
Neural Logic Rule Layers

Jan Niclas Reimann

Department of Automation Technology
South Westfalia University of Applied Science
Soest, Germany
reimann.janniclas@fh-swf.de

Andreas Schwung

Department of Automation Technology
South Westfalia University of Applied Science
Soest, Germany
schwung.andreas@fh-swf.de

Abstract

Despite their great success in recent years, deep neural networks (DNN) are mainly black boxes where the results obtained by running through the network are difficult to understand and interpret. Compared to e.g. decision trees or bayesian classifiers, DNN suffer from bad interpretability where we understand by interpretability, that a human can easily derive the relations modeled by the network. A reasonable way to provide interpretability for humans are logical rules. In this paper we propose neural logic rule layers (NLRL) which are able to represent arbitrary logic rules in terms of their conjunctive and disjunctive normal forms. Using various NLRL within one layer and correspondingly stacking various layers, we are able to represent arbitrary complex rules by the resulting neural network architecture. The NLRL are end-to-end trainable allowing to learn logic rules directly from available data sets. Experiments show that NLRL-enhanced neural networks can learn to model arbitrary complex logic and perform arithmetic operation over the input values.

1 Introduction

Deep neural networks (DNN) have shown great success in previous years in various domains like image classification, speech recognition and natural language processing. However, DNNs operate in a black box fashion where the results obtained by running through the network are in general difficult to understand. Hence, compared to other machine learning algorithms like decision trees or fuzzy classifiers, DNN suffer from low interpretability. However, for various applications, a reasoning structure which can easily be understood by a human is highly desirable. A natural form to constitute such easily understandable relations are provided by applying logical rules.

In this paper, we develop new learning modules for neural networks called neural logic rule layers (NLRL) with in-built capacity to represent logical rules. The approach relies on the concept of boolean logic which represents rules in form of algebraic equations. As every boolean logic can be represented by a combination of negations, AND and OR-rules, we concentrate on a neural network representation of these logic subtypes, i.e. we propose neural network layers embedding disjunctive and conjunctive normal forms if at least two layer are stacked. The NLRL is split into three blocks, namely the negation operator realized using a gating mechanism, the AND-connector and the OR-connector. The AND- and OR-connector thereby share their weights. Optionally, the decision whether an AND- or an OR-clause is used for reasoning can be represented by an additional gating unit. Despite the combinatorial character of the logic units, the proposed architecture is end-to-end trainable using backpropagation.

Besides their increased interpretability, NLRL inherently incorporate inductive bias into the neural network structure. Due to the restrictions raised on the network parameters for representing logical rules, comprehensible inductive bias is introduced, especially if used as a classification layer. Furthermore, if knowledge is available about the underlying classification problem, NLRL allow for the

direct incorporation of such knowledge in form of logical rules. The rules could finally be fine tuned using backpropagation. Furthermore, if the NLRL is used in the output layer, i.e. the output is the result of a single rule evaluation, it is most likely that no output neuron fires if an unknown input is presented. This is in sharp contrast to the most often used softmax layer where the most likely class is output yielding misleading results.

We experiment on a variety of different logic function and different network structures. Furthermore, we test on various nonlinear function representations which are also representable by the NLRL. The results obtained on the synthetic data sets are very encouraging motivating further investigations on e.g. image classification.

2 Neural Logic Normal Form Units

We propose neural logic normal form units, i.e. units to represent logical relations in form of AND- and OR-rules. We model the relations by using the framework of propositional logic and boolean algebra as discussed next.

2.1 Propositional Logic and boolean algebra

Logical rules provide a convenient way to model expert knowledge in various domain. Such rules can be described using basic operations as logical AND (conjunction), logical OR (disjunction) and negation using the truth tables TRUE and FALSE. Equivalently, we can represent the truth values by integer values 0 and 1 and apply corresponding arithmetic operation. Hence, assuming integer variables $x, y \in \{0,1\}$, the conjunction, disjunction and negation can equivalently be defined by the algebraic relations

$$x \wedge y = xy, \quad (1)$$

$$x \vee y = x + y - xy, \quad (2)$$

$$\neg x = 1 - x. \quad (3)$$

The generalization of the conjunction and disjunction to more than two input dimensions, i.e. $x \in \{0,1\}^n$, yields

