university of groningen / faculty of science and engineering

An Analysis of Decompositional Rule Extraction for Explainable Neural Networks

Bachelor's Project Thesis

Nicholas Kees Dupuis, s2843013, N.K.Dupuis@student.rug.nl,
Supervisor: Dr Bart Verheij, bart.verheij@rug.nl

Abstract: As artificially intelligent systems take a more important role in our society, it becomes important to be able to explain their decisions. Neural networks have recently been one of the most successful tools for producing intelligent systems, but the decisions of neural networks are inherently difficult to explain, as the internal state is incomprehensible. Rule extraction seeks to make that state accessible to humans, and bring explainability to neural networks. This paper analyses a decompositional approach to rule extraction as applied to feed forward networks trained with back propagation, and finds that explainability may come at the cost of losing robustness in the presence of noise, scalability and reasonable time complexity, and flexibility to learn different types of relationships.

1 Introduction

There has been a strong call from the research community for an increased focus on Artificial Intelligence (AI) safety, and in particular, explainability of AI systems. Explainable AI (or XAI) refers any system for which a human can understand why a decision is made and how it came to its conclusions (Gunning, 2017). The Asimolar A.I. principles, signed by 1273 AI researchers calls specifically for systems which can "provide a satisfactory explanation auditable by a competent human authority". It is thus important to be able to construct a transparent system for which the internal state is accessible to a human operator and can be interpreted unambiguously.

1.1 Why XAI?

There are several reasons why explainable AI is important. In the short term, as AI systems are being included in larger, more complex systems, they will need to be proven robust before they can be included. This is especially true when systems are used in safety critical applications in which failure may result in injuries, damages, or loss of life. It is important that these types of failure can be proven unlikely or impossible to occur. This kind of extensive verification requires a strong understanding of the inner working of such an AI system.

In the long term, it will be necessary to be able to align AI systems' values with human values (ASI, 2017). One potential obstacle to doing this successfully is that a sufficiently intelligent AI with goals which do not align with human goals would have a strong incentive to deliberately decieve its user about its plans, goals, and values in order to keep them from being altered (Bostrom, 2014). While developing AGI*, it will be imperative to verify that the beliefs and values of those systems are desirable, and match those of humankind.

1.2 Increasing Explainability with Rule Extraction

The use and efficacy of neural networks has exploded in recent years, and they are quickly taking a more prominent role in our society and economy. This poses a problem for explainable A.I, as neural networks are inherently incomprehensible. One of the methods which may help make them more explainable, and thus more safe, is rule extraction (Hailesilassie, 2016).

*Artificial General Intelligence, or AGI, refers to future systems which, unlike "narrow AI" will be able to solve problems in a wide range of domains (Goertzel and Pennachin, 2007).

Rule extraction refers to any method which takes a neural network as input and returns symbolic information which was embedded in the weights and biases of that network. When a neural network can correctly identify relationships in a set of data, it can be assumed that knowledge about those relationships exists, albeit abstractly, in the weight vectors of that network. Rule extraction seeks to make this knowledge externally accessible by constructing a symbolic rule set with declarative logical rules which are easy to understand.

1.2.1 Knowledge Acquisition for Symbolic AI

Rule extraction can further be used to increase explainability in A.I. by helping solve the knowledge acquisition problem faced by symbolic AI systems. In contrast with deep learning, symbolic systems are inherently more explainable, because knowledge is encoded in a declarative way. However, constructing knowledge bases is incredibly difficult and labor intensive, and often requires expert input. Rule extraction has the potential to assist in this task by deriving new rules from neural networks, and thus combining the learning power of deep learning with the explainability of symbolic A.I (Ultsch and Korus, 1995).

1.2.2 Decompositional Rule Extraction

The most direct form of rule extraction is decomposition, which refers to the decompilation of the internal state (weights and biases) of a neural network. Relationships between individual neurons are derived in order to ultimately derive the relationship between the input layer and the output layer.

