0.182

Table 11: Accuracies of predictions on testing sets, using our arguments, random arguments, and inverted arguments. Trained over 4 epochs. Results averaged over 40 simulations.

As expected the normal arguments are the best across all datasets, showing

us that proper annotation is important. Similarly, we can see how attempting to maximise the wrong contributions worsens the performance significantly, by looking at the results of inverted arguments. The biggest differences here can be seen for the Integer, and German credit datasets, with only two possible classes. While a random classifier should achieve an accuracy of 0.5 on these dataset, the inverted arguments worsen our results to be significantly below it. Therefore, bad annotations can actively harm our networks. This also shows us that our annotations, that we selected for these datasets are proper. In fact the random arguments hurt more than we expected. Even tough we incorporate the cross entropy in our lass, random arguments still lead to accuracies approaching those of a random classifier.
The least differences can be seen on the Japanese dataset. Either our suggested arguments are bad for this dataset, as we've previously theorised, or this dataset might just not be up to the task.

7 Conclusions

In this thesis we introduced the reader to the concepts of explanations on networks and options to interact with them during learning using attribution functions. In order to evaluate our methods we have run a multitude of tests. Using these we want to come back to our research questions that we have formulated in section 1.1.

We'll start of with **Research Question 4**: Can we improve quality of explanations of our network using Shapley values? Pretty much all of our experiments have shown this to be the case. Unfortunately evaluating explanations themselves is a difficult problem. Here we proposed methods to compare explanations to the previously defined arguments. We have seen that our sampling Shapley variant properly approximated the Shapley values and outperforms each of the other options on more complicated sets in teaching explanations more similar to the arguments.

Next we want to have a look at **Research Question 3**: What difference does it make for our network to prevent false explanations or to reinforce correct explanations? Here we have seen that penalising all irrelevant features often constrains the learner to hard. It can lead to decent results when annotations are not ambiguous in that every feature is correctly categorised without question. However, the training tasks addressed with neural networks usually don't have hard coded explanations. Otherwise, simpler approaches with better results would exist.

Our first **Research Question 1** was: Can we extend the loss [17] to properly distinguish between positive, neutral and negative explanations. In our tests we have found, that we can implement negative Arguments and thus have more similar explanation vectors in comparison to functions omitting these, although this only provided minor differences in correctly categorising these features.

Finally **Research Question 2**: Does correctly identifying explanations improve the performance of our network? There are cases where explanations do seem to help. However, in general we have found this to not be the case. Quite often the default cross entropy is just as good. In case of the Japanese dataset we have found that training the dataset only through explanations did not work. This might indicate bad arguments, which is also represented by the fact that cross entropy here outperformed methods including the argument losses. Similarly, we have seen that bad annotations actively harm prediction rates. In our default variant omitting arguments leads to an argument loss of 0. So it's always possible to just include arguments that the annotator is sure of.

Another observation that we have seen on our datasets is that baseline only harmed the networks. This might be due to the properties of our datasets. Many of their features have bucket values instead of continuous ones, which makes creating a proper baseline difficult.

7.1 Future Work

Related to our topic there are multiple possible extensions that can be done. On one hand neural networks have traditionally be comparable bad at predicting results outside of their trained value ranges. It could be interesting to see if networks that properly identify explanations can better extrapolate for such examples.

Instead of simply maximising and minimising contributions we could also attempt to maximise towards similar explanations, by using cosine similarities within the losses. This would also open the possibility of using granular explanations where features can have continuous positive or negative explanations instead of simply using negative, neutral or positive.

Another extension is the possibility of adapting sampling Shapley values to images. In our tests we have seen an improvement over other methods when using sampling Shapley contributions. By sampling areas instead of pixels it should be computationally possible to adapt this method. Here it could also be interesting to test covered objects, wherein the object itself is used as positive arguments, covered areas as negative arguments and remaining areas as neutral. Thus the network may recognise an object and where it extends beyond covered areas.

Albeit not necessarily research topic of this thesis we have encountered the problem of evaluating explanations. We are of the opinion that properly evaluating explanations is key to improving them. Unfortunately both methods for evaluation and datasets with proper annotations are next to non existent. A proper unified benchmark for evaluation would go a long way in improving comparability of methods.

