A Appendix

A.1 Experiment Setup

Relying on neural network confidence for corrective example weights

We investigate the λ hyper-parameter for NSIL introduced in Section 3.3 and its effect on the optimisation of the symbolic learner. When NSIL calculates the weights of the corrective examples, two updates occur; one based on $FNR_{H',\theta'}$, and the other based on $CONF_{\theta'}$. λ can be adjusted to vary the effect of the $CONF_{\theta'}$ update in Equations 7 and 8, as we may not want to fully rely on the neural network confidence when the neural network is under-trained. Therefore, we use $\lambda \in \{1, 0.8, 0.6, 0.4, 0.2, 0\}$ and record the accuracy of the learned knowledge at each iteration. We again select the more challenging E9P and HS tasks, and use 40% of the training data in the E9P task. We run NSIL for 10 iterations, with 5 repeats, using different randomly generated seeds. Figure 6 presents the results.

In all tasks, setting $\lambda = 1$ results in the best performance, as NSIL converges to the correct knowledge. This is our justification for using $\lambda = 1$ in Section 3.3. In the E9P task, when $\lambda = 0$ (purple dash-dot line), the accuracy improves throughout training and the symbolic learner converges to the correct knowledge at iteration 7, although has to explore more of the search space to do so. When $0 < \lambda < 1$, accurate knowledge is learned until iteration 4, at which point the accuracy begins to fluctuate. In these cases, the corrective examples with accurate neural network predictions have lower weights, which encourages the symbolic learner to choose a different label than the label given in the training set. When $\lambda = 1$, NSIL maintains the correct knowledge in later stages of training. In both HS tasks, $0 < \lambda \leq 1$ results in similar performance, whereas when $\lambda = 0$, the learned knowledge alternates at each iteration. This is due to the "broad-brush" nature of the FNR in these tasks, as many examples have the same weight and are updated simultaneously. There are only two possible labels for each set configuration, and there are many choices of digits, so the FNR for each label and set configuration applies to many corrective examples. Note this is not the case in the E9P task, as there are 19 labels and fewer choices of digits for each label. In the HS tasks, when $\lambda > 0$, the neural network confidence scores help the symbolic learner to converge to the correct knowledge by providing a positive weight update for specific examples. This breaks down the broad weights set by the FNR.

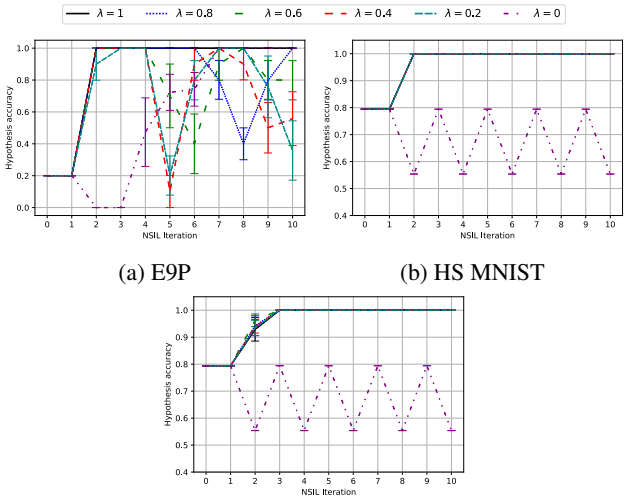
Domain knowledge and corrective examples

ASP encodings for the domain knowledge are presented in Figure 8. Let us now present sample encodings for the corrective examples. In each case, `ID` refers to a unique identifier, and `W` is the example weight calculated by Equations 5 - 8.

Cumulative Arithmetic Let us assume the Addition sub-task, $X = [$  $, 3, 4]$, $y = 8$, and $Z = [1, 3, 4]$. A corrective example pair for this data point is shown in Figures 7a and 7b.

Two-Digit Arithmetic Let us assume the E9P sub-task, $X =$ [ ,  $]$, $y = 1$, and $Z = [2, 1]$. A corrective example pair for this data point is shown in Figures 7c and 7d.

