

LEARNING ARBITRARY LOGICAL FORMULA AS A SPARSE NEURAL NETWORK MODULE

Anonymous authors

Paper under double-blind review

ABSTRACT

NeSy (Neuro-Symbolic) predictors are hybrid models composed of symbolic predictive models chained after neural networks. Most existing NeSy predictors require either given symbolic knowledge or iterative training. DSL (Deep Symbolic Learning) is the first NeSy predictor that supports fully end-to-end training from scratch, but it learns a look-up table rather than arbitrary programs or formulas. We propose the Logical Formula Learner framework, a general framework of network modules that explicitly equate a logical formula after convergence. We then propose 3 novel designs within the LFL framework with different levels of combinatorial search freedom: LFL-Type1 learns arbitrary logical formula, LFL-Type2 learns a look-up table, and LFL-Type3 has combinatorial search freedom between them. LFL-Type1 and LFL-Type2 show improvements over previous designs, and all three types can be wrapped into NeSy predictors. To our knowledge, LFL-Type1-based NeSy predictor is the first NeSy predictor that supports fully end-to-end training from scratch and explicitly learns arbitrary logical formulas.¹

1 INTRODUCTION

In a special lecture of NIPS 2019, Yoshua Bengio suggested that studies in deep learning should expand from System 1 to System 2 (Bengio, 2019), where System 1 means the intuitive, black-box parts and System 2 means the symbolic, white-box parts of either human or artificial intelligence. Works in multiple research directions can be seen as attempting to approach System 2 intelligence, such as those combining deep learning with Symbolic Regression (SR) (Makke & Chawla, 2024) or Inductive Logic Programming (ILP) (Zhang et al., 2023) methods, while the term Neuro-Symbolic (NeSy) refers to the general research field of combining neural systems and symbolic reasoning (Marra et al., 2024).

A subset of NeSy methods is NeSy predictors defined in Marconato et al. (2024), members of which present hybrid models composed of symbolic predictive models chained after neural networks. Some NeSy predictors combine neural networks with a given symbolic program or knowledge base so that the neural network can be trained by symbolic constraints (Manhaeve et al., 2018; Huang et al., 2021; Badreddine et al., 2022; Winters et al., 2022; Yang et al., 2023). Some other NeSy predictors attempt to learn the symbolic predictor jointly with the neural networks, most of them requiring iteratively training the symbolic predictor and the neural networks while fixing the other (Duan et al., 2022; Cunnington et al., 2022; Liu et al., 2023); some of them also require pre-training the neural networks beforehand. Then DSL (Deep Symbolic Learning) (Daniele et al., 2022) presents a fully differentiable NeSy predictor that allows for end-to-end training of a differentiable logic module chained after neural networks, both from scratch. The major limitation of DSL is that its differentiable logic module equates a look-up table that maps each of the input concepts' cartesian product to one of the possible output symbols, while many other NeSy predictors learn arbitrary logical circuits or programs.

The idea of designing a differentiable module that becomes equivalent to a symbolic expression is also seen in a line of research starting from EQL (Equation Learner) (Martius & Lampert, 2016), a differentiable module designed for SR. An EQL network is a neural network module that equates an arithmetic formula after convergence, archived by introducing diverse activation functions and

¹Code available at: <https://anonymous.4open.science/r/logical-formula-learner-693B>

054
055
056
057
058
059
060
061
062
063
064
065
066
067
068
069
070
071
072
073
074
075
076
077
078
079
080
081
082
083
084
085
086
087
088
089
090
091
092
093
094
095
096
097
098
099
100
101
102
103
104
105
106
107

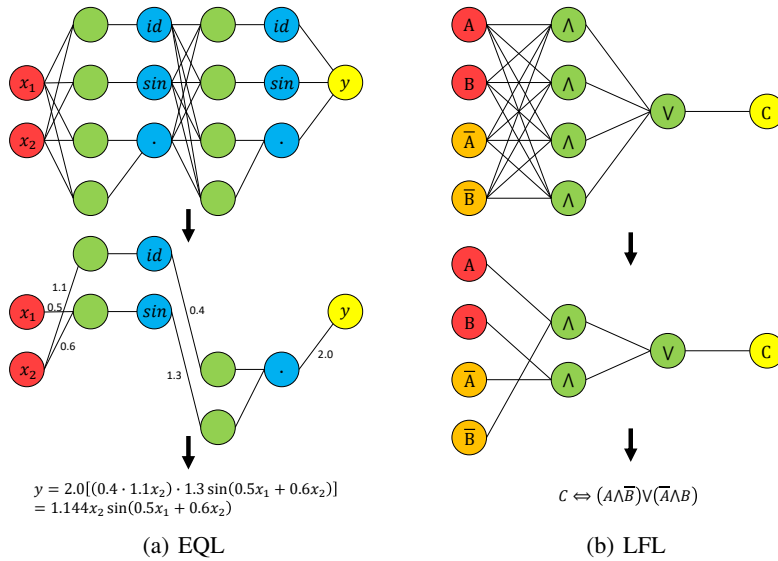


Figure 1: An intuitive illustration of how an EQL or LFL converges into symbolic expressions. In all network architecture figures of this paper, red-colored objects represent input values, yellow ones represent output values, orange ones represent intermediate values, green ones represent trainable modules or neurons, and blue ones represent transformations that contain no trainable parameters.

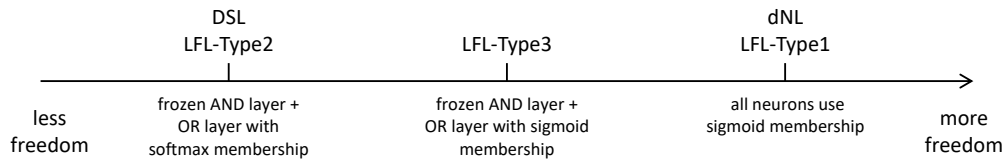


Figure 2: Comparison of LFL variations in terms of freedom of their combinatorial search spaces.

sparsity constraints into an MLP network. Another method with a similar philosophy is dNL (differentiable Neural Logic Network) (Payani & Fekri, 2019a). A dNL network is composed of logical neurons designed with Product t-norm that express AND or OR relationships and thus can be equivalent to any arbitrary logical formula, experimented mainly on ILP tasks.

Inspired by them, in this work we propose a general framework of differentiable modules that equate logical formulas after convergence, named Logical Formula Learner (LFL). An LFL network uses customized neurons that relax binary logic symbols with fuzzy logic operations and converges into a sparse network equating a relatively simple logical formula. Figure 1 shows an intuitive comparison between EQL and LFL.

Both DSL and dNL fit into the LFL framework. Apart from them, we propose three novel designs within the LFL framework with different levels of combinatorial search freedom (Figure 2): LFL-Type1 learns arbitrary logical formulas like dNL, LFL-Type2 learns a look-up table like DSL, and LFL-Type3 has combinatorial search freedom between them. LFL-Type1 works a little better than dNL on learning logical formulas from binary data when the number of hidden neurons is limited. LFL-Type2 and LFL-Type3 work as well as DSL’s logic module in NeSy predictors with better consistency between training and inference behaviours. Although directly using LFL-Type1 in NeSy predictors makes the network struggle to converge, we tackle this problem by adding an MLP as gradient shortcut. This, to our knowledge, results in the first NeSy predictor that satisfies: (1) Being end-to-end differentiable; (2) Training all modules from scratch and achieving joint convergence in

a single run; (3) Explicitly learning arbitrary logical formula (within limited complexity) with its symbolic module.

2 THE LOGICAL FORMULA LEARNER FRAMEWORK

2.1 RELAXING BINARY LOGICAL NEURONS INTO FUZZY, DIFFERENTIABLE ONES

It is straightforward to define binary neurons that describe the logical AND, OR or NOT relationships:

$$f_{binary_AND}(\mathbf{x}^n) = \bigwedge_i (\overline{m_i} \vee x_i) \quad (1)$$

$$f_{binary_OR}(\mathbf{x}^n) = \bigvee_i (m_i \wedge x_i) \quad (2)$$

$$f_{binary_NOT}(x) = \overline{x} \quad (3)$$

where f means the neuron’s output, $x_i \in \{0, 1\}$ means the neuron’s i th input, and $m_i \in \{0, 1\}$ means the ”membership” of the input which decides whether the input is a member of the AND/OR operation. The NOT operation applies on a single input.

To make our logical neurons differentiable and trainable, we need to replace the binary logical operations with fuzzy ones and the binary memberships with continuous values in range $[0, 1]$:

$$f_{AND}(\mathbf{x}^n) = \bigotimes_i ((1 - g(w_i)) \oplus x_i) = \bigotimes_i (1 - g(w_i) \otimes (1 - x_i)) \quad (4)$$

$$f_{OR}(\mathbf{x}^n) = \bigoplus_i (g(w_i) \otimes x_i) = 1 - \bigotimes_i (1 - g(w_i) \otimes x_i) \quad (5)$$

$$f_{NOT}(x) = 1 - x \quad (6)$$

where f means the neuron’s output, $x_i \in [0, 1]$ means the neuron’s i th input, \otimes and \bigotimes mean chosen differentiable fuzzy t-norm operations, \oplus and \bigoplus mean chosen differentiable fuzzy t-conorm operations, w_i means the control parameters for the membership value, and g means a chosen differentiable function that maps w_i into $[0, 1]$. An LFL network can be constructed by arbitrarily combining these differentiable neurons. The selected g may produce soft, stochastic membership values during training. During inference, we set the memberships to binary, deterministic values in $\{0, 1\}$ according to the trained parameters w_i so that the LFL module strictly equates a fuzzy logical formula.

2.2 HOW EXISTING METHODS FIT INTO THE LFL FRAMEWORK

2.2.1 DIFFERENTIABLE NEURAL LOGIC NETWORK (DNL)

The logical neurons proposed in dNL can be obtained by substituting Product t-norm and the corresponding t-conorm as chosen fuzzy logic operations and the sigmoid function σ as g into equations 4 and 5²:

$$f_{dNL_AND}(\mathbf{x}^n) = \prod_i (1 - \sigma(w_i) (1 - x_i)) \quad (7)$$

$$f_{dNL_OR}(\mathbf{x}^n) = 1 - \prod_i (1 - \sigma(w_i) x_i) \quad (8)$$

A dNL network can thus be constructed by arbitrarily combining the AND/OR neurons defined above and the NOT neurons defined in 6.

²The original design in dNL multiplies w_i with a hyperparameter c , i.e. $g(w_i) = \sigma(cw_i)$. The hyperparameter c seems redundant since scaling the trainable control parameters is equivalent to scaling the learning rate.

2.2.2 DEEP SYMBOLIC LEARNING (DSL)

As a fully differentiable NeSy predictor, a DSL network consists of black-box neural classifiers that map high-dimensional input to symbolic representations and a differentiable logic module that equates a look-up table after convergence, indicating that the logic module should also fit into the LFL framework. In this subsection, we describe an LFL module that is equivalent to the original definition of DSL’s logic module.

Say there are multiple neural classifiers with their number of classes denoted as $\{n_1, n_2, n_3, \dots\}$ and the j th unnormalized prediction of the i th classifier as $N_{ij} \in \mathbb{R}$. Each classifier’s predictions are sparsified by ϵ -greedy policy:

$$x_k = \begin{cases} \text{softmax}_j(N_{ij}), & \text{if } j \text{ is the chosen symbol of classifier } i \\ 0, & \text{otherwise} \end{cases} \quad (9)$$

where $k \in [1, \sum_i n_i]$, softmax_j means applying softmax on the j dimension, and $x_k \in (0, 1)$ is the LFL module’s k th input. For each classifier, its predicted class is chosen with probability $1 - \epsilon_1$, and a random class is chosen with probability ϵ_1 .

The LFL module uses Gödel t-norm and t-conorm. Its first layer consists of AND neurons with memberships frozen such that each AND neuron represents a member of the cartesian product of the module’s input concepts:

$$h_l = \min_k (1 - \min(m_{kl}, 1 - x_k)) \quad (10)$$

$$m_{kl} = \begin{cases} 1, & \text{if the } k\text{th symbol is a member of the } l\text{th cartesian product of all class symbols} \\ 0, & \text{otherwise} \end{cases} \quad (11)$$

where $l \in [1, \prod_i n_i]$, m_{kl} is the l th AND neuron’s k th membership value, and h_l is the neuron’s output.

The second layer consists of OR neurons with trainable memberships generated with softmax and ϵ -greedy policy, producing the final class prediction from n_p classes:

$$y_p = \max_l (\min(m_{lp}, h_l)) \quad (12)$$

$$m_{lp} = \begin{cases} \text{softmax}_p(w_{lp}), & \text{if } p \text{ is the chosen symbol of input } l \\ 0, & \text{otherwise} \end{cases} \quad (13)$$

where $p \in [1, n_p]$, m_{lp} is the p th OR neuron’s l th membership value, w_{lp} is the trainable control parameter for m_{lp} , softmax_p means applying softmax on the p dimension, and y_p is the neuron’s output. For each input h_l , its maximal m_{lp} is chosen with probability $1 - \epsilon_2$, and a random one of the memberships is chosen with probability ϵ_2 .

During each individual input sample’s forward pass, the above design assures that only one value in x , h or y is non-zero, so that the original definition of DSL could focus only on the non-zero values. The non-zero value of y then defines the network’s predicted class and truth value. Furthermore, the module can predict multiple class labels by concatenating outputs of multiple OR layers defined above.

2.3 OUR NEW DESIGNS

One difference between dNL and DSL is that the latter uses noisy input symbols and memberships with Gödel t-(co)norm, which may allow the network to explore more of the possible combinatorial search space. We also notice that the Concrete distribution (Maddison et al., 2016) provides an elegant way of adding noise to soft symbolic values. So in this subsection, we introduce three novel designs within the LFL framework, all of which utilize Gödel t-(co)norm and the Concrete distribution.

2.3.1 LFL-TYPE1: LEARNING ARBITRARY LOGICAL FORMULA LIKE DNL, BUT WITH GÖDEL T-(CO)NORM AND NOISY WEIGHTS

Substituting Gödel t-(co)norm as chosen fuzzy operations and the Binary Concrete distribution as g into 4 and 5, we get:

$$f_{LFLAND}(\mathbf{x}^n) = \min_i (1 - \min(\sigma(w_i + \eta(\log(u_i) - \log(1 - u_i))), 1 - x_i)) \quad (14)$$

$$f_{LFLOR}(\mathbf{x}^n) = \max_i (\min(\sigma(w_i + \eta(\log(u_i) - \log(1 - u_i))), x_i)) \quad (15)$$

where u_i is sampled independently from uniform distribution in $(0, 1)$ and η is a hyperparameter controlling the noise scale³. An LFL-Type1 network can thus be constructed by arbitrarily combining them and the NOT neurons 6.