There exist other families of rule extraction methods which do not examine the weights and biases of the network. These approaches do not deal with the internal state of the network, and instead treat the network as a "black box" and use it to generate samples for symbolic learning algorithms (Robert Andrews and Tickle, 1995). Decomposition directly tries to translate the complex internal state of a neural network into something legible and explainable, and will be the primary focus of this paper.

1.3 An Empirical Analysis of Rule Extraction

There exist a wide range of machine learning approaches which differ in both performance and in explainability. Deep learning has a characteristically high performance, while also having very low explainability. Any advance in XAI requires not only improving explainability, but also maintaining performance (Gunning, 2017). In order for rule extraction to be useful in explaining deep learning, it should keep those things which make deep learning powerful in the first place. These include the following:

1. Robustness in the presence of noise
2. Scalability and reasonable time complexity
3. Flexibility to learn different types of relationships

In order to test these, different types of rule sets will be generated and tested with a rule extractor. Experiments will be performed using the KT algorithm by LiMin Fu (Fu, 1994), one of the earlier decompositional rule extraction methods. While it is not correct to generalize the efficacy of the KT algorithm directly to all rule extraction algorithms, it is a heavily cited, tried and true method, and the types of problems faced will serve to shed light on the challenges of decompositional rule extraction approaches in general.

2 The KT Algorithm

The KT algorithm (Fu, 1994) takes in a general feed forward neural network as input and returns a set of rules. More detail can be found in the original paper, along with mathematical justifications for each of the design decisions, but an overview is provided here.

The algorithm tries to find a set of rules which map inputs to outputs, by finding intermediate rules which connect neurons locally. All neurons in the network are therefore treated as boolean literals, and their local relationships are described by if-then implications. To convert the neurons, which have a continuous activation, to boolean literals

with discrete values (true or false), the backpropagation algorithm used must have a sigmoid activation function with a high λ value:

$$\frac{1}{1+e^{-\lambda x}}$$

This ensures that a neuron's activation is either very high or very low, making ambiguous activations extremely unlikely. The neuron's continuous activation can now be converted into a boolean, where a high activation is true, and a low activation is false. This now allows us to derive logical relationships between the neurons of the network.

The KT algorithm can be divided into 3 steps:

1. Search: Rules defining the relationship between neurons of adjacent layers are found.
2. Re-write: Those rules are rewritten to only describe neurons in the input and output layers.
3. Rule Refinement: The accuracy of those rules are tested against some data to remove poorly performing rules.

2.1 Search

The search phase tries to derive if-then implications, where an antecedent is comprised of a set of literals connected by conjunctions, and the consequent implied by the antecedent is a single literal. The literals in the antecedent may be either positive or negated, and the consequent may likewise also be negated.

The search phase goes neuron by neuron through the network and tries to generate rules in which the consequent is the neuron currently being examined. Antecedent literals are drawn from the previous layer of neurons, whose activations directly impact the consequent neuron. The previous layer of neurons is refered to as the set of "attributes." Attributes can be either positive(A^+) or negative(A^-), referring directly to the sign of the weight which connects it to the consequent. This is not to be confused with positive or negated literals, which refers to the logical operation of negation.

Rules generated by the KT algorithm may take two forms: Confirming

$$IF \quad A_1^+, ...A_i^+, \neg A_1^-, ...\neg A_i^- \quad THEN \quad C$$

or disconfirming.

$$IF \quad \neg A_1^+, ...\neg A_i^+, A_1^-, ...A_i^- \quad THEN \quad \neg C$$

Thus if all positive literals in the antecedent are true, and all negated literals are false, it holds that the consequent must be true.

2.1.1 The Certainty Condition

For each neuron, the search phase tries to find rules which satisfy the certainty condition:

- If in the antecedent all positive literals are true (the corresponding neurons have activations of 1), and all negated literals are false (the neurons have an activation of 0), then the neuron referred to by the consequent MUST have an activation greater than 0.5. (or in the case of a disconfirming rule, less than 0.5)

In practice, this means that if we constrain the activations of the neurons referred to in the antecedent, regardless of the activations of any unconstrained neurons, the activation of the neuron referred to by the consequent must mathematically be greater than 0.5 (or less than 0.5 if the rule is disconfirming).