Hitting Set Let us assume the HS sub-task, $X = \{\{$  $\}, \{$  , 3$\}, \{$  $\}\}$, $y = 1$, and $Z = \{\{1\}, \{1, 3\}, \{4\}\}$.



(a) E9P (b) HS MNIST

(c) HS FashionMNIST

Figure 6: Knowledge accuracy with varying λ, E9P and HS tasks. Error bars indicate standard error.

A corrective example pair for this data point is shown in Figures 7e and 7f. Note that $e^{\text{pos}}_{Z,y}$ and $e^{\text{neg}}_{Z,y}$ are implemented with positive and negative LAS examples respectively.

Model details

NSIL CNN The neural component in NSIL is the CNN architecture from [Manhaeve *et al.*, 2018; Yang *et al.*, 2020]. It consists of an encoder with 2 convolutional layers with kernel size 5, and output sizes 6 and 16 respectively. Each convolutional layer is followed by a max pooling layer of size 2, stride 2. The encoder is followed by 3 linear layers of size 120, 84, and 10 (in the hitting set tasks, the last layer is of size 5), and all layers are followed by a ReLU activation function. Finally, a softmax layer returns the output probability distribution.

Baselines The baseline CNN in the Arithmetic tasks follows the baseline CNN architecture from [Manhaeve *et al.*, 2018; Yang *et al.*, 2020], which is largely the same as the NSIL CNN, except the size of the last linear layer is 19, the network accepts a pair of concatenated images as input, and a log softmax layer is used to provide a classification. The CNN-LSTM in the Hitting Set tasks uses the same CNN encoder as NSIL, applied to each image in the input sequence, followed by an LSTM layer of tunable size, a linear layer of size 1, and a sigmoid layer. The CBM and CBM-S baselines have similar architectures, consisting of the same CNN as NSIL, applied to each image separately. The only difference is that the CBM variant doesn't contain a softmax layer for the CNN. In both variants, the CNN is followed by 3 linear layers where the size of the first two layers are tunable and the last layer is of size 19 in the Arithmetic tasks, and 1 in the Hitting Set tasks. In the Addition task, ReLU activation is used after the first two linear layers for the CBM and CBM-S baselines, as this resulted in improved performance. In the other tasks, no non-linearity was used. The CNN-LSTM-NALU and CNN-LSTM-NAC baselines contain the same CNN as NSIL and the other baselines, followed by an LSTM layer of tunable size,

```
#pos(ID@W, { result(8) }, { }, {
   :- result(X), X != 8.
   start_list((4, (3, (1, empty)))).
}).
```

(a) Cumulative Addition $e^{\text{pos}}_{Z,y}$

```
#pos(ID@W, { result }, { }, {
   result :- result(X), X != 8.
   :- result(X), result(Y), Y < X.
   start_list((4, (3, (1, empty)))).
}).
```

(b) Cumulative Addition $e^{\text{neg}}_{Z,y}$

```
#pos(ID@W, { result(1) }, { }, {
   :- result(X), X != 1.
   digit(1, 2). digit(2, 1).
}).
```

(c) E9P $e^{\text{pos}}_{Z,y}$

```
#pos(ID@W, { result }, { }, {
   result :- result(X), X != 1.
   :- result(X), result(Y), Y < X.
   digit(1, 2). digit(2, 1).
}).
```

(d) E9P $e^{\text{neg}}_{Z,y}$

```
#pos(ID@W, { }, { }, {
   ss_element(1,1).
   ss_element(2,1).
   ss_element(2,3).
   ss_element(3,4).
}).
```

(e) HS $e^{\text{pos}}_{Z,y}$

```
#neg(ID@W, { }, { }, {
   ss_element(1,1).
   ss_element(2,1).
   ss_element(2,3).
   ss_element(3,4).
}).
```