2.3.2 LFL-TYPE2: LEARNING A LOOK-UP TABLE LIKE DSL, BUT WITH DENSE, NOISY SYMBOLS AND WEIGHTS

DSL used ϵ -greedy policy to produce sparse, noisy symbolic input and predictions in 9 and 13. In LFL-Type2 we replace them with dense, noisy values generated by the Concrete distribution:

$$x_k = \text{softmax}_j (N_{ij} - \eta_0 \log(-\log(u_k))) \quad (16)$$

$$m_{lp} = \text{softmax}_p (w_{lp} - \eta_1 \log(-\log(u_{lp}))) \quad (17)$$

where u_k and u_{lp} are sampled independently from uniform distribution in $(0, 1)$, η_0 and η_1 are hyperparameters controlling the noise scales. 16 is also used when wrapping LFL-Type1 or LFL-Type3 into NeSy predictors.

2.3.3 LFL-TYPE3: USING FROZEN AND LAYER LIKE DSL, BUT WITH INDEPENDENT WEIGHTS FOR THE OR LAYER

In LFL-Type1 all memberships are independently trained, while LFL-Type2 consists of a frozen, predefined layer of AND neurons and a layer of OR neurons where memberships correlate through softmax. Then it's natural to experiment with a third design, in which LFL-Type2's OR layer 17 is replaced by a layer of LFL-Type1's OR neurons where all memberships are freely trainable:

$$m_{lp} = \sigma(w_{lp} + \eta_1(\log(u_{lp}) - \log(1 - u_{lp}))) \quad (18)$$

2.4 NETWORK ARCHITECTURE FOR LFL-BASED DIFFERENTIABLE NESY PREDICTORS

In this subsection, we describe the network architectures that wrap LFL modules into NeSy predictors for direct or recurrent NeSy tasks defined in DSL, such as MNIST Sum and MNIST Multi-digit Sum. How these architectures are applied to the MNIST Arithmetic tasks is shown in Figure 3.

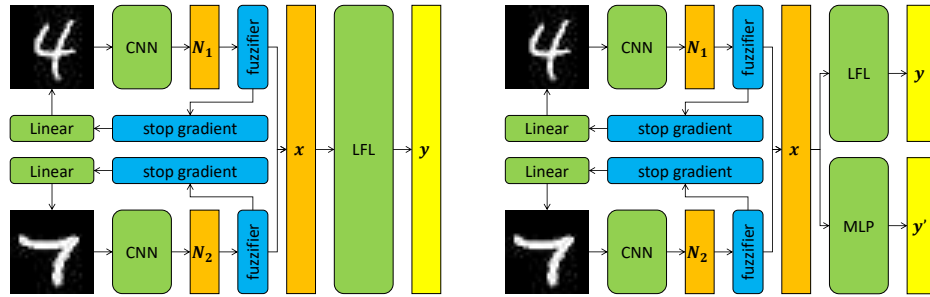
2.4.1 DSL'S VANILLA ARCHITECTURE

In DSL, the differentiable logic module is chained after CNN classifiers so that the whole network works as a NeSy predictor. In this work, we add linear reconstruction models that allow us to visualize the symbols learned by CNN classifiers, with stop gradient operations so that they don't affect other network components during training, as shown in Figure 3(a).

For recurrent NeSy tasks, recurrent values (such as c) are passed forward through the recurrent blocks, as shown in Figure 3(c).

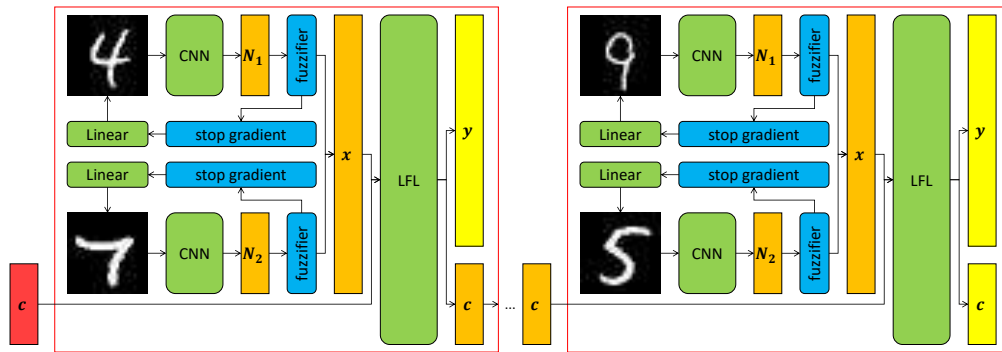
³The original paper (Maddison et al., 2016) wrote the Binary Concrete distribution in a slightly different form, where $\log \alpha$ replaced the control parameter w_i and a "temperature" parameter λ that divides both the noise and the control parameter replaced η . The same difference applies to the softmax-like Concrete distribution used in section 2.3.2.

270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323

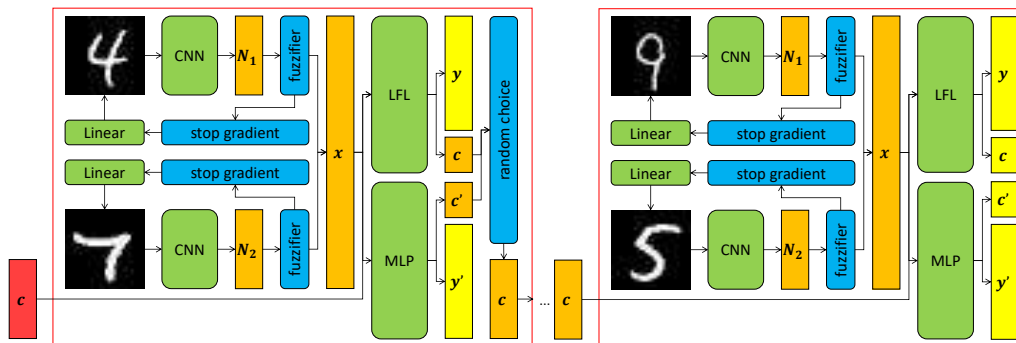


(a) Vanilla MNIST-sum

(b) MNIST-sum with MLP gradient shortcut



(c) Vanilla MNIST Multi-digit Sum



(d) MNIST Multi-digit Sum with MLP gradient shortcut

Figure 3: Network architectures for MNIST Arithmetic tasks, where N means real-valued CNN predictions, "fuzzifier" means softmax-like transformation such as 9 or 16, x and y mean the LFL module's input and output symbols, c means the carry symbols in MNIST Multi-digit Sum, "stop gradient" means detaching the tensor to block gradient backpropagation, "random choice" means randomly choosing one of the input tensors as output for each sample, and each red square contains a recurrent block for MNIST Multi-digit Sum. Parameters are shared among CNNs or linear reconstruction layers. Similar to the membership values in LFL, the fuzzifier layers also output binary, deterministic symbols during inference.

2.4.2 OUR NEW TRICK: ADDING AN MLP AS GRADIENT SHORTCUT

In our early experiments, we found that using freely trainable LFL variations like dNL or LFL-Type1 in the vanilla architecture results in a network that struggles to converge. A similar convergence problem has been encountered in training deep CNNs, where it was resolved by adding "shortcut" connections that allow gradients to back-propagate through (He et al., 2016). Inspired by this, we add an MLP that shares the same input and output with the LFL module so that gradients can back-propagate through the MLP, allowing the whole network to converge into an optimal or sub-optimal solution, as shown in Figure 3(b). The MLP uses sigmoid activation in its output layer for consistency.

For recurrent NeSy tasks, the LFL and MLP predict two different versions of the recurrent values \mathbf{c} , as shown in Figure 3(d). During training, each of them is randomly selected at sample level with probability 0.5 for the next recurrence; during inference, the LFL's prediction is used for the next recurrence.

2.5 LOSS FUNCTIONS AND REGULARIZATIONS

For all network architectures described above, the LFL's supervision loss \mathcal{L}_{sup} is measured by the binary cross-entropy loss. For DSL, the loss applies only on its non-zero prediction values.

Another loss term is an L2 reconstruction loss \mathcal{L}_{rec} that trains the linear reconstruction layer. The stop gradient operation stops the reconstruction loss from affecting other modules.

Similar to EQL, we need sparsity constraints to make the LFL modules learn logical formulas in their relatively simple forms. In DSL and LFL-Type2 all trainable memberships are generated with softmax-like transformations 13 17 so that they are always sparse. For memberships generated with sigmoid-like transformations 7 8 14 15 18, we apply a sparsity constraint loss:

$$\mathcal{L}_{reg} = \sum_k \frac{1}{n_k} \sum_{ij} \sigma(w_{ijk}) \quad (19)$$

where $\sigma(w_{ijk})$ is the median membership of the i th input of the j th neuron of the k th layer with sigmoid-like memberships, and n_k means the number of such memberships in the k th layer.

For network architectures with the MLP gradient shortcut, another binary cross-entropy loss \mathcal{L}_{MLP} applies on the MLP's prediction. In these networks we also constraint the distribution of LFLs' input symbol \mathbf{x} such that the average of \mathbf{x} in each batch is close to even distribution over each neural classifier's predicted labels:

$$\mathcal{L}_{label} = \sum_i \frac{1}{n_i} \sum_j \text{BCE}\left(\frac{1}{n_i}, \frac{1}{b} \sum_k x_{ijk}\right) \quad (20)$$

where BCE means the binary cross-entropy loss, x_{ijk} means the normalized LFL input corresponding to the j th class predicted by the i th classifier from the k th sample in a batch, b means the batch size, and n_i means classifier i 's number of predicted classes.

So the overall loss function used in our experiments is

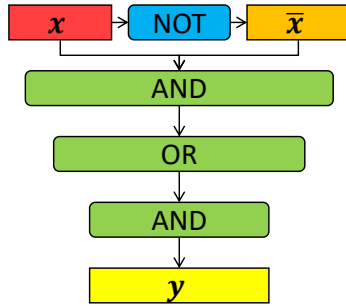
$$\mathcal{L} = \mathcal{L}_{sup} + \mathcal{L}_{rec} + \beta_1 \mathcal{L}_{reg} + \beta_2 \mathcal{L}_{MLP} + \beta_3 \mathcal{L}_{label} \quad (21)$$

for the network architectures with MLP gradient shortcut. Only \mathcal{L}_{sup} , \mathcal{L}_{rec} and \mathcal{L}_{reg} are used for those without the MLP gradient shortcut, and only \mathcal{L}_{sup} and \mathcal{L}_{rec} are used for those with DSL or LFL-Type2. β_1 , β_2 and β_3 are hyperparameters adjusting the loss terms' weights.

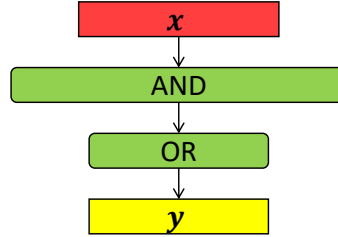
3 EXPERIMENTS

In previous works, dNL has been experimented on ILP tasks that require learning logical formulas from binary training data, and DSL has been experimented on MNIST Arithmetic tasks. We

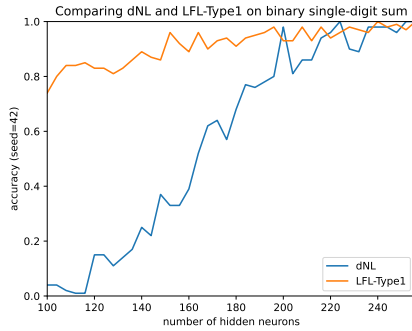
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431



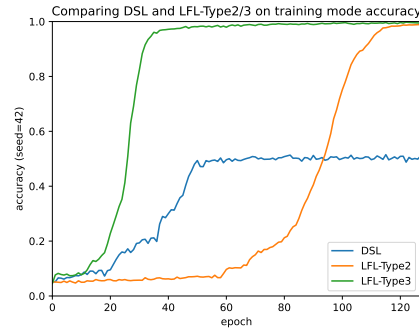
(a) Network for Experiment 3.1.1.



(b) Network for Experiment 3.1.2.



(c) Result of experiment 3.1.2.



(d) Result of experiment 3.2.

Figure 4: Figures related to the experiments.

Table 1: Results of MNIST Arithmetic experiments. **MLP** means whether MLP gradient shortcut is used, \mathcal{L}_{label} means whether \mathcal{L}_{label} is used, **Correct formula** means whether the LFL learns a correct formula. For MNIST Multi-digit Sum two accuracies are evaluated on test datasets with $n_{digit} = 3$ and $n_{digit} = 128$.

	MLP	\mathcal{L}_{label}	Task	Accuracy(%)	Correct formula
DSL	×	×	Sum	97.8	✓
LFL-Type2	×	×	Sum	98.1	✓
LFL-Type3	×	×	Sum	98.1	✓
	×	×	Multi-digit Sum	97.7/96.9	✓
LFL-Type1	✓	✓	Sum	98.3	✓
	×	✓	Sum	16.3	×
	✓	×	Sum	65.8	×
	✓	✓	Multi-digit Sum	98.5/98.1	✓
	×	✓	Multi-digit Sum	20.2/10.3	×
	✓	×	Multi-digit Sum	81.8/75.0	×

evaluate and compare LFL variations on these task types. Test set accuracies of MNIST Arithmetic experiments are shown in Table 1. Note that difference in accuracies among successful MNIST Arithmetic experiments doesn't really matter because mistakes are caused only by CNN classifiers when the LFL learns a correct formula.

3.1 COMPARING DNL AND LFL-TYPE1 ON LEARNING LOGICAL FORMULA FROM BINARY DATA

Since dNL has been proven to work well on learning simple logical formulas with 2-layer network architectures (Payani & Fekri, 2019a), we evaluate dNL and LFL-Type1 on two possibly harder tasks: learning a 3-layer logical formula with NOT neurons and learning MNIST Sum’s logical formula with limited number of hidden neurons.

3.1.1 LEARNING A 3-LAYER LOGICAL FORMULA WITH NEGATION

The 3-layer logical formula to be learned has 8 input concepts and 2 outputs. The 8 input concepts and their negations form 16 input concepts, and the formula then requires at least 8 hidden neurons in an AND layer and 4 neurons in an OR layer to learn (Appendix B). Both dNL and LFL-Type1 learn it successfully with the architecture in Figure 4(a)

3.1.2 LEARNING MNIST SUM’S FORMULA WITH LIMITED NUMBER OF HIDDEN NEURONS

The MNIST Sum task requires learning a 2-layer logical formula with 20 inputs, 19 outputs, and 100 hidden concepts. Both dNL and LFL-Type1 can learn it from binary training data with 256 hidden neurons, but LFL-Type1 maintains higher accuracies when the number of hidden neurons reduces towards 100 (Figure 4(c)). The network architecture shown in Figure 4(b) is used for this task and the following experiments where LFL-Type1 is used in NeSy predictors.

