# Supplementary Materials

## A – Details on Parameter Learning

In the supplementary materials, we want to discuss details on **parameter learning** in SLASH. Since we use co-ordinate descent for training SLASH we present the derivative of each component of the loss function defined in equation 10 since while optimization, one component has to be kept fixed.

We start with the gradient of the NPP loss function $L_{NPP}$ i.e. the negative log-likelihood, defined in equation 2

$$\begin{aligned}\frac{\partial}{\partial \xi} L_{NPP} &= \frac{\partial}{\partial x_{Q_i}} \cdot \frac{\partial x_{Q_i}}{\partial \xi} L_{NPP} \\ &= \frac{\partial x_{Q_i}}{\partial \xi} \left( -\sum_{i=1}^{n} \frac{\partial}{\partial x_{Q_i}} \log \left( P_{\xi}^{(X_{\mathbf{Q}}, C)}(x_{Q_i}) \right) \right) \\ &= \frac{\partial x_{Q_i}}{\partial \xi} \left( -\sum_{i=1}^{n} \frac{1}{\left( P_{\xi}^{(X_{\mathbf{Q}}, C)}(x_{Q_i}) \right)} \frac{\partial}{\partial x_{Q_i}} \left( P_{\xi}^{(X_{\mathbf{Q}}, C)}(x_{Q_i}) \right) \right)\end{aligned}$$

Here, we remark that $\frac{\partial x_{Q_i}}{\partial \xi}$ will be carried out by back-propagation and the expression after it is the initial gradient.

Next, we derive the gradient of the logical entailment loss function $L_{ENT}$, as defined in equation 7. One estimates the gradient as follows

$$\frac{1}{n} \frac{\partial}{\partial \mathbf{p}} L_{ENT} \geq -\frac{1}{n} \sum_{i=1}^{n} \frac{\partial \log(P_{\Pi(\theta)}(Q_i))}{\partial \mathbf{p}} \cdot \log(P^{(X_{\mathbf{Q}}, C)}(x_{Q_i})),$$

whereby

- $X_{\mathbf{Q}}$ is the set of random variables associated with the set of the queries $\mathbf{Q}$,
- $x_{Q_i}$ is a training sample, a realization of the set of random variables $X_{\mathbf{Q}}$ associated with the particular query $Q_i$,
- $P^{(X_{\mathbf{Q}}, C)}(x_{Q_i})$ is the probability of the realization $x_{Q_i}$ estimated by the NPP modelling the joint over the set $X_{\mathbf{Q}}$ and $C$ – the set of classes (the domain of the NPP),
- $\log(P_{\Pi(\theta)}(Q_i))$ – the probability of the query $Q_i$ under the program $\Pi(\theta)$ calculated by SLASH (for the reference see the equation $(5)$),
- and $\frac{\partial \log(P_{\Pi(\theta)}(Q_i))}{\partial \mathbf{p}}$ is the gradient as defined in Eq.9.

We begin with the definition of the **cross-entropy** for two vectors $y_i$ and $\hat{y}_i$:

$$H(y_i, \hat{y}_i) := \sum_{j=1}^{m} y_{ij} \cdot \log \left( \frac{1}{\hat{y}_{ij}} \right) = \sum_{j=1}^{m} \left( y_{ij} \cdot \underbrace{\log(1)}_{=0} - y_{ij} \cdot \log(\hat{y}_{ij}) \right) = -\sum_{j=1}^{m} y_{ij} \cdot \log(\hat{y}_{ij}).$$

Hereafter we substitute

$$y_i = \log(P_{\Pi(\theta)}(Q_i)) \qquad \text{and} \qquad \hat{y}_i = P^{(X_{\mathbf{Q}}, C)}(x_{Q_i})$$

and obtain

$$H(y_i, \hat{y}_i) = H \left( \log(P_{\Pi(\theta)}(Q_i)), P^{(X_{\mathbf{Q}}, C)}(x_{Q_i}) \right) = -\sum_{j=1}^{m} \log(P_{\Pi(\theta)}(Q_{ij})) \cdot \log \left( P^{(X_{\mathbf{Q}}, C)}(x_{Q_{ij}}) \right). \tag{11}$$