(f) HS $e^{\text{neg}}_{Z,y}$

```
empty(empty).
list(L) :- start_list(L).
list(T) :- list((_, T)).
head(L, H) :- list(L), L = (H, _).
tail(L, T) :- list(L), L = (_, T).
add(L, (X+Y, T)) :- list(L), L = (X, (Y, T)).
list(L) :- add(_, L).
mult(L, (X*Y, T)) :- list(L), L = (X, (Y, T)).
list(L) :- mult(_, L).
eq(L, ELT) :- list(L), L = (ELT, empty).
result(R) :- start_list(L), f(L, R).
:- result(X), result(Y), X < Y.
#predicate(base, head/2). #predicate(base, tail/2).
#predicate(base, add/2). #predicate(base, mult/2).
#predicate(base, eq/2). #predicate(base, empty/1).
#predicate(target, f/2).
P(A, B) :- Q(A, B), m1(P, Q).
P(A, B) :- Q(A, C), P(C, B), m2(P, Q).
P(A, B) :- Q(A, C), R(C, B), m3(P, Q, R), Q != R.
P(A, B) :- Q(A, B), R(A, B), m4(P, Q, R), Q != R.
P(A) :- Q(A, B), m5(P, Q, B).
P(A) :- Q(A), m6(P, Q).
P(A, B) :- Q(A), R(A, B), m7(P, Q, R).
P(A, B) :- Q(A, B), R(B), m8(P, Q, R).
#modem(2, m1(target/2, any/2)).
#modem(2, m2(target/2, any/2)).
#modem(3, m3(target/2, any/2, any/2)).
#modem(3, m4(target/2, any/2, any/2)).
#modem(2, m5(target/1, any/2)).
#modem(2, m6(target/1, any/1)).
#modem(3, m7(target/2, any/1, any/2)).
#modem(3, m8(target/2, any/2, any/1)).
```

(a) Cumulative Arithmetic tasks

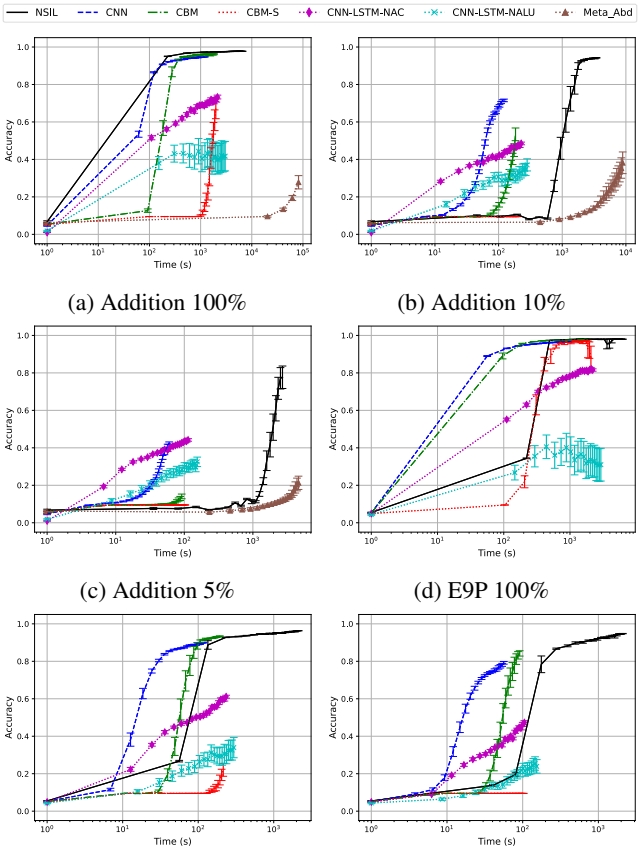
```
r(0..18). d(0..9).
even(X) :- d(X), X \\ 2 = 0.
plus_nine(X1,X2) :- d(X1), X2=9+X1.
res(X1,X2,Y) :- dig(1,X1), dig(2,X2).
:- dig(1,X1),dig(2,X2),res(X1,X2,Y1),res(X1,X2,Y2),
     Y1 != Y2.
#modeh(res(var(d),var(d),var(r))).
#modeb(var(n) = var(d)).
#modeb(var(n) = var(d) + var(d)).
#modeb(plus_nine(var(d),var(r))).
#modeb(even(var(d))).
#modeb(not even(var(d))).
```

(b) Two-Digit Arithmetic tasks

```
s(1..4). h(1..2). e(1..5).
#modeha(hs(var(h), var(e))).
#modeh(hit(var(s))).
#modeb(hs(var(h), var(e)),(positive)).
#modeb(var(e) != var(e)).
#modeb(ss_element(var(s),var(e)),(positive)).
#modeb(ss_element(3,var(e)),(positive)).
#modeb(ss_element(var(s),1),(positive)).
#modeb(hit(var(s))).
```

(c) MNIST Hitting Set tasks

Figure 8: ASP encodings of NSIL domain knowledge.