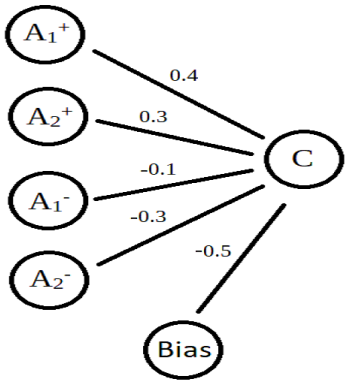


Figure 2.1: Example of a network excerpt

For example, take the network excerpt from figure 2.1, and suppose we have the following potential rules:

1. $IF \quad A_1^+, A_2^+ \quad THEN \quad C$
2. $IF \quad A_1^+, A_2^+, \neg A_2^- \quad THEN \quad C$

3. $IF \quad \neg A_1^+ \quad THEN \quad \neg C$

For C to have an activation of at least 0.5, the sum of all the incoming inputs must be greater than zero. In the case of rule 1, it is possible to have a sum greater than zero, but it is not guaranteed, because the unconstrained neurons (A_1^- and A_2^-) could bring that sum as low as -0.2. Therefore rule 1 does not satisfy the certainty condition. Rule 2 on the other hand, constrains A_2^- to an activation of zero, thus guaranteeing the sum be greater than zero. Because of this addition, rule 2 does satisfy the certainty condition. Rule 3 is disconfirming, so the sum of the inputs must be guaranteed to be less than zero. In order to reach a sum greater than zero, both A_1^+ and A_2^+ must have an activation of one to overcome the negative bias term. Since A_1^+ is constrained to an activation of zero, this becomes impossible, and thus rule 3 also satisfies the certainty condition.

The search phase performs a heuristic tree search for each neuron in the network to find confirming and disconfirming rules which satisfy the certainty condition.

2.2 Re-write

After the search phase, the algorithm has a set of rules which all satisfy the certainty condition. Many of the boolean literals will refer to "concepts", or neurons found in the hidden layer(s) and not in the input or output layers. In order for all the rules to contain only literals from the input layer in the antecedent, and literals from the output layer as consequents, the rules must be rewritten.

For example, suppose we have the rule:

- $M_1 \wedge M_2 \rightarrow C$

In which M_1 and M_2 both refer to concepts, which are in turn consequents of other rules:

- $A_1 \wedge B_1 \rightarrow M_1$
- $A_3 \rightarrow M_1$
- $A_2 \wedge \neg B_2 \rightarrow M_2$

The rewrite step creates new rules by replacing concepts with the antecedents of rules in which that concept is the consequent:

- $A_1 \wedge B_1 \wedge A_2 \wedge \neg B_2 \rightarrow C$
- $A_3 \wedge A_2 \wedge \neg B_2 \rightarrow C$

2.3 Rule Refinement

The last step of the KT algorithm takes a set of rules produced by the previous two steps, and seeks to refine them by removing poorly performing rules. This stage is technically optional, but in practice is quite useful, as the algorithm often returns many partially correct or incorrect rules alongside the correct ones.

This refinement can be done by taking the training data used to train the neural network, and using it to test each rule in the rule set. If it is often the case that when the antecedent of a rule is true and the consequent of the rule is false, then this rule ought to be discarded from the rule set. The rules which perform well under rule refinement are kept, and make up the final extracted rule set.

2.4 Parameters

There are two constants which assist in limiting the time complexity of the algorithm and need to be defined at the start:

The first is k, which refers to the maximum number of literals in a rule's antecedent. The time it takes to run the algorithm grows exponentially with the number of literals to consider, and limiting this with the constant k is a necessary condition for the algorithm to terminate in reasonable time.

The second is ϵ which is an addition to the certainty condition. ϵ effectively refers to "how certain" a rule must be to be included. Instead of comparing a sum to the threshold in the search phase, the algorithm compares the sum to the threshold plus ϵ. A positive ϵ requires the rules to be more certain, and a negative ϵ allows them to be less certain.

3 Methods

In each experiment, different rule sets will be learned by a neural network, and then subsequently extracted. The extractor's performance will be determined by comparing the initial rule set to the final rule set.