$$\bigwedge_{i=1}^n x_i = \prod_{i=1}^n x_i, \quad (4)$$

$$\bigvee_{i=1}^n x_i = \left(\bigodot_{i=2}^n (1 - x_i) \odot (-1 \ x_1) \right) \mathbf{1} + 1. \quad (5)$$

where the Kronecker operator is used to represent the algebraic OR. Based on the basic operations, additional operations can be defined where implication, exclusive OR and equivalence being the most prominent ones defined as follows

$$x \rightarrow y = \neg x \vee y \quad (6)$$

$$x \oplus y = (x \vee y) \wedge \neg(x \wedge y) = (x \wedge \neg y) \vee (\neg x \wedge y) \quad (7)$$

$$x \equiv y = \neg(x \oplus y) = (x \wedge y) \vee (\neg x \wedge \neg y) \quad (8)$$

As indicated, the above additional operations can be traced back to the basic operations. However, they require a sequence of conjunction and disjunction operations or vice versa.

Furthermore, conjunction, disjunction and negation are related via the De-Morgan laws

$$\bigwedge_{i=1}^n x_i = \neg(\bigvee_{i=1}^n \neg x_i), \quad (9)$$

$$\bigvee_{i=1}^n x_i = \neg(\bigwedge_{i=1}^n \neg x_i). \quad (10)$$

As a consequence, for arbitrary logical relations, only two of the three basic operations are required. This property will be further exploited as an alternative representation for the logical layer definition discussed next.

2.2 End-to-end training of logic layers

Using boolean algebra, rules can be equivalently defined by means of algebraic equations. To represent boolean algebra in form of a neural network architecture, we propose a corresponding layer structure consisting of three blocks, namely the input negation, a parallel path to represent the AND and OR-relation which share their weights and an output gating to decide whether AND or OR-relation shall be used. The architecture is illustrated in Fig. 1.

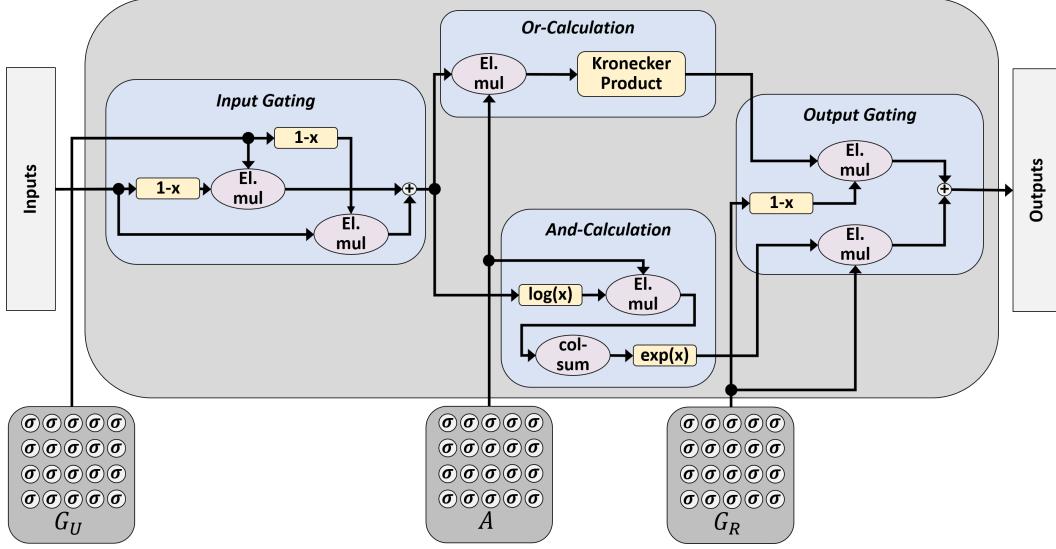


Figure 1: Architecture of NLRLs.