We remark that $m$ represent the number of classes defined in the domain of an NPP. Now, we differentiate the equation $(11)$ with the respect to $p$ depicted as in Eq. 9 to be the label of the probability of an atom $c = v$ in $r^{npp}$, denoting $P_{\Pi(\boldsymbol{\theta})}(c = v)$. Since differentiation is linear, the product rule is applicable directly:

$$\frac{\partial}{\partial \mathbf{p}} H\left(y_i, \hat{y}_i\right) = -\sum_{j=1}^{m} \left[ \frac{\partial \log(P_{\Pi(\theta)}(Q_{ij}))}{\partial \mathbf{p}} \cdot \log \left( P^{(X_{\mathbf{Q}}, C)}(x_{Q_{ij}}) \right) + \log(P_{\Pi(\theta)}(Q_{ij})) \cdot \frac{\partial \log \left( P^{(X_{\mathbf{Q}}, C)}(x_{Q_{ij}}) \right)}{\partial \mathbf{p}} \right].$$

We do not wish to consider the latter term of $\log(P_{\Pi(\theta)}(Q_i)) \cdot \frac{\partial \log \left( P^{(X_{\mathbf{Q}}, C)}(x_{Q_i}) \right)}{\partial \mathbf{p}}$ because it represents the rescaling and to keep the first since SLASH procure $\frac{\partial \log(P_{\Pi(\theta)}(Q_i))}{\partial \mathbf{p}}$ following Eq. 9. To achieve this, we estimate equation from above downwards as

$$\frac{\partial}{\partial \mathbf{p}} H\left(y_i, \hat{y}_i\right) \geq -\sum_{j=1}^{m} \frac{\partial \log(P_{\Pi(\theta)}(Q_{ij}))}{\partial \mathbf{p}} \cdot \log \left( P^{(X_{\mathbf{Q}}, C)}(x_{Q_{ij}}) \right). \tag{12}$$

Furthermore, let us recall that under i.i.d assumption we obtain from the definition of likelihood

$$LH(y, \hat{y}) = \prod_{i=1}^{n} LH(y_i, \hat{y}_i),$$

and following the negative likelihood coupled with the knowledge that the log-likelihood of $y_i$ is the log of a particular entry of $\hat{y}_i$

$$\begin{aligned} L_{ENT} &= -\log LH(y, \hat{y}) = -\sum_{i=1}^{n} \log LH(y_i, \hat{y}_i) \\ &= -\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} \cdot \log(\hat{y}_{ij}) = \sum_{i=1}^{n} \left[ -\sum_{j=1}^{m} y_{ij} \cdot \log(\hat{y}_{ij}) \right] \\ &= \sum_{i=1}^{n} H(y_i, \hat{y}_i). \end{aligned}$$

Finally, we obtain the following estimate applying inequality (12)

$$\frac{1}{n}\frac{\partial}{\partial \mathbf{p}} L_{ENT} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial \mathbf{p}} H(y_i, \hat{y}_i) \geq -\frac{1}{n}\sum_{i=1}^{n} \frac{\partial \log(P_{\Pi(\theta)}(Q_i))}{\partial \mathbf{p}} \cdot \log\left(P^{(X_{\mathbf{Q}}, C)}(x_{Q_i})\right)$$

Also, we note that the mathematical transformations listed above hold for any type of NPP and the task dependent queries (NN with Softmax, PC or PC jointly with NN). The only difference will be the second term, i.e., $\log(P^{(C|X_{\mathbf{Q}})}(x_{Q_{ij}}))$ or $\log(P^{(X_{\mathbf{Q}}|C)}(x_{Q_{ij}}))$ depending on the NPP and task. The NPP in a form of a single PC modeling the joint over $X_{\mathbf{Q}}$ and $C$ was depicted to be the example. With that, the derivation of gradients for both loss functions 2 and 7 is complete, and the training is carried out by coordinated descent.