The neural network will be performing a classification task in which, based upon a series of inputs, it must classify a data point as either true or false.

First, an initial rule set is selected to be tested. Which rule set is selected is the dependant variable

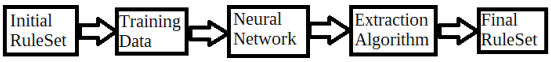


Figure 3.1: Diagram of the experimentation process

of any experiment. Next, data is generated from the initial rule set. For each data point, twelve inputs are all initialized randomly as either ones or zeros. If a confirming rule applies, the output is set to 1. If a disconfirming rule applies, the output is set to 0. If no rule applies, then the output is randomly initialized to either zero or one. This means that, while the input space will be uniformly represented, the output space may not be.

Next, the data is used to train and test a neural network. The network used will be a 3 layer network with 12 input neurons, 12 hidden neurons, and 1 output neuron. 2400 training samples will be used to train the network, after which the network will be tested 10 times, each time with 10000 different samples of randomly generated testing data. For each rule set there exists a maximum accuracy a classifier can be expected to achieve, which can be found by assuming the classifier correctly classifies all samples for which a rule applies, and then guesses on all remaining samples. If the network fails to get within half a percent of this maximum accuracy in any of the 10 times it is tested, it is trained again. This continues until all 10 tests are passed. This level of strenuous training and testing ensures with a high degree of certainty that before extraction begins, the network has correctly learned the initial rule set.

Finally, Rules are extracted from the network using the KT algorithm. (Fu, 1994)

3.1 Evaluating the Final Rule Set

An extraction is either deemed "successful" or "unsuccessful." If the final rule set can be used to classify the data with an accuracy which is at least within 1 percent of the maximum accuracy, the extraction is considered to be a success. It is possible for the extractor to return a partially correct rule set which classifies above 50 percent, but does not manage to classify as well as the original rule set and neural network. This is still deemed a failed extraction.

3.2 Parameters and Time

Because the KT algorithm requires the parameters k and ϵ to be initialized before it runs, and these values can greatly affect the success or failure of extraction, each experiment runs the algorithm many times with different parameters until a solution is found. This is done in the following way:

1. Set k to 1 and ϵ to 3 at the start
2. While ϵ is greater than or equal to -3 decrease ϵ by 0.5
3. If ϵ is less than -3, set ϵ to 3 and increment k

If this method fails to produce a solution in 2 minutes, the method is terminated.

4 Experiments

4.1 Robustness in the Presence of Noise

One of the important questions about rule extraction is how robust it is in the presence of noisy data. If a neural network can learn a rule set well with noisy data, can that rule set still be extracted?

For this experiment, 3 rule sets are tested[†]:

1. $A_1 \rightarrow C$, $\neg A_1 \rightarrow \neg C$
2. $A_1 \rightarrow C$, $A_2 \rightarrow C$, $\neg A_1 \wedge \neg A_2 \rightarrow \neg C$
3. $A_1 \wedge \neg A_2 \rightarrow C$, $\neg A_1 \wedge A_2 \rightarrow C$, $A_1 \wedge A_2 \rightarrow \neg C$, $\neg A_1 \wedge \neg A_2 \rightarrow \neg C$

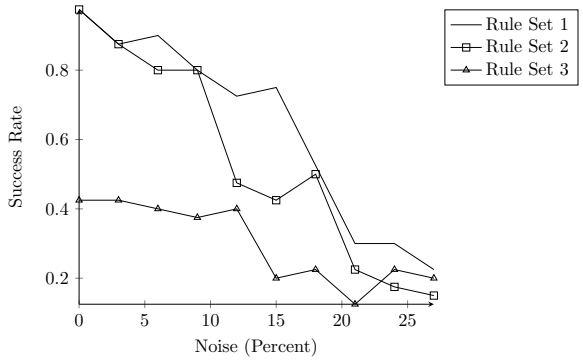
To determine robustness, "noise data" is added to the training data, or data points for which the output is entirely random, and not determined by the rule set. This is analogous to "outlier" points, or points which don't fit the general pattern in the data. Experiments were thus performed with different percentages of the data points being noise.

