A Appendix

We introduce more implementation details and experimental results in the following sub-sections.

A.1 Parallel Abduction

As described in section 3.2, $Meta_{Abd}$ tries to estimate the most probable z by abduction following Equation 3. Given training data $D = \{\langle x_i, y_i \rangle\}_{i=1}^{n}$, let $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{y} = (y_1, \ldots, y_n)$ and $\boldsymbol{z} = (z_1, \ldots, z_n)$, we have the posterior of $H \cup \boldsymbol{z}$ as follows:

$$\begin{aligned} P(H, \boldsymbol{z}|B, \boldsymbol{x}, \boldsymbol{y}, \theta) &\propto P(H, \boldsymbol{y}, \boldsymbol{z}|B, \boldsymbol{x}, \theta) \\ &= P(\boldsymbol{y}|B, H, \boldsymbol{z}) P_{\sigma^*}(H|B) P_\theta(\boldsymbol{z}|\boldsymbol{x}) \\ &= P_{\sigma^*}(H|B) \prod_{i=1}^{n} P(y_i|B, H, z_i) P_\theta(z_i|x_i), \end{aligned} \tag{6}$$

where the last equation holds because the examples are drawn i.i.d. from the underlying distribution. Therefore, the logical abduction in the expectation step of $Meta_{Abd}$ can be parallelised naturally:

1. Sample an abductive hypothesis H from the prior distribution $H \sim P_{\sigma^*}(H|B)$;
2. Parallelly abduce z_i from H and $\langle x_i, y_i \rangle$, and then calculate their scores by Equation 5;
3. Aggregate the results by Equation 6;
4. Get the best $H \cup \boldsymbol{z}$ and continue the maximisation step to optimise θ.

We applied this strategy in our implementation of $Meta_{Abd}$ and have achieved better efficiency on multi-threaded CPUs.

A.2 MNIST Cumulative Sum/Product

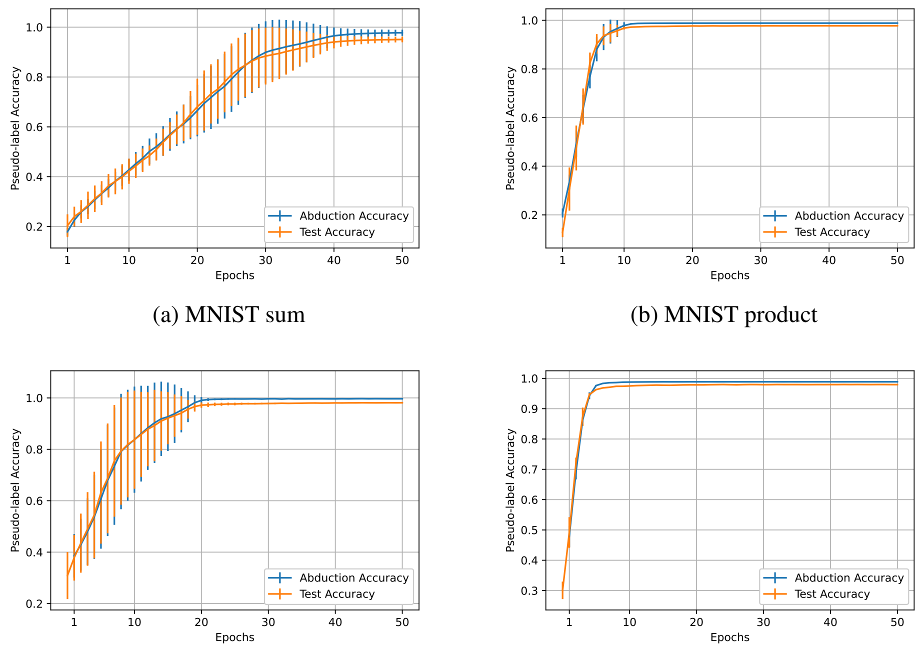
```
% Non-abducible primitives of list operations.
head([H|_],H).
tail([_|T],T).
empty([]).

% Abducible primitives for generating CLP constraints.
abduce_add([X,Y|T],[N|T],Abduced,1.0):-
     (not(ground(N)) ->
         metagol:new_var(N); number(N)),
    atomics_to_string([X,'+',Y,'#=',N], Abduced).
abduce_mult([X,Y|T],[N|T],Abduced,1.0):-
     (not(ground(N)) ->
         metagol:new_var(N); number(N)),
    atomics_to_string([X,'*',Y,'#=',N], Abduced).
abduce_eq([X|T],[N|T],Abduced,1.0):-
     (not(ground(N)) ->
         metagol:new_var(N); number(N)),
    atomics_to_string([X,'#=',N], Abduced).
```