3.2 COMPARING DSL AND LFL-TYPE2 ON MNIST SUM

LFL-Type2 has the same combinatorial search freedom as DSL’s logic module and converges as well as the latter on MNIST Sum. A minor limitation of DSL is that its ϵ -greedy search policy keeps producing random label choices with probability ϵ after convergence, creating a significant gap between training and inference behaviors. The limitation doesn’t exist in LFL-Type2’s Concrete noise distribution, as shown in Figure 4(d).

3.3 EXPERIMENTING LFL-TYPE3 AND LFL-TYPE1 ON MNIST SUM AND MULTI-DIGIT SUM

LFL-Type3 and LFL-Type1 have larger combinatorial search freedom than DSL, and we evaluate them on MNIST Sum and Multi-digit Sum. LFL-Type3 converges well on MNIST Sum and Multi-digit Sum using DSL’s vanilla network architecture, and it actually converges faster than DSL or LFL-Type2 with automatically tuned hyperparameters, as shown in Figure 4(d). LFL-Type1 also converges well with our new architecture that contains the MLP gradient shortcut. In MNIST Sum the formula learned by LFL-Type1 is the same as that of LFL-Type3. In MNIST Multi-digit Sum LFL-Type1 learns a formula different from LFL-Type3 for predicting the next carry, proving LFL-Type1’s ability to learn more arbitrary logical formulas in NeSy predictors. See Appendix D for the learned formulas.

3.4 ABLATION STUDY

Two additional tricks have been used in NeSy predictors with LFL-Type1: the MLP gradient shortcut and the classification label distribution loss 20. Removing either of them results in failure of joint convergence on MNIST-Sum and MNIST Multi-digit Sum.

4 RELATED WORKS

4.1 NEURO-SYMBOLIC PREDICTORS

The recent interest in NeSy predictors started with DeepProbLog (Manhaeve et al., 2018), in which programs written in ProbLog (De Raedt et al., 2007) are translated into differentiable arithmetic circuits connected with neural networks, allowing for end-to-end training. ABLSim (Huang et al., 2021) trains neural networks by minimizing the inconsistency between them and given background

486 knowledge. Logic Tensor Networks (Badreddine et al., 2022) encode symbolic knowledge into
 487 differentiable regularizations for training neural networks. DeepStochLog (Winters et al., 2022)
 488 extends DeepProbLog with stochastic definite clause grammars. NeurASP (Yang et al., 2023) im-
 489 proves pre-trained neural networks with given Answer Set Programs.

490 The above methods require given symbolic knowledge, while some other NeSy predictors require
 491 iterative training. DeepLogic (Duan et al., 2022) iteratively trains a non-differentiable Deep Logic
 492 Module with neural networks. NSIL (Cunnington et al., 2022) iteratively trains an ASP program
 493 with neural classifiers. NTOC (Liu et al., 2023) iteratively trains neural networks and symbolic rules,
 494 where the symbolic rules can be translated into fixed differentiable circuits. Then DSL (Daniele
 495 et al., 2022) is our most closely related work, in which a differentiable logic module that equates a
 496 look-up table and fits into the LFL framework is wrapped into a NeSy predictor.

497 498 499 4.2 DIFFERENTIABLE MODULES FOR SYMBOLIC REGRESSION

500 This work takes initial inspiration from EQL (Martius & Lampert, 2016). An EQL network equates
 501 an arithmetic expression after convergence in a way similar to LFL networks, originally proposed
 502 for Symbolic Regression. EQL⁺ (Sahoo et al., 2018) and GMEQL (Chen, 2020) explore different
 503 EQL variations, while OccamNet (Dugan et al., 2020) and KAN (Liu et al., 2024) extend the idea
 504 with more network designs. EQL’s differentiability allows people to connect it with CNNs for joint
 505 learning (Kim et al., 2020) and NeSy RL (Luo et al., 2024).

506 507 508 4.3 INTEGRATING NEURAL NETWORKS FOR INDUCTIVE LOGIC PROGRAMMING

509 As another closely related method, dNL fits into the LFL framework and has been experimented
 510 on ILP tasks (Payani & Fekri, 2019a). NLN (Payani & Fekri, 2019b) extends dNL with XOR
 511 neurons, and Payani & Fekri (2020) uses dNL to incorporate symbolic knowledge for RL. Works
 512 such as ∂ ILP (Evans & Grefenstette, 2018), NLIL (Yang & Song, 2019), and DLM (Zimmer et al.,
 513 2021) also integrate neural networks for ILP tasks. LNN (Riegel et al., 2020; Sen et al., 2022)
 514 proposes customized neurons with Łukasiewicz t-(co)norm for ILP tasks, but their neuron designs
 515 don’t strictly fit into the LFL framework.

516 517 518 5 CONCLUSION AND FUTURE WORK

519 In this work, we present the Logical Formula Learner framework and three novel designs within
 520 it. The LFL framework summarizes previous designs into a general framework of network modules
 521 that explicitly equate a logical formula after convergence. The proposed LFL-Type1 and LFL-Type2
 522 show improvements over previous designs, and LFL-Type3 works fine with combinatorial search
 523 freedom between the two. Furthermore, by wrapping LFL-Type1 into NeSy predictors with MLP
 524 gradient shortcut, we obtain the first end-to-end differentiable NeSy predictor that converges from
 525 scratch in one single run and explicitly learns an arbitrary logical formula.

526 Future directions based on this work include:

- 527
528
529 1. Experimenting with other possible neuron designs within the LFL framework, such as using
 530 other t-(co)norm choices or introducing annealing schedules for the noise scales.
- 531
532 2. Using logical neurons defined in this work in layers other than fully connected ones. For
 533 example, using them in convolutional layers may produce white-box CNNs with better
 534 interpretability or adversarial robustness.
- 535
536 3. Integrating LFL networks into other neural network applications, such as image or video
 537 caption, model-free RL, model-based RL, etc. Particularly, integrating LFLs into model-
 538 based RL agents may allow such agents to describe their internal representations with sound
 539 symbolic logic, opening up a new path toward reliable language generation and understand-
 ing.

REFERENCES

- 540
541
542 Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna:
543 A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM*
544 *SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.
- 545 Samy Badreddine, Artur d’Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor
546 networks. *Artificial Intelligence*, 303:103649, 2022.
- 547 Yoshua Bengio. From system 1 deep learning to system 2 deep learning. In *Neural Information*
548 *Processing Systems*, 2019.
- 549 Gang Chen. Learning symbolic expressions via gumbel-max equation learner networks. *arXiv*
550 *preprint arXiv:2012.06921*, 2020.
- 551 Daniel Cunnington, Mark Law, Jorge Lobo, and Alessandra Russo. Neuro-symbolic learning of
552 answer set programs from raw data. *arXiv preprint arXiv:2205.12735*, 2022.
- 553 Alessandro Daniele, Tommaso Campari, Sagar Malhotra, and Luciano Serafini. Deep symbolic
554 learning: Discovering symbols and rules from perceptions. *arXiv preprint arXiv:2208.11561*,
555 2022.
- 556 Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its
557 application in link discovery. In *IJCAI*, volume 7, pp. 2462–2467. Hyderabad, 2007.
- 558 Aaron Defazio and Samy Jelassi. A momentumized, adaptive, dual averaged gradient method.
559 *Journal of Machine Learning Research*, 23(144):1–34, 2022.
- 560 Xuguang Duan, Xin Wang, Peilin Zhao, Guangyao Shen, and Wenwu Zhu. Deeplogic: Joint learning
561 of neural perception and logical reasoning. *IEEE Transactions on Pattern Analysis and Machine*
562 *Intelligence*, 45(4):4321–4334, 2022.
- 563 Owen Dugan, Rumen Dangovski, Allan Costa, Samuel Kim, Pawan Goyal, Joseph Jacobson, and
564 Marin Soljačić. Occamnet: A fast neural model for symbolic regression at scale. *arXiv preprint*
565 *arXiv:2007.10784*, 2020.
- 566 Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of*
567 *Artificial Intelligence Research*, 61:1–64, 2018.
- 570 Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recog-
571 nition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp.
572 770–778, 2016.
- 573 Yu-Xuan Huang, Wang-Zhou Dai, Le-Wen Cai, Stephen H Muggleton, and Yuan Jiang. Fast ab-
574 ductive learning by similarity-based consistency optimization. *Advances in Neural Information*
575 *Processing Systems*, 34:26574–26584, 2021.
- 576 Samuel Kim, Peter Y Lu, Srijon Mukherjee, Michael Gilbert, Li Jing, Vladimir Čeperić, and Marin
577 Soljačić. Integration of neural network-based symbolic regression in deep learning for scientific
578 discovery. *IEEE transactions on neural networks and learning systems*, 32(9):4166–4177, 2020.
- 579 Anji Liu, Hongming Xu, Guy Van den Broeck, and Yitao Liang. Out-of-distribution generalization
580 by neural-symbolic joint training. In *Proceedings of the AAAI Conference on Artificial Intelli-*
581 *gence*, volume 37, pp. 12252–12259, 2023.
- 582 Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić,
583 Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint*
584 *arXiv:2404.19756*, 2024.
- 585 Lirui Luo, Guoxi Zhang, Hongming Xu, Yaodong Yang, Cong Fang, and Qing Li. Insight: End-
586 to-end neuro-symbolic visual reinforcement learning with language explanations. *arXiv preprint*
587 *arXiv:2403.12451*, 2024.
- 588 Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous
589 relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

- 594 Nour Makke and Sanjay Chawla. Interpretable scientific discovery with symbolic regression: a
595 review. *Artificial Intelligence Review*, 57(1):2, 2024.
596
- 597 Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt.
598 Deepproblog: Neural probabilistic logic programming. *Advances in neural information process-*
599 *ing systems*, 31, 2018.
600
- 601 Emanuele Marconato, Stefano Teso, Antonio Vergari, and Andrea Passerini. Not all neuro-symbolic
602 concepts are created equal: Analysis and mitigation of reasoning shortcuts. *Advances in Neural*
603 *Information Processing Systems*, 36, 2024.
604
- 605 Giuseppe Marra, Sebastijan Dumančić, Robin Manhaeve, and Luc De Raedt. From statistical re-
606 lational to neurosymbolic artificial intelligence: A survey. *Artificial Intelligence*, pp. 104062,
607 2024.
- 608 Georg Martius and Christoph H Lampert. Extrapolation and learning equations. *arXiv preprint*
609 *arXiv:1610.02995*, 2016.
610
- 611 Ali Payani and Faramarz Fekri. Inductive logic programming via differentiable deep neural logic
612 networks. *arXiv preprint arXiv:1906.03523*, 2019a.
613
- 614 Ali Payani and Faramarz Fekri. Learning algorithms via neural logic networks. *arXiv preprint*
615 *arXiv:1904.01554*, 2019b.
616
- 617 Ali Payani and Faramarz Fekri. Incorporating relational background knowledge into reinforcement
618 learning via differentiable inductive logic programming. *arXiv preprint arXiv:2003.10386*, 2020.
619
- 620 Ryan Riegel, Alexander Gray, Francois Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus
621 Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, et al. Logical neural
622 networks. *arXiv preprint arXiv:2006.13155*, 2020.
623
- 624 Subham Sahoo, Christoph Lampert, and Georg Martius. Learning equations for extrapolation and
625 control. In *International Conference on Machine Learning*, pp. 4442–4450. Pmlr, 2018.
626
- 627 Prithviraj Sen, Breno WSR de Carvalho, Ryan Riegel, and Alexander Gray. Neuro-symbolic induc-
628 tive logic programming with logical neural networks. In *Proceedings of the AAAI conference on*
629 *artificial intelligence*, volume 36, pp. 8212–8219, 2022.
- 630 Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and
631 their roles for better empirical performance. *arXiv preprint arXiv:2304.11127*, 2023.
632
- 633 Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural
634 stochastic logic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*,
635 volume 36, pp. 10090–10100, 2022.
636
- 637 Yuan Yang and Le Song. Learn to explain efficiently via neural logic inductive learning. *arXiv*
638 *preprint arXiv:1910.02481*, 2019.
639
- 640 Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set
641 programming. *arXiv preprint arXiv:2307.07700*, 2023.
642
- 643 Zheng Zhang, Levent Yilmaz, and Bo Liu. A critical review of inductive logic programming tech-
644 niques for explainable ai. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.
- 645 Matthieu Zimmer, Xuening Feng, Claire Glanois, Zhaohui Jiang, Jianyi Zhang, Paul Weng, Dong Li,
646 Jianye Hao, and Wulong Liu. Differentiable logic machines. *arXiv preprint arXiv:2102.11529*,
647 2021.

A IMPLEMENTATION DETAILS

All experiments are implemented with Python. The neural network modules are implemented with Pytorch, and symbolic expressions are extracted from LFL modules' trained weights with SymPy. MadGrad optimizer (Defazio & Jelassi, 2022) is used for all trainable modules, with learning rates independently tuned for CNN, LFL, and MLP modules. Random seed 42 is used for all experiments, and hyperparameters are selected automatically with Optuna's (Akiba et al., 2019) implementation of Tree-Structured Parzen Estimator (Watanabe, 2023). Pytorch's GPU version is required to run the experiments efficiently.

B DATASET DESCRIPTION

Experiment 3.1.1 uses a binary dataset with size $2^8 = 256$. The inputs and outputs are generated with the following logical formula:



$$\begin{aligned}
 \mathbf{x} &= [x_0, x_1, \dots, x_7] \\
 \mathbf{h}_1 &= [x_0 \wedge x_1, x_2 \wedge x_3, x_4 \wedge x_5, x_6 \wedge x_7, \overline{x_0} \wedge \overline{x_1}, \overline{x_2} \wedge \overline{x_3}, \overline{x_4} \wedge \overline{x_5}, \overline{x_6} \wedge \overline{x_7}] \\
 \mathbf{h}_2 &= [h_{10} \vee h_{11}, h_{12} \vee h_{13}, h_{14} \vee h_{15}, h_{16} \vee h_{17}] \\
 \mathbf{y} &= [h_{20} \wedge h_{21}, h_{22} \wedge h_{23}]
 \end{aligned} \tag{22}$$





where \mathbf{x} means the dataset's input, \mathbf{y} means its output, \mathbf{h}_1 and \mathbf{h}_2 mean intermediate variables that hidden neurons might learn to represent.

Experiment 3.1.2 uses a binary dataset with size $10^2 = 100$. The inputs and outputs are generated with the following logical formula:

$$\begin{aligned}
 \mathbf{l} &= [l_0, l_1, \dots, l_9] \\
 \mathbf{r} &= [r_0, r_1, \dots, r_9] \\
 \mathbf{y} &= [y_0, y_1, \dots, y_{18}] \\
 y_k &= \bigvee_{i+j=k} (l_i \wedge r_j)
 \end{aligned} \tag{23}$$

where \mathbf{l} and \mathbf{r} mean input symbols representing values of the two digits, y_k means the output symbol representing that the sum is k . This is the same formula that LFL modules are expected to learn when wrapped into a NeSy predictor for MNIST Sum.