Each experiment was performed 40 times, and the success rate was recorded. These results can be seen in figure 4.1.

Evidently, the addition of outliers has a strong and immediate effect on the performance of the rule extractor. Clearly the quality of the data is an important factor.

[†]literals A_i refer to the twelve input neurons, and C refers to the output neuron

Figure 4.1: Results of Noise Data



Real world data is seldom without noise, and one of the reasons neural networks are so useful is their robustness in the presence of noise. If a rule extraction technique is to be viable for solving the problem of transparency it should work well in cases where the neural network was trained on noisy data.

4.2 Scalability and Time Complexity

Another concern when using rule extraction is scalability. Naturally the ideal would be for rule extraction to scale as well as the neural networks used, such that any deep learning system could be extracted from, and thus explained. This scalability depends on two factors: time complexity and accuracy as rule sets get larger.

In order to test these, two experiments will be performed. The first will test the effect of adding more rules. Each rule will be a single input which implies the output. The second experiment will test one rule, but the antecedent will have a variable number of attributes.

Each experiment will consist of 20 trials, and the success rate and the average time will be recorded in seconds. If the algorithm fails to find a solution after 2 minutes, this will be considered a failed extraction, and the time will not be recorded. (see results in table 4.1 and 4.2)

The accuracy and time complexity are naturally correlated, as the algorithm fails whenever it takes too long (2 minutes) to find the answer. Observe the following extractor output for the rule set $A_1 \wedge A_2 \rightarrow C$: (see figure 4.2)

Table 4.1: Results of adding rules

	Succ rate	Avg. Time
1 Rule	90%	0.78
2 Rules	85%	2.8
3 Rules	85%	7.5
4 Rules	65%	2.3
5 Rules	70%	6.1

Table 4.2: Results of antecedent size

	Succ rate	Avg. Time
1 Att.	95%	0.74
2 Att.	90%	15.6
3 Att.	50%	9.0
4 Att.	50%	63.6
5 Att.	0%	N.A.

Figure 4.2: Sample extractor output

Success (y/n)	Time
y	9
y	1
y	2
y	2
y	34
y	7
y	1
n	120+
y	47
y	2
y	8
y	2
y	7
y	1
y	52
y	2
y	1
y	7
y	96
n	120+

What is apparent is that the time to extract is generally quite low, with several outliers. When testing the rule with 3 attributes ($A_1 \wedge A_2 \wedge A_3 \rightarrow C$), it was found that in half of the trials the solution needed at least 120 seconds to be found (the extractor terminated at 2 minutes), and yet in the other half of the trials the solution was found in just 9 seconds on average. The extractor is deterministic, and will perform exactly the same if tested twice on the same network. Both training of neural networks and the generation of training data, however, are stochastic, and it appears the result networks can either be easy to extract from or quite hard.

This could be explained by the fact that each time the extractor increments k, or the number of antecedents a rule may have, the time it takes to search grows rapidly (exponentially). Thus, the time to extract may be expected to increase exponentially, rather than linearly. There is a fundamental limit to k, namely the largest layer of the network (in this case 12 neurons), but as the extractor seldom makes it past $k = 5$ before the 2 minutes are up, it is unlikely this limit would ever be reached in reasonable time.

In order to determine whether it might be expected for those cases which fail to terminate in 2 minutes would eventually find a solution given unlimited time, a single trial experiment was performed with a single rule with 5 antecedent attributes (the case which in the previous experiment, given 20 trials, failed to ever terminate in under 2 minutes).

Given unlimited time, the extractor did find a solution in 6 minutes and 4 seconds. This is a good sign for sure, but considering the network it extracted from managed to finish training in just a couple seconds, this is a problem. Consider cases where training a neural network can take hours or days. If scaling up the problem size would have the same effect on the rule extractor, we might not be able to expect a solution for months, or even years.

4.3 Flexibility

Neural networks are quite good at learning many different kinds of relationships. Does rule extraction share that kind of flexibility? To test this, we will consider different kinds of input space partitions.