Figure 4: Background knowledge used in the MNIST cumulative sum/product tasks.

The background knowledge used in the MNIST cumulative sum/product experiments is shown in Figure 4. We demonstrate how it works by the following example.

Example (Constraint abduction) Given a training example `f([` `,` `,` `,` `],15)`, $Meta_{Abd}$ will try to learn a program of the dyadic predicate `f` to satisfy (i.e., logically prove) the example. The program to be learned is the *abductive* hypothesis H. The learning process is similar to generic Meta-Interpretive Learning (Muggleton et al., 2014) except that it abduces some ground expressions (the `Abduced` atom in Figure 4) according to the definition of the abducible primitives. In the



(a) MNIST sum (b) MNIST product

(c) MNIST sum with 1-shot CNN pre-train (d) MNIST product with 1-shot CNN pre-train

Figure 5: Pseudo-label accurracy during $Meta_{Abd}$ and $Meta_{Abd+\text{1-shot CNN}}$ learning.

MNIST sum/product tasks, the `Abduced` atoms are strings like "X+#=3", which is a CLP(Z)[5] constraint. According to the definition in Figure 4, when the Prolog variable is not grounded (i.e., constant), the abducible variable will create a new variable to represent `N`; if the Prolog variable is grounded to a number, which means it is the final output in our example, then there is no need to generate a new variable to represent it. Assume that the currently sampled H is the cumulative sum program in Figure 3a, then for the example `f([,,,],15)` $Meta_{Abd}$ can abduce four CLP(Z) constraints: "+#=N1", "N1+#=N2", "N2+#=N3" and "N3#=15". Note that the scores of the abducibles in Figure 4 are all 1.0, which means that these constraints are *hard constraints* that have to be satisfied.

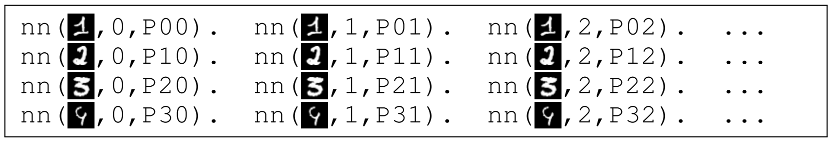
After abducing the constraints, $Meta_{Abd}$ will call the CLP(Z) to solve them, giving a small set of pseudo-labels z that satisfy those constraints. Then, $Meta_{Abd}$ will try to calculate the scores of the abduced $H \cup z$ according to Equation 5. $P_{\sigma^*}(H|B)$ is directly given by H's complexity, i.e., the size of the program; $P_\theta(z|x)$ is given by the probabilistic facts by the perception neural network, which are shown in Figure 6. The predicate "`nn(Img,Label,Prob)`" means the probability of `Img` being an instance of `Label` is `Prob`. To get the probability of all pseudo-labels of an image sequence, $Meta_{Abd}$ simply multiplies the probabilities of each image:

$$p_\theta(z|x) = \prod_j p_\theta(z_j|x_j),$$

where x_j is the j-th image in x (first argument of predicate `nn`), z_j is the abduced pseudo-label of x_j (second argument of `nn`), and the probability is the third argument of `nn`.