For the feedforward path we obtain the following description:

$$\text{Negation Gating: } \hat{x} = (1 - \sigma(g^u)) \circ x + \sigma(g^u) \circ (1 - x), \quad (11)$$

$$\text{AND-rule: } \text{AND} = \exp(\mathbf{A}(\log(|\hat{x}| + \epsilon))), \quad (12)$$

$$\text{OR-rule: } \text{OR} = ((1 - a_n \hat{x}_n) \odot \dots \odot (1 - a_2 \hat{x}_2) \odot (-1 a_1 \hat{x}_1)) \mathbf{1} + 1, \quad (13)$$

$$\text{Output Gating: } y = (1 - \sigma(g^r)) \circ \text{AND} + \sigma(g^r) \circ \text{OR}. \quad (14)$$

where \mathbf{A} , \mathbf{g}^u and \mathbf{g}^r are the learnable weights. The weight vector \mathbf{g}^i models the negation of the logical input variable while \mathbf{g}^o is the output gating, which decides whether conjunction or disjunction are used by the neurons of the layer. The component $a_{i,j}$ of the weight matrix \mathbf{A} assigns the input variable x_i to the j th rule in the layer. Note that all weights have to be restricted to $[0,1]$. To allow for an end-to-end training of the variables, we define the weights as continuous variables which are send subsequently through a sigmoidal activation following the same approach as in Trask et al. (2018). Note also, that the logic relations as modeled by the forward path hold only if the inputs to the layer are boolean values, i.e. $x \in \{0,1\}^n$. However, similar to the weights, to assure an end-to-end training we use sigmoidal activations in the preceding layer to restrict the values to the interval $[0,1]$.

As previously discussed, we can restrict ourselves to the negation normal form (Robinson and Voronkov, 2001), i.e. the use of negation operator together with either conjunction or disjunction operator to represent arbitrary logical rules. Hence, using the same architecture as in Fig. 1, we can alternatively implement either the conjunction or disjunction path alone. In this case, the output gating is replaced by a negation gating of the output variable. This latter representation provides some computational advantages, as the parallel computation of conjunction and disjunction is avoided. Particularly, due to the Kronecker product, the calculation of the disjunction scales exponentially with the input dimension and the number of neurons in the layer and is hence computationally expensive. Hence, we always use the conjunction operator in this setting.

2.3 Representational Power of Stacked NLL

We already discussed the representation of a single NLRL-neuron as either representing a conjunction or disjunction rule which is made learnable by the output gating. Alternatively, we can implement the

negation normal form by using the conjunction solely and allow for negation of the output. Also, we can fix the representation by design, i.e. omit the output gating completely. If NLRL are stacked layer-wise, the representational power drastically increases. Stacking two layers of NLRLs allows for arbitrary logic relations in form of logic normal forms, i.e. either conjunctive or disjunctive normal forms which particularly includes the previously defined implication, exclusive or and equivalence operators.

As a by-product, the NLRLs are also able to represent various forms of functions. Due to the multiplicative nature of the algebraic AND-rule, multiplication of inputs as well as polynomial relations, where the maximum achievable order of the polynomial equals the number of layers, are learnable. The OR-rule can also represent addition and subtraction operations. Combining both algebraic AND and OR-rule, arbitrary complex relations combining the above functions can be learned.

The constraints on the optimization parameters and the in-built rule base structure of NLRL already provide considerable inductive bias to the learning process when NLRL are used. However, there are application examples where additional a priori knowledge is available, e.g. by experts or experienced users. Sharing this additional knowledge with the learning process can be beneficial. Human knowledge can often be represented by logic rules or fuzzy rules, hence NLRL naturally allow for a direct incorporation of such rules by correspondingly initializing the optimization parameters instead of random initialization.

Furthermore, as we use differentiable activation functions for the inputs, they are restricted to the interval $[0, 1]$. By interpreting the input activations as membership functions, the corresponding NLRL can be interpreted as representing a fuzzy rule.

Finally, we consider the extrapolation behavior of NLRLs. To this end we consider a typical classification scenario where we assume that a feature extractor, e.g. some convolutional layers, have been stacked downstream of the logic layers. We compare the behavior of NLRL with classical fully connected layers with a final softmax layer for multiclass classification. If we assume that an input is provided to the network, that does not belong to any of the previously seen classes, e.g. an outlier, the softmax layer will output the most likely, but wrong class affiliation. Hence, the result could be misleading. In contrast, the NLRL will simply give no output, i.e. no rule will be fired, such that the results can be directly interpreted as a not classified input.