MNIST Sum is a standard benchmark for testing NeSy predictors, first used in Manhaeve et al. (2018). Each sample of the MNIST Sum task takes 2 handwritten digits as input and requires a model to predict their sum in $[0, 18]$ as a 19-class classification task. For example, a sample of the dataset has input (, ) and classification label 11.

MNIST Multi-digit Sum is an extension of MNIST Sum that requires a model to predict labels for summing two multi-digit handwritten numbers. The two numbers' length n_{digits} is a property of the dataset. The inputs and labels are provided in reverse order. For example, a sample of the dataset with $n_{digits} = 2$ has input [(, )], (, ) and classification label [6,0,1] (since $59+47=106$).

The MNIST Sum training dataset uses two MNIST images from its training set for each sample, so its actual size is $60000^2 = 3.6 \times 10^9$, making it impractical to actually train a full epoch. In our implementations, we take 8192 randomly generated samples as a pseudo-epoch of the dataset. To generate a sample, we first randomly choose two digits in 0 to 9, then randomly select two handwritten digit images that correspond to the chosen digits.

The MNIST Multi-digit Sum training dataset is implemented in a similar way, where 8192 randomly generated samples are treated as a pseudo-epoch.

The MNIST Sum test dataset uses MNIST test set images instead of training set images and treats 65536 samples as a pseudo-epoch. The MNIST Multi-digit Sum test dataset also uses MNIST test set images, treating 65536 samples as a pseudo-epoch for $n_{digit} = 3$ and 8192 samples as a pseudo-epoch for $n_{digit} = 128$. The accuracies in Table 1 are evaluated on one pseudo-epoch of test datasets.

C HYPERPARAMETERS

Since the linear reconstruction layer and its reconstruction loss term don't affect other network components, we fix their loss weight to 1 and learning rate to 0.001. For our novel LFL variations, we describe the manually chosen hyperparameters in text and put the hyperparameters chosen by Optuna in tables. For DSL we use default hyperparameters found in the original implementation.

C.1 EXPERIMENT 3.1.1

Both dNL and LFL-Type1 are trained for 256 epochs, with 32 and 16 neurons in their hidden layers. dNL's Hyperparameters:

	Meaning	Value
lr	learning rate	60.74940936219056
β_1	regularization loss weight	0.13347185900881423

LFL-Type1's Hyperparameters:

	Meaning	Value
lr	learning rate	28.353628505319445
η_1	noise scale of the 1st layer	0.8044418562388825
η_2	noise scale of the 2nd layer	0.09504842110818068
η_3	noise scale of the 3rd layer	0.42150252547988554
β_1	regularization loss weight	0.05891286543711857

C.2 EXPERIMENT 3.1.2

Both dNL and LFL-Type1 are trained for 4096 epochs in every run, allowing them to fully converge with a limited number of hidden neurons. The hyperparameters are chosen with 256 hidden neurons in each network.

dNL's Hyperparameters:

	Meaning	Value
lr	learning rate	3.721338105780261
β_1	regularization loss weight	0.5163968434483255

LFL-Type1's Hyperparameters:

	Meaning	Value
lr	learning rate	20.743324016859198
η_1	noise scale of the 1st layer	0.9772471411825822
η_2	noise scale of the 2nd layer	0.8185423593180111
β_1	regularization loss weight	0.1485845565420301

C.3 DSL ON MNIST SUM

The network is trained for 128 pseudo-epochs.

	Meaning	Value
lr_{LFL}	the logic module's learning rate	0.11639833786002995
lr_{CNN}	the CNN's learning rate	0.001
ϵ_{symbol}	ϵ of the fuzzifier	0.2807344052335263
ϵ_{rule}	ϵ of the OR layer	0.1077119516324264

C.4 LFL-TYPE2 ON MNIST SUM

The network is trained for 128 pseudo-epochs.

756

757

758

759

760

761

762

C.5 LFL-TYPE3 ON MNIST SUM

The network is trained for 128 pseudo-epochs.

766

767

768

769

770

771

772

773

774

775

776

C.6 LFL-TYPE3 ON MNIST MULTI-DIGIT SUM

The network is trained for up to 256 pseudo-epochs on datasets with $n_{digits} = 1, 2, 3$ sequentially, with early stopping if the average supervision BCE loss falls below 0.001 at any epoch.

779

780

781

782

783

784

785

786

787

788

C.7 LFL-TYPE1 ON MNIST SUM

The network is trained for 128 pseudo-epochs, with 512 hidden neurons in the LFL-Type1 module.

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

C.8 LFL-TYPE1 ON MNIST MULTI-DIGIT SUM

The network is trained for up to 256 pseudo-epochs on datasets with $n_{digits} = 2, 3$ sequentially (training on $n_{digits} = 1$ would allow LFL-Type1 to completely ignore input carries), with early stopping if the average supervision BCE loss falls below 0.001 at any epoch. There are 8192 hidden neurons in the LFL-Type1 module.

808

809

	Meaning	Value
lr_{LFL}	the LFL's learning rate	0.03233368589774149
lr_{CNN}	the CNN's learning rate	0.0013455349127554875
η_0	noise scale of the fuzzifier	0.21398889643701835
η_1	noise scale of the OR layer	0.9837843300240907

	Meaning	Value
lr_{LFL}	the LFL's learning rate	9.613947918454464
lr_{CNN}	the CNN's learning rate	0.0019151515603726996
η_0	noise scale of the fuzzifier	0.9113745711049871
η_1	noise scale of the OR layer	0.5785108825130485
β_1	regularization loss weight	0.07854073377893649

	Meaning	Value
lr_{LFL}	the LFL's learning rate	4.505968658051522
lr_{CNN}	the CNN's learning rate	0.0047565461690141616
η_0	noise scale of the fuzzifier	0.8632755389984255
η_1	noise scale of the OR layer	0.9467031818141409
β_1	regularization loss weight	0.09120659551465411

	Meaning	Value
lr_{LFL}	the LFL's learning rate	3.4946338976075064
lr_{CNN}	the CNN's learning rate	0.0006208460070100641
lr_{MLP}	the MLP's learning rate	0.0627384382035572
η_0	noise scale of the fuzzifier	0.3761108683081919
η_1	noise scale of the AND layer	0.9385093053957093
η_2	noise scale of the OR layer	0.5471963095839101
β_1	regularization loss weight	0.4498676093863323
β_2	MLP loss weight	21.454211508035232
β_3	label distribution loss weight	3.7151918234461507

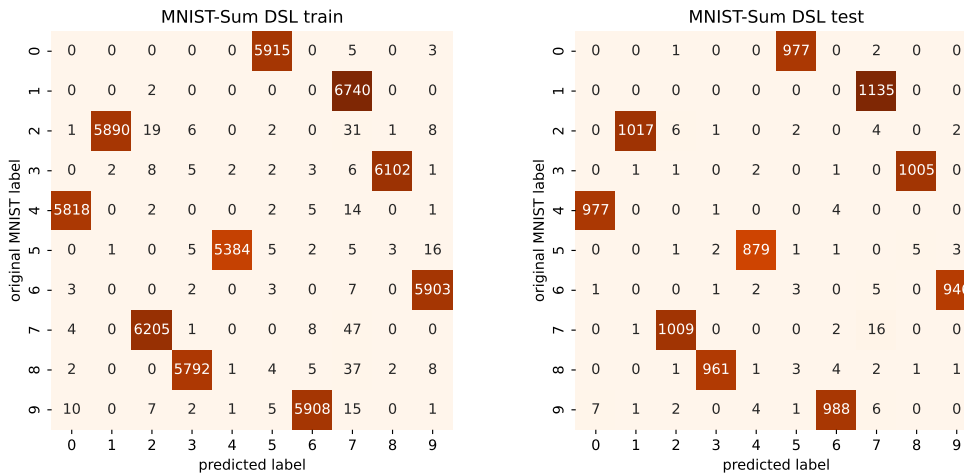
	Meaning	Value	
810			
811	lr_{LFL}	the LFL's learning rate	11.331405298874264
812	lr_{CNN}	the CNN's learning rate	0.000571482276406197
813	lr_{MLP}	the MLP's learning rate	0.012437761948519307
814	η_0	noise scale of the fuzzifier	0.3115310029581702
815	η_1	noise scale of the AND layer	0.8485090728796351
816	η_2	noise scale of the OR layer	0.9770926024145439
817	β_1	regularization loss weight	0.13687607266487536
818	β_2	MLP loss weight	8.473281653324152
819	β_3	label distribution loss weight	1.266068955037883

D DETAILED EXPERIMENT RESULTS

Here we demonstrate what the NeSy predictors have learned with confusion matrices of their neural classifiers and logical formulas learned by their LFL modules. In these logical formulas we represent digit symbols with their linear reconstruction layer's reconstruction of corresponding one-hot tensors. The reconstructed images are colored in yellow or cyan, representing symbols predicted from two input images. In MNIST Multi-digit Sum we denote the carry symbols as C_0 for input carry being 0, C_1 for input carry being 1, C'_0 for output carry being 0 and C'_1 for output carry being 1.

D.1 DSL ON MNIST SUM

Confusion matrices on MNIST training and test dataset:



Learned formula:

$$y_0 = (0 \wedge 0)$$

$$y_1 = (0 \wedge 1) \vee (1 \wedge 0)$$

$$y_2 = (2 \wedge 0) \vee (0 \wedge 2) \vee (1 \wedge 1)$$

$$y_3 = (2 \wedge 1) \vee (0 \wedge 3) \vee (1 \wedge 2) \vee (3 \wedge 0)$$

$$y_4 = (4 \wedge 0) \vee (2 \wedge 2) \vee (0 \wedge 4) \vee (1 \wedge 3) \vee (3 \wedge 1)$$

$$y_5 = (4 \wedge 1) \vee (2 \wedge 3) \vee (5 \wedge 0) \vee (0 \wedge 5) \vee (1 \wedge 4) \vee (3 \wedge 2)$$

$$y_6 = (4 \wedge 2) \vee (2 \wedge 4) \vee (5 \wedge 1) \vee (0 \wedge 6) \vee (1 \wedge 5) \vee (3 \wedge 3) \vee (6 \wedge 0)$$

$$y_7 = (4 \wedge 3) \vee (2 \wedge 5) \vee (7 \wedge 0) \vee (5 \wedge 2) \vee (0 \wedge 7) \vee (1 \wedge 6) \vee (3 \wedge 4) \vee (6 \wedge 1)$$

$$y_8 = (4 \wedge 4) \vee (2 \wedge 6) \vee (7 \wedge 1) \vee (8 \wedge 0) \vee (5 \wedge 3) \vee (0 \wedge 8) \vee (1 \wedge 7) \vee (3 \wedge 5) \vee (6 \wedge 2)$$

$$\begin{aligned}
y_9 &= (4 \wedge 5) \vee (2 \wedge 7) \vee (7 \wedge 2) \vee (8 \wedge 1) \vee (5 \wedge 4) \vee (0 \wedge 9) \vee (9 \wedge 0) \vee (1 \wedge 8) \vee (3 \wedge 6) \vee (6 \wedge 3) \\
y_{10} &= (4 \wedge 6) \vee (2 \wedge 8) \vee (7 \wedge 3) \vee (8 \wedge 2) \vee (5 \wedge 5) \vee (9 \wedge 1) \vee (1 \wedge 9) \vee (3 \wedge 7) \vee (6 \wedge 4) \\
y_{11} &= (4 \wedge 7) \vee (2 \wedge 9) \vee (7 \wedge 4) \vee (8 \wedge 3) \vee (5 \wedge 6) \vee (9 \wedge 2) \vee (3 \wedge 8) \vee (6 \wedge 5) \\
y_{12} &= (4 \wedge 8) \vee (7 \wedge 5) \vee (8 \wedge 4) \vee (5 \wedge 7) \vee (9 \wedge 3) \vee (3 \wedge 9) \vee (6 \wedge 6) \\
y_{13} &= (4 \wedge 9) \vee (7 \wedge 6) \vee (8 \wedge 5) \vee (5 \wedge 8) \vee (9 \wedge 4) \vee (6 \wedge 7) \\
y_{14} &= (7 \wedge 7) \vee (8 \wedge 6) \vee (5 \wedge 9) \vee (9 \wedge 5) \vee (6 \wedge 8) \\
y_{15} &= (7 \wedge 8) \vee (8 \wedge 7) \vee (9 \wedge 6) \vee (6 \wedge 9) \\
y_{16} &= (7 \wedge 9) \vee (8 \wedge 8) \vee (9 \wedge 7) \\
y_{17} &= (8 \wedge 9) \vee (9 \wedge 8) \\
y_{18} &= 9 \wedge 9
\end{aligned}$$

D.2 LFL-TYPE2 ON MNIST SUM

Confusion matrices on MNIST training and test dataset:

MNIST-Sum LFL-Type2 train										MNIST-Sum LFL-Type2 test											
original MNIST label	0	1	2	3	4	5	6	7	8	9	original MNIST label	0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	0	5920	1	0	0	0	0	0	0	0	0	0	977	1	1	1	0
1	0	6739	0	0	0	0	0	0	3	0	1	0	1135	0	0	0	0	0	0	0	0
2	0	12	5927	1	0	3	1	4	10	0	2	3	4	1019	1	0	1	0	0	4	0
3	0	1	2	6114	3	1	0	3	4	3	3	0	0	1	1007	1	0	0	0	1	0
4	5822	8	0	0	0	2	0	0	1	9	4	978	1	0	0	0	0	0	0	0	3
5	0	1	1	8	5403	1	5	1	0	1	5	0	0	0	12	876	2	1	1	0	0
6	2	2	0	0	0	7	5907	0	0	0	6	1	2	0	1	1	4	948	1	0	0
7	1	22	2	0	0	0	0	0	6238	2	7	0	6	2	0	0	1	0	2	1016	1
8	1	9	0	4	4	7	2	5815	0	9	8	1	0	1	2	1	7	1	959	0	2
9	6	6	0	3	5	8	0	2	7	5912	9	3	2	0	0	3	1	1	0	5	994
	0	1	2	3	4	5	6	7	8	9		0	1	2	3	4	5	6	7	8	9