List of Tables

List of Figures

8 References

[1] Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases. p. 487–499. VLDB '94, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1994)

[2] Amgoud, L., Caminada, M., Cayrol, C., Lagasquie, M.C., Prakken, H.: Towards a consensual formal model: inference part. Deliverable of ASPIC project (2004)

[3] Ancona, M., Öztireli, C., Gross, M.H.: Explaining deep neural networks with a polynomial time algorithm for shapley values approximation. In: ICML (2019)

[4] Arous, I., Dolamic, L., Yang, J., Bhardwaj, A., Cuccu, G., Cudré-Mauroux, P.: Marta: Leveraging human rationales for explainable text classification. Proceedings of the AAAI Conference on Artificial Intelligence **35**(7), 5868–5876 (May 2021), https://ojs.aaai.org/index.php/AAAI/article/view/16734

[5] Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)

[6] Ghazimatin, A., Pramanik, S., Saha Roy, R., Weikum, G.: Elixir: Learning from user feedback on explanations to improve recommender models. p. 3850–3860. WWW '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3442381.3449848, https://doi.org/10.1145/3442381.3449848

[7] Gunning, D., Aha, D.: Darpa's explainable artificial intelligence (xai) program. AI Magazine **40**, 44–58 (06 2019). https://doi.org/10.1609/aimag.v40i2.2850

[8] Jeyakumar, J.V., Noor, J., Cheng, Y.H., Garcia, L., Srivastava, M.: How can i explain this to you? an empirical study of deep neural network explanation methods. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 4211–4222. Curran Associates, Inc. (2020), https://proceedings.neurips.cc/paper/2020/file/2c29d89cc56cdb191c60db2f0bae796b-Paper.pdf

[9] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6980

[10] Lapuschkin, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions

by layer-wise relevance propagation. PLoS ONE **10**, e0130140 (07 2015). https://doi.org/10.1371/journal.pone.0130140

[11] Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 4765–4774. Curran Associates, Inc. (2017), http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[12] McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics **5**(4), 115–133 (1943)

[13] Mozina, M., Zabkar, J., Bratko, I.: Argument based machine learning. Artif. Intell. **171**(10-15), 922–937 (2007), https://doi.org/10.1016/j.artint.2007.04.007

[14] Okhrati, R., Lipani, A.: A multilinear sampling algorithm to estimate shapley values. In: Proc. of ICPR. ICPR (2020)

[15] Owen, G.: Multilinear extensions of games. Management Science **18**(5-part-2), 64–79 (1972)

[16] Ribeiro, M.T., Singh, S., Guestrin, C.: "why should i trust you?": Explaining the predictions of any classifier. p. 1135–1144. KDD '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939778, https://doi.org/10.1145/2939672.2939778

[17] Ross, A.S., Hughes, M.C., Doshi-Velez, F.: Right for the right reasons: Training differentiable models by constraining their explanations. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 2662–2670. ijcai.org (2017). https://doi.org/10.24963/ijcai.2017/371, https://doi.org/10.24963/ijcai.2017/371

[18] Shapley, L.S.: A Value for n-Person Games, pp. 307–318. Princeton University Press (1953). https://doi.org/doi:10.1515/9781400881970-018, https://doi.org/10.1515/9781400881970-018

[19] Shrikumar, A., Greenside, P., Kundaje, A.: Learning important features through propagating activation differences. In: Proceedings of the 34th International Conference on Machine Learning - Volume 70. p. 3145–3153. ICML'17, JMLR.org (2017)

[20] Stammer, W., Schramowski, P., Kersting, K.: Right for the right concept: Revising neuro-symbolic concepts by interacting with their explanations. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021. pp. 3619–3629. Computer Vision Foundation / IEEE (2021), https://openaccess.thecvf.com/content/CVPR2021/html/Stammer_Right_for_the_Right_Concept_Revising_Neuro-Symbolic_Concepts_by_Interacting_CVPR_2021_paper.html

[21] Sun, J., Lapuschkin, S., Samek, W., Zhao, Y., Cheung, N.M., Binder, A.: Explanation-guided training for cross-domain few-shot classification. In: 2020 25th International Conference on Pattern Recognition (ICPR). pp. 7609–7616 (2021). https://doi.org/10.1109/ICPR48806.2021.9412941

[22] Sundararajan, M., Najmi, A.: The many shapley values for model explanation. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 9269–9278. PMLR (13–18 Jul 2020), https://proceedings.mlr.press/v119/sundararajan20b.html

[23] Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. Proceedings of Machine Learning Research, vol. 70, pp. 3319–3328. PMLR (2017), http://proceedings.mlr.press/v70/sundararajan17a.html

[24] Teach, R.L., Shortliffe, E.H.: An analysis of physician attitudes regarding computer-based clinical consultation systems. Computers and Biomedical Research **14**(6), 542–558 (1981). https://doi.org/https://doi.org/10.1016/0010-4809(81)90012-4, https://www.sciencedirect.com/science/article/pii/0010480981900124

[25] Teso, S., Kersting, K.: Explanatory interactive machine learning. In: Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society. p. 239–245. AIES '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3306618.3314293, https://doi.org/10.1145/3306618.3314293

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Druck-Exemplaren überein.

Datum und Unterschrift:

27.02.2022 J. Gemander

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

Date and Signature:

27.02.2022 J. Gemander