3 Related Work

Our rule base architecture in NN can be classified to the research direction of informed machine learning, see von Rueden et al. (2019) for a recent survey. Informed machine learning summarizes approaches of incorporating various forms of a priori available domain knowledge into the learning process which constitutes a form of inductive bias DesJardins and Gordon (1995). The approaches differ in the type of knowledge (e.g. natural sciences, world knowledge or expert intuition), the way the knowledge is represented (e.g. rules, constraints, simulations or graphs) and where the knowledge is incorporated into the learning pipeline (e.g. training data, training algorithm or hypothesis space) (von Rueden et al., 2019). Our NLRL incorporate world and expert knowledge in form of rules into the hypothesis space. Related approaches also integrating logical rules have already been presented. In Hu et al. (2016), a regularization of NN is performed by using logical relations as regularizer while Diligenti et al. (2017a) incorporate logic rules in form of constraints into the training scheme. Similarly, Diligenti et al. (2017b) propose semantic based regularization. First approaches working directly in the hypothesis space are Markov Logic Networks (Richardson and Domingos, 2006) where a first order logic base is augmented with trainable weights. A hypothesis space approach for fast construction of data sets for machine learning problems has been presented in Ratner et al. (2016), while Sachan et al. (2018) present a similar approach for solving physics problems in text books. In contrast to the previously mentioned approaches, we propose to integrate rules in the hypothesis space by biasing the NN to represent logical rules. This allows for a direct end-to-end learnable integration of logical rules in NN which to the best of our knowledge has not been presented previously.

Another branch in the literature is the development of networks structures for logic testing with application in software testing (Evans et al., 2018). In Allamanis et al. (2017) continuous semantic representations of symbolic expressions are learned using tree based NN which represent logic

relations. Evans et al. (2018) investigate the ability of NN to model and understand logic entailments. Selsam et al. (2019) propose to learn solvers for propositional satisfiability using NN. However, both network architecture as well as application domain, i.e. testing logical terms vs. training logical relations, considerably differ from the NLRL. We leave the question whether NLRL can also be applied to these problems for future research.

A different branch of research related are binarized neural networks where weights and activations are binary variables Hubara et al. (2016) which particularly allows for implementation on resource constraint systems which is their main focus. However, modeling logic relations has not been reported for binary neural networks. The proposed NLRL share some common ideas with neural arithmetic logic units proposed in Trask et al. (2018). Particularly, the set up of end-to-end training is similar for both architectures. However, NALUs are designed for basic arithmetic operation like addition and multiplication while we focus on representation of propositional formulas which are not representable with NALUs.

Due to the rule structure and the input activations which can be interpreted as membership values, NLRL have some relations to fuzzy systems. Especially, in the field of neuro-fuzzy systems Shihabudheen and Pillai (2018), various architectures combining fuzzy logic and neural networks have been proposed where, however, most of the architectures are based on single layer radial basis function networks. Furthermore, training of these networks is restricted to the parameters of the input activations and the rule weighting while rule generation is done via heuristics Mitra and Hayashi (2000). In contrast, we propose an end-to-end trainable network architecture where all components of the rule are trainable which has not been done before in the neuro-fuzzy context.

4 Experiments

We present preliminary experiments on a given set of elementary logic and arithmetic operations and leave further experiments on e.g. classification tasks for future work. To this end we consider two input variables x and y and perform the basic logic operations AND , OR , NOT and XOR , as well as multiplication, addition, identity operation and constant functions, see Tab. 2. The data set

Function	$CS = 2$, 2-2-2-2-10, AND-NONEG	$CS = 4$, 2-4-4-10, AND-NONEG	$CS = 6$, 2-6-6-10, AND-OR	$CS = 8$, 2-8-8-8-10, AND-NONEG	$CS = 10$, 2-10-10-10, AND-NEG
$x/2 + y/2$	71.9%	100%	100%	100%	99.7%
$x \cdot \neg y$	99.8%	100%	93.2%	100%	100%
$x AND y$	93.7%	100%	99.3%	100%	99.6%
$x OR y$	57.0%	100%	97.1%	99.7%	98.4%
$x XOR y$	32.6%	78.8%	88.1%	80.3%	75.0%
x	90.2%	100%	91.1%	99.9%	100%
y	100%	100%	100%	100%	100%
$\neg x$	94.3%	100%	99.9%	100%	100%
$\neg y$	100%	100%	100%	100%	100%
0.7	100%	100%	100%	100%	100%
Overall Acc.	83.95%	97.88%	96.87%	97.99%	97.27%

Table 1: Individual scores for the considered logical and arithmetic operations with different network architectures. For each CS, only the best performing network is reported.

is generated by sampling random variables x and y in the interval $[0, 1]$ and evaluating the above functions such that 10 output values, one per function, are obtained. We sample a data set of size 100000, where 10000 are hold out for testing.