Learned formula:

$$\begin{aligned}
y_0 &= 0 \wedge 0 \\
y_1 &= (1 \wedge 0) \vee (0 \wedge 1) \\
y_2 &= (1 \wedge 1) \vee (2 \wedge 0) \vee (0 \wedge 2) \\
y_3 &= (1 \wedge 2) \vee (2 \wedge 1) \vee (3 \wedge 0) \vee (0 \wedge 3) \\
y_4 &= (4 \wedge 0) \vee (1 \wedge 3) \vee (2 \wedge 2) \vee (3 \wedge 1) \vee (0 \wedge 4) \\
y_5 &= (4 \wedge 1) \vee (1 \wedge 4) \vee (2 \wedge 3) \vee (3 \wedge 2) \vee (5 \wedge 0) \vee (0 \wedge 5) \\
y_6 &= (4 \wedge 2) \vee (1 \wedge 5) \vee (2 \wedge 4) \vee (3 \wedge 3) \vee (5 \wedge 1) \vee (0 \wedge 6) \vee (6 \wedge 0) \\
y_7 &= (4 \wedge 3) \vee (1 \wedge 6) \vee (2 \wedge 5) \vee (3 \wedge 4) \vee (5 \wedge 2) \vee (0 \wedge 7) \vee (6 \wedge 1) \vee (7 \wedge 0) \\
y_8 &= (4 \wedge 4) \vee (1 \wedge 7) \vee (2 \wedge 6) \vee (3 \wedge 5) \vee (5 \wedge 3) \vee (0 \wedge 8) \vee (6 \wedge 2) \vee (8 \wedge 0) \vee (7 \wedge 1) \\
y_9 &= (4 \wedge 5) \vee (1 \wedge 8) \vee (2 \wedge 7) \vee (3 \wedge 6) \vee (5 \wedge 4) \vee (0 \wedge 9) \vee (6 \wedge 3) \vee (8 \wedge 1) \vee (7 \wedge 2) \vee (9 \wedge 0) \\
y_{10} &= (4 \wedge 6) \vee (1 \wedge 9) \vee (2 \wedge 8) \vee (3 \wedge 7) \vee (5 \wedge 5) \vee (6 \wedge 4) \vee (8 \wedge 2) \vee (7 \wedge 3) \vee (9 \wedge 1)
\end{aligned}$$

$$\begin{aligned}
y_{11} &= (4 \wedge 7) \vee (2 \wedge 9) \vee (3 \wedge 8) \vee (5 \wedge 6) \vee (6 \wedge 5) \vee (8 \wedge 3) \vee (7 \wedge 4) \vee (9 \wedge 2) \\
y_{12} &= (4 \wedge 8) \vee (3 \wedge 9) \vee (5 \wedge 7) \vee (6 \wedge 6) \vee (8 \wedge 4) \vee (7 \wedge 5) \vee (9 \wedge 3) \\
y_{13} &= (4 \wedge 9) \vee (5 \wedge 8) \vee (6 \wedge 7) \vee (8 \wedge 5) \vee (7 \wedge 6) \vee (9 \wedge 4) \\
y_{14} &= (5 \wedge 9) \vee (6 \wedge 8) \vee (8 \wedge 6) \vee (7 \wedge 7) \vee (9 \wedge 5) \\
y_{15} &= (6 \wedge 9) \vee (8 \wedge 7) \vee (7 \wedge 8) \vee (9 \wedge 6) \\
y_{16} &= (8 \wedge 8) \vee (7 \wedge 9) \vee (9 \wedge 7) \\
y_{17} &= (8 \wedge 9) \vee (9 \wedge 8) \\
y_{18} &= 9 \wedge 9
\end{aligned}$$

D.3 LFL-TYPE3 ON MNIST SUM

Confusion matrices on MNIST training and test dataset:

MNIST-Sum LFL-Type3 train										MNIST-Sum LFL-Type3 test										
original MNIST label	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
0	0	2	0	0	0	0	0	0	5917	4	0	0	0	0	0	0	0	0	0	0
1	0	6741	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	2	0	0	5953	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0
3	1	3	11	0	38	6075	0	3	0	0	0	0	0	0	0	0	0	0	0	0
4	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5839
5	0	2	5402	0	0	2	1	0	14	0	0	0	0	0	0	0	0	0	0	0
6	0	2	0	0	0	0	0	0	1	5912	3	0	0	0	0	0	0	0	0	0
7	2	66	1	6162	32	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
8	2	14	1	0	8	0	5813	5	7	1	0	0	0	0	0	0	0	0	0	0
9	5925	2	0	1	0	0	0	0	1	0	20	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9

Learned formula:

$$\begin{aligned}
y_0 &= 0 \wedge 0 \\
y_1 &= (1 \wedge 0) \vee (0 \wedge 1) \\
y_2 &= (1 \wedge 1) \vee (2 \wedge 0) \vee (0 \wedge 2) \\
y_3 &= (1 \wedge 2) \vee (2 \wedge 1) \vee (3 \wedge 0) \vee (0 \wedge 3) \\
y_4 &= (1 \wedge 3) \vee (2 \wedge 2) \vee (3 \wedge 1) \vee (0 \wedge 4) \vee (4 \wedge 0) \\
y_5 &= (1 \wedge 4) \vee (5 \wedge 0) \vee (2 \wedge 3) \vee (3 \wedge 2) \vee (0 \wedge 5) \vee (4 \wedge 1) \\
y_6 &= (1 \wedge 5) \vee (5 \wedge 1) \vee (2 \wedge 4) \vee (3 \wedge 3) \vee (0 \wedge 6) \vee (6 \wedge 0) \vee (4 \wedge 2) \\
y_7 &= (1 \wedge 6) \vee (5 \wedge 2) \vee (7 \wedge 0) \vee (2 \wedge 5) \vee (3 \wedge 4) \vee (0 \wedge 7) \vee (6 \wedge 1) \vee (4 \wedge 3) \\
y_8 &= (1 \wedge 7) \vee (5 \wedge 3) \vee (7 \wedge 1) \vee (2 \wedge 6) \vee (3 \wedge 5) \vee (8 \wedge 0) \vee (0 \wedge 8) \vee (6 \wedge 2) \vee (4 \wedge 4) \\
y_9 &= (9 \wedge 0) \vee (1 \wedge 8) \vee (5 \wedge 4) \vee (7 \wedge 2) \vee (2 \wedge 7) \vee (3 \wedge 6) \vee (8 \wedge 1) \vee (0 \wedge 9) \\
&\vee (6 \wedge 3) \vee (4 \wedge 5) \\
y_{10} &= (9 \wedge 1) \vee (1 \wedge 9) \vee (5 \wedge 5) \vee (7 \wedge 3) \vee (2 \wedge 8) \vee (3 \wedge 7) \vee (8 \wedge 2) \vee (6 \wedge 4) \vee (4 \wedge 6) \\
y_{11} &= (9 \wedge 2) \vee (5 \wedge 6) \vee (7 \wedge 4) \vee (2 \wedge 9) \vee (3 \wedge 8) \vee (8 \wedge 3) \vee (6 \wedge 5) \vee (4 \wedge 7) \\
y_{12} &= (9 \wedge 3) \vee (5 \wedge 7) \vee (7 \wedge 5) \vee (3 \wedge 9) \vee (8 \wedge 4) \vee (6 \wedge 6) \vee (4 \wedge 8) \\
y_{13} &= (9 \wedge 4) \vee (5 \wedge 8) \vee (7 \wedge 6) \vee (8 \wedge 5) \vee (6 \wedge 7) \vee (4 \wedge 9)
\end{aligned}$$

$$y_{14} = (9 \wedge 5) \vee (5 \wedge 9) \vee (7 \wedge 7) \vee (8 \wedge 6) \vee (6 \wedge 8)$$

$$y_{15} = (9 \wedge 6) \vee (7 \wedge 8) \vee (8 \wedge 7) \vee (6 \wedge 9)$$

$$y_{16} = (9 \wedge 7) \vee (7 \wedge 9) \vee (8 \wedge 8)$$

$$y_{17} = (9 \wedge 8) \vee (8 \wedge 9)$$

$$y_{18} = 9 \wedge 9$$

D.4 LFL-TYPE3 ON MNIST MULTI-DIGIT SUM

Confusion matrices on MNIST training and test dataset:

original MNIST label	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	5923	0	0	0	0
1	0	0	0	6742	0	0	0	0	0	0
2	0	0	5957	0	0	0	1	0	0	0
3	1	0	0	0	1	0	0	0	6128	1
4	5841	0	0	0	0	0	1	0	0	0
5	0	0	0	0	5419	0	0	0	2	0
6	0	5917	0	0	0	0	0	0	1	0
7	1	0	1	0	0	0	6263	0	0	0
8	1	1	0	0	1	0	0	5847	1	0
9	13	0	0	0	0	0	3	3	0	5930

original MNIST label	0	1	2	3	4	5	6	7	8	9
0	1	0	0	1	1	974	2	1	0	0
1	1	0	1	1129	1	1	0	0	2	0
2	2	1	1018	0	0	2	3	2	4	0
3	1	0	2	0	3	0	1	1	1002	0
4	977	0	0	1	0	0	0	1	0	3
5	0	1	1	0	878	2	0	1	7	2
6	2	946	0	2	4	4	0	0	0	0
7	0	0	3	2	1	0	1018	1	1	2
8	0	0	1	0	3	0	0	964	5	1
9	14	0	1	0	2	0	6	8	0	978

Learned formula:

$$y_0 = (C_0 \wedge 4 \wedge 6) \vee (C_0 \wedge 6 \wedge 4) \vee (C_0 \wedge 2 \wedge 8) \vee (C_0 \wedge 1 \wedge 9) \vee (C_0 \wedge 5 \wedge 5) \vee (C_0 \wedge 0 \wedge 0) \vee (C_0 \wedge 7 \wedge 3) \vee (C_0 \wedge 8 \wedge 2) \vee (C_0 \wedge 3 \wedge 7) \vee (C_0 \wedge 9 \wedge 1) \vee (C_1 \wedge 4 \wedge 5) \vee (C_1 \wedge 6 \wedge 3) \vee (C_1 \wedge 2 \wedge 7) \vee (C_1 \wedge 1 \wedge 8) \vee (C_1 \wedge 5 \wedge 4) \vee (C_1 \wedge 0 \wedge 9) \vee (C_1 \wedge 7 \wedge 2) \vee (C_1 \wedge 8 \wedge 1) \vee (C_1 \wedge 3 \wedge 6) \vee (C_1 \wedge 9 \wedge 0)$$

$$y_1 = (C_0 \wedge 4 \wedge 7) \vee (C_0 \wedge 6 \wedge 5) \vee (C_0 \wedge 2 \wedge 9) \vee (C_0 \wedge 1 \wedge 0) \vee (C_0 \wedge 5 \wedge 6) \vee (C_0 \wedge 0 \wedge 1) \vee (C_0 \wedge 7 \wedge 4) \vee (C_0 \wedge 8 \wedge 3) \vee (C_0 \wedge 3 \wedge 8) \vee (C_0 \wedge 9 \wedge 2) \vee (C_1 \wedge 4 \wedge 6) \vee (C_1 \wedge 6 \wedge 4) \vee (C_1 \wedge 2 \wedge 8) \vee (C_1 \wedge 1 \wedge 9) \vee (C_1 \wedge 5 \wedge 5) \vee (C_1 \wedge 0 \wedge 0) \vee (C_1 \wedge 7 \wedge 3) \vee (C_1 \wedge 8 \wedge 2) \vee (C_1 \wedge 3 \wedge 7) \vee (C_1 \wedge 9 \wedge 1)$$

$$y_2 = (C_0 \wedge 4 \wedge 8) \vee (C_0 \wedge 6 \wedge 6) \vee (C_0 \wedge 2 \wedge 0) \vee (C_0 \wedge 1 \wedge 1) \vee (C_0 \wedge 5 \wedge 7) \vee (C_0 \wedge 0 \wedge 2) \vee (C_0 \wedge 7 \wedge 5) \vee (C_0 \wedge 8 \wedge 4) \vee (C_0 \wedge 3 \wedge 9) \vee (C_0 \wedge 9 \wedge 3) \vee (C_1 \wedge 4 \wedge 7) \vee (C_1 \wedge 6 \wedge 5) \vee (C_1 \wedge 2 \wedge 9) \vee (C_1 \wedge 1 \wedge 0) \vee (C_1 \wedge 5 \wedge 6) \vee (C_1 \wedge 0 \wedge 1) \vee (C_1 \wedge 7 \wedge 4) \vee (C_1 \wedge 8 \wedge 3) \vee (C_1 \wedge 3 \wedge 8) \vee (C_1 \wedge 9 \wedge 2)$$

$$y_3 = (C_0 \wedge 4 \wedge 9) \vee (C_0 \wedge 6 \wedge 7) \vee (C_0 \wedge 2 \wedge 1) \vee (C_0 \wedge 1 \wedge 2) \vee (C_0 \wedge 5 \wedge 8) \vee (C_0 \wedge 0 \wedge 3) \vee (C_0 \wedge 7 \wedge 6) \vee (C_0 \wedge 8 \wedge 5) \vee (C_0 \wedge 3 \wedge 0) \vee (C_0 \wedge 9 \wedge 4) \vee (C_1 \wedge 4 \wedge 8) \vee (C_1 \wedge 6 \wedge 6) \vee (C_1 \wedge 2 \wedge 0) \vee (C_1 \wedge 1 \wedge 1) \vee (C_1 \wedge 5 \wedge 7) \vee (C_1 \wedge 0 \wedge 2) \vee (C_1 \wedge 7 \wedge 5) \vee (C_1 \wedge 8 \wedge 4) \vee (C_1 \wedge 3 \wedge 9) \vee (C_1 \wedge 9 \wedge 3)$$

$$y_4 = (C_0 \wedge 4 \wedge 0) \vee (C_0 \wedge 6 \wedge 8) \vee (C_0 \wedge 2 \wedge 2) \vee (C_0 \wedge 1 \wedge 3) \vee (C_0 \wedge 5 \wedge 9) \vee (C_0 \wedge 0 \wedge 4) \vee (C_0 \wedge 7 \wedge 7) \vee (C_0 \wedge 8 \wedge 6) \vee (C_0 \wedge 3 \wedge 1) \vee (C_0 \wedge 9 \wedge 5) \vee (C_1 \wedge 4 \wedge 9) \vee (C_1 \wedge 6 \wedge 7) \vee (C_1 \wedge 2 \wedge 1) \vee (C_1 \wedge 1 \wedge 2) \vee (C_1 \wedge 5 \wedge 8) \vee (C_1 \wedge 0 \wedge 3) \vee (C_1 \wedge 7 \wedge 6) \vee (C_1 \wedge 8 \wedge 5) \vee (C_1 \wedge 3 \wedge 0) \vee (C_1 \wedge 9 \wedge 4)$$

$$y_5 = (C_0 \wedge 4 \wedge 1) \vee (C_0 \wedge 6 \wedge 9) \vee (C_0 \wedge 2 \wedge 3) \vee (C_0 \wedge 1 \wedge 4) \vee (C_0 \wedge 5 \wedge 0) \vee (C_0 \wedge 0 \wedge 5) \vee (C_0 \wedge 7 \wedge 8) \vee (C_0 \wedge 8 \wedge 7) \vee (C_0 \wedge 3 \wedge 2) \vee (C_0 \wedge 9 \wedge 6) \vee (C_1 \wedge 4 \wedge 0) \vee (C_1 \wedge 6 \wedge 8) \vee (C_1 \wedge 2 \wedge 2) \vee (C_1 \wedge 1 \wedge 3) \vee (C_1 \wedge 5 \wedge 9) \vee (C_1 \wedge 0 \wedge 4) \vee (C_1 \wedge 7 \wedge 7) \vee (C_1 \wedge 8 \wedge 6) \vee (C_1 \wedge 3 \wedge 1) \vee (C_1 \wedge 9 \wedge 5)$$