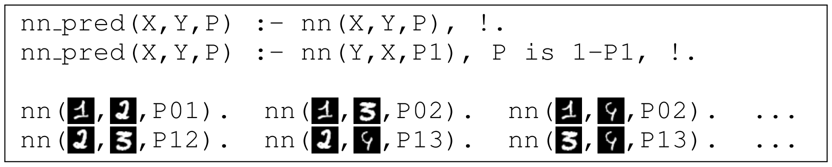
We also report the pseudo-label accuracy of abduction and perception during training, which are shown in Figure 5. The blue lines are the accuracy of the abduced labels (i.e., the accuracy of the expectation of z) in each EM iteration; the orange lines are the accuracy of the perceptual neural net's classification accuracy on the MNIST test set. As we can observe, the convergence speed of cumulative sum is slower, because its the posterior distribution on pseudo-labels ($P(H, z|B, x, y, \theta)$) is much denser than that of cumulative product. After applying the 1-shot CNN pre-train, whose test

[5]https://github.com/triska/clpz



```
nn( ,0,P00).   nn( ,1,P01).   nn( ,2,P02).   ...
nn( ,0,P10).   nn( ,1,P11).   nn( ,2,P12).   ...
nn( ,0,P20).   nn( ,1,P21).   nn( ,2,P22).   ...
nn( ,0,P30).   nn( ,1,P31).   nn( ,2,P32).   ...
```

Figure 6: Monadic probabilistic facts generated by neural network in the sum/product tasks.



```
nn_pred(X,Y,P) :- nn(X,Y,P), !.
nn_pred(X,Y,P) :- nn(Y,X,P1), P is 1-P1, !.

nn( , ,P01).   nn( , ,P02).   nn( , ,P02).   ...
nn( , ,P12).   nn( , ,P13).   nn( , ,P13).   ...
```

Figure 7: Dyadic probabilistic facts generated by neural network in the sorting task.

```
% List operations.
head([H|_],H).
tail([_|T],T).
empty([]).

% Background knowledge about permutation
permute(L1,O,L2):-
    length(L1,N),
    findall(S,between(1,N,S),O1),
    % generate permutation with Prolog's built-in predicate
    catch(permutation(O1,O),_,fail),
    permute1(L1,O,L2).
% permute the image list with order O
permute1([],[],_).
permute1([S|List],[O|Os],List2):-
    nth1(O,List2,S),
    permute1(List,Os,List2).

% Abducible primitives.
abduce_nn_pred([X,Y|_],nn_pred(X,Y),Score):-
    nn_pred(X,Y,Score).
```

Figure 8: Background knowledge used in the MNIST sorting task.

accuracy is shown at 0 epoch in the figures, the convergence speed of MNIST cumulative sum is significantly improved because the EM algorithm is less-likely to be trapped in local optimums.

A.3 MNIST SORTING

Different to the MNIST cumulative sum/product tasks which learn a perceptual neural network predicting the digit in each single image, in the MNIST sorting task, $Meta_{Abd}$ uses a perceptual neural network to learn an unknown binary relation between two images. Examples are shown in Figure 7. The neural network uses the same convnet as before to take the input from a pair of images (the first two arguments of predicate `nn`), and then a Multi-Layered Perception (MLP) is used to predict the probability `PIJ`. The first two clauses translate the neural network's output `nn` to the probabilistic facts for $Meta_{Abd}$'s abduction.

Example (Dyadic facts abduction) Background knowledge of the MNIST sorting task is shown in Figure 8. Different to the previous example which abduces the label of each input image, in the sorting task, the facts being extracted from raw data are dyadic relationship between two images. Given an training example with input $x = [\ ,\ ,\ ,\ ]$, the perceptual neural network will process all the pairwise combinations among them and output a score as shown in Figure 7. Because the pairwise combinations are just a half of pairwise permutations, we also provided a symmetric rule to complete them (the first two clauses in Figure 7). During $Meta_{Abd}$'s induction, the abduced