We compare three different NLRL structures on the data set, each with input layer of size two, two or three NLRL respectively, output layer of size 10 and varying numbers of neurons per layer. First, we employ NLRL with inbuilt AND and OR-relations (termed AND-OR) and corresponding output gating. Second, we employ NLRL in negation normal form, i.e. using AND-relations and negation at input and output only (termed AND-NEG). Third, we remove the negation gating of the output as this creates some redundancy of the negation operation when NLRL are stacked (termed AND-NONEG). In fact, the output negation and subsequent input negation can both change their weights without

Architecture	AND-OR	AND-NEG	AND-NONEG
2-2-2-10	69.54%	73.80%	83.86%
2-2-2-2-10	75.59%	75.42%	83.95%
2-4-4-10	97.77%	90.67%	97.88%
2-4-4-4-10	87.36%	90.19%	97.12%
2-6-6-10	96.87%	93.92%	94.15%
2-6-6-6-10	96.56%	90.95%	93.87%
2-8-8-10	96.95%	96.51%	94.40%
2-8-8-8-10	97.05%	93.16%	97.99%
2-10-10-10	96.91%	97.27%	94.28%
2-10-10-10-10	95.42%	69.23%	94.43%

Table 2: Overall accuracy for the different network architectures, connection sizes and number of layers.

affecting the results. We test with different numbers of neurons, also termed connection size (CS), per logic rule layer, namely with CS of 2, 4, 6, 8 and 10. Hence the network size is 2-CS-(CS)-10. We employ a learning rate of 10 with a minibatch size of 20 and put $\epsilon = 10^{-5}$ for numerical stability. Neither batch normalization nor dropout is used. The networks are initialized randomly by uniformly sampling from the interval $[-0.5, 0.5]$. The results of the network training are presented in Fig. 2. Note, that we employ early-stopping which is why some curves suddenly stop before the maximum iteration.

As can be seen in the results, the AND-OR network architectures consistently converges earlier to optimal results than networks without the option to represent OR-relations. However this comes with the cost of higher computation times as shown later. This is obviously due to the better representation abilities of these networks. Thus, networks with only AND-operations (AND-NEG and AND-NONEG) have to model OR-relations by orchestrating negation and AND-relations accordingly which requires more training. Among the networks solely using the AND-operation, we can infer that networks without negation gating (AND-NONEG) perform better than with negation (AND-NEG). Particularly, the convergence speed is drastically reduced, especially with higher network sizes. As discussed these networks have inbuilt redundancy in the negation operation, if multiple layers are stacked. This makes training more difficult especially as a cyclic behavior during training can occur.

In general, networks with three layers converge slower compared to 2-layer architectures which is not surprising due to the higher numbers of parameters. The final results are better compared to 2-layer architectures. By considering the networks size, which is represented mainly by the connection size (CS) between subsequent layers, we can see that the performance increases with increasing CS until a saturation occurs at CS 8. Increasing the CS further does not increase the performance indicating that the necessary network size has been reached.

Considering the course of the loss function over iterations, we observe a stepwise behavior. This is caused due to the combinatorical nature of the optimization problem such that a reduction of the loss is achieved if particular combination of boolean parameters converge to specific values. This is most likely amplified due to the derivative of the sigmoid function, enabling big parameter changes in a region around zero. Hence, the steps can be related to individual rules having converged.

Fig. 3 shows the learned surfaces of the logic and arithmetic relations to be trained for one of the best performing networks to give a qualitative representation of the learned manifold, which underlines the performance of the network on the test data set.

Finally, we compare the different network architectures concerning their computation times in Fig.4. As already stated, the AND-OR net has a considerable higher computation times compared to the AND-NEG and AND-NONEG networks.