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

original MNIST label	0	1	2	3	4	5	6	7	8	9
0	0	0	3	0	0	0	0	2	5918	0
1	6736	0	1	1	0	4	0	0	0	0
2	2	2	1	1	0	0	3	5945	0	0
3	0	0	0	0	12	0	0	1	0	6118
4	1	0	0	0	5836	0	1	0	4	0
5	0	0	3	0	5415	0	0	0	0	3
6	0	0	5916	0	0	0	1	0	1	0
7	4	1	0	1	0	6254	2	2	1	0
8	2	5838	5	0	5	0	0	0	0	1
9	1	2	0	12	1	2	0	5929	0	2

original MNIST label	0	1	2	3	4	5	6	7	8	9
0	1	1	1	0	0	1	0	0	976	0
1	1132	0	0	0	0	2	0	0	0	1
2	0	1	0	2	0	6	1020	0	0	3
3	0	1	0	0	6	0	0	0	0	1003
4	0	0	2	978	0	0	1	1	0	0
5	0	0	1	0	889	0	0	0	0	2
6	1	1	949	3	1	0	0	0	2	1
7	2	1	0	1	0	1020	2	1	0	1
8	0	964	0	0	4	0	1	1	1	3
9	0	1	0	9	5	3	0	989	1	1

1097

Learned formula:

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

$$y_0 = 0 \wedge 0$$

$$y_1 = (1 \wedge 0) \vee (0 \wedge 1)$$

$$y_2 = (1 \wedge 1) \vee (2 \wedge 0) \vee (0 \wedge 2)$$

$$y_3 = (1 \wedge 2) \vee (2 \wedge 1) \vee (0 \wedge 3) \vee (3 \wedge 0)$$

$$y_4 = (1 \wedge 3) \vee (4 \wedge 0) \vee (2 \wedge 2) \vee (0 \wedge 4) \vee (3 \wedge 1)$$

$$y_5 = (1 \wedge 4) \vee (4 \wedge 1) \vee (5 \wedge 0) \vee (2 \wedge 3) \vee (0 \wedge 5) \vee (3 \wedge 2)$$

$$y_6 = (1 \wedge 5) \vee (6 \wedge 0) \vee (4 \wedge 2) \vee (5 \wedge 1) \vee (2 \wedge 4) \vee (0 \wedge 6) \vee (3 \wedge 3)$$

$$y_7 = (1 \wedge 6) \vee (6 \wedge 1) \vee (4 \wedge 3) \vee (5 \wedge 2) \vee (7 \wedge 0) \vee (2 \wedge 5) \vee (0 \wedge 7) \vee (3 \wedge 4)$$

$$y_8 = (1 \wedge 7) \vee (8 \wedge 0) \vee (6 \wedge 2) \vee (4 \wedge 4) \vee (5 \wedge 3) \vee (7 \wedge 1) \vee (2 \wedge 6) \vee (0 \wedge 8) \vee (3 \wedge 5)$$

$$y_9 = (1 \wedge 8) \vee (8 \wedge 1) \vee (6 \wedge 3) \vee (4 \wedge 5) \vee (5 \wedge 4) \vee (7 \wedge 2) \vee (2 \wedge 7) \vee (9 \wedge 0) \vee (0 \wedge 9) \vee (3 \wedge 6)$$

$$y_{10} = (1 \wedge 9) \vee (8 \wedge 2) \vee (6 \wedge 4) \vee (4 \wedge 6) \vee (5 \wedge 5) \vee (7 \wedge 3) \vee (2 \wedge 8) \vee (9 \wedge 1) \vee (3 \wedge 7)$$

$$y_{11} = (8 \wedge 3) \vee (6 \wedge 5) \vee (4 \wedge 7) \vee (5 \wedge 6) \vee (7 \wedge 4) \vee (2 \wedge 9) \vee (9 \wedge 2) \vee (3 \wedge 8)$$

$$y_{12} = (8 \wedge 4) \vee (6 \wedge 6) \vee (4 \wedge 8) \vee (5 \wedge 7) \vee (7 \wedge 5) \vee (9 \wedge 3) \vee (3 \wedge 9)$$

$$y_{13} = (8 \wedge 5) \vee (6 \wedge 7) \vee (4 \wedge 9) \vee (5 \wedge 8) \vee (7 \wedge 6) \vee (9 \wedge 4)$$

$$y_{14} = (8 \wedge 6) \vee (6 \wedge 8) \vee (5 \wedge 9) \vee (7 \wedge 7) \vee (9 \wedge 5)$$

$$y_{15} = (8 \wedge 7) \vee (6 \wedge 9) \vee (7 \wedge 8) \vee (9 \wedge 6)$$

$$y_{16} = (8 \wedge 8) \vee (7 \wedge 9) \vee (9 \wedge 7)$$

$$y_{17} = (8 \wedge 9) \vee (9 \wedge 8)$$

$$y_{18} = 9 \wedge 9$$

D.6 LFL-TYPE1 ON MNIST MULTI-DIGIT SUM

Confusion matrices on MNIST training and test dataset:

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

MNIST Multi-digit Sum LFL-Type1 train

0	0	0	0	0	0	0	0	0	5923	0
1	6742	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	5958	0	0	0	0
3	0	1	0	0	1	0	0	6129	0	0
4	0	0	0	1	0	0	0	0	0	5841
5	0	5419	0	0	0	0	0	2	0	0
6	0	0	0	0	0	0	5916	0	1	1
7	0	0	0	6264	0	1	0	0	0	0
8	1	0	5850	0	0	0	0	0	0	0
9	1	0	0	1	5946	1	0	0	0	0
	0	1	2	3	4	5	6	7	8	9

original MNIST label

predicted label

MNIST Multi-digit Sum LFL-Type1 test

0	1	0	0	2	0	0	0	0	977	0
1	1133	1	0	1	0	0	0	0	0	0
2	1	0	2	4	0	1023	1	1	0	0
3	1	3	1	0	0	1	0	1004	0	0
4	0	0	0	0	6	1	0	0	0	975
5	0	884	0	0	0	2	1	5	0	0
6	2	1	1	0	0	0	950	0	3	1
7	3	0	1	1019	2	2	0	0	1	0
8	0	1	968	0	0	2	1	1	1	0
9	0	4	0	2	998	0	1	0	1	3
	0	1	2	3	4	5	6	7	8	9

original MNIST label

predicted label

1151

Learned formula:

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

$$\begin{aligned}
 y_0 &= (0 \wedge 4) \vee (C_0 \wedge 1 \wedge 9) \vee (C_0 \wedge 5 \wedge 5) \vee (C_0 \wedge 8 \wedge 2) \vee (C_0 \wedge 7 \wedge 3) \vee (C_0 \wedge 9 \wedge 1) \vee \\
 &(C_0 \wedge 2 \wedge 8) \vee (C_0 \wedge 6 \wedge 4) \vee (C_0 \wedge 3 \wedge 7) \vee (C_0 \wedge 0 \wedge 0) \vee (C_0 \wedge 4 \wedge 6) \vee (C_1 \wedge 1 \wedge 8) \vee \\
 &(C_1 \wedge 5 \wedge 4) \vee (C_1 \wedge 8 \wedge 1) \vee (C_1 \wedge 7 \wedge 2) \vee (C_1 \wedge 9 \wedge 0) \vee (C_1 \wedge 2 \wedge 7) \vee (C_1 \wedge 6 \wedge 3) \vee \\
 &(C_1 \wedge 3 \wedge 6) \vee (C_1 \wedge 0 \wedge 9) \vee (C_1 \wedge 4 \wedge 5) \vee (9 \wedge 2 \wedge 2) \vee (8 \wedge 9 \wedge 6) \vee (7 \wedge 2 \wedge 4) \vee \\
 &(7 \wedge 3 \wedge 0) \vee (C_1 \wedge 7 \wedge 7 \wedge 4) \vee (1 \wedge 9 \wedge 4 \wedge 5) \vee (1 \wedge 2 \wedge 6 \wedge 4) \vee (1 \wedge 2 \wedge 0 \wedge \\
 &1) \vee (1 \wedge 2 \wedge 9 \wedge 4) \vee (7 \wedge 4 \wedge 1 \wedge 0) \vee (1 \wedge 7 \wedge 6 \wedge 0) \vee (5 \wedge 8 \wedge 7 \wedge 4) \vee (C_1 \wedge \\
 &8 \wedge 7 \wedge 4 \wedge 0) \vee (1 \wedge 3 \wedge 1 \wedge 5 \wedge 3) \vee (5 \wedge 9 \wedge 3 \wedge 1 \wedge 3) \vee (5 \wedge 2 \wedge 3 \wedge 1 \wedge 5 \wedge 3) \\
 y_1 &= (C_0 \wedge 1 \wedge 0) \vee (C_0 \wedge 5 \wedge 6) \vee (C_0 \wedge 8 \wedge 3) \vee (C_0 \wedge 7 \wedge 4) \vee (C_0 \wedge 9 \wedge 2) \vee (C_0 \wedge \\
 &2 \wedge 9) \vee (C_0 \wedge 6 \wedge 5) \vee (C_0 \wedge 3 \wedge 8) \vee (C_0 \wedge 0 \wedge 1) \vee (C_0 \wedge 4 \wedge 7) \vee (C_1 \wedge 1 \wedge 9) \vee \\
 &(C_1 \wedge 5 \wedge 5) \vee (C_1 \wedge 8 \wedge 2) \vee (C_1 \wedge 7 \wedge 3) \vee (C_1 \wedge 9 \wedge 1) \vee (C_1 \wedge 2 \wedge 8) \vee (C_1 \wedge 6 \wedge \\
 &4) \vee (C_1 \wedge 3 \wedge 7) \vee (C_1 \wedge 0 \wedge 0) \vee (C_1 \wedge 4 \wedge 6) \vee (1 \wedge 1 \wedge 0) \vee (5 \wedge 6 \wedge 3) \vee (7 \wedge \\
 &3 \wedge 7) \vee (3 \wedge 2 \wedge 0) \vee (3 \wedge 6 \wedge 4) \vee (4 \wedge 9 \wedge 3) \vee (C_1 \wedge 2 \wedge 7 \wedge 4) \vee (C_1 \wedge 1 \wedge 7 \wedge \\
 &4) \vee (5 \wedge 1 \wedge 3 \wedge 4) \vee (7 \wedge 9 \wedge 1 \wedge 5) \vee (7 \wedge 6 \wedge 7 \wedge 2) \vee (7 \wedge 0 \wedge 4 \wedge 3) \vee (2 \wedge \\
 &7 \wedge 2 \wedge 4) \vee (8 \wedge 7 \wedge 2 \wedge 6 \wedge 4) \vee (8 \wedge 6 \wedge 9 \wedge 2 \wedge 0) \vee (3 \wedge 4 \wedge 8 \wedge 7 \wedge 4) \\
 y_2 &= (C_0 \wedge 1 \wedge 1) \vee (C_0 \wedge 5 \wedge 7) \vee (C_0 \wedge 8 \wedge 4) \vee (C_0 \wedge 7 \wedge 5) \vee (C_0 \wedge 9 \wedge 3) \vee (C_0 \wedge 2 \wedge \\
 &0) \vee (C_0 \wedge 6 \wedge 6) \vee (C_0 \wedge 3 \wedge 9) \vee (C_0 \wedge 0 \wedge 2) \vee (C_0 \wedge 4 \wedge 8) \vee (C_1 \wedge 1 \wedge 0) \vee (C_1 \wedge 5 \wedge \\
 &6) \vee (C_1 \wedge 8 \wedge 3) \vee (C_1 \wedge 7 \wedge 4) \vee (C_1 \wedge 9 \wedge 2) \vee (C_1 \wedge 2 \wedge 9) \vee (C_1 \wedge 6 \wedge 5) \vee (C_1 \wedge 3 \wedge \\
 &8) \vee (C_1 \wedge 0 \wedge 1) \vee (C_1 \wedge 4 \wedge 7) \vee (1 \wedge 9 \wedge 3) \vee (1 \wedge 6 \wedge 3) \vee (5 \wedge 0 \wedge 3) \vee (8 \wedge 7 \wedge \\
 &3) \vee (7 \wedge 2 \wedge 3) \vee (1 \wedge 5 \wedge 2 \wedge 6) \vee (1 \wedge 6 \wedge 3 \wedge 0) \vee (5 \wedge 9 \wedge 6 \wedge 3) \vee (7 \wedge 2 \wedge 6 \wedge \\
 &3) \vee (7 \wedge 3 \wedge 0 \wedge 4) \vee (7 \wedge 3 \wedge 1 \wedge 2) \vee (7 \wedge 0 \wedge 1 \wedge 3) \vee (3 \wedge 5 \wedge 7 \wedge 4) \vee (1 \wedge 5 \wedge \\
 &3 \wedge 0) \vee (1 \wedge 7 \wedge 2 \wedge 0) \vee (1 \wedge 2 \wedge 3 \wedge 0) \vee (1 \wedge 8 \wedge 2 \wedge 1 \wedge 3) \vee (3 \wedge 7 \wedge 2 \wedge 3 \wedge 0) \\
 y_3 &= (C_0 \wedge 1 \wedge 2) \vee (C_0 \wedge 5 \wedge 8) \vee (C_0 \wedge 8 \wedge 5) \vee (C_0 \wedge 7 \wedge 6) \vee (C_0 \wedge 9 \wedge 4) \vee (C_0 \wedge 2 \wedge 1) \vee \\
 &(C_0 \wedge 6 \wedge 7) \vee (C_0 \wedge 3 \wedge 0) \vee (C_0 \wedge 0 \wedge 3) \vee (C_0 \wedge 4 \wedge 9) \vee (C_1 \wedge 1 \wedge 1) \vee (C_1 \wedge 5 \wedge 7) \vee (C_1 \wedge \\
 &8 \wedge 4) \vee (C_1 \wedge 7 \wedge 5) \vee (C_1 \wedge 9 \wedge 3) \vee (C_1 \wedge 2 \wedge 0) \vee (C_1 \wedge 6 \wedge 6) \vee (C_1 \wedge 3 \wedge 9) \vee (C_1 \wedge 0 \wedge 2) \vee \\
 &(C_1 \wedge 4 \wedge 8) \vee (5 \wedge 3 \wedge 8) \vee (7 \wedge 4 \wedge 4) \vee (9 \wedge 0 \wedge 5) \vee (3 \wedge 4 \wedge 4) \vee (5 \wedge 2 \wedge 0) \vee (7 \wedge 0 \wedge \\
 &4) \vee (1 \wedge 5 \wedge 8 \wedge 2) \vee (8 \wedge 5 \wedge 2 \wedge 3) \vee (3 \wedge 2 \wedge 6 \wedge 4) \vee (4 \wedge 1 \wedge 7 \wedge 2) \vee (C_1 \wedge 8 \wedge 2 \wedge 1 \wedge \\
 &0) \vee (C_1 \wedge 3 \wedge 1 \wedge 7 \wedge 3) \vee (1 \wedge 4 \wedge 8 \wedge 7 \wedge 4) \vee (9 \wedge 6 \wedge 9 \wedge 0 \wedge 4) \vee (0 \wedge 8 \wedge 7 \wedge 9 \wedge 2) \\
 y_4 &= (C_0 \wedge 1 \wedge 3) \vee (C_0 \wedge 5 \wedge 9) \vee (C_0 \wedge 8 \wedge 6) \vee (C_0 \wedge 7 \wedge 7) \vee (C_0 \wedge 9 \wedge 5) \vee (C_0 \wedge 2 \wedge \\
 &2) \vee (C_0 \wedge 6 \wedge 8) \vee (C_0 \wedge 3 \wedge 1) \vee (C_0 \wedge 0 \wedge 4) \vee (C_0 \wedge 4 \wedge 0) \vee (C_1 \wedge 1 \wedge 2) \vee (C_1 \wedge 5 \wedge \\
 &8) \vee (C_1 \wedge 8 \wedge 5) \vee (C_1 \wedge 7 \wedge 6) \vee (C_1 \wedge 9 \wedge 4) \vee (C_1 \wedge 2 \wedge 1) \vee (C_1 \wedge 6 \wedge 7) \vee (C_1 \wedge \\
 &3 \wedge 0) \vee (C_1 \wedge 0 \wedge 4) \vee (C_1 \wedge 0 \wedge 3) \vee (C_1 \wedge 4 \wedge 9) \vee (1 \wedge 8 \wedge 7) \vee (5 \wedge 9 \wedge 3) \vee (7 \wedge \\
 &2 \wedge 0) \vee (3 \wedge 0 \wedge 4) \vee (3 \wedge 8 \wedge 0) \vee (4 \wedge 5 \wedge 0) \vee (7 \wedge 3 \wedge 0) \vee (1 \wedge 5 \wedge 1 \wedge 6) \vee \\
 &(1 \wedge 5 \wedge 1 \wedge 0) \vee (1 \wedge 9 \wedge 0 \wedge 0) \vee (1 \wedge 6 \wedge 4 \wedge 9) \vee (5 \wedge 3 \wedge 0 \wedge 8) \vee (7 \wedge 6 \wedge 4 \wedge \\
 &0) \vee (9 \wedge 1 \wedge 7 \wedge 9) \vee (0 \wedge 4 \wedge 5 \wedge 4) \vee (0 \wedge 5 \wedge 9 \wedge 0) \vee (4 \wedge 1 \wedge 7 \wedge 2) \vee (5 \wedge \\
 &4 \wedge 1 \wedge 8 \wedge 9) \vee (9 \wedge 2 \wedge 3 \wedge 4 \wedge 9) \vee (2 \wedge 3 \wedge 1 \wedge 6 \wedge 0) \vee (5 \wedge 8 \wedge 9 \wedge 6 \wedge 4)
 \end{aligned}$$