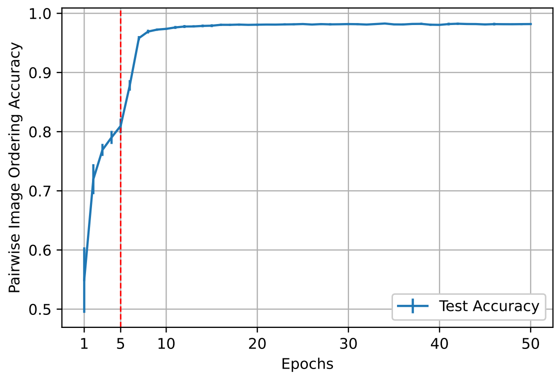


Figure 9: MNIST pairwise ordering (`nn_pred`) accuracy during learning.

```
metarule([P,Q],[P,A],[[Q,A]]).
metarule([P,Q],[P,A],[[Q,A,B],[P,B]]).
metarule([P,Q,R],[P,A],[[Q,A,B],[R,B]]).
metarule([P,Q,R],[P,A,B],[[Q,A],[R,A,B]]).
metarule([P,Q],[P,A,B],[[Q,A,B]]).
metarule([P,Q,R],[P,A,B],[[Q,A,B],[R,A,B]]).
metarule([P,Q,R],[P,A,B],[[Q,A,B,C],[R,C]]).
metarule([P,Q,R],[P,A,B],[[Q,A,B],[R,B]]).
metarule([P,Q,R],[P,A,B],[[Q,A,C],[R,C,B]]).
```

Figure 10: Meta-rules used in all the experiments.

facts are the pairwise probabilistic facts themselves instead of CLP(Z) constraints like before, so the `Score` is the probability of each probabilistic fact. In other words, in the sorting task, *the abduction of z* (the truth values of the probabilistic facts) *is performed simultaneously with logical induction*. Recall the Prolog code of $Meta_{Abd}$ in Figure 2, there is a greedy process that keeps the current most probable abduction with `getmaxprob(Max)` and `setmaxprob(Max)`. The greedy strategy is used to prune the search space of z, it excludes the facts with low probability and quickly find a locally optimal z (truth value assignment), which will be used as pseudo-labels to train the perceptual neural network in the maximisation step.

Figure 9 shows the perception accuracy during training. The test pairs contains 10,000 randomly sampled images from the MNIST test set. The vertical line at epoch 5 shows the time point when $Meta_{Abd}$ switching from the sub-task (learning concept of "sorted" with target predicate `s`) to the main tasks (learning permutation sort). The results in this figure verifies that the perception model is successfully re-used in this experiment.

A.4 Reproducibility

We introduce more experimental details in this subsection. All experiments are completed on a PC with AMD Ryzen 3900X CPU and Nvidia 2080Ti GPU. The data and source codes of $Meta_{Abd}$ will be available after the publication of this work.

A.4.1 META-RULES

The meta-interpreter of $Meta_{Abd}$ uses a set of meta-rules to guide the induction of the logic theory H. We use the meta-rules from the higher-order meta-interpreter $Metagol_{ho}$[6] (Cropper et al., 2020), which are shown in Figure 10.It has been shown that these meta-rules have universal Turing expressivity and can represent higher-order programs (Cropper et al., 2020).

[6]https://github.com/andrewcropper/mlj19-metaho

A.4.2 Neural Network & Hyperparameters

The convnet in our experiments is from PyTorch's MNIST tutorial[7] as Trask et al. (2018) suggested. The LSTM and RNN models in the MNIST cumulative sum/product experiments have 2 hidden layers with dimension 64; the `NAC` and `NALU` modules have 2 hidden layers with dimension 32. In the MNIST sorting experiments, we set the hyperparameter $\tau = 1.0$ for NeuralSort, which is the default value in the original codes[8]. Moreover, the output of NeuralSort is a vector with floating numbers, in order to reproduce the result from the original paper, we rank the output scores to generate the final prediction of orderings.

[7] https://github.com/pytorch/examples/tree/master/mnist

[8] https://github.com/ermongroup/neuralsort