5 Conclusion

We presented neural logic rule layers which are able to represent arbitrary logical relations using propositional logic and negation gating. The proposed network layer is generally applicable and can be used in arbitrary network architectures. Furthermore, the proposed architecture allows for

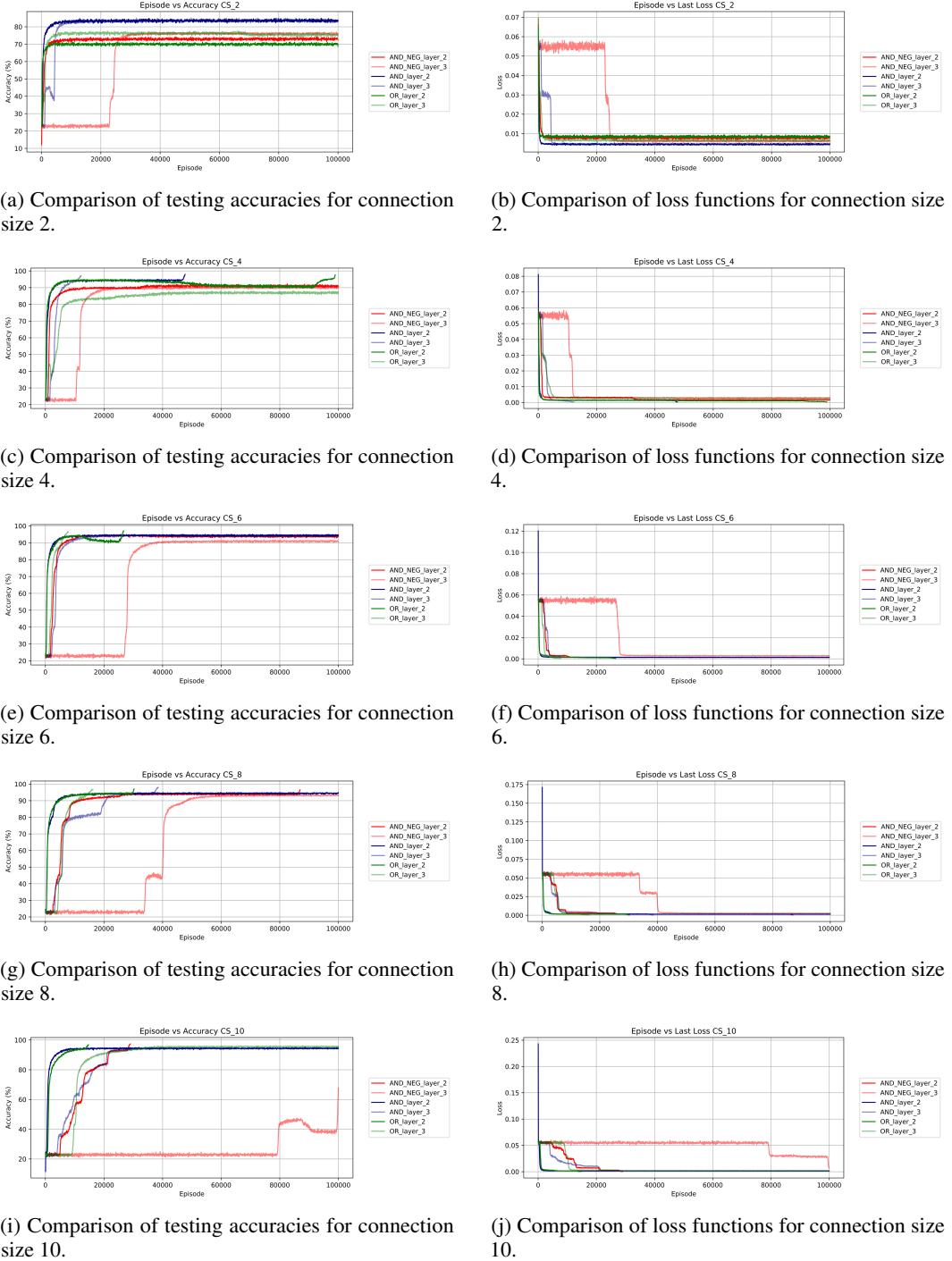


Figure 2: Comparison of testing accuracy and loss function for different connection sizes and network architectures.

a direct incorporation of available human knowledge in form of predefined rules. NLRL have some nice properties, i.e. they are able to not only represent logical rules but also certain types of nonlinear functions and share some relations to fuzzy logic. In fact, depending on the chosen input activation function which can be considered as membership functions in the context of fuzzy logic, the architecture allows for an end-to-end training of the rules parameter, the logic relations and the number of active rules. The results applied on some synthetic data sets reveal the applicability of the proposed architecture.

In future research we will apply and test NLRL on various data sets, including image classification. Furthermore, we will explore the incorporation of NLRL in recurrent neural network architectures. This would result in RNN structures representing finite state automata which is especially applicable for data analytics and reinforcement learning in discrete environments like discrete production systems.