$$\begin{aligned}
1188 \quad & y_5 = (C_0 \wedge 1 \wedge 4) \vee (C_0 \wedge 5 \wedge 0) \vee (C_0 \wedge 8 \wedge 7) \vee (C_0 \wedge 7 \wedge 8) \vee (C_0 \wedge 9 \wedge 6) \vee (C_0 \wedge 2 \wedge \\
1189 \quad & 3) \vee (C_0 \wedge 6 \wedge 9) \vee (C_0 \wedge 3 \wedge 2) \vee (C_0 \wedge 0 \wedge 5) \vee (C_0 \wedge 4 \wedge 1) \vee (C_1 \wedge 1 \wedge 3) \vee (C_1 \wedge 5 \wedge \\
1190 \quad & 9) \vee (C_1 \wedge 8 \wedge 6) \vee (C_1 \wedge 7 \wedge 7) \vee (C_1 \wedge 9 \wedge 5) \vee (C_1 \wedge 2 \wedge 2) \vee (C_1 \wedge 6 \wedge 8) \vee (C_1 \wedge 3 \wedge 1) \vee \\
1191 \quad & (C_1 \wedge 0 \wedge 4) \vee (C_1 \wedge 4 \wedge 0) \vee (7 \wedge 2 \wedge 4) \vee (7 \wedge 3 \wedge 0) \vee (C_1 \wedge 7 \wedge 4 \wedge 5) \vee (C_1 \wedge 3 \wedge 8 \wedge \\
1192 \quad & 0) \vee (2 \wedge 5 \wedge 7 \wedge 2) \vee (6 \wedge 1 \wedge 8 \wedge 7) \vee (C_1 \wedge 9 \wedge 5 \wedge 3 \wedge 0) \vee (C_1 \wedge 0 \wedge 8 \wedge 2 \wedge 3) \vee (9 \wedge \\
1193 \quad & 2 \wedge 1 \wedge 8 \wedge 0) \vee (1 \wedge 8 \wedge 7 \wedge 2 \wedge 3) \vee (C_1 \wedge 5 \wedge 9 \wedge 2 \wedge 4 \wedge 6) \vee (9 \wedge 4 \wedge 7 \wedge 9 \wedge 2 \wedge 3) \\
1194 \quad & y_6 = (C_0 \wedge 1 \wedge 5) \vee (C_0 \wedge 5 \wedge 1) \vee (C_0 \wedge 8 \wedge 8) \vee (C_0 \wedge 7 \wedge 9) \vee (C_0 \wedge 9 \wedge 7) \vee (C_0 \wedge 2 \wedge \\
1195 \quad & 4) \vee (C_0 \wedge 6 \wedge 0) \vee (C_0 \wedge 3 \wedge 3) \vee (C_0 \wedge 0 \wedge 6) \vee (C_0 \wedge 4 \wedge 2) \vee (C_1 \wedge 1 \wedge 4) \vee (C_1 \wedge 5 \wedge \\
1196 \quad & 0) \vee (C_1 \wedge 8 \wedge 7) \vee (C_1 \wedge 7 \wedge 8) \vee (C_1 \wedge 9 \wedge 6) \vee (C_1 \wedge 2 \wedge 3) \vee (C_1 \wedge 6 \wedge 9) \vee (C_1 \wedge 3 \wedge \\
1197 \quad & 2) \vee (C_1 \wedge 0 \wedge 5) \vee (C_1 \wedge 4 \wedge 1) \vee (5 \wedge 7 \wedge 3) \vee (6 \wedge 0 \wedge 4) \vee (C_1 \wedge 1 \wedge 6 \wedge 3) \vee (1 \wedge 5 \wedge \\
1198 \quad & 1 \wedge 8) \vee (5 \wedge 8 \wedge 9 \wedge 0) \vee (7 \wedge 0 \wedge 4 \wedge 8) \vee (7 \wedge 5 \wedge 2 \wedge 0) \vee (7 \wedge 8 \wedge 7 \wedge 2) \vee (9 \wedge \\
1199 \quad & 4 \wedge 2 \wedge 6) \vee (2 \wedge 4 \wedge 8 \wedge 7) \vee (4 \wedge 5 \wedge 9 \wedge 3) \vee (1 \wedge 5 \wedge 7 \wedge 6) \vee (C_1 \wedge 8 \wedge 3 \wedge 0 \wedge 7) \vee \\
1200 \quad & (1 \wedge 5 \wedge 0 \wedge 4 \wedge 3) \vee (5 \wedge 7 \wedge 2 \wedge 3 \wedge 7) \vee (7 \wedge 6 \wedge 4 \wedge 9 \wedge 4) \vee (8 \wedge 7 \wedge 1 \wedge 8 \wedge 7 \wedge 4) \\
1201 \quad & y_7 = (C_0 \wedge 1 \wedge 6) \vee (C_0 \wedge 5 \wedge 2) \vee (C_0 \wedge 8 \wedge 9) \vee (C_0 \wedge 7 \wedge 0) \vee (C_0 \wedge 9 \wedge 8) \vee (C_0 \wedge 2 \wedge \\
1202 \quad & 5) \vee (C_0 \wedge 6 \wedge 1) \vee (C_0 \wedge 3 \wedge 4) \vee (C_0 \wedge 0 \wedge 7) \vee (C_0 \wedge 4 \wedge 3) \vee (C_1 \wedge 1 \wedge 5) \vee (C_1 \wedge 5 \wedge \\
1203 \quad & 1) \vee (C_1 \wedge 8 \wedge 8) \vee (C_1 \wedge 7 \wedge 9) \vee (C_1 \wedge 9 \wedge 7) \vee (C_1 \wedge 2 \wedge 4) \vee (C_1 \wedge 6 \wedge 0) \vee (C_1 \wedge 3 \wedge \\
1204 \quad & 3) \vee (C_1 \wedge 0 \wedge 6) \vee (C_1 \wedge 4 \wedge 2) \vee (1 \wedge 8 \wedge 7) \vee (1 \wedge 6 \wedge 9) \vee (5 \wedge 7 \wedge 4) \vee (8 \wedge 0 \wedge 4) \vee \\
1205 \quad & (2 \wedge 0 \wedge 6) \vee (2 \wedge 7 \wedge 3) \vee (3 \wedge 2 \wedge 0) \vee (5 \wedge 7 \wedge 3) \vee (C_1 \wedge 3 \wedge 9 \wedge 3) \vee (C_1 \wedge 7 \wedge 3) \vee \\
1206 \quad & 0) \vee (1 \wedge 8 \wedge 6 \wedge 0) \vee (5 \wedge 8 \wedge 3 \wedge 9) \vee (8 \wedge 2 \wedge 1 \wedge 9) \vee (7 \wedge 9 \wedge 4 \wedge 8) \vee (7 \wedge 6 \wedge 3 \wedge \\
1207 \quad & 9) \vee (7 \wedge 6 \wedge 3 \wedge 0) \vee (1 \wedge 2 \wedge 4 \wedge 5 \wedge 4) \vee (1 \wedge 2 \wedge 2 \wedge 6 \wedge 3) \vee (9 \wedge 6 \wedge 4 \wedge 5 \wedge 0) \\
1208 \quad & y_8 = (C_0 \wedge 1 \wedge 7) \vee (C_0 \wedge 5 \wedge 3) \vee (C_0 \wedge 8 \wedge 0) \vee (C_0 \wedge 7 \wedge 1) \vee (C_0 \wedge 9 \wedge 9) \vee (C_0 \wedge 2 \wedge 6) \vee \\
1209 \quad & (C_0 \wedge 6 \wedge 2) \vee (C_0 \wedge 3 \wedge 5) \vee (C_0 \wedge 0 \wedge 8) \vee (C_0 \wedge 4 \wedge 4) \vee (C_1 \wedge 1 \wedge 6) \vee (C_1 \wedge 5 \wedge 2) \vee (C_1 \wedge \\
1210 \quad & 8 \wedge 9) \vee (C_1 \wedge 7 \wedge 0) \vee (C_1 \wedge 9 \wedge 8) \vee (C_1 \wedge 2 \wedge 5) \vee (C_1 \wedge 6 \wedge 1) \vee (C_1 \wedge 3 \wedge 4) \vee (C_1 \wedge 0 \wedge \\
1211 \quad & 7) \vee (C_1 \wedge 4 \wedge 3) \vee (9 \wedge 6 \wedge 4) \vee (9 \wedge 2 \wedge 4) \vee (1 \wedge 6 \wedge 3) \vee (5 \wedge 9 \wedge 6 \wedge 9) \vee (7 \wedge 4 \wedge \\
1212 \quad & 9 \wedge 4) \vee (9 \wedge 2 \wedge 6 \wedge 4) \vee (9 \wedge 0 \wedge 9 \wedge 4) \vee (9 \wedge 1 \wedge 8 \wedge 4) \vee (3 \wedge 1 \wedge 7 \wedge 3) \vee (4 \wedge 4 \wedge 1 \wedge \\
1213 \quad & 3 \wedge 0) \vee (1 \wedge 7 \wedge 2 \wedge 3) \vee (1 \wedge 9 \wedge 3 \wedge 0) \vee (C_1 \wedge 7 \wedge 2 \wedge 4 \wedge 0) \vee (6 \wedge 3 \wedge 8 \wedge 2 \wedge 3) \\
1214 \quad & y_9 = (C_0 \wedge 1 \wedge 8) \vee (C_0 \wedge 5 \wedge 4) \vee (C_0 \wedge 8 \wedge 1) \vee (C_0 \wedge 7 \wedge 2) \vee (C_0 \wedge 9 \wedge 0) \vee (C_0 \wedge 2 \wedge \\
1215 \quad & 7) \vee (C_0 \wedge 6 \wedge 3) \vee (C_0 \wedge 3 \wedge 6) \vee (C_0 \wedge 0 \wedge 9) \vee (C_0 \wedge 4 \wedge 5) \vee (C_1 \wedge 1 \wedge 7) \vee (C_1 \wedge 5 \wedge \\
1216 \quad & 3) \vee (C_1 \wedge 8 \wedge 0) \vee (C_1 \wedge 7 \wedge 1) \vee (C_1 \wedge 9 \wedge 9) \vee (C_1 \wedge 2 \wedge 6) \vee (C_1 \wedge 6 \wedge 2) \vee (C_1 \wedge 3 \wedge \\
1217 \quad & 5) \vee (C_1 \wedge 0 \wedge 8) \vee (C_1 \wedge 4 \wedge 4) \vee (5 \wedge 0 \wedge 1) \vee (0 \wedge 2 \wedge 0) \vee (7 \wedge 3 \wedge 0) \vee (C_1 \wedge 1 \wedge \\
1218 \quad & 8 \wedge 2) \vee (C_1 \wedge 1 \wedge 3 \wedge 0) \vee (C_1 \wedge 8 \wedge 9 \wedge 1) \vee (1 \wedge 5 \wedge 7 \wedge 0) \vee (5 \wedge 8 \wedge 2 \wedge 4) \vee (8 \wedge \\
1219 \quad & 6 \wedge 0 \wedge 9) \vee (2 \wedge 3 \wedge 9 \wedge 0) \vee (3 \wedge 1 \wedge 8 \wedge 6) \vee (0 \wedge 1 \wedge 7 \wedge 0) \vee (1 \wedge 5 \wedge 6 \wedge 4) \vee \\
1220 \quad & (5 \wedge 2 \wedge 3 \wedge 0) \vee (C_1 \wedge 1 \wedge 6 \wedge 3 \wedge 0) \vee (C_1 \wedge 7 \wedge 2 \wedge 4 \wedge 0) \vee (1 \wedge 9 \wedge 2 \wedge 3 \wedge 4) \\
1221 \quad & C'_0 = (C_0 \wedge 0) \vee (C_0 \wedge 0) \vee (1 \wedge 3) \vee (C_0 \wedge 1 \wedge 1) \vee (C_0 \wedge 1 \wedge 5) \vee (C_0 \wedge 1 \wedge 8) \vee (C_0 \wedge 1 \wedge 7) \vee \\
1222 \quad & (C_0 \wedge 1 \wedge 2) \vee (C_0 \wedge 1 \wedge 6) \vee (C_0 \wedge 1 \wedge 3) \vee (C_0 \wedge 1 \wedge 4) \vee (C_0 \wedge 5 \wedge 1) \vee (C_0 \wedge 5 \wedge 2) \vee (C_0 \wedge \\
1223 \quad & 5 \wedge 3) \vee (C_0 \wedge 5 \wedge 4) \vee (C_0 \wedge 8 \wedge 1) \vee (C_0 \wedge 7 \wedge 1) \vee (C_0 \wedge 7 \wedge 2) \vee (C_0 \wedge 2 \wedge 1) \vee (C_0 \wedge 2 \wedge 5) \vee \\
1224 \quad & (C_0 \wedge 2 \wedge 7) \vee (C_0 \wedge 2 \wedge 2) \vee (C_0 \wedge 2 \wedge 6) \vee (C_0 \wedge 2 \wedge 3) \vee (C_0 \wedge 2 \wedge 0) \vee (C_0 \wedge 2 \wedge 4) \vee (C_0 \wedge \\
1225 \quad & 6 \wedge 1) \vee (C_0 \wedge 6 \wedge 2) \vee (C_0 \wedge 6 \wedge 3) \vee (C_0 \wedge 3 \wedge 1) \vee (C_0 \wedge 3 \wedge 5) \vee (C_0 \wedge 3 \wedge 2) \vee (C_0 \wedge 3 \wedge 6) \vee \\
1226 \quad & (C_0 \wedge 3 \wedge 3) \vee (C_0 \wedge 3 \wedge 4) \vee (C_0 \wedge 0 \wedge 5) \vee (C_0 \wedge 0 \wedge 7) \vee (C_0 \wedge 0 \wedge 2) \vee (C_0 \wedge 0 \wedge 6) \vee (C_0 \wedge \\
1227 \quad & 0 \wedge 4) \vee (C_0 \wedge 4 \wedge 1) \vee (C_0 \wedge 4 \wedge 5) \vee (C_0 \wedge 4 \wedge 2) \vee (C_0 \wedge 4 \wedge 3) \vee (C_0 \wedge 4 \wedge 0) \vee (C_0 \wedge 4 \wedge 4) \vee \\
1228 \quad & (C_1 \wedge 1 \wedge 1) \vee (C_1 \wedge 1 \wedge 5) \vee (C_1 \wedge 1 \wedge 7) \vee (C_1 \wedge 1 \wedge 2) \vee (C_1 \wedge 1 \wedge 6) \vee (C_1 \wedge 1 \wedge 3) \vee (C_1 \wedge \\
1229 \quad & 1 \wedge 0) \vee (C_1 \wedge 1 \wedge 4) \vee (C_1 \wedge 5 \wedge 1) \vee (C_1 \wedge 5 \wedge 2) \vee (C_1 \wedge 5 \wedge 3) \vee (C_1 \wedge 5 \wedge 0) \vee (C_1 \wedge 8 \wedge 0) \vee \\
1230 \quad & (C_1 \wedge 7 \wedge 1) \vee (C_1 \wedge 7 \wedge 0) \vee (C_1 \wedge 2 \wedge 1) \vee (C_1 \wedge 2 \wedge 5) \vee (C_1 \wedge 2 \wedge 2) \vee (C_1 \wedge 2 \wedge 6) \vee (C_1 \wedge \\
1231 \quad & 2 \wedge 3) \vee (C_1 \wedge 2 \wedge 0) \vee (C_1 \wedge 2 \wedge 4) \vee (C_1 \wedge 6 \wedge 1) \vee (C_1 \wedge 6 \wedge 2) \vee (C_1 \wedge 6 \wedge 0) \vee (C_1 \wedge 3 \wedge \\
1232 \quad & 1) \vee (C_1 \wedge 3 \wedge 5) \vee (C_1 \wedge 3 \wedge 2) \vee (C_1 \wedge 3 \wedge 3) \vee (C_1 \wedge 3 \wedge 0) \vee (C_1 \wedge 3 \wedge 4) \vee (C_1 \wedge 0 \wedge 1) \vee \\
1233 \quad & (C_1 \wedge 0 \wedge 5) \vee (C_1 \wedge 0 \wedge 8) \vee (C_1 \wedge 0 \wedge 7) \vee (C_1 \wedge 0 \wedge 2) \vee (C_1 \wedge 0 \wedge 6) \vee (C_1 \wedge 0 \wedge 3) \vee (C_1 \wedge \\
1234 \quad & 0 \wedge 0) \vee (C_1 \wedge 0 \wedge 4) \vee (C_1 \wedge 4 \wedge 1) \vee (C_1 \wedge 4 \wedge 2) \vee (C_1 \wedge 4 \wedge 3) \vee (C_1 \wedge 4 \wedge 0) \vee (C_1 \wedge 4 \wedge 4) \\
1235 \quad & 4) \vee (1 \wedge 7 \wedge 6) \vee (1 \wedge 9 \wedge 4) \vee (5 \wedge 3 \wedge 8) \vee (5 \wedge 5 \wedge 8) \vee (7 \wedge 6 \wedge 9) \vee (7 \wedge 0 \wedge 0) \vee (7 \wedge \\
1236 \quad & 8 \wedge 4) \vee (9 \wedge 2 \wedge 4) \vee (0 \wedge 7 \wedge 6) \vee (1 \wedge 5 \wedge 8 \wedge 0) \vee (1 \wedge 7 \wedge 6 \wedge 4) \vee (1 \wedge 2 \wedge 3 \wedge 4) \vee \\
1237 \quad & (5 \wedge 7 \wedge 7 \wedge 9) \vee (5 \wedge 3 \wedge 3 \wedge 4) \vee (7 \wedge 5 \wedge 7 \wedge 3) \vee (3 \wedge 1 \wedge 9 \wedge 4) \vee (1 \wedge 8 \wedge 9 \wedge 0 \wedge 3) \\
1238 \quad & C'_1 = (8 \wedge 6) \vee (7 \wedge 7) \vee (9 \wedge 7) \vee (3 \wedge 9) \vee (3 \wedge 4) \vee (C_0 \wedge 1 \wedge 9) \vee (C_0 \wedge 5 \wedge 5) \vee (C_0 \wedge \\
1239 \quad & 5 \wedge 8) \vee (C_0 \wedge 5 \wedge 7) \vee (C_0 \wedge 5 \wedge 9) \vee (C_0 \wedge 5 \wedge 6) \vee (C_0 \wedge 8 \wedge 5) \vee (C_0 \wedge 8 \wedge 8) \vee (C_0 \wedge 8 \wedge \\
1240 \quad & 7) \vee (C_0 \wedge 8 \wedge 9) \vee (C_0 \wedge 8 \wedge 2) \vee (C_0 \wedge 8 \wedge 6) \vee (C_0 \wedge 8 \wedge 3) \vee (C_0 \wedge 8 \wedge 4) \vee (C_0 \wedge 7 \wedge 5) \vee \\
1241 \quad & (C_0 \wedge 7 \wedge 8) \vee (C_0 \wedge 7 \wedge 9) \vee (C_0 \wedge 7 \wedge 6) \vee (C_0 \wedge 7 \wedge 3) \vee (C_0 \wedge 7 \wedge 4) \vee (C_0 \wedge 9 \wedge 1) \vee (C_0 \wedge \\
& 9 \wedge 5) \vee (C_0 \wedge 9 \wedge 8) \vee (C_0 \wedge 9 \wedge 9) \vee (C_0 \wedge 9 \wedge 2) \vee (C_0 \wedge 9 \wedge 6) \vee (C_0 \wedge 9 \wedge 3) \vee (C_0 \wedge 9 \wedge
\end{aligned}$$