Any rule set is essentially a description of how to classify different parts of the input space. Take the following rule set:
$A_1 \rightarrow C$, $A_2 \rightarrow C$

This can also be described by the input space diagram show in figure 4.3.

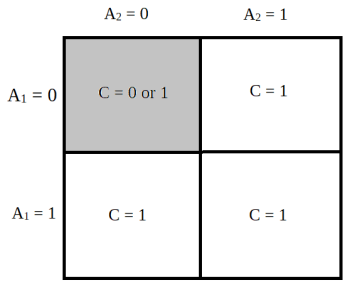


Figure 4.3: Diagram of the input space

In this case, the inputs A_1 and A_2 are positively correlated with the output C. A similar rule set can be constructed such that both inputs are negatively correlated with the output:
$\neg A_1 \rightarrow C$, $\neg A_2 \rightarrow C$

The input space diagram for this rule set is also similar, only it has now been flipped. (see figure 4.4)

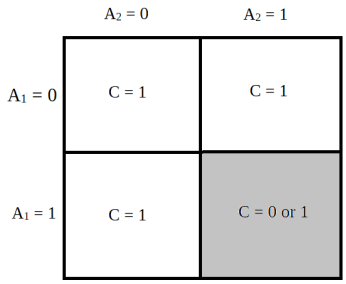


Figure 4.4: Diagram of the flipped input space

Flipping the input space has no measurable effect on the neural network's ability to learn relationships. The following experiment seeks to determine whether or not rule extraction shares this flexibility. To test this, 6 rule sets are constructed in which the inputs positively correlate to the output, and 6 similar but "flipped" rule sets are constructed which negatively correlate to the output: (see table 4.3)

Table 4.3: Rule Sets to be tested

	Original	Flipped
Set 1a	$A_1 \to C$	$\neg A_1 \to C$
Set 1b	$\neg A_1 \to \neg C$	$A_1 \to \neg C$
Set 2a	$A_1 \to C$ $A_2 \to C$	$\neg A_1 \to C$ $\neg A_2 \to C$
Set 2b	$\neg A_1 \to \neg C$ $\neg A_2 \to \neg C$	$A_1 \to \neg C$ $A_2 \to \neg C$
Set 3a	$A_1 \to C$ $A_2 \to C$ $A_3 \to C$	$\neg A_1 \to C$ $\neg A_2 \to C$ $\neg A_3 \to C$
Set 3b	$\neg A_1 \to \neg C$ $\neg A_2 \to \neg C$ $\neg A_3 \to \neg C$	$A_1 \to \neg C$ $A_2 \to \neg C$ $A_3 \to \neg C$

After performing each experiment 20 times, the following results are achieved: (see table 4.4)

Table 4.4: Results of Flipping the Input Space

	Original	Flipped
Set 1a	85%	50%
Set 1b	35%	85%
Set 2a	85%	15%
Set 2b	30%	90%
Set 3a	75%	25%
Set 3b	65%	90%

One observation which is striking, is not the effect of correlation, but rather that those rule sets with negated antecedents all performed worse than their twin rule set with positive antecedents. To test this observation, the experiment was run again on the same rule sets, dividing them this time into those with positive antecedents, and those with a negated antecedents. (see table 4.5)

Table 4.5: Results of testing a negated antecedent

	Positive Ant.	Negated Ant.
Set 1a	90%	55%
Set 1b	100%	20%
Set 2a	95%	50%
Set 2b	90%	35%
Set 3a	75%	45%
Set 3b	95%	25%

The KT algorithm itself has no explicit preference for positive or negative antecedents in its tree search. The only other explanation for this difference in accuracy is that the structure of the neural network extracted from is different in a way that is more challenging to extract from. This is surprising, considering that both networks trained on rules with positive antecedents and the nets trained on rules with negative antecedents classify with the same maximum accuracy.

What this does show is that two very similar rules can differ significantly in their difficulty with respect to the rule extractor, and that to achieve a high success rate the rules may need to adhere to a particular format. This is not particularly flexible, and poses another limit to the utility of rule extraction.