References

- A. Trask, F. Hill, S. Reed, J. Rae, C. Dyer and P. Blunsom, “Neural Arithmetic Logic Units,” *Proc. of 32nd Conference on Neural Information Processing Systems*, 2018
- K.V. Shihabudheen and G.N. Pillai, “Recent advances in neuro-fuzzy system: A survey,” *Knowledge-Based Systems*, 152, pp. 136-162, 2018
- S. Mitra, and Y. Hayashi, “Neuro-Fuzzy Rule Generation: Survey in Soft Computing Framework,” *IEEE Transactions on Neural Networks*, 11(3), pp. 748-768, 2000
- L. von Rueden, S. Mayer, J. Garske, C. Bauckhage, and J. Schuecker, “Informed Machine Learning - Towards a Taxonomy of Explicit Integration of Knowledge into Machine Learning,” *arXiv:1903.12394v1*, 2019
- R. Evans, D. Saxton, D. Amos, P. Kohli, and E. Grefenstette, “Can Neural Networks Understand Logical Entailment?,” *Proc. of 2018th International Conference on Learning Representations*, 2018
- D. Selsam, M. Lamm, B. Bunz, P. Liang, D.L. Dill, and L. de Moura “Learning a SAT solver from single-bit supervision,” *Proc. of 2019th International Conference on Learning Representations*, 2019
- M. Allamanis, P. Chanthirasegaran, P. Kohli, and C. Sutton “Learning Continuous Semantic Representations of Symbolic Expressions,” *Proc. of the 34th International Conference on Machine Learning*, 2017
- Z. Hu, X. Ma, Z. Liu, E. Hovy, and E.P. Xing “Harnessing Deep Neural Networks with Logic Rules,” *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016
- M. Diligenti, M. Gori, and C. Sacca “Semantic-based regularization for learning and inference,” *Artificial Intelligence*, 244, pp. 143-165, 2017
- M. Diligenti, S. Roychowdhury, and M. Gori “Integrating prior knowledge into deep learning,” *Proc. of the 16th IEEE International Conference on Machine Learning and Applications*, 2017
- M. DesJardins, and D.F. Gordon “Evaluation and selection of biases in machine learning,” *Machine Learning Journal*, 5, pp. 1-17, 1995
- A.J. Ratner, C.M. De Sa, S. Wu, D. Selsam, and C. Ré “Data programming: Creating large training sets, quickly,” *Advances in Neural Information Processing Systems*, 2016
- M. Sachan, K.A. Dubey, T.M. Mitchell, D. Roth, and E.P. Xing “Learning pipelines with limited data and domain knowledge: A study in parsing physics problems,” *Advances in Neural Information Processing Systems*, 2018
- M. Richardson and P. Domingos “Markov logic networks,” *Machine Learning*, 62(1-2), pp. 107-136, 2006
- A.J.A. Robinson and A. Voronkov “Handbook of Automated Reasoning,” *Elsevier*, 2001
- I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio “Binarized Neural Networks,” *Advances in Neural Information Processing Systems*, 2016