1242 $(4) \vee (C_0 \wedge 2 \wedge 8) \vee (C_0 \wedge 2 \wedge 9) \vee (C_0 \wedge 6 \wedge 5) \vee (C_0 \wedge 6 \wedge 8) \vee (C_0 \wedge 6 \wedge 7) \vee (C_0 \wedge 6 \wedge 9) \vee$
1243 $(C_0 \wedge 6 \wedge 6) \vee (C_0 \wedge 6 \wedge 4) \vee (C_0 \wedge 3 \wedge 8) \vee (C_0 \wedge 3 \wedge 7) \vee (C_0 \wedge 4 \wedge 8) \vee (C_0 \wedge 4 \wedge 7) \vee (C_0 \wedge$
1244 $4 \wedge 9) \vee (C_0 \wedge 4 \wedge 6) \vee (C_1 \wedge 1 \wedge 2) \vee (C_1 \wedge 1 \wedge 9) \vee (C_1 \wedge 5 \wedge 5) \vee (C_1 \wedge 5 \wedge 8) \vee (C_1 \wedge 5 \wedge$
1245 $7) \vee (C_1 \wedge 5 \wedge 9) \vee (C_1 \wedge 5 \wedge 6) \vee (C_1 \wedge 5 \wedge 4) \vee (C_1 \wedge 8 \wedge 1) \vee (C_1 \wedge 8 \wedge 5) \vee (C_1 \wedge 8 \wedge 8) \vee$
1246 $(C_1 \wedge 8 \wedge 7) \vee (C_1 \wedge 8 \wedge 9) \vee (C_1 \wedge 8 \wedge 2) \vee (C_1 \wedge 8 \wedge 3) \vee (C_1 \wedge 8 \wedge 4) \vee (C_1 \wedge 7 \wedge 5) \vee (C_1 \wedge$
1247 $7 \wedge 8) \vee (C_1 \wedge 7 \wedge 9) \vee (C_1 \wedge 7 \wedge 2) \vee (C_1 \wedge 7 \wedge 6) \vee (C_1 \wedge 7 \wedge 3) \vee (C_1 \wedge 7 \wedge 4) \vee (C_1 \wedge 9 \wedge$
1248 $1) \vee (C_1 \wedge 9 \wedge 5) \vee (C_1 \wedge 9 \wedge 8) \vee (C_1 \wedge 9 \wedge 9) \vee (C_1 \wedge 9 \wedge 2) \vee (C_1 \wedge 9 \wedge 6) \vee (C_1 \wedge 9 \wedge 3) \vee$
1249 $(C_1 \wedge 9 \wedge 0) \vee (C_1 \wedge 9 \wedge 4) \vee (C_1 \wedge 2 \wedge 8) \vee (C_1 \wedge 2 \wedge 7) \vee (C_1 \wedge 2 \wedge 9) \vee (C_1 \wedge 6 \wedge 5) \vee (C_1 \wedge$
1250 $6 \wedge 8) \vee (C_1 \wedge 6 \wedge 7) \vee (C_1 \wedge 6 \wedge 9) \vee (C_1 \wedge 6 \wedge 6) \vee (C_1 \wedge 6 \wedge 3) \vee (C_1 \wedge 6 \wedge 4) \vee (C_1 \wedge 3 \wedge$
1251 $8) \vee (C_1 \wedge 3 \wedge 7) \vee (C_1 \wedge 3 \wedge 6) \vee (C_1 \wedge 0 \wedge 9) \vee (C_1 \wedge 4 \wedge 5) \vee (C_1 \wedge 4 \wedge 8) \vee (C_1 \wedge 4 \wedge 7) \vee$
1252 $(C_1 \wedge 4 \wedge 9) \vee (C_1 \wedge 4 \wedge 6) \vee (1 \wedge 8 \wedge 7) \vee (5 \wedge 0 \wedge 3) \vee (5 \wedge 4 \wedge 5) \vee (5 \wedge 5 \wedge 8) \vee (5 \wedge$
1253 $5 \wedge 9) \vee (8 \wedge 4 \wedge 5) \vee (7 \wedge 4 \wedge 2) \vee (2 \wedge 7 \wedge 0) \vee (6 \wedge 3 \wedge 0) \vee (6 \wedge 1 \wedge 3) \vee (3 \wedge 8 \wedge$
1254 $3) \vee (0 \wedge 2 \wedge 0) \vee (1 \wedge 2 \wedge 0) \vee (8 \wedge 7 \wedge 0) \vee (6 \wedge 3 \wedge 4) \vee (C_1 \wedge 8 \wedge 2 \wedge 4) \vee (C_1 \wedge 6 \wedge$
1255 $4 \wedge 2) \vee (C_1 \wedge 6 \wedge 1 \wedge 8) \vee (C_1 \wedge 0 \wedge 4 \wedge 3) \vee (C_1 \wedge 1 \wedge 7 \wedge 4) \vee (C_1 \wedge 5 \wedge 8 \wedge 2) \vee (C_1 \wedge$
1256 $7 \wedge 2 \wedge 4) \vee (1 \wedge 5 \wedge 2 \wedge 2) \vee (1 \wedge 7 \wedge 2 \wedge 3) \vee (1 \wedge 9 \wedge 0 \wedge 0) \vee (1 \wedge 2 \wedge 9 \wedge 4) \vee (5 \wedge$
1257 $8 \wedge 2 \wedge 8) \vee (5 \wedge 8 \wedge 5 \wedge 0) \vee (5 \wedge 0 \wedge 1 \wedge 9) \vee (8 \wedge 5 \wedge 2 \wedge 3) \vee (7 \wedge 4 \wedge 1 \wedge 0) \vee (9 \wedge$
1258 $2 \wedge 1 \wedge 2) \vee (2 \wedge 4 \wedge 7 \wedge 3) \vee (2 \wedge 8 \wedge 9 \wedge 4) \vee (2 \wedge 8 \wedge 3 \wedge 4) \vee (0 \wedge 4 \wedge 5 \wedge 7) \vee (0 \wedge$
1259 $5 \wedge 7 \wedge 3) \vee (1 \wedge 5 \wedge 3 \wedge 4) \vee (7 \wedge 9 \wedge 0 \wedge 4) \vee (1 \wedge 7 \wedge 3 \wedge 1 \wedge 6) \vee (5 \wedge 2 \wedge 3 \wedge 7)$
1260 $(6) \vee (7 \wedge 9 \wedge 6 \wedge 4 \wedge 7) \vee (7 \wedge 6 \wedge 1 \wedge 7 \wedge 3) \vee (9 \wedge 2 \wedge 4 \wedge 7 \wedge 0) \vee (4 \wedge 8 \wedge 9 \wedge 3 \wedge 4)$

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295