5 Discussion

The experiments of this paper tested the presence of the following three characteristics in rule extraction:

1. Robustness in the presence of noise
2. Scalability and reasonable time complexity
3. Flexibility to learn different relationships

Firstly, the extractor proved not to be very robust when noise was included in the data. A small amount of noise had an immediate effect on the performance of the extractor, and a large amount of noise completely crippled its ability to derive the rule sets. The presence of noise in data is often a reason to choose deep learning over other machine learning approaches, and so rule extraction failing to handle noise would prove a major limitation to its application.

Secondly, when the rule set was scaled up, both by adding more rules and by increasing the size of those rules, the performance dropped significantly. Especially in the case of increasing the size of rules, experimental results suggest an exponential time complexity, which would not be considered reasonable.

This is consistent with other research which shows that rule extraction, specifically those methods which aim for perfect network fidelity like the method used in this paper, are in the general case

NP-hard (Chorowski and Zurada, 2011). This suggests an upper limit on the problem sizes that rule extraction can be expected to handle.

Thirdly, it was shown that a small change in the rule set, namely flipping the input space partition, could produce a large negative effect on the performance of the extractor. Specifically, when the classification problem involves matching inputs of 1 to an output, the extractor performs far better than when it involves matching inputs of 0 to that same output. This is not very flexible and poses a problem, particularly because when trying to discover rules, we can't be expected to know if those rules match a particular format the rule extractor can handle well.

What this shows is that qualities which make neural networks so useful across so many domains may be limiting factors for the use of rule extraction to provide explainability. It may be necessary to make sacrifices in the name of explainability by limiting problems to be within a domain for which rule extraction works well.

6 Conclusion

When discussing machine learning methods, we should consider both learning performance as well as explainability. This can be done by plotting these qualities on a graph like the one published by Darpa shown in figure 6.1 (Gunning, 2017).

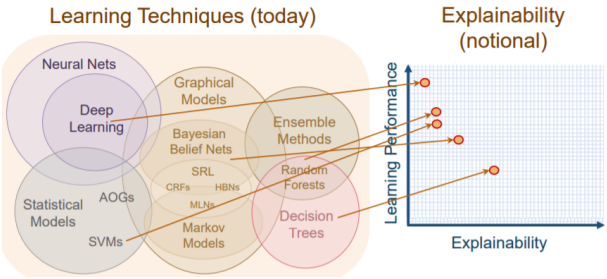


Figure 6.1: Current Machine Learning Methods

For rule extraction to be useful, it must not only make deep learning more explainable, but it must also avoid harming learning performance. There already exist methods with lower learning performance which are more explainable than deep learning, and so for rule extraction to be an improvement on the state of the art, it must be able to combine the high learning performance of deep learning with the explainability of logical rules.

The KT algorithm requires sacrificing many of the things which make deep learning powerful in order to successfully derive the rule set. If rule extraction is to truly make deep learning explainable, it should place as few limitations as possible on learning performance, and the KT fails to do so. Future research should focus on mitigating these limitations as best as possible in order successfully bring the power of deep learning to explainable artificial intelligence.

References

Asimolar AI Principles. `https://futureoflife.org/ai-principles/`, 2017.

Nick Bostrom. *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press, 2014.

J. Chorowski and J. M. Zurada. Extracting rules from neural networks as decision diagrams. *IEEE Transactions on Neural Networks*, 22(12):2435–2446, 2011.

LiMin Fu. Rule generation from neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 24:1114–1124, 1994.

Ben Goertzel and Cassio Pennachin. *Artificial general intelligence*, volume 2. Springer, 2007.

David Gunning. Explainable artificial intelligence (xai). *Defense Advanced Research Projects Agency (DARPA), nd Web*, 2017.

Tameru Hailesilassie. Rule extraction algorithm for deep neural networks: A review. *arXiv preprint arXiv:1610.05267*, 2016.

Joachim Diederich Robert Andrews and Alan B Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8:373–389, 1995.

Alfred Ultsch and Dieter Korus. Automatic acquisition of symbolic knowledge from subsymbolic neural networks. *3rd European Congress on Intelligent Techniques and Soft Computing*, 1:326–331, 1995.