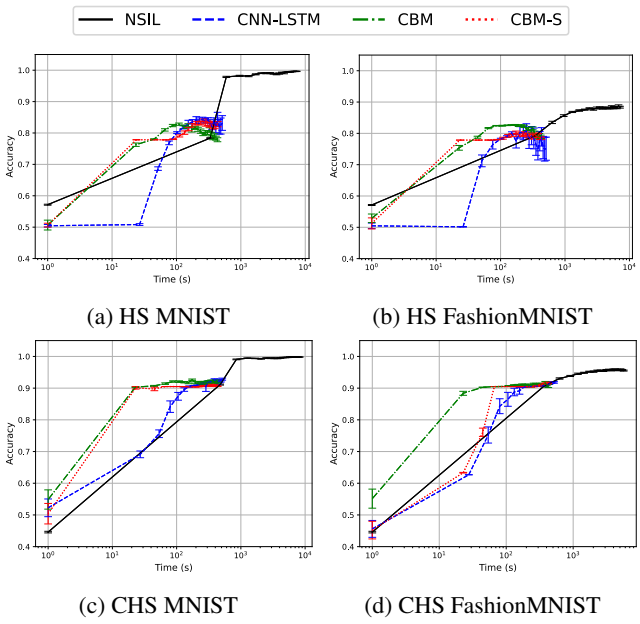


(a) Addition 100% (b) Addition 10%

(c) Addition 5% (d) E9P 100%

(e) E9P 10% (f) E9P 5%

Figure 9: Two-Digit Arithmetic learning time vs. accuracy with reducing training set sizes.



(a) HS MNIST (b) HS FashionMNIST

(c) CHS MNIST (d) CHS FashionMNIST

Figure 10: Hitting Set learning time vs. accuracy.

and then either an NALU or NAC layer. Finally, we use the stochastic gradient descent optimiser for all neural networks [Ruder, 2016], and tune the learning rate and momentum.[4]

Machine details

All experiments are performed on the following infrastructure: RHEL 8.5 x86_64 with Intel Xeon E5-2667 CPUs (20 cores total), and an NVIDIA A100 GPU, 150GB RAM.

A.2 Learning Time Comparison

Figures 9 and 10 show the learning time vs. accuracy comparison for the Arithmetic and Hitting Set tasks respectively. For each method we plot the time taken to complete 20 epochs, and each point shows the accuracy after an epoch of training (1 epoch = 1 NSIL iteration). Error bars indicate standard error. As observed in both task domains, NSIL requires more time to complete 20 epochs, but achieves a greater accuracy than the neural methods. Finally, NSIL has comparable learning time to $Meta_{Abd}$ on the addition tasks.

[4]Please refer to https://github.com/DanCunnington/NSIL for details of all the hyper-parameters used in our experiments.

A.3 Asset Licenses

The ILASP system is free to use for research,[5] FastLAS[6] and the FashionMNIST dataset[7] are both open-source with an MIT license, the MNIST dataset is licensed with Creative Commons Attribution-Share Alike 3.0,[8] and the CNN models used from DeepProbLog are open-source and licensed with Apache 2.0.[9]

[5]https://ilasp.com/terms

[6]https://github.com/spike-imperial/FastLAS/blob/master/LICENSE

[7]https://github.com/zalandoresearch/fashion-mnist/blob/master/LICENSE

[8]See bottom paragraph: https://keras.io/api/datasets/mnist/

[9]https://github.com/ML-KULeuven/deepproblog/blob/master/LICENSE