**Backpropagation for joint NN and PC NPPs:** If within the SLASH program, $\Pi(\boldsymbol{\theta})$, the NPP forwards the data tensor through a NN first, i.e., the NPP models a joint over the NN's output variables by a PC, then we rewrite (8) to

$$\sum_{i=1}^{n} \frac{\partial \log\left(P_{\Pi(\boldsymbol{\theta})}(Q_i)\right)}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{n} \frac{\partial \log\left(P_{\Pi(\boldsymbol{\theta})}(Q_i)\right)}{\partial \mathbf{p}} \times \frac{\partial \mathbf{p}}{\partial \boldsymbol{\theta}} \times \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{\gamma}}. \tag{13}$$

Thereby, $\boldsymbol{\gamma}$ is the set of the NN's parameters and $\frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{\gamma}}$ is computed by the backward propagation through the NN.

## B – SLASH Programs

Here, the interested reader will find the SLASH programs which we compiled for our experiments. Figure 5 presents the one for the MNIST Addition task, Figure 7 – for the Generative MNIST Addition task, and Figure 9 – for the set prediction task with slot attention encoder and the subsequent CoGenT test. Note the use of the "+" and "-" notation for indicating whether a random variable is given or being queried for.

```
# Define images
img(i1). img(i2).
# Define Neural-Probabilistic Predicate
npp(digit(X), [0,1,2,3,4,5,6,7,8,9]) :- img(X).
# Define the addition of digits given two images and the resulting sum
addition(A, B, N) :- digit(+A, -N1), digit(+B, -N2), N = N1 + N2.
```

Figure 5: SLASH Program for MNIST addition. The same program was used for the training with missing data.

```
# Is 7 the sum of the digits in img1 and img2?
:- addition(image_id1, image_id2, 7)
```

Figure 6: Example SLASH Query for MNIST addition. The same type of query was used for the training with missing data

```
# Define images
img(i1). img(i2).
# Define Neural-Probabilistic Predicate
npp(digit(X), [0,1,2,3,4,5,6,7,8,9]) :- img(X).
# Define the addition of digits given one images and the resulting sum
# Thereby, we add to additional conditions:
## 1. Make the variable N "safe", i.e., one guarantees that there is only
## a countable mount of stable models to be found: N = 0..18
## 2. Insist on the fact that both variables A and B representing images
## are as matter of fact different: A != B
## 3. Since the second image is not available, one has only the prior
## over tha classes P(C) at disposal: digit(-B, -N2)
addition(A, B, N) :- digit(+A, -N1), digit(-B, -N2), N = 0..18, A != B, N - N1 = N2.
```

Figure 7: SLASH Program for the generative MNIST addition.

```
# Is img2 the difference between the sum of the digits and the image encoded in img1?
:- addition(image_id1, _, 2)
```

Figure 8: Example SLASH Query for generative MNIST addition.

```
# Define slots
slot(s1). slot(s2). slot(s3). slot(s4).
# Define identifiers for the objects in the image
# (there are up to four objects in one image).
obj(o1). obj(o2). obj(o3). obj(o4).
# Assign each slot to an object identifier
{assign_one_slot_to_one_object(X, O): slot(X)}=1 :- obj(O).
# Make sure the matching is one-to-one between slots
# and objects identifiers.
:- assign_one_slot_to_one_object(X1, O1),
   assign_one_slot_to_one_object(X2, O2),
   X1==X2, O1!=O2.
# Define all Neural-Probabilistic Predicates
npp(color_attr(X), [red, blue, green, grey, brown,
             magenta, cyan, yellow, bg]) :- slot(X).
npp(shape_attr(X), [circle, triangle, square, bg]) :- slot(X).
npp(shade_attr(X), [bright, dark, bg]) :- slot(X).
npp(size_attr(X), [big,small,bg]) :- slot(X).
# Object O has the attributes C and S and H and Z if ...
has_attributes(O, C, S, H, Z) :- slot(X), obj(O),
                 assign_one_slot_to_one_object(X, O),
                 color(+X, -C), shape(+X, -S),
                 shade(+X, -H), size(+X, -Z).
```

Figure 9: SLASH Program for ShapeWorld4. The same program was used for the CoGenT experiments.

```
# Does object o1 have the attributes red, circle, bright, small?
:- has_attributes(o1, red, circle, bright, small)
```

Figure 10: Example SLASH Query for ShapeWorld4 experiments. In other words, this query corresponds to asking SLASH: “Is object 1 a small, bright red circle?”.