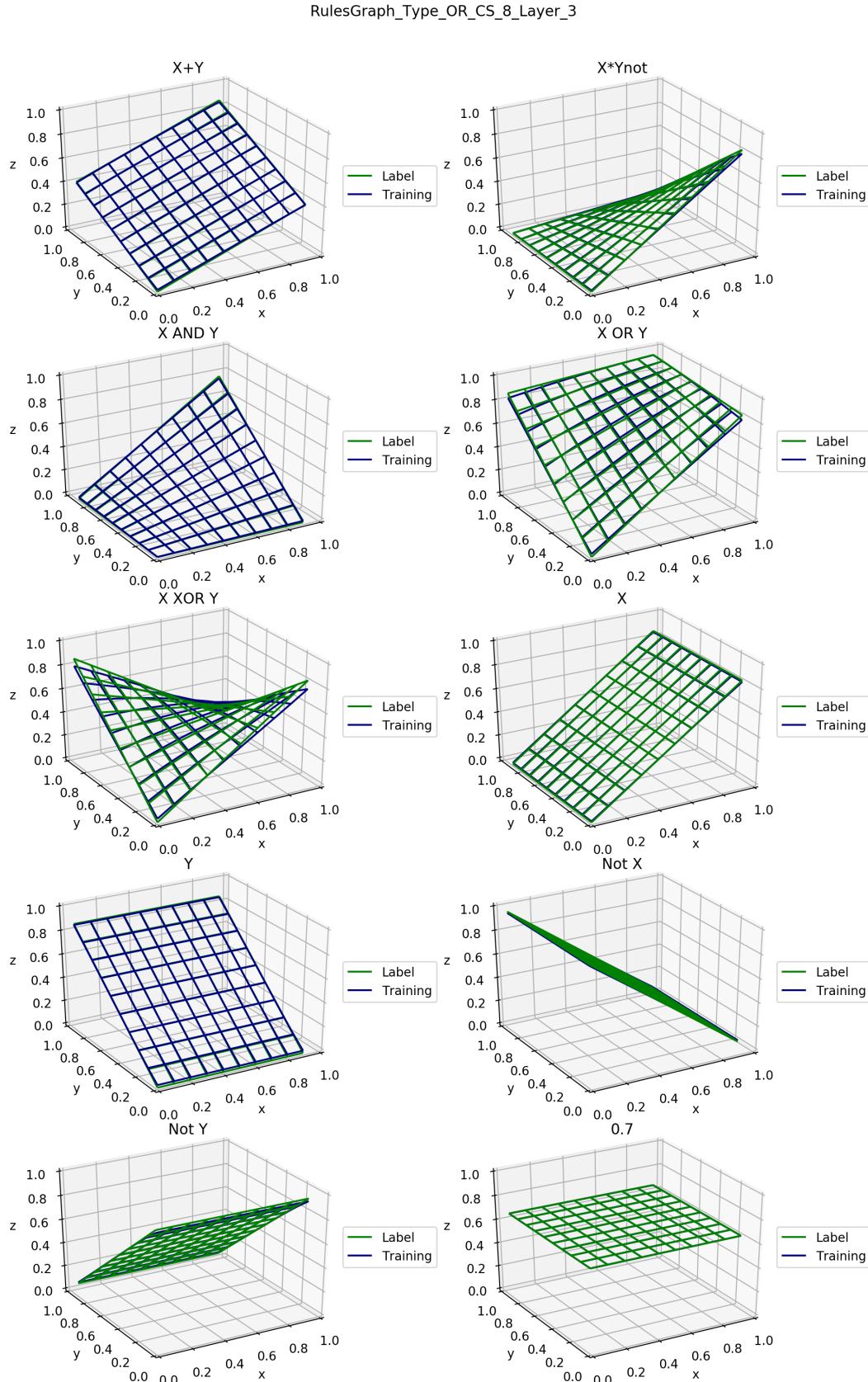


Figure 3: Illustrations of the learned surfaces representing the logic and arithmetic relations for the network architecture 2-8-8-10.

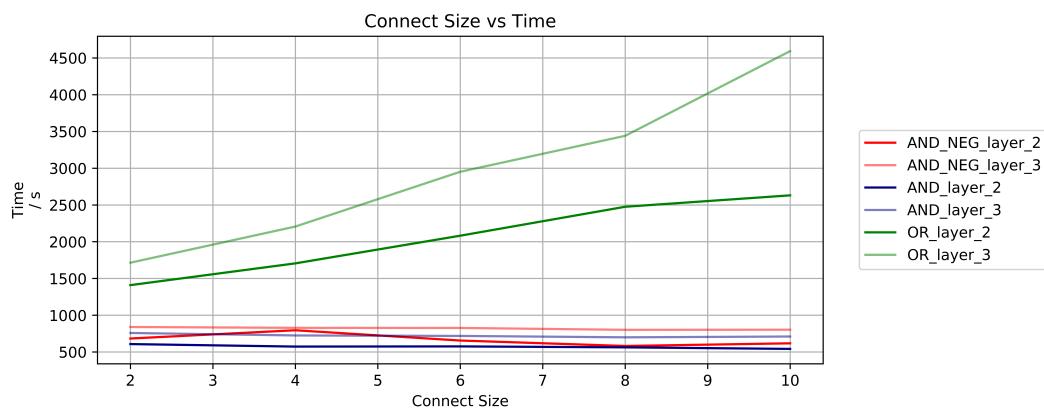


Figure 4: Training times for the various architectures depending on the connection size.