# C – Experimental Details

## ShapeWorld4 Generation

The ShapeWorld4 and ShapeWorld4 CoGenT data sets were generated using the original scripts of (Kuhnle and Copestake 2017) (https://github.com/AlexKuhnle/ShapeWorld). The exact scripts will be added together with the SLASH source code.

## Average Precision computation (ShapeWorld4)

For the baseline slot encoder experiments on ShapeWorld4, we measured the average precision score as in (Locatello et al. 2020). In comparison to the baseline slot encoder, when applying SLASH Attention, however, we handled the case of a slot not containing an object, e.g., only background variables, differently. Whereas (Locatello et al. 2020) add an additional binary identifier to the multi-label ground truth vectors, we have added a background (bg) attribute to each category (*cf.* Fig. 9). A slot is thus considered to be empty (i.e., not containing an object) if each NPP returns the maximal conditional probability for the *bg* attribute.
As the ShapeWorld4 prediction task only included discrete object properties both for Slot Attention and for SLASH Attention, the distance threshold for the average precision computation was infinity (thus corresponding to no threshold).

## Model Details

For those experiments using NPPs with PC, we have used Einsum Networks (EiNets) for implementing the probabilistic circuits. EiNets are a novel implementation design for SPNs introduced by (Peharz et al. 2020) that minimize the issue of computational costs that initial SPNs had suffered. This is accomplished by combining several arithmetic operations via a single monolithic einsum-operation.

For all experiments, the ADAM optimizer (Kingma and Ba 2015) with $\beta 1 = 0.9$ and $\beta 2 = 0.999$, $\epsilon = 1e - 8$ and no weight decay was used.

**MNIST-Addition Experiments** For the MNIST-Addition experiments, we ran the DeepProbLog and NeurASP programs with their original configurations, as stated in (Manhaeve et al. 2018) and (Yang, Ishay, and Lee 2020), respectively. For the SLASH MNIST-Addition experiments, we have used the same neural module as in DeepProbLog and NeurASP, when training SLASH with the neural NPP (SLASH (DNN)) represented in Tab. 2. When using a PC NPP (SLASH (PC)) we have used an EiNet with the Poon-Domingos (PD) structure (Poon and Domingos 2011) and normal distribution for the leafs. The formal hyperparameters for the EiNet are depicted in Tab. 3.

The learning rate and batch size for the DNN were 0.005 and 100, for DeepProbLog, NeurASP and SLASH (DNN). For the EiNet, these were 0.01 and 100.

| Type | Size/Channels | Activation | Comment |
|---|---|---|---|
| Encoder | - | - | - |
| Conv 5 x 5 | 1x28x28 | - | stride 1 |
| MaxPool2d | 6x24x24 | ReLU | kernel size 2, stride 2 |
| Conv 5 x 5 | 6x12x12 | - | stride 1 |
| MaxPool2d | 16x8x8 | ReLU | kernel size 2, stride 2 |
| Classifier | - | - | - |
| MLP | 16x4x4,120 | ReLU | - |
| MLP | 120,84 | ReLU | - |
| MLP | 84,10 | - | Softmax |

Table 2: Neural module – LeNet5 for MNIST-Addition experiments.

| Variables | Width | Height | Number of Pieces | Class count |
|---|---|---|---|---|
| 784 | 28 | 28 | [4,7,28] | 10 |

Table 3: Probabilistic Circuit module – EiNet for MNIST-Addition experiments.

**ShapeWorld4 Experiments** For the baseline slot attention experiments with the ShapeWorld4 data set, we have used the architecture presented in Tab. 4. For further details on this, we refer to the original work of (Locatello et al. 2020). The slot encoder had a number of 4 slots and 3 attention iterations over all experiments.

For the SLASH Attention experiments with ShapeWorld4, we have used the same slot encoder as in Tab. 4, however, we replaced the final MLPs with 4 individual EiNets with Poon-Domingos structure (Poon and Domingos 2011). Their hyperparameters are represented in Tab. 5.

The learning rate and batch size for SLASH Attention were 0.01 and 512, for ShapeWorld4 and ShapeWorld4 CoGenT. The learning rate for the baseline slot encoder were 0.0004 and 512.

| Type | Size/Channels | Activation | Comment |
|---|---|---|---|
| Conv 5 x 5 | 32 | ReLU | stride 1 |
| Conv 5 x 5 | 32 | ReLU | stride 1 |
| Conv 5 x 5 | 32 | ReLU | stride 1 |
| Conv 5 x 5 | 32 | ReLU | stride 1 |
| Position Embedding | - | - | - |
| Flatten | axis: [0, 1, 2 x 3] | - | flatten x, y pos. |
| Layer Norm | - | - | - |
| MLP (per location) | 32 | ReLU | - |
| MLP (per location) | 32 | - | - |
| Slot Attention Module | 32 | ReLU | - |
| MLP | 32 | ReLU | - |
| MLP | 16 | Sigmoid | - |

Table 4: Baseline slot encoder for ShapeWorld4 experiments.

| EiNet | Variables | Width | Height | Number of Pieces | Class count |
|---|---|---|---|---|---|
| Color | 32 | 8 | 4 | [4] | 9 |
| Shape | 32 | 8 | 4 | [4] | 4 |
| Shade | 32 | 8 | 4 | [4] | 3 |
| Size | 32 | 8 | 4 | [4] | 3 |

Table 5: Probabilistic Circuit module – EiNet for ShapeWorld4 experiments.

## D – Additional Results on Missing Pixel MNIST Addition

| | DeepProbLog | **SLASH** (PC) |
|---|---|---|
| 50% | $\mathbf{79.94 \pm 7.2}$ | $72.2 \pm 12.15$ |
| 80% | $31.6 \pm 6.08$ | $\mathbf{44.2 \pm 8.23}$ |
| 90% | $16.94 \pm 1.76$ | $\mathbf{29.6 \pm 5.77}$ |
| 97% | $12.33 \pm 0.47$ | $\mathbf{17.6 \pm 2.97}$ |

Table 6: Additional MNIST Addition Results. Test accuracy corresponds to the percentage of correctly classified test images. Both models (DeepProbLog and SLASH (PC)) were trained on the full MNIST data, but tested on images with missing pixels. Test accuracies are presented in percent. The amount of missing data was varied between 50% and 97% of the pixels per image.

In addition to the setting considered in Evaluation 2, we adjusted the settings for the MNIST Addition task with missing data into the following way.

Training was performed with the full MNIST data set, however the test data set contained different rates of missing pixels. Whereas using an NPP with a PC allows, among other things, to compute marginalization "out of the box" without requiring an update to the architecture or a retraining, this is not so trivial for purely neural-based predicates as in DeepProbLog. Thus, we allowed SLASH (PC) to marginalize over the missing pixels, where this was not directly possible for DeepProbLog.
The results can be seen in Tab. 6 for $50\%$, $80\%$, $90\%$ and $97\%$ of missing pixels per image in the test set. We observe that at $50\%$, DeepProbLog outperforms SLASH by a small margin. For all other rates, we observe that SLASH (PC) reaches significantly higher test accuracies than DeepProbLog. However, we remark that in this setting, SLASH produces larger standard deviations in comparison to DeepProbLog. These results indicate that the conclusions, drawn in the main part of our work, remain true also in this setting of handling missing data.

## E – Computational time Analysis

| | **RTE** (ms) |
|---|---|
| DeepProbLog | $10m : 9s$ |
| NeurASP | $1m : 24s$ |
| **SLASH** (PC) | $1m : 15s$ |
| **SLASH** (DNN) | $\mathbf{0m : 38s}$ |

Table 7: Run time per epoch analysis for MNIST Addition.

Tab. 7 depicts the average run time training epoch (RTE) in ms per DPPL over the whole training window. Also, here we investigate SLASH both with a PC NPP and a DNN NPP. Through an efficient and batch-wise implementation, SLASH (DNN) greatly surpasses both DeepProbLog and NeurASP in terms of RTE, although all three of them use the same DNN. Notably, SLASH (PC) is also much faster than DeepProbLog and slightly faster than NeurASP, although the PC performs full probabilistic inference. These results were measured on a machine equipped with an AMD Ryzen 7 3800X 8-Core Processor and an Nvidia GeForce RTX 2080 Ti GPU. For SLASH, we were using eight threads for parallel calls upon ASP solver.